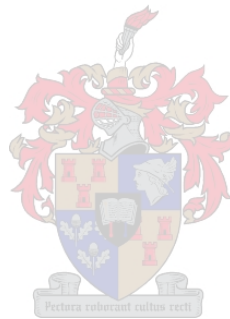


# A decision support system for scheduling the harvesting and wine making processes at a winery

Adri van der Merwe



Thesis presented in partial fulfilment of the requirements for the degree  
**MSc (Operations Research)**  
Department of Logistics,  
University of Stellenbosch, South Africa



# Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

December 2009



# Abstract

Technological advances made over the past century have had a major impact on traditional wineries. Software solutions for management issues are widely available and give rise to the prospect of computerized decision support systems assisting in various aspects of managing a winery. The most popular applications seem to be concerned with supply chain management or harvest scheduling. Such projects are under way all over the globe and great success has been achieved to this effect. However, prior to this study no such project has been considered in South Africa.

The phrase *active cellar scheduling problem* refers to the assignment of grape batches to processors inside the cellar where bottlenecks often occur during the busy harvesting period. The phrase *harvest scheduling problem*, on the other hand, refers to selecting the best possible dates to harvest the respective vineyard blocks in order to preserve grape quality.

A mixed integer programming model for the active cellar scheduling problem is derived in this thesis, but proves to be too time consuming to solve exactly via the branch-and-bound method. A meta-heuristic tabu search approach is therefore designed to solve the problem approximately instead. When applied to a small, fictitious cellar, it is found that the tabu search method often solves the problem optimally. The computer processing time associated with the tabu search approach also constitutes a significant (often thousand-fold) improvement over that of the branch-and-bound approach for realistically sized problem instances.

A generic tabu search is also designed to solve the over-arching harvest scheduling problem for a general winery. This schedule is found by referring to the smaller tabu search of the active cellar scheduling in order to verify the impact that harvesting moves have on activities in the cellar. One harvesting schedule is considered a better schedule than another when it has a lower harvest evaluation score, determined by the placement of the vineyard blocks in the harvesting schedule. The harvest evaluation score takes into account the combination of vineyard blocks selected for harvesting on the same day (and their effect on the active cellar) as well as the ripeness and quality of the grapes.

Both tabu searches are finally included in a flexible, computerized decision support system, called VINDSS. This system is found to produce good harvesting schedules when compared to an actual five day schedule during the 2009 harvesting period at Wamakersvallei, a winery serving as case study for this thesis.



# Uittreksel

Tegnologiese vooruitgang oor die afgelope eeu het 'n groot invloed op tradisionele wynkelders gehad. Sagteware-oplossings wat besluitsteun tot bestuursaanleenthede bied, is algemeen beskikbaar en het gelei tot die rekenaarmatige implementering van besluitsteunstelsels vir wynkelders. Dit blyk dat die mees populêre besluitsteuntoepassings in die wynindustrie te make het met besluite rakende van voorsieningskettings en oes-skedulering. Sulke besluitsteunprojekte is wêreldwyd onderweg en het alreeds groot sukses behaal. Daar is egter tot dusver geen so 'n projek in Suid-Afrika onderneem nie.

Die frase *aktiewe kelderskeduleringsprobleem* verwys na die toekenning van druifvrugte aan masjiene binne die kelder waar bottelnekke algemeen tydens die besige parstydperk voorkom. Die frase *oes-skeduleringsprobleem*, daarenteen, verwys na die seleksie van bes moontlike oesdatums vir elk van die wingerdblokke om sodoende druifkwaliteit te verseker.

'n Gemengde heeltallige programmeringsmodel is vir die aktiewe kelderskeduleringsprobleem ontwikkel, maar die rekenaaroplossingstyd van hierdie benadering blyk te lank te wees om die probleem eksak deur middel van 'n vertak-en-begrens metode op te los. 'n Meta-heuristiese tabu soektog is dus ontwikkel om die probleem benaderd op te los. Wanneer hierdie benadering op 'n klein, fiktiewe kelder toegepas word, word optimale oplossings dikwels verkry. Verder toon die rekenaaroplossingstyd van die tabu soektog 'n groot (in sommige gevalle byna 'n duisendvoudige) verbetering op dié van die eksakte oplossingsmetode.

'n Generiese tabu soektog is ook ontwikkel om die oorkepelende oes-skeduleringsprobleem vir 'n algemene wynkelder op te los. So 'n oes-skedule word gevind deur na die kleiner tabu soektog vir die aktiewe kelderskedulering te verwys om sodoende die effekte van veranderinge in die oes-skedule op die prosesse binne die aktiewe kelder na te speur. Een oes-skedule word beter as 'n ander skedule beskou wanneer dit met 'n beter oes-evalueringswaarde gepaard gaan, soos deur die plasing van die wingerdblokke in die skedule bepaal. Die oes-evalueringswaarde neem die moontlike kombinasies van wingerdblokke wat op dieselfde dag geoes word, in ag (en ook die effek wat dit op aktiwiteite in die kelder het), asook die rypheid en kwaliteit van die druive.

Beide tabu soektogte word in 'n plooibare, rekenaar-geïmplementeerde besluitsteunstelsel, bekend as VINDSS, ingesluit. Daar word gevind dat hierdie stelsel goeie oes-skedules lewer wanneer dit vergelyk word met 'n werklike vyf-dag skedule tydens die 2009 parseisoen van Wamakersvallei, die kelder wat as gevalliestudie vir hierdie tesis gedien het.





# Terms of Reference

Although this project is of a generic nature, Wamakersvallei Winery, situated in Wellington, South Africa, served as a case study for this thesis. The initial problem identified was to focus on the merging of Wamakersvallei Winery with two other Wellington cellars. There would thus be three facilities to which grapes may be transported for processing during the harvesting season. A decision support system was sought to assist in deciding to which facility a vineyard block should be assigned, taking into account the processes required for the production of wine as well as the processors available at each of the facilities. However, the merger never took place and the project was steered in a different direction with the active cellar and harvest scheduling problems currently considered as the main focus. The first two years of the project was funded by the CSIR in the form of a bursary to the author.

Prof JH van Vuuren was the supervisor for this thesis. At the commencement of this project in February 2007, he occupied the position of Associate Professor at the Department of Applied Mathematics of the University of Stellenbosch. During the course of this project, Prof JH van Vuuren was promoted to full Professor and Subject Head of Operations Research and Quantitative Management at the University of Stellenbosch. Dr FE van Dyk, of the Logistics and Quantitative Methods section of the Built Environments Department of The Council for Scientific and Industrial Research (CSIR), acted as co-supervisor of this project. Facilities of both the Applied Mathematics division, Department of Mathematical Sciences, and the Operations Research and Quantitative Management division, Department of Logistics, of the University of Stellenbosch were used during the course of this project which was completed in June 2009.



# Acknowledgements

The author of this thesis hereby wishes to personally express her gratitude towards those who played a significant role during the progress of this thesis:

- Prof van Vuuren for his dedication, guidance and support during the good times and especially for his understanding and patience during the more difficult times throughout the duration of this project and also for his mutual interest and support of the South African wine industry.
- Dr FE van Dyk for the idea of this project and her dedication, guidance and support during the duration of this project, as well as all the kind words of encouragement before workshop presentations and the opportunity to attend such a workshop in Santiago, Chile.
- The Department of Mathematical Sciences (Division of Applied Mathematics) as well as the Department of Logistics (Division Operations Research and Quantitative Management) of the University of Stellenbosch (South Africa), for the use of their computing facilities.
- The Council for Scientific and Industrial Research (CSIR) for funding the first two years of this project and also for granting me the opportunity to represent the CSIR along with Dr FE van Dyk at a workshop in Santiago, Chile.
- Wamakersvallei Winery for their support of this project and their willingness to provide information and data, as well as the occasional (significant) discount on purchases made at their cellar.
- All of my Applied Mathematics and Operations Research office colleagues over the past two and a half years for creating a fun and supportive environment to work in as well as a strong basis of moral and technical support, especially Frank Ortmann for helping with the proofreading of my thesis and for always finding time to help others with L<sup>A</sup>T<sub>E</sub>X emergencies.
- My family whom I love dearly, parents for the moral and financial support over the years and for giving me the opportunity to attain my University of Stellenbosch education, as well as my brother for the support and also the relaxing lunches and conversations empowering me to push through during the difficult times.
- Last but not at all least, my friends for their moral support and for understanding the pressure of the last month during which my time was dedicated only towards completing my thesis, those who joined me for an unforgettable time in Thailand resulting in a refreshed vitality with which I could focus on my thesis and Daneel Smuts who often bore the brunt of my thesis frustration and supported me throughout this thesis more than anyone could realize.

---

---

# Table of Contents

<b>List of Figures</b>	<b>vii</b>
<b>List of Algorithms</b>	<b>ix</b>
<b>List of Tables</b>	<b>xi</b>
<b>Glossary</b>	<b>xv</b>
<b>List of Acronyms</b>	<b>xix</b>
<b>List of Reserved Symbols</b>	<b>xxi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 A brief history of South African wine . . . . .	1
1.2 Informal problem description . . . . .	4
1.3 Scope and objectives . . . . .	5
1.4 Thesis overview . . . . .	5
<b>2 The South African wine industry</b>	<b>7</b>
2.1 Organisational structure of the South African wine industry . . . . .	7
2.1.1 The development of the organisational structure . . . . .	8
2.1.2 The current organisational structure . . . . .	9
2.2 Wine production in South Africa . . . . .	11
2.2.1 Wine producing regions . . . . .	12
2.2.2 Wine production statistics . . . . .	14
2.2.3 The import and export of wine in South Africa . . . . .	17
2.3 From the vine to wine . . . . .	21
2.3.1 Viticulture (grape growing) . . . . .	21
2.3.2 Oenology (wine making) . . . . .	25

---

2.4	Chapter overview . . . . .	31
<b>3</b>	<b>Methodological background</b>	<b>33</b>
3.1	Introduction to the scheduling problem . . . . .	34
3.1.1	Machine environment . . . . .	34
3.1.2	Processing restrictions and constraints . . . . .	36
3.1.3	Performance measures and optimality criteria . . . . .	37
3.1.4	Representation of task sets and the corresponding schedules . . . . .	38
3.2	A concise survey of literature . . . . .	42
3.2.1	The job shop scheduling problem . . . . .	43
3.2.2	Decision support in the wine industry . . . . .	45
3.3	Solving scheduling problems via mathematical programming . . . . .	46
3.3.1	Mixed integer programming in the context of scheduling . . . . .	46
3.3.2	The branch-and-bound method . . . . .	48
3.4	Tabu search methodology . . . . .	50
3.5	Chapter overview . . . . .	53
<b>4</b>	<b>Formal problem statement</b>	<b>55</b>
4.1	Wamakersvallei Winery . . . . .	55
4.1.1	Cellar location and layout . . . . .	55
4.1.2	Cellar machinery and location . . . . .	57
4.1.3	The staff at Wamakersvallei . . . . .	61
4.1.4	The product . . . . .	62
4.2	From the vineyards to the press . . . . .	65
4.2.1	Harvesting the grapes . . . . .	65
4.2.2	Scheduling the arrival of the grapes . . . . .	67
4.2.3	On arrival at the winery . . . . .	68
4.3	Production flow and layout inside the cellar . . . . .	70
4.3.1	Production flow of white grapes . . . . .	71
4.3.2	Production flow of red grapes . . . . .	72
4.4	EzyWine . . . . .	73
4.5	Chapter overview . . . . .	74
<b>5</b>	<b>Mathematical formulation of the cellar scheduling problem</b>	<b>75</b>
5.1	Defining the workspace mathematically . . . . .	75
5.1.1	Jobs and their characteristics . . . . .	76

5.1.2	Processors and their characteristics . . . . .	76
5.1.3	Further parameters and variables . . . . .	77
5.2	Model formulation disregarding pipe assignment . . . . .	80
5.2.1	The constraint sets . . . . .	81
5.2.2	Objective function . . . . .	88
5.3	Chapter overview . . . . .	91
<b>6</b>	<b>Tabu Search solution of the cellar scheduling problem</b>	<b>93</b>
6.1	The initial solution . . . . .	95
6.1.1	Assignment of jobs to the tipping bins . . . . .	95
6.1.2	Further assignment of Type I jobs . . . . .	96
6.1.3	Further assignment of Type II jobs . . . . .	99
6.1.4	Further assignment of Type III jobs . . . . .	100
6.2	Evaluating a solution . . . . .	102
6.2.1	The cellar packing algorithm . . . . .	102
6.2.2	Evaluating an assignment to the red fermentation tanks . . . . .	107
6.3	Generating candidate moves and selecting the best move . . . . .	109
6.3.1	The general ejection chain move and further communal move aspects . . . . .	110
6.3.2	Move Type A: Tipping bins . . . . .	115
6.3.3	Move Type B: separators . . . . .	116
6.3.4	Move Type C: presses . . . . .	117
6.3.5	Move Type D: separator or press assignment of $T_{j_2}$ . . . . .	118
6.3.6	Move Type E: red fermentation tanks . . . . .	119
6.4	Solving the cellar scheduling problem with a tabu search . . . . .	121
6.5	Chapter overview . . . . .	127
<b>7</b>	<b>The harvest scheduling problem at Wamakersvallei Winery</b>	<b>129</b>
7.1	Defining the harvest scheduling attributes . . . . .	129
7.2	The initial harvest schedule . . . . .	131
7.3	Evaluating a harvest schedule . . . . .	133
7.3.1	Creating cellar scheduling scenarios . . . . .	133
7.3.2	Calculating the harvest evaluation score . . . . .	137
7.4	Generating candidate moves and selecting the best move . . . . .	141
7.5	Solving the harvest scheduling problem with the tabu search . . . . .	145
7.6	Chapter overview . . . . .	145

<b>8</b>	<b>The Wamakersvallei decision support system</b>	<b>147</b>
8.1	Required changes to the tabu searches . . . . .	148
8.1.1	A new generation approach for candidate moves on the presses . . . . .	148
8.1.2	Job generation for the active cellar scheduling problem . . . . .	149
8.1.3	Generating moves in the harvest scheduling problem . . . . .	150
8.1.4	Evaluation a harvesting schedule . . . . .	150
8.2	The decision support system applied to Wamakersvallei . . . . .	152
8.2.1	Importing data . . . . .	152
8.2.2	Solving the harvest scheduling problem with VinDSS . . . . .	154
8.2.3	Generating the candidate list of moves . . . . .	158
8.3	A short analysis of the performed tabu search . . . . .	159
8.4	The suggested schedule vs Wamakersvallei selection . . . . .	161
8.5	Chapter overview . . . . .	163
<b>9</b>	<b>Conclusion</b>	<b>165</b>
9.1	Thesis summary . . . . .	165
9.2	Suggestions and recommendations . . . . .	166
9.3	Possible future work . . . . .	167
9.3.1	Improving the cellar and harvest scheduling problems . . . . .	167
9.3.2	Improving the mathematical representation of winery characteristics . . . . .	169
9.3.3	Improving the functionality of VINDSS . . . . .	169
	<b>References</b>	<b>171</b>
<b>A</b>	<b>Processor specifications</b>	<b>181</b>
A.1	Tank capacities and processor numbering . . . . .	181
A.1.1	Tank capacities . . . . .	181
A.1.2	Processor numbers . . . . .	188
A.2	Grape Intakes . . . . .	188
<b>B</b>	<b>Mathematical formulation of the scheduling problem</b>	<b>191</b>
B.1	IP formulation without pipe assignment . . . . .	191
B.2	Solving the problem instance in Example 5.1 . . . . .	193
B.3	The IP formulation including pipe assignment . . . . .	198
<b>C</b>	<b>Wamakervallei harvesting application data</b>	<b>205</b>
C.1	Sample sugar levels . . . . .	205



---

C.2 Example information . . . . .	217
<b>D VinDSS user manual</b>	<b>219</b>
D.1 Importing data from Excel . . . . .	219
D.2 Generating a harvesting schedule from the imported data . . . . .	220



---

# List of Figures

2.1	The institutional organization of the South African wine industry . . . . .	9
2.2	The institutional organization of the South African Wine Industry Council . . . . .	10
2.3	The institutional organization of the South African Wine Industry Trust . . . . .	11
2.4	The South African wine producing regions . . . . .	13
2.5	A schematic illustration of viticulture practices in South Africa . . . . .	22
2.6	A flowchart for making white table wine . . . . .	26
2.7	A flowchart for making dry red table wine . . . . .	30
3.1	The precedence relation of the nine jobs in Example 3.3 . . . . .	40
3.2	A Gantt chart representation of a solution to a scheduling problem . . . . .	40
3.3	A single vertex of the disjunctive graph representation . . . . .	41
3.4	The uncompleted disjunctive graph of Example 3.4 . . . . .	42
3.5	The completed disjunctive graph from Figure 3.4 . . . . .	42
3.6	The branching tree for of Example 3.5 . . . . .	50
4.1	The view of the Wamakersvallei Winery building from the parking area . . . . .	56
4.2	A map of the Cape winelands, with Wellington at the top, middle . . . . .	57
4.3	A floor plan of the cellar layout at Wamakersvallei . . . . .	58
4.4	Tipping bins, crusher and destemmer . . . . .	59
4.5	Stainless steel fermentation tanks . . . . .	61
4.6	Wine bottles displaying two of the Wamakersvallei labels . . . . .	63
4.7	A typical grape harvester . . . . .	66
4.8	A form containing information regarding samples received . . . . .	67
4.9	An example of the list of vineyard blocks selected by the Wamakersvallei team . . . . .	69
4.10	The weighing station at Wamakersvallei . . . . .	70
4.11	The layout of pipes in the Wamakersvallei cellar . . . . .	71
4.12	Production options for white wine at Wamakersvallei wine cellar . . . . .	72

---

4.13	Production options for red wine at Wamakersvallei wine cellar . . . . .	73
5.1	The cellar graph representation of the active cellar at Wamakersvallei Winery . .	78
5.2	The cellar graph for the active cellar used in Example 5.1 . . . . .	86
5.3	Minimum makespan Gantt chart for Example 5.1 . . . . .	89
5.4	Total completion time Gantt chart for Example 5.1 . . . . .	90
6.1	An outline of the Tabu Search applied to the active cellar scheduling problem . .	94
6.2	The initial cellar graph for the active cellar situation in Example 6.5 . . . . .	104
6.3	The final cellar graph for the active cellar from Figure 6.2 . . . . .	105
6.4	The Gantt chart for the active cellar assignment . . . . .	107
8.1	A screen shot of the user interface of VINDSS after importing data . . . . .	153
8.2	A screen shot of the user interface of VINDSS displaying a solution . . . . .	162
B.1	The numbered cellar graph for the small, fictitious active cellar . . . . .	199
D.1	A screen shot of the exact format of the Excel input file . . . . .	220
D.2	A screen shot of the drop down month selection function in VINDSS . . . . .	222

---

# List of Algorithms

3.1	The general framework of a tabu search . . . . .	52
6.1	Initial active cellar assignment for the tipping bins . . . . .	97
6.2	Further initial active cellar assignment for Type I jobs . . . . .	98
6.3	Further initial active cellar assignment for Type II jobs . . . . .	99
6.4	Further initial active cellar assignment for Type III jobs . . . . .	101
6.5	Evaluating a possible solution . . . . .	103
6.6	Initialize times( $\mathbf{o}, \mathbf{t}, \mathbf{f}, m$ ) . . . . .	103
6.7	Update with arrival times( $\mathbf{o}, e$ ) . . . . .	104
6.8	Apply forward move( $\mathbf{o}, \mathbf{o}_m, \mathbf{t}, \mathbf{f}, m_1$ ) . . . . .	106
6.9	Evaluate a red fermentation tank assignment . . . . .	108
6.10	The general ejection chain move . . . . .	111
6.11	Check correct order . . . . .	112
6.12	checkAndAddPair(pairs, $j_1, j_2$ ) . . . . .	113
6.13	Update influenced machine . . . . .	114
6.14	Selecting the best candidate solution . . . . .	115
6.15	Move on tipping bins . . . . .	115
6.16	Move on separators . . . . .	116
6.17	Move on presses . . . . .	117
6.18	Move on $b$ of Type II jobs . . . . .	118
6.19	Move on red fermentation tanks . . . . .	119
6.20	createFeasibleList( $\mathbf{L}_e$ ) . . . . .	120
6.21	createRealListAndEvaluate( $\mathbf{L}_f$ ) . . . . .	121
6.22	The cellar scheduling tabu search . . . . .	122
7.1	Generating the order of assignment for the initial harvesting schedule . . . . .	132
7.2	Generating arrival times . . . . .	137
7.3	Evaluating a harvest schedule . . . . .	139
7.4	determineSugarLevelScore( $\mathbf{H}_x$ ) . . . . .	140

7.5	Generating the list of swap moves . . . . .	141
7.6	<code>createListOfSwapMoves(<math>b, d, p, \mathbf{H}</math>)</code> . . . . .	143
7.7	<code>chooseAndApplyBestMove(<math>\mathbf{L}, \mathbf{\Omega}, b</math>)</code> . . . . .	144
7.8	The harvest scheduling tabu search . . . . .	145
8.1	<code>determineSecondSugarLevelScore(<math>\mathbf{H}_x</math>)</code> . . . . .	152

---

# List of Tables

2.1	Geographic distribution of South African wine grape vineyards . . . . .	12
2.2	South African grape varieties as percentage of total area . . . . .	14
2.3	The distribution between red and white wines produced . . . . .	15
2.4	The white grape varieties as a percentage of total white grape area in South Africa	16
2.5	The red grape varieties as a percentage of total red grape area in South Africa .	16
2.6	The wine production and per capita consumption per country . . . . .	17
2.7	The volume share per country of the world wine exports . . . . .	18
2.8	The value share per country of the world wine exports . . . . .	18
2.9	The volume share per country of the world wine imports . . . . .	19
2.10	The value share per country of the world wine imports . . . . .	19
2.11	Packaged and bulk natural wine exports ranked per receiving country . . . . .	20
2.12	Imports to South Africa, bottled and bulk . . . . .	21
3.1	The predecessor requirements for each of the nine jobs in Example 3.3 . . . . .	39
3.2	The processors ( $P_i$ ) assigned to each of the tasks $T_{jk}$ in Example 3.4 . . . . .	41
3.3	A summary of the optimization algorithms for the scheduling problem . . . . .	43
3.4	A summary of the heuristics for the scheduling problem . . . . .	44
3.5	The characteristics of the single machine scheduling problem of Example 3.6 . . .	52
3.6	The neighbourhood of schedule $s_2$ in Example 3.6 along with the move values . .	53
3.7	The neighbourhood of schedule $s_3$ in Example 3.6 along with the move values . .	53
3.8	The neighbourhood of schedule $s_4$ in Example 3.6 along with the move values . .	53
4.1	The functions and theoretical total capacities (in litres) of the different stores . .	62
4.2	A price list of the La Cave and Bains Way wines . . . . .	63
4.3	A price list of the Wamakersvallei dessert wines and 33° South wines . . . . .	64
4.4	Summary of grape grading according to sugar level . . . . .	68
4.5	Summary of the annual grape intakes at Wamakersvallei cellar . . . . .	70

5.1	The different tasks required in order to process a load of red grapes . . . . .	80
5.2	The different tasks required in order to process a load of white grapes . . . . .	80
5.3	The allowed capacities $c_i$ of the machines ( $P_4, \dots, P_{15}$ ) . . . . .	86
5.4	The jobs to be scheduled and the allowed red fermentation tanks of Example 5.1 . . . . .	87
5.5	The duration of processing $p_{ijk}$ for task $T_{jk}$ in Example 5.1 . . . . .	87
6.1	The distribution of assigning Type I or Type II jobs . . . . .	96
6.2	The final order matrix $\mathbf{o}$ in Example 6.3 in table form . . . . .	101
6.3	The makespan $\mathbf{v}(a)$ for every move $a$ in list $\mathbf{L}_r$ . . . . .	126
6.4	The number of optimal schedules found with the active cellar tabu search . . . . .	126
7.1	The most recent sample sugar levels of the cellar considered in Example 7.1 . . . . .	131
7.2	The frequency of $s_b$ batches for vineyard blocks smaller than 21 tonnes . . . . .	134
7.3	The frequency of $s_b$ batches for vineyard blocks between 21 and 51 tonnes . . . . .	134
7.4	The frequency of $s_b$ batches for vineyard blocks between 51 and 71 tonnes . . . . .	134
7.5	The frequency of batch sizes for vineyard blocks larger than 91 tonnes . . . . .	135
7.6	The resulting jobs after splitting the blocks in Example 7.4 . . . . .	136
7.7	The types 0, 1, 2 and 3 of the harvesting schedule entries . . . . .	142
7.8	The swap move restrictions . . . . .	142
8.1	The list of blocks constructed from the sample sugar levels . . . . .	155
8.2	The initial harvesting schedule, $\mathbf{H}_I$ . . . . .	158
8.3	An analysis of harvesting schedules $\mathbf{H}_{50}$ and $\mathbf{H}_{100}$ . . . . .	160
8.4	An analysis of $\mathbf{H}_{100}$ and the Wamakersvallei harvest . . . . .	163
A.1	Physical capacities of the tanks found in Stores A to F of Wamakersvallei cellar . . . . .	182
A.2	Values of $P_i$ for machinery in the active cellar, excluding fermentation tanks. . . . .	188
A.3	Daily grape intakes at Wamakersvallei Winery from 2000 to 2006 . . . . .	189
B.1	The values of all $\mu_{ijk}$ for the processors, jobs and their tasks in Example 5.1 . . . . .	193
B.2	All non-zero setup times $s_{ij\ell}$ for Example 5.1, expressed in hours . . . . .	194
B.3	The different tasks with pipe assignments to process a load of red grapes . . . . .	198
B.4	The different tasks with pipe assignments to process a load of white grapes . . . . .	198
B.5	The duration $p_{ijk}$ of processing of task $T_{jk}$ regarding pipe assignment . . . . .	200
B.6	All non-zero setup times $s_{ij\ell}$ for Example 5.1, expressed in hours . . . . .	200
B.7	The values of $u_{i_1, i_2}$ when including pipe assignments . . . . .	201
C.1	The sugar levels of the samples received from January 26th to February 3rd . . . . .	205



---

C.2	The sugar levels of the samples received from February 4th to February 12th . . .	211
C.3	Jobs $J_1, \dots, J_{40}$ generated as part of the first scenario in §8.2 . . . . .	218
D.1	The abbreviations used to refer to the cultivars when importing data . . . . .	221



---

# Glossary

- A juice** The free-run juice that is released from a press or separator before pressing.
- B juice** The juice released from a press after grape skins have been pressed.
- Balling** The concentration of a sucrose solution as the weight percentage of sucrose at 17.5°C.
- Brix** A measurement of the dissolved sucrose level.
- Buffer tank** Used to collect juice when grapes are pressed, which may then be transported to fermentation tanks.
- Cooperative cellar** A cellar which works on a communal basis processing grapes of their farmer members into wine. All the grapes are pooled, so that farmers with poor quality grapes benefit most.
- Corporate cellar** A cooperative cellar that has been transformed into a company that may therefore make a profit.
- Distilling wine** Wine that is specially prepared for distillation to spirits and is intended for use in brandy or other spirits, for fortification of wine or for industrial purposes.
- Dedicated processors** A set of processors, each with a specialized function.
- Estate cellar** A small farm with its own cellar where all the grapes used for producing wine must come from the farm. An estate cellar may not buy in any grapes from other growers.
- Feasible active cellar schedule** A schedule for which no two processing time intervals overlap on the same machine in which a number of problem-specific characteristics are met. Multiple processors may be used, but no two time intervals may be allocated to the same *job*.
- Feasible harvest cellar schedule** A harvesting schedule for which no *vineyard block* is assigned to more than one day.
- Fermentation** The process by which sugar is transformed to alcohol.
- Fermentation tank** A stainless steel tank which is temperature-controlled and in which fermentation of grapes occurs.
- Flexible job shop** A combination of the *job shop* and *parallel processors* environments, where a *job* follows its own predetermined route through *work centres*.

**Flexible flow shop** A combination of the *flow shop* and the *parallel processors* environments, where a number of *flow shop stages* operate in series and in which all *jobs* follow the same production route.

**Flow shop** A manufacturing facility where all *jobs* consist of the same number of tasks which follow the same production route on the processors.

**Flow shop stage** A set of parallel processors used in the flexible flow shop machine environment.

**Fortified wine** Non-sparkling wine which has been fortified with wine spirit, including the volume of wine spirit used in the fortification process.

**Job** A load of grapes that is received at a cellar for which processing consists of a fixed number of tasks.

**Job shop** A manufacturing facility where each job follows its own predetermined production route.

**Job shop with machine repetition** A *job shop* where more than one task of the same job is allowed to be processed on a specific processor.

**Lees** Old English for the sediment that settles at the bottom of a container during maturation of wine.

**Maceration** Leaving the partially fermented wine on the skins of red grapes to draw out more tannin, colour and flavour during the red winemaking process.

**Multi-purpose processor** A machine that is equipped with different tools for processing tasks.

**Must** The thick liquid that forms as a result of crushing grapes. It is a mixture of grape juice, stem fragments, grape skins, seeds and pulp.

**Natural wine** Non-fortified and non-sparkling wine, including *perlé* wine. It further includes any grape juice or must (concentrate) used to sweeten such natural wine.

**Non-alcoholic** Refers to unfermented, undiluted or concentrated juice from grapes destined for use in non-alcoholic products such as fruit juices.

**Non-preemptive schedule** A production schedule in which a task may not be interrupted.

**Oenologist** A person who studied oenology and who makes wine as part of their daily activities.

**Oenology** The scientific study of making wine.

**Open shop** A production facility where each *job* has to be processed on each of the *processors*.

**Parallel processors** A set of identical processors in a production facility performing the same function.

**Perlé wine** Wine that is carbonated to the extent that the pressure in the container in which it is sold is between 75 and 300 kPa.

**Phylloxera** Louse-like aphids which attack only grapevines, killing these vines by attacking their roots.

- Pomace** The skins, stalks and seeds that remain after making wine.
- Precedence constraint** A constraint that requires one or more production tasks to be completed before another production task may commence.
- Precedence diagram** A diagram showing elemental tasks and their precedence requirements.
- Preemptive schedule** A schedule allowing the processing of a task to be interrupted at any point in time and a different job to commence processing on the machine instead.
- Press** A machine used to separate juice (or wine) from grapes and grape skins, which does so by applying pressure to grapes or pomace.
- Private cellar** A cellar which is allowed to buy in (all) grapes for the production of its wine.
- Processing time** The time required on a specific processor (machine) to complete a specified job or task.
- Processor** A machine on which a job may be processed in a manufacturing facility.
- Rebate wine** Wine especially prepared for double distillation in a pot still and then, as distillate, matured for a period of at least three years in oak casks with a capacity of no more than 340 litres.
- Release date** The time a job arrives in a production system. Also the earliest time that a job can be processed.
- Recirculation** A *job* in a *job shop* visiting each *processor* at most once.
- Scheduling** The allocation of resources (such as production *centres* or workers) to production tasks over a set period of time.
- Single machine processing** The case where there is only one *processor* in the system and each job to be performed therefore consists of a single *task*.
- Sparkling wine** Wine carbonated (either by fermentation or by impregnation with carbon dioxide) to the extent that the pressure in the container in which it is sold is more than 300 kPa. It includes any grape juice or must (concentrate) used to sweeten such sparkling wine.
- Task** A production process performed by a single *processor* at a production facility. The tasks performed on a batch of production material are referred to collectively as a *job*.
- Terroir** The total natural environment of any viticultural site, including climate, soil and slope of the area.
- Tipping bin** A container into which the grapes are offloaded on arriving at a cellar.
- Uniform processors** A set of parallel processors operating at different speeds.
- Unrelated processors** A set of parallel processors for which the processing speeds are job-dependent.
- Viticulture** That part of horticulture involving the cultivation of grape vines.
- Weight** A real number assigned to a job indicating its priority relative to the other available jobs.

**Work centre** A collection of parallel processors (instead of single processors) used in the pre-determined route of a *job* in a *job centre* within a production facility.

---

## List of Acronyms

- BAWSI** Black Association of the Wine & Spirit Industry
- BEE** Black Economic Empowerment
- BUSCO** wine industry Business Support Company
- DEVCO** wine industry Development Support Company
- KWV** Cape Wine Growers' Co-operation
- NAFU** National African Farmers Union
- RUDNET** Rural Development Network
- SALBA** South African Liquor Brand-owners Association
- SAWB** South African Wine & Brandy Company
- SAWFA** South African Wine Farmers Association
- SAWIS** South African Wine Industry Information & Systems
- SAWIT** South African Wine Industry Trust
- WCSA** Wine Cellars of South Africa
- WCSC** Wine Charter Steering Committee
- WIDA** Wine Industry Development Association
- WIECO** Wine Industry Empowerment Company
- WIP** Wine Industry Strategy Plan
- WOSA** Wines of South Africa





---

## List of Reserved Symbols

The symbols listed below are reserved for a specific use. However, other symbols may be used throughout the thesis in an unreserved fashion.

- $a_{ijk}$  equal to 1 if task  $T_{jk}$  is assigned to machine  $P_i$ , or 0 otherwise.
- $\mathcal{B}$  the set of  $N$  vineyard blocks  $\{B_1, \dots, B_N\}$  under consideration.
- $B_b$  a vineyard block; may consist of a set of  $s_b$  jobs.
- $\mathbf{b}(j)$  equal to 1 if job  $J_j$  is assigned to a separator, or 0 otherwise.
- $c_i$  the capacity of processor  $P_i$ .
- $D$  the number of days for which a harvest scheduling solution is generated.
- $d$  day  $d$  of the harvesting schedule.
- $e_j$  the estimated time of arrival for job  $J_j$ .
- $f_{jk}$  the finishing time of task  $T_{jk}$ .
- $\mathcal{J}$  a set of  $n$  jobs  $\{J_1, J_2, \dots, J_n\}$ .
- $J_j$  a job; may consist of a set of  $k_j$  tasks  $\{T_{j1}, T_{j2}, \dots, T_{jk_j}\}$  when working with multi-operational environments.
- $k_j$  the number of tasks that make up job  $J_j$ .
- $m$  the number of processors available for processing.
- $N$  the number of vineyard blocks under consideration.
- $n$  the number of jobs requiring processing.
- $\mathbf{o}$  the order matrix of a suggested active scheduling problem solution, where row  $i$  refers to processor  $P_i$ , the entries refer to jobs and the columns to the order in which the jobs are processed on processor  $P_i$ .
- $\mathbf{om}$  the machine order matrix, where row  $j$  refers to job  $J_j$ , the entries to processors and the columns to the order in which jobs are assigned to the different processors.
- $\mathbf{ot}$  the tank order matrix, where row  $j$  refers to job  $J_j$  (only of Type I) and the entries to red fermentation tanks.
- $\mathcal{P}$  a set of  $m$  processors  $\{P_1, P_2, \dots, P_m\}$ .
- $P_i$  the  $i$ -th processor.
- $p_{ijk}$  the time required by processor  $P_i$  to process task  $T_{jk}$ .
- $p_{ij}$  the time required in order to process all allowed tasks  $T_{jk}$  ( $k = 1, \dots, k_j$ ) of job  $J_j$  on processor  $P_i$ .
- $q_{j\ell}$  equal to 1 if jobs  $J_j$  and  $J_\ell$  are allowed to be mixed, or 0 otherwise.
- $s_{ij\ell}$  the setup time required to have machine  $P_i$  ready to process any task from job  $J_\ell$  directly after processing a task of job  $J_j$ .
- $\mathcal{T}$  the set of all tasks  $\{T_{11}, \dots, T_{1k_1}, T_{21}, \dots, T_{2k_2}, \dots, T_{n1}, \dots, T_{nk_n}\}$ .
- $T_{jk}$  task  $k$  of job  $J_j$ ; refers to a stage in the processing of wine.
- $t_{jk}$  the starting time of task  $T_{jk}$ .

- $tq_{ij}$  equal to 1 if it is allowed to assign job  $J_j$  to red wine fermentation tank  $P_i$ , or 0 otherwise.
- $u_{i_1i_2}$  equal to 1 if processor  $P_{i_2}$  is allowed to be used directly after processor  $P_{i_1}$ , or 0 otherwise.
- $v_i$  the current volume of tank  $P_i$ .
- $w_j$  the physical weight of job  $J_j$ .
- $x_{ij\ell}$  equal to 1 if a task of job  $J_\ell$  follows a task of Job  $J_j$  directly on processor  $P_i$ , or 0 otherwise.
- $\alpha$  the best completion time uncovered by the cellar scheduling tabu search.
- $\epsilon$  the best red evaluation score uncovered by the cellar assignment tabu search.
- $\kappa$  an index denoting cultivar.
- $\mu_{ijk}$  equal to 1 if task  $T_{jk}$  is allowed to be performed on processor  $P_i$ , or 0 otherwise.
- $\mu_{ij}$  equal to 1 if some task  $T_{jk}$  of job  $J_j$  is allowed to be performed on processor  $P_i$ , or 0 otherwise.
- $\vartheta$  the best harvest evaluation score achieved in the harvest scheduling problem.
- $\Omega$  the harvest evaluation score employed to evaluate harvesting schedules.
- $\Omega_c$  the contribution of a single generated cellar scheduling problem to the harvest evaluation score  $\Omega$ .
- $\Omega_s$  the contribution of expected sugar levels of the blocks scheduled for harvesting with respect to the harvest evaluation score  $\Omega$ .
- $\Omega_{\bar{x}}$  the average contribution of each generated cellar scheduling problem to the harvest evaluation score  $\Omega$ .

---

---

# CHAPTER 1

---

## Introduction

### Contents

1.1 A brief history of South African wine . . . . .	1
1.2 Informal problem description . . . . .	4
1.3 Scope and objectives . . . . .	5
1.4 Thesis overview . . . . .	5

Throughout the ages, wine has been a part of civilization. The earliest evidence of wine production dates back to between 6000 and 5000 BC from archaeological sites in Georgia and Iran. The archaeological evidence from around the third millennium BC points to the domestication of the grapevine in the Early Bronze Age sites of the Near East. In Egypt wine played an important role in ancient ceremonial life and became a part of recorded history. Wine was also common in classical Greece and in the Roman Empire [145]. Wine is both alive in this age and able to link us to civilizations of the past, bestowing all with a sensory stimulation beyond compare. Even more important, wine has proven to be a rewarding and curiously multidisciplinary subject for study. In this chapter the focus of and problems considered in this thesis are introduced, starting with the history of South African wine in §1.1. In §1.2 the problem under investigation is described informally and in §1.3 the scope and objectives of this thesis are given. An overview of the thesis is also given in §1.4.

### 1.1 A brief history of South African wine

The first European settler of South Africa, Jan van Riebeeck, established the Dutch East India Company's fuelling station at the Cape of Good Hope in 1652. He soon after sent to Europe for vine cuttings in the belief that sailors would suffer less from scurvy if they drank wine [42]. In 1655 the first vines arrived, mainly of French origin. Even though he was a rather unwilling pioneer and no expert on winemaking, on 2 February 1659 he recorded: 'Today, praise be to God, wine was pressed for the first time from Cape grapes' [100]. This led to the planting of vines on a larger scale at Roschheuvel, known today as Bishopscourt, Wynberg.

Vine cuttings were distributed amongst servants of the company who had been freed to farm their own land, called Free Burghers, to encourage them to plant vineyards. Initially they were reluctant to do so [154]. There were also many other setbacks in the beginning, mainly because of the ignorance of viticulture amongst the grape farmers. Fine quality wine was not produced

until Simon van der Stel was appointed as van Riebeeck's successor as governor of the Cape in 1679. Van der Stel was a wine enthusiast and very knowledgeable about viticulture and winemaking. He planted a 750 hectare vineyard on his legendary farm Constantia in 1685. The winemaking circumstances were improved even further with the arrival of the French Huguenots in the Cape in 1688. Although they did not have direct winemaking experience, they brought with them their culture and knowledge of vineyard and cellar practice [119]. The Huguenots had very little money, since they were religious refugees and had to make a living with only the bare essentials. They also had to adapt their established French winemaking techniques to the new conditions of the Cape [119]. The Dutch had almost no wine tradition, so the culture and skills of the French left a permanent impression on the South African wine industry. Thanks to Van der Stel, with the help of French Huguenot refugees, the quality of Cape wine started to improve.

Van der Stel made Groot Constantia into a model wine estate and reorganised the local farming community by introducing crop quotas. He also established Stellenbosch<sup>1</sup>, the second oldest Cape colony and the first settlement inland from Cape Town. Simon van der Stel's son, Willem Adriaan, succeeded his father as governor of the Cape. Willem Adriaan contributed greatly to the improvement of viticulture in the Cape and had a Gardener's Almanac which reflected a detailed account of the progress he made [119]. This almanac also serves as the first official record of a vineyard in the Cape. Unfortunately he was despised by the Free Burghers for his tyrannical style and corrupt practices and in 1708 the Free Burghers rebelled. This meant the end of his career as governor of the Cape and he was banished to Holland where he spent the rest of his life in exile [119].

Groot Constantia regained much greater fame when a talented and ambitious grower, Hendrik Cloete, who descended from Van Riebeeck's under gardener, bought the farm in 1788 [119, 100]. They had such great success that the Constantia wines came to fill the glasses of the famous. According to the *The Oxford Companion to Wine* [100], 'their fame was never matched by any other New World wines and at their height they commanded more prestige, more fabulous prices, and enjoyed more crowned patronage than the most celebrated wine of Europe'. Even Napoleon Bonaparte is known to have 'yearned for the sweet wines of Constantia' when he was imprisoned on St. Helena [119].

In 1886 misfortune struck the wine industry of the Cape. The epidemic of louse-like aphids, commonly called *Phylloxera*, destroyed most of the South African and European vines [119]. *Phylloxera* attacks only grapevines and kills vines by attacking their roots. At the time there was no known cure [100]. This led to a 20-year recuperation period and to make up for the lost time, growers had re-established some 80 million high yielding vines, such as Cinsaut, by the early 1900s. This led to an uncontrolled overproduction and with the lack of a market, as well as the Anglo Boer War, the industry erupted into chaos [154]. Surplus wine was literally poured into the local rivers. In order to stabilize the industry, the Cape Wine Growers' Co-operative<sup>2</sup> (KWV) was formed by Western Cape wine farmers, headed by Charles Kohler, on 8 January 1918. They had the legal power to limit production and set minimum prices [100] and their aim was 'to overcome the oppressive constraints of wine surpluses and the exploitation of wine and brandy contractors by improving co-operation and by raising the quality of South African wine and brandy' [76]. This initiative was supported in Parliament by then Prime Minister, General Jan Smuts.

Members of KWV had to sell all their wine through the KWV and contribute a levy of 10% on their sales. An annual surplus of wine was declared by the KWV and removed from the market.

<sup>1</sup>Stellenbosch may be translated directly as 'Stel's forest'.

<sup>2</sup>Known at the time as Koperatieve Wijnbouwers Vereniging van Zuid-Afrika Bpkt.

This ‘surplus’ of wine would be distilled and stored by manufacturing-wholesalers on behalf of KWV who would, in return, not ‘compete with the established wine or spirit trade or distilling or manufacturing interests in Africa south of the equator’ [118].

The KWV almost immediately began constructing new cellars to store brandy and spirits and acquired the distillery and some cellars from the SA Motor Fuel Supply Company near Paarl Station. Depots were established in Stellenbosch, Worcester, Montagu and Robertson. They also began an export trade relationship with Vine Products, a British company [76].

The South African Wine Farmers Association (SAWFA) was founded during the 1930s with KWV and Vine Products each controlling half the shares. A new important trading partnership was formed with Sainsbury’s in Toronto following the Ottawa Agreement. Additional markets opened in Sweden and Norway in 1935 which led to another factory being opened in Worcester for the production of grape juice and KWV Eau De Cologne [76].

In 1939 Europe was destroyed in the throes of the Second World War which solved the immediate problem of surplus brandy disposal [118]. Upon seeing such a gap in the wine market, SAWFA started fanatically updating its distillery and storage facilities. Financial support was offered to Stellenbosch/Elsenburg Agricultural College by the KWV in order to support this proposal. At the same time KWV gained legal control over the minimum price [76].

Also, KWV withdrew from buying grapes for export production which led to farmers needing access to cellars to produce *good wine*, since prices of good wine had increased relative to distilling wine. This caused the number of co-operative wine cellars (who now supplied wine rather than grapes) to increase from 6 to 19 between 1939 and 1944, to 30 by 1950, to 46 by 1955 and finally to 69 by the end of 1975 [118].

In the wake of the war KWV acquired 100% of the shares in SAWFA and started focusing on the improvement of fertilisation and trellising of vines which resulted in larger crops. By 1950s, the harvest had increased from a quarter million hectolitres to three million hectolitres. By the end of the 1950s KWV’s cellars received 12 000 visitors a year [76].

In 1961 white wines were prepared using controlled fermentation techniques for the first time through the initiative of the KWV’s new German cellar master, Willi Hacker. The KWV also reached the interest of the general public by starting public wine courses. Truly phenomenal changes occurred in the wine industry when the KWV provided financial support for the erection of an office and laboratory complex for viticulture and oenological research at Nietvoorbij near Stellenbosch. The European principle of wine and beer sales in restaurants were promoted by the KWV chief executive, Jean de Villiers. Restraining legislation was lifted, allowing the sale of ‘white man’s liquor’ to black, Asian and coloured South Africans.

During the 1980s the South African wine industry was brought to its knees when the United States, the United Kingdom, and 23 other nations passed laws placing various trade sanctions on South Africa. Individual cities and provinces around the world implemented various laws and local regulations forbidding registered corporations under their jurisdiction from doing business with South African firms, factories, or banks [142].

The 1980s also gave way to the formation of both the Brandy Foundation, with the aim to aid the development of premium brandies, as well as the Cape Wine and Spirits Educational Trust with the title of Cape Wine Master being the highest qualification [76]. KWV also installed high-tech bottling lines at the cellar in Paarl in attempt to maintain and improve efficient wine production through new technology.

The 1990s earned the KWV international acclaim, starting with the largest brandy cellar of its kind in the world, the KWV House of Brandy in Worcester, opening its doors to the public in 1992. The following year the KWV celebrated its 75th anniversary and was honoured by being awarded the State President's Export Award. After trade sanctions were lifted in 1994, South African wine exports reached unexpected heights. The conversion of the KWV from a co-operative to a company was set in motion at the start of 1996, directed by a Board chaired by Lourens Jonker and KWV's new managing director, Dr Willem Barnard. The end of 1997 marked the end of this transformation, making the KWV's wine producer members shareholders in the company, KWV Group Limited [76]. As a result of this transformation, The South African Wine Industry Trust was formed, managed in conjunction with the Department of Agriculture and financed by the KWV. The objective of this organisation is to 'gather valuable input over a broad spectrum of issues related to the best application of funds available for wine export, research, employee empowerment and socio-economic upliftment' [76]. Technical services previously provided by the KWV was now continued by a new producer services division called VinPro. An internal restructuring of the group took place in 1999 and resulted in KWV South Africa (Pty) Ltd and KWV International (Pty) Ltd, the latter being awarded the 1999 State President's Award for Export. The current role of the KWV and the other organizations involved in the wine industry today, are further discussed in Chapter 2.

## 1.2 Informal problem description

The South African wine industry has come a long way from the early days of Jan van Riebeeck. Today (2008) there are approximately 4 000 wine producers and 560 wine cellars of which more than 300 were built during the last decade. The majority of these producers and cellars are situated in the Western Cape containing more than 93% of all South African vines [103].

Even though the topic of this thesis is of a generic nature, Wamakersvallei Winery is used as a case study in order to demonstrate workability of the methodology developed. Wamakersvallei Winery is a modern South African winery situated in Wellington (60 km from Cape Town) in the Western Cape Province. As with most cellars, traditional methods of making wine is being replaced by more technologically advanced practices and machinery at Wamakersvallei. These advances require the concurrent introduction of software to assist in the management of such a winery. At Wamakersvallei Winery there is currently the opportunity to assist in two scheduling problems experienced.

Grapes are received from more than 80 different suppliers, each further divided into vineyard blocks. During the harvesting period, roughly spanning mid January to mid April, grape samples are received from the suppliers on a two-weekly basis per vineyard block, and sugar, pH and acidity analyses are performed on these samples. Based on these analyses, the viticulturist and winemaker, along with the cellar manager, manually sort through the large volumes of data in order to agree on a fitting harvesting block selection for each day. This decision is most often made during a tedious meeting occurring at the end of each harvesting day, when deciding upon the selection of blocks to be harvested during the next day.

The main criteria for the selection of vineyard blocks to be harvested on a specific day is that the vineyard blocks selected for harvesting should consist of grapes that are fully ripened and that there should be enough space at the cellar to receive and process the grapes. Two scheduling problems therefore arise. The first is the scheduling of vineyard blocks selected for harvesting over a set period of time rather than only one day in advance and the second is the scheduling of grapeloads being processed on the different machinery inside the harvesting cellar every day.

## 1.3 Scope and objectives

In order to lend decision support to the Wamakersvallei viticulturist and winemakers, three objectives are pursued in this thesis:

- I. To assist the winemaker in the scheduling process of assigning grapeloads from the different producers to the tipping bins and further processors within the active cellar.
- II. To assist the viticulturist in the sometimes difficult scheduling decisions by generating a suggested harvesting schedule (in the sense of suggesting vineyard blocks to be harvested) stretching over a set number of days.
- III. To develop a decision support system which may be used to import sample data from Microsoft Excel [86] and then suggest a harvesting schedule for a set number of days along with the assignment of grapeloads consisting of red grapes to the fermentation tanks. The focus should be on processing the grapes within business hours as well as ensuring the ripeness and retaining the quality of grapes.

## 1.4 Thesis overview

Apart from this introductory chapter, this thesis consists of a further eight chapters. Chapter 2 provides the reader with a basic understanding of wine making and the South African wine industry. More specifically, the development of the South African wine industry organisational structure as well as its current state is discussed. Some statistics are given on wine production in South Africa as well as the import and export of wine. Furthermore, viticultural practices and a basic overview of wine making methods are considered.

In Chapter 3, the necessary background information is presented in order to properly identify the scheduling problems experienced at a winery as well as the methodological possibilities to consider when designing a decision support system for wineries. The chapter contains a review of the notation commonly used in classical scheduling problems, and also throughout this thesis, as well as an overview of the three-field  $\alpha|\beta|\gamma$  notation used to classify classical scheduling problems. Typical representations of schedules are also considered. A concise survey of literature is conducted with respect to the job shop scheduling problem as well as other applications of optimization and decision support in the wine industry. Finally, this chapter also contains an overview of the exact methods with which the scheduling problems are solved in this thesis. The first exact solution method consists of formulating the scheduling problem as a mixed integer programming model and then applying the well-known branch-and-bound method to solve the problem. A meta-heuristic tabu search approach is also considered in this thesis and the methodological background section therefore focusses on the tabu search method as well.

The goal in Chapter 4 is to provide the reader with the necessary information to fully understand the origin of the scheduling problems experienced at Wamakersvallei. In the first section of this chapter, some important aspects of Wamakersvallei are discussed, such as its physical location and current cellar layout, the machinery used, products delivered and staff employed. The work

methods of the employees and the process of ensuring the grape quality that is received at the cellar, is also considered. Furthermore, aspects such as the harvesting process, the scheduling of vineyard blocks to be harvested and the required procedures are described. The different processes and orders of production are presented, focussing mainly on the production of white and red wines, excluding the minority of cases such as Rosé and desert wines. The EzyWine [43] data management system used at Wamakersvallei is also discussed.

The first scheduling problem identified at Wamakersvallei, referred to as the *active cellar scheduling problem*, is concerned with the assignment and ordering of the processes occurring in the part of the cellar identified as the problem area during the harvesting period (*i.e.* where bottlenecks often occur). The purpose of Chapter 5 is to derive a mathematical programming model with which an exact solution may be found to the cellar scheduling problem.

An alternative tabu search approach towards solving the active cellar scheduling problem is developed and outlined in Chapter 6. The goal of this tabu search is to indicate whether or not a feasible production schedule may be found for a specific harvesting day if a given set of vineyard blocks were to be harvested on that day. A solution is feasible if it adheres to the constraint set defined in the mathematical programming model of Chapter 5 and if the solution may be carried out during the business hours of the cellar.

In the seventh chapter, the tabu search method developed in order to solve the second scheduling problem identified at Wamakersvallei, called the *harvest scheduling problem*, is discussed. A good harvesting schedule should ensure that vineyard blocks are harvested as close to their optimal maturity dates as possible. Furthermore, the restrictions imposed by the physical capacity of the cellar, are used in order to further distinguish between harvesting schedules. The measured sugar levels of the vineyard block samples are employed to ensure the ripeness of harvested grapes. The tabu search method of Chapter 6 is used to take into account the physical capacities and time limit imposed by the selected operating hours.

Chapter 8 contains an overview of the newly developed decision support system, called VINDSS. The tabu searches developed in Chapters 6 and 7 are applied to Wamakersvallei 2009 harvesting data, with some minor updates, in order to illustrate the working of VINDSS. A short analysis of the performance of the tabu search is performed and the final harvesting solution is compared to the Wamakersvallei harvest that occurred, based on the same data.

Finally, a short thesis summary is presented in Chapter 9, as well as some future suggestions that may be applied to improve the working of VINDSS and the general tabu search approach implemented in this thesis.



---

---

## CHAPTER 2

---

# The South African wine industry

### Contents

2.1	Organisational structure of the South African wine industry . . . . .	7
2.1.1	<i>The development of the organisational structure</i> . . . . .	8
2.1.2	<i>The current organisational structure</i> . . . . .	9
2.2	Wine production in South Africa . . . . .	11
2.2.1	<i>Wine producing regions</i> . . . . .	12
2.2.2	<i>Wine production statistics</i> . . . . .	14
2.2.3	<i>The import and export of wine in South Africa</i> . . . . .	17
2.3	From the vine to wine . . . . .	21
2.3.1	<i>Viticulture (grape growing)</i> . . . . .	21
2.3.2	<i>Oenology (wine making)</i> . . . . .	25
2.4	Chapter overview . . . . .	31

The South African wine industry is an important economic sector of the country, especially in the Western and Northern Cape provinces. It comprises a R13 billion sector [117], producing 3.3% of the global wine production (number 9 in the world) during 2003, when it was the 8th largest exporter of wine globally. The South African wine industry supports an estimated 350 000 farm workers and their dependants, over 4 500 commercial producers and 3 300 cellar personnel [117]. The annual taxes and excise duties earned by the state from the wine industry may be as high as R1.8 billion. Another R3.5 billion is generated annually in earnings through wine tourism [117]. Hence it is clear that the South African wine industry plays a significant role in the well-being of this country. It is a complex industry that is only now starting to develop the necessary organisational structures.

This chapter contains a more in-depth look at the South African wine industry, starting with its organisational structure in §2.1. Aspects concerned with the production of wine in South Africa as well as some statistics on the subject follow in §2.2 and in §2.3 the actual process of wine making is considered.

## 2.1 Organisational structure of the South African wine industry

The South African wine industry has come a long way since its inception during the 17th century. The development of the organizations involved until the end of the 20th century, which was for

quite some time only the KWV, was briefly discussed in §1.1. A further look will now be taken at the wine industry of the 21st century, first at the development of the associated organisational structure (in §2.1.1) and then at the current organisational structure and its role (in §2.1.2).

### 2.1.1 The development of the organisational structure

When the KWV changed from a co-operative to a company in 1997 the industry embarked on a deregulating process. The first step was that the remaining statutory powers be placed under the control of a body that represented the whole industry, hence the forming of the South African Wine Industry Trust (SAWIT). This meant that by the end of the 20th century the South African wine industry was no longer subject to the restraining structures of regulation that had maintained farm incomes, but repressed innovation [118].

These changes left the future of the industry laying bright ahead. Unfortunately, to date, the industry has only been transformed partially. Large quantities of standard, high-yielding grapes are still produced in order to produce large quantities of cheap wine for which demand is declining. The dependence of the industry on unschooled workers still proves to be an area of concern, as does the lack of opportunities for new entrants to the industry [118].

The deregulation of the wine industry also led to the forming of a vast number of organizations representing the different sectors of the industry and/or performing statutory functions. Examples are Wines of South Africa (WOSA), busying themselves with generic promotion, Winetech, a technical research, development and transfer unit, SA Wine Industry Information & Systems (SAWIS) and so forth. The need was realized to form a head industry body to represent and shape the highly disjointed representative and business structure of the industry. Therefore, the SA Wine & Brandy Company (SAWB) was established in 2002 by VinPro (representatives of wine producers), Wine Cellars of South Africa (WCSA) (representing the cellars), the SA Liquor Brand-owners Association (SALBA) (representatives of the wholesale trade) and BAWSI, the Black Association of the Wine & Spirit Industry.

After the establishment of the SAWB, the Wine Industry Strategy Plan (WIP, approved in 2003 by the Minister of Agriculture) was developed as required by the Ministry of Agriculture [118]. The goals of WIP was to ‘increase global competitiveness and profitability, to generate equitable access and participation within the wine value chain, to enable environmentally sustainable production systems and to promote socially responsible consumption of the produce of the vine’ [117]. In 2003 SAWIT, in coordination with the SAWB as representatives of the wine industry, also drafted a Wine-BEE (Black Economic Empowerment) Charter & Industry Scorecard to set the standard for dealing with matters that may present themselves during the next decade [118]. The structure of the SAWB is shown graphically in Figure 2.1.

Activities of the SAWB were then mainly directed by the WIP. A board of directors consisting of representatives from the wine producers (VinPro), the cellars (WCSA), the workers (BAWSI), the trade (SALBA) and an independent chair oversaw the organizations which may be divided into four business units. The first is the Development Unit whose role concerns the development of human resources, economic development and empowerment. The next category comprises all of the organizations involved in market development and promotion such as WOSA, SA Brandy Foundation and SA National Wine Association. The third category is concerned with knowledge and information systems (including inspection services) coordinated by SAWIS. The last and fourth category is one of technology innovation and transfer, which is coordinated by Winetech.

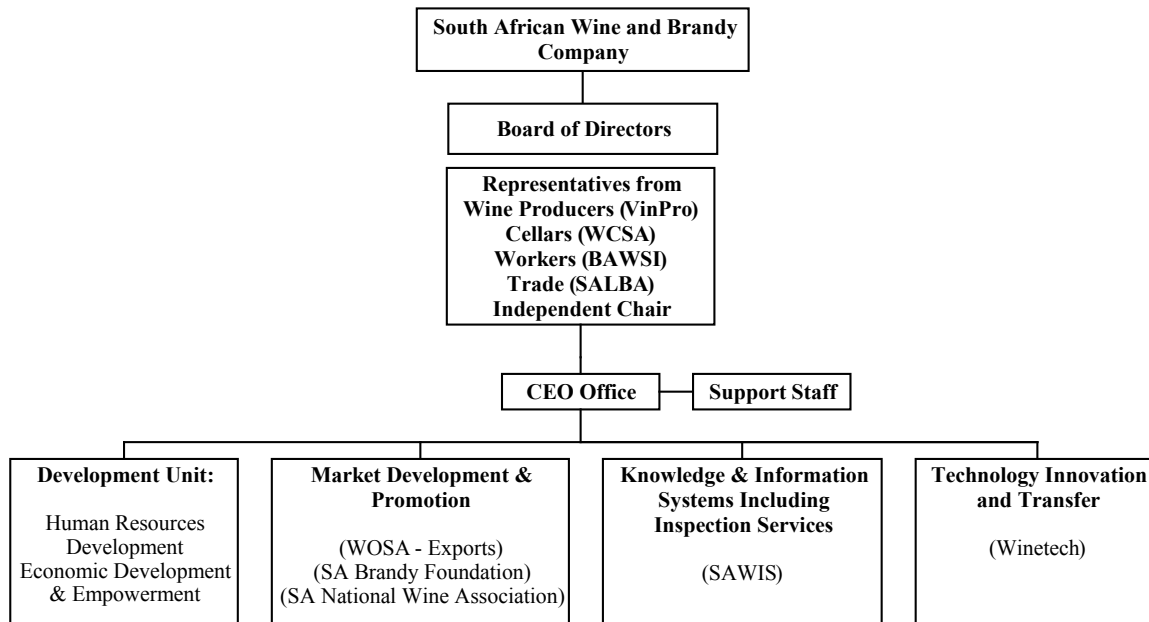


Figure 2.1: The institutional organization of the South African wine industry [118].

The institutional structure of the representative wine industry organization, as shown in Figure 2.1, has been under review by the SAWB and SAWIT in order to ‘provide an efficient and fully representative industry governance structure’ [118]. In June 2006 a new structure was launched, the South African Wine Council which now serves as the apex organization [154]. The structure of the SA Wine Council and the roles of involved representative organizations in the current organisational structure of the South African wine industry is described in the next section.

### 2.1.2 The current organisational structure

From 2006 major changes have occurred in the South African wine industry. The most significant of these is without doubt the restructuring of the SAWB to form the SA Wine Council on 30 June 2006 and is briefly discussed in this section.

There are also other relevant structures, other than the SA Wine Council. These include the Wine & Spirits Board, SAWIT (the ministerial appointment to act as industry trust as mentioned in §2.1.1), RUDNET (a consortium of rural and civil society non-government organizations), a number of voluntary, private or professional wine societies and also service providing groups or bodies [118]. There is also the Wine Charter Steering Committee (WCSC) whose purpose is to ‘fundamentally transform the wine industry within the context of a growing and profitable wine industry’ [149]. Of these, one of importance, if not the most important organization is SAWIT. The structure and functions of SAWIT are also briefly described in this section.

#### The restructuring of SAWB

With the restructuring of SAWB to form the SA Wine Council in June 2006, Dr Johan van Rooyen stayed on as the CEO of SA Wine Council and on 1 October 2006 Professor Kader Asmal, a member of parliament and a former Minister of Education and of Water Affairs,

took over as chairperson [154]. Soon after his appointment as chairperson, Asmal was recorded as saying that ‘fragmentation is one of the major factors preventing the South African wine industry from reaching its full potential’. Therefore, according to Asmal, the role of SA Wine Council should be to ‘harness different bodies and energies to create a unified vehicle through which the wine industry can increase its competitiveness and grow to the economic pillar it deserves to be’ [55].

The SA Wine Council is managed by a Board of Directors with representatives from VinPro, WCSA, SALBA, Labour (consisting of inter thirteen different trade unions and labour organisations), Civil Society (consisting of BAWSI, the Rural Development Network (RUDNET)), Emerging Agriculture (such as the National African Farmers Union (NAFU)), the chairpersons of Winetech, WOSA, SAWIS and the Development & Transformation Unit [151].

The consultation process also resulted in the introduction of the Advisory Forum which consults to and engages with the SA Wine Council Board of Directors. The Advisory Forum includes representatives from SALBA, VinPro, WCSA, SAWIT, BAWSI, NAFU and RUDNET [68].

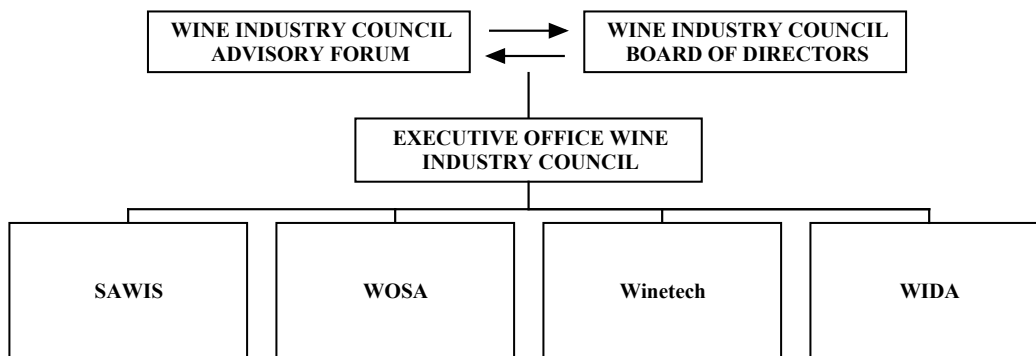


Figure 2.2: The institutional organization of the South African Wine Industry Council [150].

The current institutional structure of the SA Wine Council is shown graphically in Figure 2.2. The SA Wine Council is divided into four business units. The first is SAWIS, with their main functions being the collection, processing and dissemination of industry information, as well as the administration of the industry’s Wine of Origin system [108]. The second is WOSA who, amongst other things, have been mandated to control wine tourism and local marketing of wine in South Africa [154]. Winetech, the third subdivision, has as its core business the ‘improvement of the South African wine industry’s competitive position with the support of the newest research, technological development, training and transfer of technology’ [150]. The last subdivision is the Wine Industry Development Association (WIDA), focusing on sectoral determination and employment conditions<sup>1</sup> [85].

### The organisational structure and function of SAWIT

As the ministerial appointed trust, SAWIT has been the main contributor to the reconstruction of SAWB to form the SA Wine Council. Their goal is currently to ‘generate equitable access and participation in the South African wine industry on the part of those historically marginalized and still on the fringes of the industry to meaningfully participate in a globally competitive,

<sup>1</sup>Since the completion of this thesis, the SA Wine Council has been dispersed. The four business units illustrated in Figure 2.2, now function independently and there is no overseeing council or body [128].

profitable and sustainable industry, as well as facilitating or contributing in the cohesiveness of the industry and speaking in one voice' [116].

SAWIT consists of three subdivisions, as shown in Figure 2.3. The first is the wine industry Business Support Company (BUSCO), the second the Wine Industry Empowerment Company (WIECO) and the third the wine industry Development Support Company (DEVCO). BUSCO manages technology transfer, wine research and development. Through BUSCO, SAWIT is also able to provide funds to Winetech. WIECO endorses and facilitates the economic empowerment of previously disadvantaged communities. DEVCO precipitates social transformation through direct support and intervention, with black economic empowerment at the heart of the organisation, in order to ensure global competitiveness of the South African wine industry [116]. DEVCO funds organizations such as RUDNET and WOSA as well as other projects serving its goal.

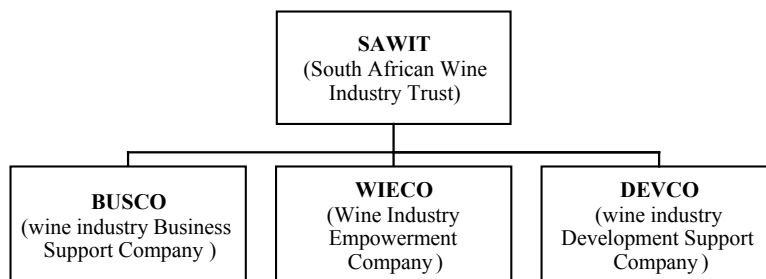


Figure 2.3: The institutional organization of the South African Wine Industry Trust [116].

## 2.2 Wine production in South Africa

Now that the organisational structure of the South African wine industry has been discussed briefly, wine production in South Africa may be considered. This section now includes the different regions of wine production in South Africa as well as the different types of cellars found in South Africa. It also includes various statistics from the industry in terms of the preferred varieties of white and red grapes and also some statistics on import and export. First, some basic definitions are required.

The meaning of the word 'wine' in terms of the South African wine industry is much wider than its everyday meaning. There exist products such as rebate wine and distilling wine that form the building blocks of brandy and have always formed a significant part of South Africa's wine industry. The South African wine industry therefore encompasses wine (natural, fortified and sparkling), rebate wine, distilling wine, brandy and other spirits distilled from distilling wine, as well as grape juice and grape juice concentrate for use in wine and non-alcoholic products [103].

Throughout this thesis, the following definitions (as defined by SAWIS in its annual report of 2006 [103]) applies unless otherwise indicated.

The term wine includes natural, fortified and sparkling wine. *Natural wine* is non-fortified and non-sparkling wine, including perlé wine<sup>2</sup>. It further includes any grape juice or must as well as grape juice or must concentrate used in the sweetening of such natural wine. *Fortified wine* is

<sup>2</sup>Perlé wine is wine carbonated to the extent that the pressure in the container in which it is sold is between 75 and 300 kPa.

non-sparkling wine which has been fortified with wine spirit, including the volume of wine spirit used in the fortification process. The last product included in the term ‘wine’ is *sparkling wine* which is wine carbonated (either by fermentation or by impregnation with carbon dioxide) to the extent that the pressure in the container in which it is sold is more than 300 kPa. As with natural wine, it includes any grape juice or must as well as grape juice or must concentrate used in the sweetening of such sparkling wine.

*Rebate wine*<sup>3</sup> is specially prepared for double distillation in a pot still and then, as distillate, maturation for a period of at least three years in oak casks with a capacity of no more than 340 litres. *Distilling wine* is wine specially prepared for distillation to spirits intended for use in brandy or other spirits, for fortification of wine or for industrial purposes. The term *non-alcoholic* refers to unfermented, undiluted or concentrated juice from grapes destined for use in non-alcoholic products such as fruit juices.

The different wine producing regions of South Africa are described in §2.2.1 and is followed by a brief description of the different types of cellars found in South Africa. Statistics of the industry are reviewed in §2.2.2 and in §2.2.3 a brief look is taken at wine import and export in South Africa.

### 2.2.1 Wine producing regions

There seems to be no official division of the wine producing regions since different sources mostly have different divisions. However, the regions are commonly divided into five subregions: Little Karoo, Breede River Valley, Boberg, Coastal Region and Orange River [108]. In order to best suit the purposes of this project, the regions as used by SAWIS for statistical purposes are adopted in this thesis. They are listed as Malmesbury, Olifants River, Paarl, Robertson, Stellenbosch and Worcester, the Little Karoo and Orange River areas, as shown in Figure 2.4.

The majority of these regions occur in the Western Cape, *i.e.*, all save two, namely the Little Karoo and Orange River areas. In Table 2.1, the distribution of the grapevines throughout these regions are listed. It is clear from these percentages that the majority of vines, as well as the majority of wine production, occurs in the Western Cape Province. It contains 84.5% of the South African vines and 82.3% of the wine producing area hectares.

Wine regions	Number of vines	% of total vines	Area (hectares)	% of total hectares
Malmesbury	37 767 450	12.30	14 883	14.60
Olifants River	27 381 384	8.92	9 861	9.67
Paarl	53 613 681	17.47	17 413	17.08
Robertson	47 308 545	15.41	13 802	13.54
Stellenbosch	53 086 710	17.29	17 265	16.93
Worcester	67 698 826	22.05	20 588	20.19
Little Karoo	9 269 687	3.02	2 966	2.94
Orange River	10 829 502	3.53	5 149	5.05
Total	306 955 785	100.00	101 957	100.00

Table 2.1: Geographic distribution of South African wine grape vineyards per wine region during 2007 (Excluding sultana) [105].

<sup>3</sup>Rebate wine is not discount or bulk wine, but is specially prepared for use in manufacturing brandy.

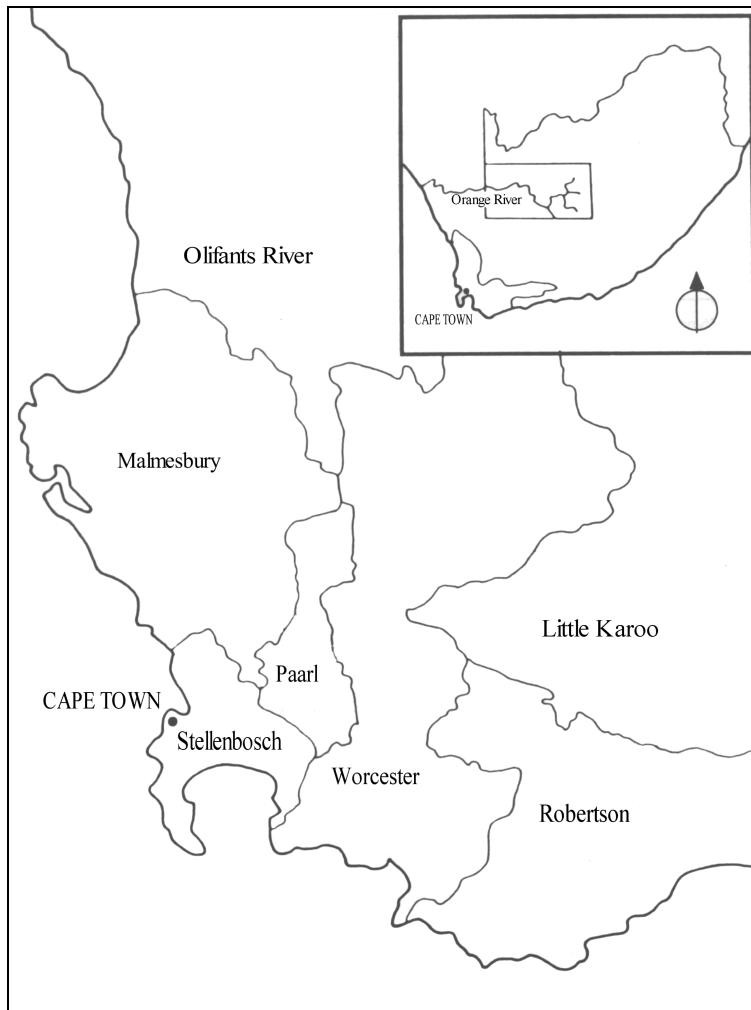


Figure 2.4: The South African wine producing regions [103].

In all of these regions, wine cellars may further be divided into four different types of cellars. This is due to the significant changes occurring in the South African wine industry since the conversion of the KWV from a co-operative to a company. The first is the cooperative cellar. *Cooperative cellars* work on a communal basis processing grapes of their farmer members into wine. All the grapes are pooled; hence farmers with poor quality grapes benefit most. There are currently 65 cooperative cellars in South Africa that are responsible for pressing about 80% of the total South African harvest.

The second type of cellar is a *corporate cellar*. These cellars have been transformed into companies. They may therefore pay the producer based on the quality of the grapes delivered and may make a profit. On request of the wine industry, both corporate cellars and cooperative cellars will be referred to as *producer cellars* as of 2009 [128].

The third type, namely *estate cellars*, are small farms with their own cellars where all the grapes for producing wine must come from the farm, *i.e.* they are not allowed to buy in grapes or wine.

The last type of cellar is *private cellars*. These cellars are allowed to buy in (all) grapes for the production of their wine.

## 2.2.2 Wine production statistics

The South African wine industry produces a very large variety of cultivars and blends with its ideal climates and land qualities for wine grape growing; this aspect of growing grapes is further discussed in §2.3.1. The area under cultivation of each of the different cultivars over the last eight years is given in Table 2.2.

VARIETY	2000	2001	2002	2003	2004	2005	2006	2007
Chenin blanc	24.1	22.3	20.6	19.6	19.1	18.8	18.7	18.8
Colombar(d)	12.2	11.8	11.4	11.2	11.2	11.3	11.4	11.6
Chardonnay	6.4	6.3	6.4	6.8	7.3	7.8	8.0	8.7
Sauvignon blanc	5.7	6.1	6.7	6.9	6.9	7.5	8.2	8.1
Hanepoot	4.3	3.8	3.4	3.1	2.8	2.6	2.5	2.4
Cape Riesling	2.3	1.9	1.6	1.4	1.2	1.1	1.0	1.1
Sémillon	1.1	1.0	1.0	1.0	1.0	1.1	1.1	1.0
Weisser Riesling	0.6	0.4	0.4	0.3	0.3	0.3	0.2	0.2
Other white varieties	7.1	5.5	4.7	4.3	4.1	3.9	3.9	3.9
<b>Total white varieties</b>	<b>63.8</b>	<b>59.4</b>	<b>56.2</b>	<b>54.6</b>	<b>54.0</b>	<b>54.3</b>	<b>55.1</b>	<b>55.8</b>
Cabernet Sauvignon	9.5	11.0	12.4	13.0	13.5	13.4	13.1	12.8
Shiraz	6.0	7.5	8.4	8.6	9.4	9.6	9.6	9.7
Merlot	5.2	6.0	6.6	6.7	7.0	6.8	6.7	6.6
Pinotage	7.0	7.3	7.2	6.8	6.7	6.4	6.2	6.0
Cinsaut noir	3.7	3.6	3.3	3.1	3.0	2.8	2.5	2.4
Ruby Cabernet	2.1	2.4	2.5	2.5	2.6	2.6	2.5	2.4
Cabernet franc	0.6	0.6	0.8	0.9	0.9	1.0	1.0	1.0
Pinot noir	0.6	0.5	0.6	0.5	0.5	0.5	0.6	0.6
Other red varieties	1.5	1.7	2.0	3.2	2.4	2.6	2.7	2.7
<b>Total red varieties</b>	<b>36.1</b>	<b>40.6</b>	<b>43.7</b>	<b>45.3</b>	<b>46.0</b>	<b>45.7</b>	<b>44.9</b>	<b>44.2</b>
Total white and red	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Total hectares	93 656	94 412	96 233	98 605	100 207	101 607	102 146	101 957
Sultana (Hectares)	11 910	11 919	11 765	11 595	11 392	10 983	10 571	9 958

Table 2.2: South African grape varieties as percentage of total area [105].

All of the cultivars listed in Table 2.2, save Hanepoot<sup>4</sup> and Sultana<sup>5</sup>, are used solely for the production of wine. The first aspect one notices in Table 2.2 is the gradual shift in the production of white and red wines. In 2000 more than 60% of the industry consisted of the processing of

<sup>4</sup>Used for wine production as well as raisins and table grapes.

<sup>5</sup>Used only for raisins and table grapes.



white grapes<sup>6</sup>. By 2007 the growing of red grape varieties had grown to constitute almost half the wine grapes produced. These grapes are not only used to produce natural wine, but also the other types of wine listed in the beginning of §2.2. Together with the fact that red grapes may be used (without contact to their skins) to produce white and rosé wines, the percentages given in Table 2.2 do not directly imply the same results in the production of actual red and white wines. The relationship between white and red wine grapes utilized towards making red and white wines is given in Table 2.3.

Year	Wine grapes utilised for total wine making purposes		Wine Produced	
	% Red	% White	% Red	% White
1997	11.6	88.4	15.1	84.9
1998	13.2	86.8	15.2	84.8
1999	13.4	86.6	16.3	83.7
2000	15.0	85.0	21.0	79.0
2001	21.1	78.9	25.2	74.8
2002	22.7	77.3	27.9	72.1
2003	28.1	71.9	32.1	67.9
2004	29.9	70.1	36.2	63.8
2005	33.5	66.5	38.9	61.1
2006	33.4	66.6	36.5	63.5
2007	34.4	65.6	36.1	63.9

Table 2.3: The distribution between red and white wines produced [103, 105].

From this table it is clear that the increase in red wine production is not as dramatic as the increase in the production of red grapes, but was nevertheless at three times the percentage in 2005 compared to what it was in 1997. According to WOSA this is due to shifting market demands and the growth of red wine consumption. The industry has rapidly increased its plantings of red wine varieties to over 80% of all new plantings in 2001, which once again fell to 51% in 2003 [154].

Still focusing on Table 2.2 and concentrating first on the white grape varieties, it is clear that even though Chenin Blanc is still the most common white grape varietal, its percentage of the total area used has dropped by 5% from 2000 to 2007. This is also the largest decline of all the listed varieties. It may be argued that since the percentage of South African wines that are exported have doubled during this period, the rather remarkable change in the production of wine grapes may be due to the more significant influence of the international market. If this is the case, it would seem that the global demand for Chardonnay and Sauvignon Blanc have both been increasing over this period. Even more so since the total percentage of white grape varieties have been decreasing. In order to eliminate the influence of this factor on the interpretation of the data, the white wine grape varieties as a percentage of the total white grape area only is shown in Table 2.4. From the table it is also clear that the only varieties showing an increase, when expressed as the percentage of total white grape area, is Columbar, Chardonnay, Sauvignon blanc and Sémillon.

Returning to the interpretation of the information in Table 2.2, but now turning the focus to red grape varieties, it is apparent that there has been an increase in all red grape varieties in terms of the total area of grapes, *i.e.* all save one, Cinsaut noir. In Chapter 1 the breakout

<sup>6</sup>This is already a significant decrease from the 75% of 1998 [103].

Variety	2000	2001	2002	2003	2004	2005	2006	2007
Chenin blanc	37.77	37.73	36.65	35.90	35.44	34.56	33.94	33.69
Columbar(d)	19.12	19.97	20.28	20.51	20.78	20.77	20.69	20.79
Chardonnay	10.03	10.66	11.39	12.45	13.54	14.34	14.52	15.59
Sauvignon blanc	8.93	10.32	11.92	12.64	12.80	13.79	14.88	14.52
Hanepoot	6.74	6.43	6.05	5.68	5.19	4.78	4.54	4.3
Cape Riesling	3.61	3.21	2.85	2.56	2.23	2.02	1.81	1.97
Sémillon	1.72	1.69	1.78	1.83	1.86	2.02	2.00	1.79
Weisser Riesling	0.94	0.68	0.71	0.55	0.56	0.55	0.36	0.36
Other white varieties	11.13	9.31	8.36	7.88	7.61	7.17	7.08	6.99
Total	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00

Table 2.4: The white wine grape varieties as a percentage of total white grape area in South Africa over the period 2000–2007.

of the disease, Phylloxera, was mentioned and also that in a desperate attempt to revive the wine industry large quantities of high yielding vines (80 million to be precise) was planted by the early 1900s, the most common variety being Cinsaut. Since one of the goals of the newly transformed South African wine industry has been to break away from its legacy and produce more high quality wines, the decrease in Cinsaut noir seems very sensible. In order to acquire an unbiased view of the increase or decrease of specific red grape cultivars, the percentages of the various red grape varieties of the total red grape area are given in Table 2.5.

Variety	2000	2001	2002	2003	2004	2005	2006	2007
Cabernet Sauvignon	26.24	27.09	28.31	28.70	29.35	29.32	29.18	28.95
Shiraz	16.57	18.47	19.18	18.98	20.43	21.01	21.38	21.95
Merlot	14.36	14.78	15.07	14.79	15.22	14.88	14.92	14.93
Pinotage	19.34	17.98	16.44	15.01	14.57	14.00	13.81	13.57
Cinsaut noir	10.22	8.87	7.53	6.84	6.52	6.13	5.57	5.43
Ruby Cabernet	5.80	5.91	5.71	5.52	5.65	5.69	5.57	5.43
Cabernet franc	1.66	1.48	1.83	1.99	1.96	2.19	2.23	2.26
Pinot noir	1.66	1.23	1.37	1.10	1.09	1.09	1.34	1.36
Other red varieties	4.14	4.19	4.57	7.06	5.22	5.69	6.01	6.11
Total	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00

Table 2.5: The red wine grape varieties as a percentage of total red grape area in South Africa over the period 2000–2007.

As expected, Cinsaut noir still shows the largest decrease in its vineyard area, with the percentage of the total grape area in 2007 being half of that in 2000. The other relatively large decrease is with the uniquely South African varietal, Pinotage<sup>7</sup>. Most of the other red grape varieties show either an increase, with Shiraz showing the largest increase and Cabernet Sauvignon, also showing an increase and still the most popular, or they seem to be relatively unchanged.

<sup>7</sup>Pinotage was cultivated in 1925 in South Africa by Abraham Izak Perold, the first Professor of Viticulture at Stellenbosch University. Pinotage is a cross between Cinsaut and Pinot Noir. The name, Pinotage, is due to the fact that Cinsaut is known as Hermitage in South Africa, hence the portmanteau name of Pinotage.

The conclusions derived from Table 2.2 for both red and white grapes is supported by a statement from WOSA that ‘the local wine industry as a whole is strengthening its focus on five noble varieties and is primarily replanting, on a large scale, Cabernet Sauvignon, Merlot and Shiraz, and Chardonnay and Sauvignon Blanc’ [154]. They also note that the most dramatic increase in the planting of a specific grape variety is with that of the Shiraz grape, which is also evident in Table 2.2.

Statistics are also available on which grape varieties are favoured by which of the wine producing regions. The region, as defined in §2.2.1, with the largest percentage of white grapes, is the Worcester area with almost a quarter of all white grapes planted there. Worcester is also the largest region in terms of area. Red grapes are planted mostly in Stellenbosch, Paarl and Malmesbury, which collectively has 64% of all areas planted with red grapes [105].

### 2.2.3 The import and export of wine in South Africa

South Africa was responsible for 3.5% of wine production globally in 2006 and also plays a significant role in the global wine industry [153]. In this section, the top ten exporting and importing countries are listed and interpreted according to volume. This is followed by a summary of South African wine exporting.

#### Global wine import and export

In 2006 twenty eight thousand four hundred million litres of wine were produced worldwide [153] of which South Africa produced 1 013 million litres, ranking as the seventh largest wine producer. However, when it comes to the per capita consumption of wine, South Africa does not even reach the top 15. Tables 2.7–2.10 further support this notion by showing large quantities of wine being exported and very little being imported. In terms of production and per capita consumption of wine, the French are in the lead with the Italians not too far behind. From Table 2.6, it may be deduced that France produced almost 20% of the global wine production in 2006 and the top three producing countries, France, Italy and Spain, collectively produced half of the wine. In Tables 2.7 and 2.8 the volumes and values of wines exported are listed per country for the ten largest exporting countries and in Tables 2.9 and 2.10 the volumes and values of wines imported are listed per country for the ten largest importing countries.

		Wine production			Per capita consumption			
		2000	2003	2006		2000	2003	2006
1	France	5 754	4 749	5 340	France	58.2	54.7	53.9
2	Italy	5 409	4 665	4 712	Italy	53.5	51.1	47.2
3	Spain	4 179	4 037	4 010	Portugal	45.9	52.6	46.9
4	United States of America	2 660	2 350	2 338	Croatia	42.0	42.5	42.3
5	Argentina	1 254	1 323	1 540	Switzerland	43.1	41.4	38.1
6	Australia	806	1 086	1 430	Spain	34.5	33.6	31.6
<b>7</b>	<b>South Africa</b>	<b>837</b>	<b>956</b>	<b>1 013</b>	Uruguay	32.0	32.0	32.5
8	Germany	1 008	819	900	Denmark	30.5	31.8	33.7
9	Portugal	784	715	715	Hungary	31.5	31.6	31.8
10	Chile	667	687	845	Argentina	33.7	32.1	28.1

Table 2.6: The wine production and per capita consumption per country [153].

Rank	Country	2000	2003	2006
1	Italy	28.2	17.9	20.7
2	France	24.2	21.1	16.8
3	Spain	14.5	15.9	19.2
4	Australia	4.5	7	8.5
5	Chile	4	5.2	5.6
6	United States of America	4.1	4.4	4.2
7	Germany	3.7	3.6	3.6
8	Moldova	2.8	5	2.2
<b>9</b>	<b>South Africa</b>	<b>2.0</b>	<b>4.7</b>	<b>3.2</b>
10	Portugal	2.7	3.4	3.7
	Other	9.3	11.8	12.3
	Total	100.0	100.0	100.0

Table 2.7: The volume share per country of the world wine exports expressed as a percentage of the total [153].

Rank	Country	2000	2003	2006
1	France	39.8	38.8	34.4
2	Italy	18.1	17.8	17.7
3	Spain	9.6	10	9.6
4	Australia	6.6	9.1	9.4
5	Chile	4.5	4.3	4.1
6	United States of America	4.1	3.5	3.7
7	Germany	2.8	3.2	3.6
8	Portugal	3.6	2.3	2.9
<b>9</b>	<b>South Africa</b>	<b>1.9</b>	<b>2.4</b>	<b>2.3</b>
10	Argentina	1.2	0.8	1.7
	Other	7.8	7.8	10.6
	Total	100.0	100.0	100.0

Table 2.8: The value share per country of the world wine exports expressed as a percentage of the total [153].

From Table 2.7 it may be deduced that once again the top three countries exporting wine in terms of volume exported produce over half of the wine exported worldwide. These top three exporting countries are also the top three producing countries globally.

In terms of value, as listed in Table 2.8, the two countries who export the highest value of wine own more than half of the value of wine exported globally. These two countries, France and Italy, are also the two largest producing countries as well as the two countries with the highest per capita consumption of wine.

Another conclusion that may be drawn from Tables 2.7 and 2.8 is that the French export mostly superior quality wines (or perhaps merely more expensive wines). This assumption may be made due to the fact that even though they do not export the largest quantities of wine (rather the second largest) they still earn the highest value of exported wines. South Africa is consistently ranked as number nine in both wine exported when listed according to the volume and to the value of the exported wine.

Rank	Country	2000	2003	2006
1	Germany	19.2	18.2	16.7
2	United Kingdom	16.2	17.1	15.6
3	United States of America	7.7	8.3	8.9
4	France	9.3	6.3	7.6
5	Russia	3.8	7.6	5.7
6	Netherlands	5.7	4.6	4.5
7	Belgium Luxembourg	4.6	3.9	3.5
8	Other Central Eastern Europe	3.2	3.7	4.5
9	Canada	3.5	3.6	3.9
10	Denmark	2.8	2.5	2.5
	Other	24	24.2	26.6
	Total	100.0	100.0	100.0

Table 2.9: The volume share per country of the world wine imports expressed as a percentage of the total [153].

Rank	Country	2000	2003	2006
1	United Kingdom	18.1	18.9	19
2	United States of America	15.7	18	17.3
3	Germany	14.5	13.3	11.3
4	Japan	5.6	4.7	4.7
5	Belgium Luxembourg	5.7	5	4.5
6	Netherlands	5.7	4.4	4.5
7	Canada	4.5	4.6	5.2
8	Switzerland	4.5	4.4	3.5
9	France	3.4	3.3	2.9
10	Denmark	2.8	3.6	2.3
	Other	19.5	19.8	24.8
	Total	100.0	100.0	100.0

Table 2.10: The value share per country of the world wine imports expressed as a percentage of the total [153].

In Tables 2.9 and 2.10 the top ten importing countries are listed in terms of volume and value respectively. The largest importer of wine in terms of volume is Germany and in terms of value, the United Kingdom. The top three wine importing countries in terms of volume and value once again support more than half the market concerned.

As expected, the numbers concerning the importing of wine differs greatly from those concerned with export. Countries who produce large quantities of wine or good quality wine do not need to import wine as much as those who do not produce high enough quantities to satisfy the expected consumption of wine. This is made clear by top exporting and producing countries such as Italy and Spain not even appearing under the top ten wine importing countries in both Tables 2.9 and 2.10. South Africa, ranked as the ninth largest exporter and seventh largest producer, is not even in the top 15 importers of wine in terms of volume or value [153].

### South African wine exports

The countries who import South African wines are listed in Table 2.11 together with the quantity of wines imported from South Africa in 2006 and 2007, given in litres and expressed as a percentage of the total volume of wine exported by South Africa.

Country	2006		2007	
	Volume (in litres)	% of total	Volume (in litres)	% of total
United Kingdom	81 925 610	30.44	87 051 366	28.13
Germany	41 551 559	15.44	59 546 131	19.24
Netherlands	36 806 310	13.67	29 038 337	9.38
Sweden	23 121 009	8.59	25 967 932	8.39
Denmark	12 007 674	4.46	13 364 361	4.32
Canada	11 998 202	4.46	12 996 576	4.20
U.S.A	10 669 427	3.96	10 667 279	3.45
Belgium	9 125 380	3.39	9 572 926	3.09
Angola	319 557	0.12	8 544 964	2.76
France	8 548 123	3.18	6 945 047	2.24
New Zealand	1 838 603	0.68	5 770 212	1.86
Republic of Ireland	5 287 064	1.96	4 734 845	1.53
Switzerland	3 184 649	1.18	4 425 953	1.43
Finland	3 736 715	1.39	4 055 992	1.31
Russia	1 199 522	0.45	2 544 415	0.82
Kenya	1 759 528	0.65	2 187 420	0.71
Norway	1 481 691	0.55	1 602 197	0.52
Japan	1 333 878	0.50	1 354 792	0.44
Tanzania	973 470	0.36	1 301 083	0.42
China	488 214	0.18	1 249 820	0.40
Nigeria	642 418	0.24	1 240 161	0.40
Mauritius	1 011 245	0.38	1 077 454	0.35
Poland	635 844	0.24	1 009 626	0.33
United Arab Emirates	726 250	0.27	899 838	0.29
Latvia	934 741	0.35	755 750	0.24
Australia	449 249	0.17	750 909	0.24
Other countries	7 410 647	2.75	10 795 126	3.49
<b>Total</b>	<b>269 166 579</b>	<b>100.00</b>	<b>309 450 512</b>	<b>100.00</b>

Table 2.11: Packaged and bulk natural wine exports ranked per receiving country in litres [105].

Unfortunately the same information on South African wine imports do not seem to be available, (only the volumes imported are available, but not the countries from where they are imported). These volumes are given in Table 2.12 in terms of the bottled and bulk natural, fortified and sparkling wines imported during 2006 and 2007.

The volume of wine imported during 2006 (19 071 365 litres) decreased to 14 101 397 litres in 2007. This is due to the significant decrease in the import of natural white bulk wine. The level of import of fortified wines decreased by almost a third, whereas the import levels of sparkling wines almost tripled in its volume from 2006 to 2007.

	2006			2007		
	Bottled	Bulk	Total	Bottled	Bulk	Total
Natural white wine	119 207	18 379 934	18 499 141	143 914	12 879 226	13 023 140
Natural red wine	266 234		266 234	297 838		297 838
Fortified wine	32 807		32 807	24 447		24 447
Sparkling wine	273 183		273 183	755 972		755 972
Total	691 431	18 379 934	19 071 365	1 222 171	12 879 226	14 101 397

Table 2.12: Imports to South Africa, bottled and bulk [105].

## 2.3 From the vine to wine

This section deals with the production of wine from wine grapes and what is needed both in the vineyard and in the cellar to make good wine. This process consists of two fields of study. The first consists of the cultivation of grapes in the vineyard and is referred to as *viticulture*. Viticulture and some of the aspects concerned are discussed in §2.3.1. The second field of study is called *oenology* (sometimes spelled in the American version *enology*) and refers to the art and science of making wine. Oenology is discussed in §2.3.2.

### 2.3.1 Viticulture (grape growing)

Viticulture is the part of horticulture which deals solely with the growing of the grape vine [134]. It is practised consciously by the viticulturist and often instinctively by grape-growers or vine-growers [100].

The market requirements for wine determines optimal viticulture practices and it is therefore important that the market segment in which the wine will compete is specified from the start so that the practices may be adapted to suit the desired result [38].

The sequence of vineyard development processes from initial planning through to picking starts with the selection of the vineyard site, choice of the rootstock, vine variety and clone. This is followed by soil testing and soil preparation, choice of vine density and trellis system<sup>8</sup>. Once all of these aspects have been decided upon vine planting may start, followed by training and pruning of the vines. Vine pests, vine diseases and weeds should be controlled throughout vineyard development. The last phase includes sampling the fruits and finally harvesting the ripened fruits [100]. Duties of the viticulturist during the last mentioned process may include: monitoring and controlling pests and diseases, fertilizing, irrigating, canopy management, monitoring fruit development and characteristics, deciding when to harvest, and vine pruning during the winter months [144].

With these tasks in mind, viticulture practices may be divided roughly into two classes. The first is *terroir* decisions which may be defined as the ‘total natural environment of any viticultural site’ [100]. The second is good practices concerning viticulture [38]. These two classes may once again be subdivided into different topics as illustrated in Figure 2.5 and are briefly discussed in this section.

<sup>8</sup>The trellis system is the support structures for the vine framework required for a given training system.

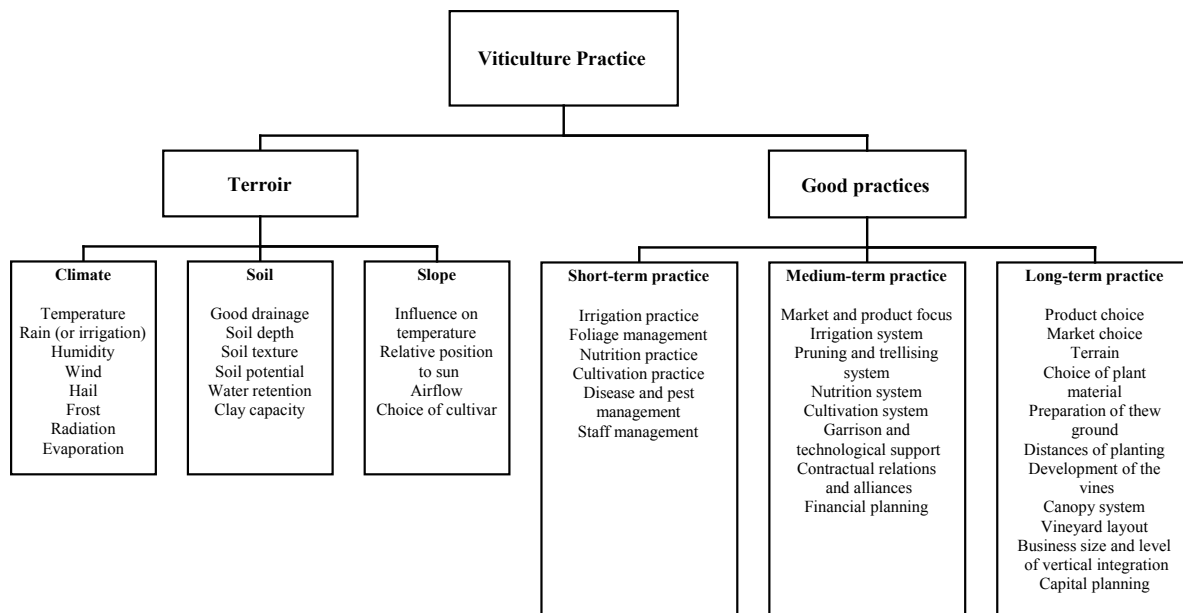


Figure 2.5: A schematic illustration of viticulture practices in South Africa [38].

## Terroir

Terroir may be subdivided into three categories namely climate, soil and slope [38].

**Climate** The climate may very well be the most important determining environmental factor for viticulture practice. Environmental aspects such as temperature, rain (or irrigation), humidity, wind, hail, frost, radiation (direct radiation is warmer than diagonal radiation) and evaporation all play a role in the selection of the proper viticultural site [38]. Temperature is certainly one of the most important elements of climate, since cold-temperature injury from winter cold along with diseases encouraged by hot, humid weather are major limiting factors to grape production. There are also other environmental factors that influence the growth and outcome of the vineyard, the volume and quality of the harvest, and the quality of the wine that is produced. These factors include frequent rainfall during the growth season and spring frosts [38, 134].

The best suited climate for a vineyard is suggested to be a Mediterranean climate, situated between 30 and 50 degrees of latitude north and south [38]. Wine production in South Africa mainly takes place at a latitude of 27 to 34 degrees south placing it in the desired climate region [154].

Another important factor for grape production is the length of the growing season which is determined by the first and last frosts of the year. A minimum frost-free period of 150 days is required for the earliest ripening cultivars [134].

**Soil** Grapevines perform best when they have healthy, well-developed root systems and a wide range of soils suitable for grapevines exist. It has also been shown that it is the physical characteristics of soil that effect the characteristic of the wine, not the chemical characteristics. Physical characteristics include good aeration, loose texture and moderate fertility, a good surface and internal drainage capability [134]. Proper soil drainage is a necessity for successful vine culture. Root growth in poorly drained soils is usually limited to the



top two feet or less compared to the well-drained soils where roots may penetrate six feet or more. This restriction of the root system may cause vine growth and fruit yields to decrease, resulting in plant survival being limited to only a few years [134]. Too much damp and nourishing reserves in the soil may also stimulate excessive growth of the grapevine that is unfavourable for the quality of the wine [38].

Deeper soil has higher sustainability and enhances the ability to guarantee stable growing conditions. The soil should also be of medium and open texture and have medium potential that will not induce excessive growth and cause a distorted balance between growth and the accumulation of essential components in the grapes. Water retention is determined by the depth, texture and structure of the soil as well as the quantity of organic material and rocks present. The ideal water retention is medium to high (15–25 mm/meter). In dry land conditions, soil with a clay capacity of 10–30 % may improve water retention [38].

**Slope** Positioning a vineyard relative to the sun or on hills and valleys plays an important role in the vineyard outcome. The most significant influence of a hill is on the temperature of the area. The slope of a hill causes a 0.6°C decline in temperature with an increase in height of 100m [38]. Hills also provide warmer temperatures during the night and no extreme temperatures during the day. Southern and south eastern facing hills are cooler than northern and north western facing hills. Hills also provide better airflow with cooler air coming down the hill. The gradient of the viticultural site may also have an effect on which cultivar is best suited for the land [38].

### Good viticultural practices

According to a study by the Department of Logistics at the University of Stellenbosch, good viticultural practices may once again be subdivided into three categories, namely short, medium and long-term decisions and management considerations. Each of these three categories is described briefly in this section.

**Short-term decisions and management considerations** Decisions should be made on the type of *irrigation practice* that will be followed. This is an important decision and even though the decided irrigation practice will be followed for at least the following two years it will still influence the irrigation system used in the long run. The *foliage-management* also has an impact on the growth and the strength of the vines. As *nutrition practice*, fertilizers should only be applied in agreement with soil analyses, since excessive fertilization causes water pollution and induces excessive growth and leaf density. This is an unfavourable situation for the vineyard in the long run. With *cultivation practice* mechanical cultivation should be limited to a minimum. A system of minimum cultivation by the use of cover crops is advised. For *disease and pest management* chemical control should be eliminated as far as possible in management practices and control measures. To successfully coordinate *staff management*, staff should be handled in such a way as to ensure good working conditions and relations in the long run. In order to ensure no deviations in the long run *harvesting management and quality control* should be done effectively and continuously. These short-term decisions should be recorded for future use [38].

**Medium-term decisions and management considerations** The first consideration is that of *market and product focus* which includes all decisions influenced by what is popular at

the time and how these production needs will be met. The *irrigation system* relies on the resource, cultivar, pruning practices and canopy system, it should also suit the soil type and climate. The right *pruning and trellising system* must be used, since it has a direct influence on the harvest and is also a means of manipulating the vineyard. The system should accommodate the vivacity of the vines to avoid forming canopies that are too dense, rather allowing maximum airflow in order to reduce the risk of disease occurrence. A proper *nutrition system* should be derived to ensure moderate use of fertilizers. This is of the utmost importance, since it should guarantee moderately growing vineyards. In order to produce the optimal grape and wine quality each individual situation requires the right fertilizing practice. A *cultivation system* consists of the manipulation of soil in existing vineyards to create and keep the most favourable physical conditions of the soil for the roots of the vines. There should be a proper *garrison and technological support system* to ensure that everyone in the industry is trained to understand and work successfully in the whole process. This aspect of decision and management considerations should also ensure that the right technological support is available throughout the process. *Contractual relations and alliances* require that decisions are made on the possibility of producing on contract basis as well as based on the possibility of forming alliances. Another important aspect is *financial planning*. This planning should typically be done to see what expenses are expected to occur during the following ten years, keeping in mind factors such as equipment that must be replaced [38].

**Long-term decisions and management considerations** With *product choice*, there should be a focus area and the end product of the process (typically the wine produced, *e.g.* Sauvignon blanc) must be specified. The *market choice* should be specified from the beginning. Selection of the viticultural site or *terrain* and decisions regarding soil preparation actions and the suitability of the soil for a specific cultivar should be based on proper profile studies. Deciding on the right *choice of plant material* is crucial to the well being of the vineyard. The type of soil dictates the choice of rootstock and the right type of clone must also be selected, since different clones can improve the complexity of a specific cultivar. Certified material should be used whenever available. Certain rootstock are more resistant to pests and diseases in particular environments than others; therefore the most pest and disease resistant rootstock should be used. The *preparation of the soil* should only be done once the type of soil has been identified through profile studies. Soil that is deeply cultivated has a better resistance against unfavourable weather. The growth of the vines and thus the quality of the wine is also influenced by the *distances of planting*. The rows should be a standardised 2.2m apart (based on the potential of the soil) and the distances between the vines should be between one and two metres. The *development of the vines* should be done in a way that ensures consistent vigour and quality in the vineyard. Enough leaf-roof for optimal ripeness is ensured by a large enough *canopy system* which should also suit the potential of the soil and the cultivar. The *vineyard layout* is influenced by the choice of the direction which is determined by the climate. The direction should also be chosen to prevent soil erosion, allow maximum airflow and reduce the occurrence of disease. However, the most important decision involves the *business size and level of vertical integration* as this will influence the success and potential of the business in the long run. *Capital planning* is yet another area of focus in which the producer must decide on the percentage of available land that will be prepared and used during production. At this stage he should also consider the possibility of his existing practices being expanded in the future [38].

The quality of a wine is affected greatly by the quality of the grapes used. The viticulturist therefore often has an important relationship with the winemaker. Viticultural preparation also includes steps to determine whether grapes are fully ripened and that the requirements of the winemaker are met. These requirements include aspects such as flavour compounds, colour compounds, sugar levels and acidity [39], and is examined by means of repeated tasting and tests. Once these requirements are met, the grapes may be harvested and transported to the cellar by means of a truck or tractor depending on the yield. The grapes are then graded — this refers to the process of determining the type and quality of grapes arriving at the cellar. Grading techniques may include the visual inspection of grapes, batch sampling or any other means [39].

It is a respected statement that good wine begins with good grapes. It is therefore essential that the grapes be analysed at the vineyard as well as on arrival at the cellar. The quality, origin and certain characteristics should be carefully documented, since the quality of the grapes may differ on arrival at the cellar from the quality of the grapes as previously documented in the vineyard, this is due to aspects such as sun exposure. The process and science of making the wine from the grapes at the cellar is described in §2.3.2.

### 2.3.2 Oenology (wine making)

The word oenology (and enology) is derived from the Greek *oinos* (wine) and *logos* (logic). Wine logic is still an accurate description of this science as it is based on the application of analysis, equipment, materials and technology in designed methodologies to achieve predetermined results [134]. An *oenologist* is a person who studied oenology and has wine making as practical part of their activities, whereas a winemaker may have no formal qualifications [98].

Wines are generally categorized by experts in five classes: Table wines, sparkling wines, dessert wines, aperitif wines and pop wines. These wines differ in the grape variety used and also in the vinification method [145]. The overwhelming majority of the wine produced in the world falls into the table wine category. The vinification methods used for white and red table wines are therefore outlined in this section.

#### The production of white wine

There is no precise recipe for the making of a bottle of white wine and the winemaker is left with a large number of choices depending on the type of grape used, the behaviour of the wine during the different stages of production and its desired outcome. A flowchart of the various procedures involved in the production of white wine, as will be described in this section, is shown in Figure 2.6. The figure represents a generic method of white wine making only, and may differ from one winemaker to the next.

The composition and soundness of the grapes used for making wine is of major importance and dictates the quality of the resulting wine. The degree of ripeness of the grapes at harvest depends on the type of wine to be made with normal maturity being between 18° and 22.5° Balling<sup>9</sup> for dry white wine. However, other varieties such as Chardonnay, may benefit from more mature grapes [98].

When making dessert wines, oxidative practices are used, but with the making of table wines the modern approach is to use reductive oenology. This practice is greatly centred on preventing

---

<sup>9</sup>Brix is used to describe the sugar content in grapes and wine, also commonly used is Balling.

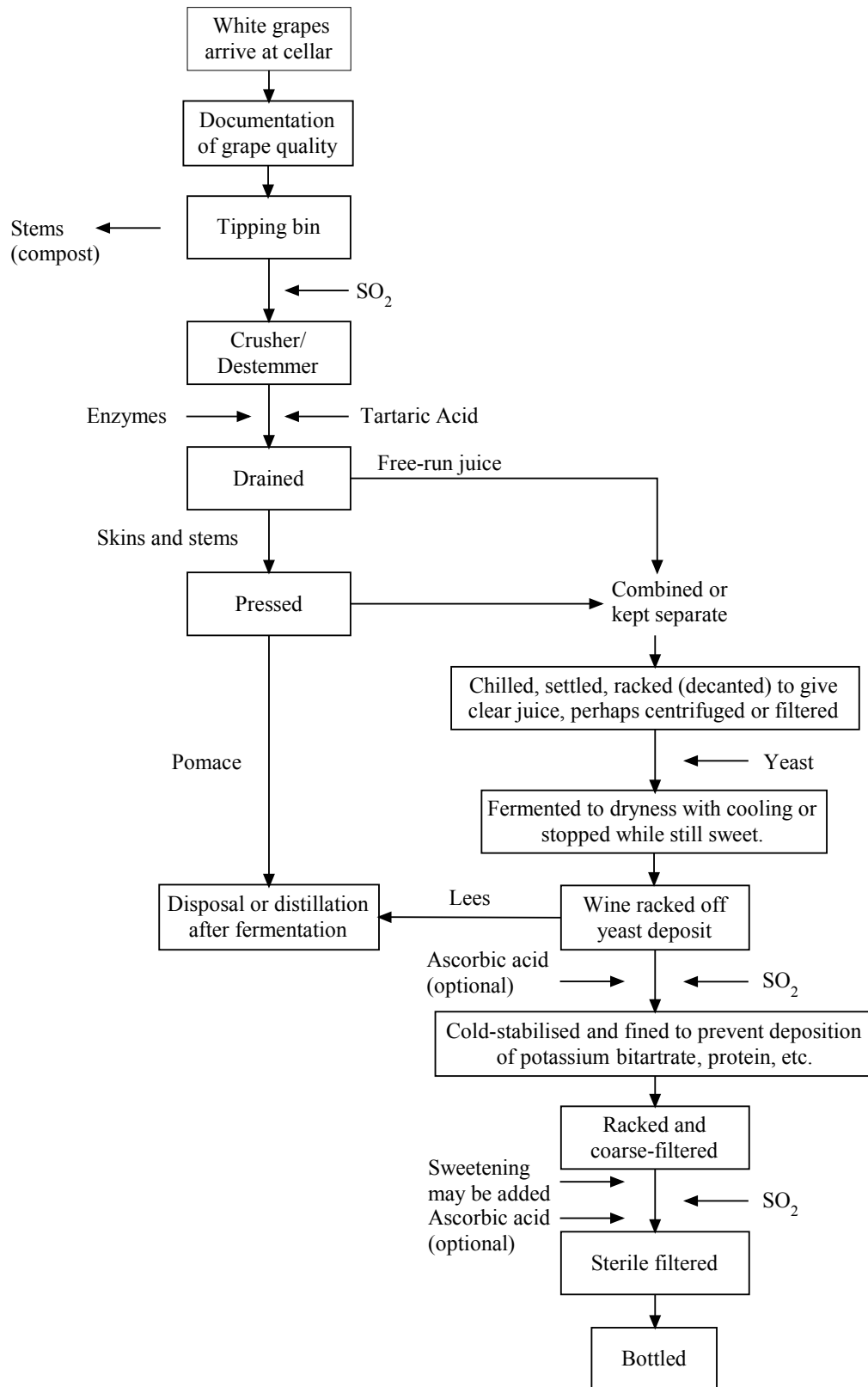


Figure 2.6: A flowchart for making white table wine [39, 98].

contact of the must and wine with air in order to avoid oxidation together with the effect of temperature control [98].

In optimal conditions, grapes harvested should be cool (between 8° and 16°C). In warmer areas, this may be achieved by machine harvesting at night. If this is not possible, must cooling may be applied. Sulphur dioxide and ascorbic acid may be added to the grapes directly after harvesting [98].

On arriving at the cellar the grape quality is properly analysed and documented, and the batch is deposited into an assigned *tipping bin* accordingly. The grapes are destemmed and crushed<sup>10</sup>, preferably in a roller crusher which destems before crushing [98]. Crushing occurs in order to break grape skins and release the juice. The thick liquid that forms as result from the crusher-destemmer, called *must*, is neither grape juice nor wine but a mixture of grape juice, stem fragments, grape skins, seeds and pulp [100]. Tartaric acid may be added, if required, to ensure that the pH of the resulting juice is within a range of 3.0 to 3.4. Certain enzymes<sup>11</sup> may also be added at this stage to hasten the yield of the drained juice and should be added as soon as possible after crushing to allow the longest period of time for them to act [98].

In order to ensure that the must is of the correct temperature, a *mash cooler* may be used, which pertains to a heat exchange device and cools the incoming pulp or grapes before fermentation. It is considered essential to maintain as low a temperature as possible during the early stages of the process, since South Africa has a relatively hot climate. A considerable loss in flavour may occur with higher temperatures in the beginning phase of the wine making process, especially with white wine. It may not always be possible to chill the grapes on arrival at the cellar or during destemming processes, in which case it should be done as soon as possible after. The type of mash cooler used depends on the cellar environment [39].

The winemaker may soak the grape skins in the juice if he wishes to extract flavours and aromas trapped in the skins [5]. Certain varieties benefit more from skin contact and the length of the contact depends on the temperature. Depending on the variety, skin contact may be for up to 18 hours at 5 to 10°C. Skin contact is of greatest value with quality grapes from cool regions [98]. Any colour grape may be used to make white wine as the skin is separated from the juice during fermentation, when dark grapes are used however, the skins should be separated from the juice as gently and soon as possible. The majority of white wines are nevertheless made from grapes with yellow or green skins [100].

The must is now drained, preferably under carbon dioxide, at a temperature not exceeding 15°C (to separate the liquid from the solids) and the skins are pressed [98]. Sulphur dioxide may again be added at the pressing stage. The pressing may be achieved using a hydraulic, pneumatic or horizontal press [5]. The sweet pomace<sup>12</sup> that is left in the press after pressing has occurred should be disposed of and removed from the winery immediately as it attracts insects and other pests [134].

At this stage, the juice may be clarified (and settled using pectic enzymes) to an extent depend-

---

<sup>10</sup>With certain production methods, such as Champagne production in France, the grapes may be pressed without being crushed. This method is seldomly used in South Africa.

<sup>11</sup>The activity of these enzymes are strongly temperature-dependant. At 10°C they have between 15 and 25 percent of their activity, with optimum activity at a temperature of 45 to 50°C. At 60°C their activity rapidly starts to decrease until they become completely inactive at 80°C. Consequently, a conflict exists between the desire for cool temperatures for grape-handling operations and the need for efficient activity of the enzymes.

<sup>12</sup>In white wine production the pomace is the sweet, pale brownish-green mass of grape skins, stems, seeds and pulp left in the press after pressing, but still before any fermentation has taken place, whereas in red wine production the pomace is a similar mass of grape debris coloured blackish red left once the free-run wine has been drained after primary fermentation [100].

ing on the preference of the winemaker. It may also be filtered in order to remove suspended solids. These operations should occur at a temperature not exceeding 15°C. The juice is now inoculated with a selected pure yeast culture. Diammonium phosphate may also be added as supplementary inorganic nitrogen to restrain the possible ensuing formation of hydrogen sulphide during fermentation and also to assist in obtaining complete fermentation of the sugar.

The next step is the temperature controlled fermentation which occurs in a fermentation vessel at temperatures cooler than room temperature [100] from 10 to 18 degrees Celsius [96]. These vessels are usually made of stainless steel (for high-volume everyday wines) since it holds the advantage that both cleaning and temperature control are much easier than with other vessel materials [100]. A fermentation vessel should be filled to about 70 % of its capacity. This allows room for foaming to occur without damaging the fermentation lock. Oak barrels are still used for some premium style wines (mostly red, or Chardonnay and Sauvignon Blanc [98]) and concrete or fibreglass may also be used [5]. The yeast now converts sugars into ethanol (alcohol) and this process is referred to as *alcoholic fermentation* [100]. The reduction in sugar content during alcoholic fermentation is approximately linear and preferably between 0.7 and 1.4° Brix per day [98]. Bentonite may also be added during fermentation in order to avoid bentonite fining as a separate subsequent operation [98].

There exists a secondary fermentation, referred to as *malolactic fermentation*, which is usually associated with red wines only, but in modern wine making it may also be applied to a select few white wine styles. In certain white wines it is definitely not wanted and is therefore discussed in §2.3.2 where the production of red wine is considered.

After fermentation, the wine as allowed to settle, is then racked (decanted) under carbon dioxide off *gross lees* (the old English word for the sediment that settles at the bottom of a container [100]) and is protected against invasion of lactic bacteria [5]. At racking, sulphur dioxide is added to induce a preferred pH of between 3.1 and 3.4; ascorbic acid may also be added [98]. Racking may be delayed for weeks with some varieties, such as Chardonnay, during which time malolactic fermentation may occur. Only a small percentage of white wine comes into contact with wood — therefore barrel fermentation and ageing is sometimes present in the production of certain white wines, especially wines made from the Chardonnay grape.

Cool temperature and minimized exposure to air as well as the minimum handling of wine are of renowned importance in white wine making [98]. A hazy translucent appearance is common in young white wines (2–4 months old) mainly due to proteins and other organic residues. New wines may also contain excessive potassium and tartrate ions which eventually crystallizes as potassium bitartrate (*cream of tartar*). Once the various batches of wine are blended to form a uniform bulk, or *coupage* as it is known in French, the wines may be fined and stabilised. Finings such as gelatin, kieselsol or bentonite clay mixtures are recommended for clarification or fining of the haze [134]. After the finings have been added, the wine tanks are filled, sealed and allowed to rest for 4–6 hours at normal room temperature after which cold stabilisation may commence. It may be cold stabilised by chilling to between -4°C and 2°C for up to three weeks [134, 98]. Potassium bitartrate crystals may be added if rapid stabilisation is required [98].

The chilled and stabilised wine is then racked for a second time to remove the crystalline tartrate deposit under inert gas. It is of utmost importance that the wine should not be exposed to air in its cold state since the solubility of dissolved oxygen is much greater in cold than in warm wine. This should be noted especially during the pumping of the wine. The pH of the wine should now be analysed, as well as the free sulphur dioxide, ascorbic acid and sugar (to suit the specific style of the wine) after which it is ready for bottling.

## The production of red wine

Red wine making is even less predictable than the making of white wine and is still considered something of an art. This is in part due to the greater number of variables on which the composition and quality of red wine depends. These are aspects such as the time and temperature of the juice on skins during fermentation, various methods of colour extraction, the extent and type of wood ageing and more handling of the wine [98]. A typical red wine making method is shown in Figure 2.7 in the form of a flowchart.

The grape maturity at harvest depends on the style of wine intended and the fruit flavour present in the grapes. The riper the grapes, the heavier the wine is in body and (generally) in flavour, as well as the higher in alcohol. The grapes used for red wine making usually range from 18 to 25.2° Balling [98].

Once the mature red (black) grapes have been harvested, the quality and further details of the grapes are documented and the preferably cool grapes are loaded into the tipping bin. They are then destemmed and crushed, after which the stems are discarded. The grape bunches may be wholly or partially crushed during crushing [96]. Stems may also be included in the fermentation if the winemaker wishes to do so, but it is not usual [98]. The destemmed and crushed must is then pumped to a fermentation vessel where tartaric acid, a small quantity sulphur dioxide and a selected yeast culture may be added. During the alcoholic fermentation, yeast converts sugars into alcohol. The formed alcohol assists with the extraction of pigments and tannins from the skins, which contribute to the flavour. Leaving the partially fermented wine on the skins to draw out more tannin, colour and flavour is referred to as *maceration* [5]. Without maceration, wine made from dark-skinned grapes is merely pink [100]. The combined process of fermentation and maceration may take anything between 24 hours and three weeks depending on the colour of the final product required [96]. During the fermentation, the process of colour extraction on the skins may be carried out in several ways. The floating skins (the cap) may be punched down at regular intervals during the fermentation so that they mix with the fermenting juice. There are also various systems by which the juice is taken from under the cap of skins and irrigated over the skins in an intermittent manner. Skins may also be submerged with head boards so that the skins are in constant contact with the juice and there exist specifically designed red wine fermenters to assist in this process. Colour may also be extracted with the use of heat [98]. During fermentation, winemakers may add oak chips or shavings to create an oak complexity. Oxidation and harsh flavour may result if this process is not approached very carefully. It is important to note that these additions only impart oak complexity and are never a satisfactory replacement for oak-cask maturation [98].

Once the winemaker is satisfied with the colour, flavour and tannin extraction, the free-run wine may be drained from the fermented must and the rest of the must pressed. As with white wine making, the pressed wine may be blended with the drained wine or kept separate, depending on the style of wine required. At this stage, the wine may be racked off yeast deposits. The wine usually undergoes *malolactic fermentation*, also referred to as secondary fermentation since it almost never precedes alcoholic fermentation. Malolactic fermentation consists of the conversion of malic acid, naturally present in new wine, into the weaker lactic acid and carbon dioxide [100]. This happens naturally, but may also be artificially induced via the injection of lactic bacteria [5]. During this stage the wine may be given a light fining of bentonite or egg white to settle the suspended material and, if necessary, the acidity may be adjusted with the addition of tartaric acid to between pH 3.3 and 3.6 [98].

The wine is then matured in either oak *barrisques*, historically of a 225 litre capacity, or larger steel tanks depending on the required style of the wine. Wood is porous and therefore exposes

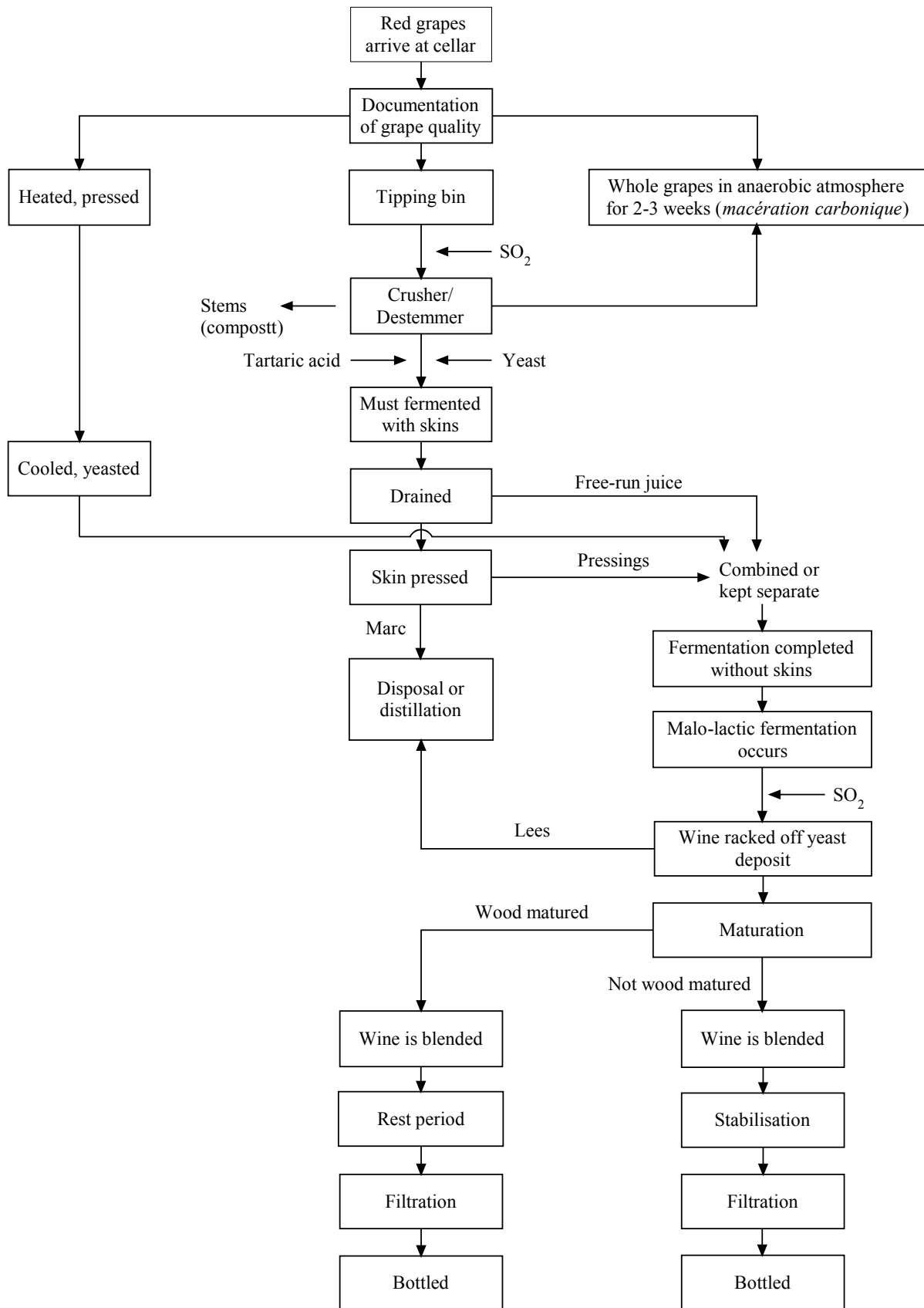


Figure 2.7: A flowchart for making dry red table wine [39, 98].



the wine to some oxygen [100]. During maturation the wine extracts tannin, flavour and colour from the wood. Barrel maturation also encourages clarification and stabilization of the wine in the most natural way [100]. The extra dimensions of flavour the wood provides depend on the origin of the oak, the size and age of the barriques, the number of times they have been used and the length of time the wine spends in them. Over time the wine is transferred between vats or barriques to rack off its lees and to allow limited aeration [5]. The wine may need two or three rackings before it can be filtered.

After ageing the wine is blended and tasted again, analysed and may be stabilised, then filtered and lightly sulphited if necessary. This process includes the removal of unwanted suspended substances not visible to the human eye, such as proteins, by chemical means to stabilize the wine and improve clarity. The fining process may also occur naturally during a lengthy ageing in the barrel. The wine may now be bottled; bottle-ageing under cool temperatures improves the quality.

## 2.4 Chapter overview

The goal of this chapter was to provide the reader with a basic understanding of wine and of the South African wine industry. In §2.1, the development of the South African wine industry organisational structure as well as its current state were discussed. Some statistics on wine production in South Africa as well as the import and export of wine were given in §2.2, and in §2.3 viticultural practices and an overview of wine making methods were discussed.



---

---

## CHAPTER 3

---

# Methodological background

### Contents

3.1	Introduction to the scheduling problem . . . . .	34
3.1.1	<i>Machine environment</i> . . . . .	34
3.1.2	<i>Processing restrictions and constraints</i> . . . . .	36
3.1.3	<i>Performance measures and optimality criteria</i> . . . . .	37
3.1.4	<i>Representation of task sets and the corresponding schedules</i> . . . . .	38
3.2	A concise survey of literature . . . . .	42
3.2.1	<i>The job shop scheduling problem</i> . . . . .	43
3.2.2	<i>Decision support in the wine industry</i> . . . . .	45
3.3	Solving scheduling problems via mathematical programming . . . . .	46
3.3.1	<i>Mixed integer programming in the context of scheduling</i> . . . . .	46
3.3.2	<i>The branch-and-bound method</i> . . . . .	48
3.4	Tabu search methodology . . . . .	50
3.5	Chapter overview . . . . .	53

Scheduling is concerned with the allocation of scarce resources (such as equipment, labour and space) to jobs, activities, tasks or customers over time and is characterized by a virtually unlimited number of problem types. The objective in scheduling is to optimize one or more (possibly conflicting) performance measures [21, 80].

The study of scheduling dates back to the start of the twentieth century with the work of Henry Gantt<sup>1</sup> and other pioneers. The scheduling problems of the 1950s, considered when researchers were faced with the problem of managing various activities occurring in workshops, were very simple. From there onwards, these problems steadily grew in complexity and during the 1970s many scheduling problems were shown to be NP-hard [80]. A number of efficient algorithms have been developed to provide optimal solutions to simple problems, whilst heuristics are typically used for the most complex scheduling problems.

The notation that is commonly used in classical scheduling problems, and also throughout this thesis, is reviewed in §3.1. This section also contains an overview of the three-field  $\alpha|\beta|\gamma$  notation used to classify classical scheduling problems as well as an overview of typical representations of schedules, given in §3.1.4. A concise survey of literature is presented in §3.2, containing both references to literature on the scheduling problem and other applications of optimization and

---

<sup>1</sup>Henry Laurence Gantt was an industrial who developed the now famous Gantt chart in the 1910s [93].

decision support in the wine industry. The optimization methods applied to solve the scheduling problems in this thesis are discussed in §§3.3 and 3.4, the exact and meta-heuristic methods respectively.

### 3.1 Introduction to the scheduling problem

Scheduling problems may generally be characterized by three sets: a set  $\mathcal{J} = \{J_1, \dots, J_n\}$  of  $n$  jobs, a set  $\mathcal{P} = \{P_1, \dots, P_m\}$  of  $m$  processors (*machines*) and an optional set of  $s$  additional resources denoted by  $\mathcal{R} = \{R_1, \dots, R_s\}$ . The subscript  $j$  generally refers to a job and the subscript  $i$  to a processor [14, 16, 80].

The process of scheduling generally entails an assignment of processors from  $\mathcal{P}$  and optional resources from  $\mathcal{R}$  to jobs from  $\mathcal{J}$  over a set period of time which may be divided into smaller *time intervals* [14, 21].

A job  $J_j \in \mathcal{J}$  may be characterized by its processing times, release date, due date, deadline and weight. These five characteristics are all considered as *static* data, since they do not depend on the schedule. On the other hand, data that are not fixed as such, are referred to as *dynamic* data [94]. The *processing time*, denoted by  $p_{ij}$ , refers to the time required by job  $J_j$  to be processed on machine  $P_i$ . The subscript  $i$  of  $p_{ij}$  may be omitted if the job  $J_j$  may be processed on any machine [80]. The *release date*,  $r_j$ , of job  $j$  is the time instance at which the job arrives in the system. It is therefore the earliest time instance at which the processing of a job can start. The *due date*,  $d_j$  represents the time job  $j$  is expected to be complete, as determined by a production schedule. If a job is completed after the due date, it incurs a penalty cost. However, the *deadline*,  $\bar{d}_j$ , of job  $j$  is the hard, real-time limit by which the product must be completed and which may not be exceeded. The importance of a job  $j$  is reflected by its *weight*,  $w_j$  [14, 16, 80].

A schedule is *feasible* if each processor handles at most one job at a time, if no job occurs on more than one processor at a specific time and, in addition, if a number of problem-specific characteristics are met. A schedule is said to be *optimal* when a given optimality criterion is minimized.

A scheduling problem may be described by a triplet  $\alpha|\beta|\gamma$ , where the field  $\alpha$  describes the machine environment and contains a single entry. The processing characteristics and constraints are described in the field  $\beta$  and may contain no entry or multiple entries. The field  $\gamma$  describes the objective to be minimized and usually contains a single entry [93]. These three fields are discussed in some detail in §§3.1.1–3.1.3.

#### 3.1.1 Machine environment

The first field,  $\alpha$ , describing the machine environment may take on the values 1,  $P_m$ ,  $Q_m$ ,  $R_m$ ,  $J_m$ ,  $FJ_c$ ,  $F_m$ ,  $FF_c$  or  $O_m$ .

If  $\alpha = 1$ , then there is only a *single processor* in the system and each job therefore consists of a single operation. This is the simplest case of all possible machine environments and is a special case of all other more complicated machine environments [21, 80, 93].

When  $\alpha \in \{P_m, Q_m, R_m\}$ , parallel processors are used. In this case each job may be processed on any of the processors  $P_1, \dots, P_m$ . If  $\alpha = P_m$ , then there are  $m$  *identical processors in parallel*. Each job  $j$  requires only one operation and may be processed on any of the  $m$  parallel processors.

If  $\alpha = Q_m$ , there are  $m$  *uniform processors*, that is  $m$  processors in parallel, but operating at different speeds. *Unrelated processors* are present when  $\alpha = R_m$ , in which case the  $m$  parallel processors can each process the jobs at a different speed, *i.e.* the speeds are job-dependent [21, 80, 93].

A job  $J_j$  may be further divided into a set of  $k_j$  tasks,  $T_{j1}, \dots, T_{jk_j}$ , where two consecutive tasks in the sequence are to be performed on different processors. Such an environment may be referred to as *multi-operational*. The set of tasks is written as

$$\mathcal{T} = \{T_{11}, \dots, T_{1k_1}, T_{21}, \dots, T_{2k_2}, \dots, T_{n1}, \dots, T_{nk_n}\}$$

with the *vector of processing times*,

$$\mathbf{p}_{jk} = [p_{jk}(1), p_{jk}(2), \dots, p_{jk}(m)]^T,$$

representing the time required by the different processors to perform a task. Here  $p_{jk}(i)^2$  is the time required by processor  $P_i$  to process task  $T_{jk}$ , for all  $k = 1, \dots, k_j, j = 1, \dots, n$  and  $i = 1, \dots, m$ . Furthermore, a *set of processors*  $\mu_{jk} \subseteq \{P_1, \dots, P_m\}$  is associated with each task,  $T_{jk}$ . Task  $T_{jk}$  may be processed on any of the processors specified in  $\mu_{jk}$  [21]. In most cases, all  $\mu_{jk}$  are one element sets (in such a case the processors are referred to as *dedicated processor*) or all  $\mu_{jk}$  are equal to  $\mathcal{P}$ , the full set of processors (in which case the processors are referred to as *parallel processors*). A *multi-purpose processor* is typically used in flexible manufacturing where processors are equipped with different tools, which means that a task can be processed on any machine with the appropriate tool. Instances also exist where all processors in the set  $\mu_{jk}$  are utilized by task  $T_{jk}$  simultaneously during the entire processing period. This type of scheduling problem is referred to as a *multiprocessor task scheduling problem* [21]. The multi-operation models are denoted by  $\alpha \in \{J_m, FJ_c, F_m, FF_c, O_m\}$ . A brief overview of each of these multi-operation models may now be given.

In a *job shop* with  $m$  dedicated processors, indicated by  $\alpha = J_m$ , each job has its own predetermined route that it must follow during production, *i.e.* a special precedence relation of the form  $T_{j1} \prec T_{j2} \prec \dots \prec T_{jk_j}$  exists for all  $j = 1, \dots, n$ . Such a precedence constraint in the set  $\mathcal{T}$  means that the processing of  $T_{j1}$  must be completed before the processing of  $T_{j2}$  can be started, and so forth. A job shop with  $\mu_{jk} = \mu_{j(k+1)}$  is referred to as a *job shop with machine repetition*. *Recirculation* is the term used to describe the situation where a job may visit a processor more than once in a job shop. The *flexible job shop*, denoted by  $\alpha = FJ_c$ , is a combination of the job shop and the parallel machine environment<sup>3</sup>. Rather than having  $m$  processors, there are  $c$  work centres, each with a number of identical processors that function in parallel. Each job still has its own predetermined route that it must follow through the shop during production, and job  $j$  requires processing at each work centre on any of the processors. In this case, if a job may visit a work centre more than once on its route through the shop, it is once again referred to as *recirculation* [21, 93].

If  $\alpha = F_m$ , the situation with  $m$  processors in series arises, called a *flow shop*. The criteria for a flow shop are that each job has to be processed on each of the  $m$  processors and that all jobs must follow the same route (*i.e.* first on  $P_1$  then on  $P_2$ , etc.). A flow shop may therefore be expressed as a special case of the job shop in which  $k_j = m$  for  $j = 1, \dots, n$  (the number of tasks required for each job is equal to the number of processors) and  $\mu_{jk} = \{P_k\}$  for each

<sup>2</sup>It is sometimes necessary to work with a set of variables rather than vectors, in such a case  $p_{jk}(i)$  may be denoted as  $p_{ijk}$ .

<sup>3</sup>Flexible job shop problems are sometimes referred to as job shop scheduling problems with multi-purpose machines[21]

$k = 1, \dots, m$  and  $j = 1, \dots, n$  (the task  $T_{kj}$  for each job is processed only on processor  $P_k$ ). Once a job has been completed on one processor, it queues at the next processor. If these queues operate according to the *First In First Out (FIFO)* principle, the flow shop is referred to as a *permutation* flow shop and the field  $\beta$  includes the entry *prmu*. Similar to the flexible job shop, the *flexible flow shop* ( $\alpha = FF_c$ ) is a combination of the flow shop and the parallel processor environments<sup>4</sup>. However, in such a case there are  $c$  *flow shop stages* in series, instead of  $m$  processors in series, each with a number of identical processors in parallel at each stage. Each job requires processing once at every stage, first at stage 1, then at stage 2, and so forth. A stage serves as a collection of parallel processors on which processing of a job is required on any one processor [21, 80, 93].

An *open shop*, denoted by  $\alpha = O_m$ , comprises  $m$  processors where each job has to be processed on each of the  $m$  processors. Some of the processing times may be set to zero. There are no restrictions with regard to the routing of each job through the machine environment and therefore the open shop may be seen as a flow shop without precedence relations between tasks [93].

### 3.1.2 Processing restrictions and constraints

The second field,  $\beta$ , is used to describe the processing restrictions and constraints, and may consist of a variety of entries. Possible entries in the field  $\beta$  are described in this section [16, 93].

A schedule is referred to as *preemptive* (indicated by the presence of the parameter *pmtn* in the field  $\beta$ ) if the processing of a job may be interrupted at any point in time and a different job put on the processor instead. The amount of processing received by such an interrupted job is not lost. When preemption of all tasks is disallowed, the schedule is referred to as *non-preemptive* (in which case the entry, *pmtn*, is omitted from the field  $\beta$ ).

Any specified additional resources are indicated by the presence of the entry *res* in the field  $\beta$ .

*Precedence constraints* require that one or more tasks may have to be completed before another task is allowed to be processed. If the order of execution of at least two tasks in  $\mathcal{T}$  is restricted by such a constraint, the tasks in the set  $\mathcal{T}$  are called *dependent*. Without the presence of such a constraint, the tasks may be referred to as *independent*. Precedence constraints are implied when the entry *prec* is included in the field  $\beta$ . Several special forms of precedence constraints exist. For example, a constraint that requires each job to have at most one successor, is referred to as an *intree* and when each job has at most one predecessor, the constraints are referred to as an *outtree*, which may be denoted by *intree* and *outtree* in the field  $\beta$  respectively. *Unconnected activity networks* are denoted by  $\beta = uan$  and a *chain*, which occurs if each task has at most one predecessor and at most one successor, is indicated by writing  $\beta = chains$ . If no such entry is present in the field  $\beta$ , independent tasks are implied. Precedence relations of a set of tasks may be represented by an acyclic<sup>5</sup> directed graph [21].

To indicate that processing of a job  $J_j$  is not allowed to start its processing before its release date, the symbol  $r_j$  is included in the field  $\beta$ , thereby allowing release dates to be specified for each job. If the symbol is omitted, the processing of job  $J_j$  may start at any time [21, 80].

Some entries appearing in the field  $\beta$  are self-explanatory. For example, the entry ( $p_j = p$ ) implies that all processing times are equal and when ( $\underline{p} \leq p_j \leq \bar{p}$ ) is included, no processing

<sup>4</sup>Flexible flow shop problems are sometimes referred to as flow shop scheduling problems with multi-purpose machines [21].

<sup>5</sup>Acyclic is a term used to describe graphs which contain no cycles.

time  $p_j$  is allowed to be less than  $\underline{p}$  or greater than  $\bar{p}$ . If no entry regarding such processing times appears, the tasks have arbitrary processing times. If  $\bar{d}$  is present in the field  $\beta$ , deadlines are imposed on the performance of a task set, but if the  $\bar{d}$  is omitted, no deadlines are assumed in the system. Also, if  $(d_j = d)$  is included, it means that all due dates are equal (although due dates are usually not explicitly specified in the field  $\beta$ ). The maximum number of tasks in a job in the case of job shops may be limited by including the entry  $(k_j \leq \ell)$ .

If the symbol,  $M_j$ , appears in the field  $\beta$  when the machine environment is that of  $P_m$ , not all  $m$  processors are capable of processing a job  $j$ . Therefore, the set of processors defined by  $M_j$  denotes the set of machines that can process job  $j$ .

When *recrc* is present in the field  $\beta$ , recirculation may occur in a job shop or flexible job shop. When *prmu* is present, then the queues in front of each processor in a flow shop, operate according to the FIFO principle. It is therefore implied that the permutation (or order) in which jobs are processed on the first processor, is maintained throughout the system.

The last example of field  $\beta$  entries, is the no-wait (*nwt*) entry. This entry is applicable to flow shops and when present, processing of a job must immediately start on the consecutive processor once a job has been completed on the one processor. Therefore the starting time of a job at the first processor has to be delayed in order to ensure that the job can go through the flow shop without having to wait for any processor.

### 3.1.3 Performance measures and optimality criteria

The last field indicates an optimality criterion, serving as a means to measure the performance of a schedule. The scheduling problem entails finding a feasible schedule which minimizes the specified objective function. In order to define some of the most generally used objective functions, some terminology is required. If a variable, say  $X_j$ , is used to describe a certain property of the derived schedule, the associated cost function is then defined by  $f_j(X_j)$ . Such a variable may be the *completion time* of job  $j$  and is denoted by  $C_j$ . The completion time of a job indicates the time required in order to complete the job<sup>6</sup>. It is important to note the difference between the processing time of a job and its completion time. The processing time refers to the amount of time required in order to process the job on a specific machine, whereas the completion time denotes the time instance when a job is completed, therefore including the time required to process its predecessors and any unassigned time intervals preceding the job. An objective function may also be a function of the due dates. The *lateness* of a job  $J_j$  is defined as

$$L_j = C_j - d_j$$

and is a positive value when the job is completed late or a negative value when job  $J_j$  is completed early. The *tardiness* of a job  $J_j$  is defined as

$$T_j = \max\{0, L_j\},$$

resulting in a positive value when the job is completed early and a value of zero when the job is completed late. The *unit penalty* of job  $J_j$  is defined as

$$U_j = \begin{cases} 0 & \text{if } C_j \leq d_j \\ 1 & \text{otherwise,} \end{cases}$$

<sup>6</sup>In the case of a non-flexible multi-operation model, such as the job shop scheduling problem, the completion time of job  $J_j$  refers to the sum of the completion times of the individual tasks  $T_{jk}$ . Therefore, in such a multi-operation model,  $C_j = \sum_{k=1}^{k_j} C_{jk}$ .

indicating whether or not a job has been completed before its due date. Objective functions are most often expressed as one of two types of functions. First there are the *bottleneck objectives*,

$$f_{\max}(X) = \max \{f_j(X_j) \mid j = 1, \dots, n\},$$

indicating that the objective is to minimize the largest (maximum) value occurring in a range of cost functions, denoted by  $f_j(X_j)$ . The other is the *sum objectives*,

$$\sum f(X) = \sum_{j=1}^n f_j(X_j),$$

which aims to minimize the sum of the individual cost functions, denoted by  $f_j(X_j)$ .

The most common objective function to be minimized is the *makespan* defined as  $C_{\max} = f_{\max}(C)$ . The objective here is to minimize the longest completion time<sup>7</sup>. High utilization of processors is achieved if a minimum makespan is used [93]. The *maximum lateness* of a scheduling problem is defined as  $L_{\max} = f_{\max}(L)$  and measures the worst violation of the due dates, *i.e.* attempting to minimize the largest value of  $L_j$ . The *total weighted completion time*,  $\sum w_j C_j = \sum f(w_j C_j)$ , gives an indication of the total holding or inventory costs incurred by the schedule. The completion time is often referred to as the *flow time* in which case the total weighted completion time is referred to as the *weighted total flow time*. The *total weighted tardiness* may be defined as  $\sum w_j T_j = \sum f(w_j T_j)$ , whilst the *weighted number of tardy jobs* is given as  $\sum w_j U_j = \sum f(w_j U_j)$ .

When an objective function is nondecreasing with respect to all  $C_1, \dots, C_n$ , it is referred to as *regular* performance measures. Therefore all the objective functions defined so far are regular. When an objective function involves variables such as *earliness*  $E_j = \max\{0, d_j - C_j\}$ , *absolute deviation*  $D_j = |C_j - d_j|$ , or *squared deviation*  $S_j = (C_j - d_j)^2$ , it is not a regular objective function [21].

If it is not possible to schedule jobs (or tasks) earlier without violating some constraint, the schedule is called *active*. Whereas, a schedule is referred to as *semiactive* if no job (or task) can be processed earlier without changing the processing order or violating the constraints [21].

In order to illustrate the  $\alpha|\beta|\gamma$  notation, two examples are now given. The following examples, illustrating the  $\alpha|\beta|\gamma$  notation, are continued as a series of further examples in order to illustrate the possible representation of a feasible schedule.

**Example 3.1** *The scheduling problem  $P_3|prec;p_j = 2|C_{\max}$  refers to scheduling jobs on three identical parallel processors for which the processing of any task requires 2 units of time. An optimal schedule will be the one that minimizes the makespan. ■*

**Example 3.2** *The scheduling problem  $J_4||C_{\max}$  refers to the problem of minimizing the completion time on a four processor job shop. Processing times should still be specified. ■*

### 3.1.4 Representation of task sets and the corresponding schedules

The job or task set may be defined in a variety of manners, depending on the required set of activities to be processed. Precedence constraints may be specified by means of a graphical

<sup>7</sup>If the feasible schedule is represented as a graph, the makespan is equal to the length of a longest path in the graph.



representation by which an acyclic directed graph  $G = (V, A)$  with  $V = \{1, \dots, n\}$  representing the corresponding jobs and  $(j, k) \in A$  if and only if  $J_j \prec J_k$  [21]. Such a graph is referred to as a *task-on-node representation*, but there is also a so-called *task-on-arc representation* where a task is defined as an arc and dummy arcs are included [16]. The most popular way of then graphically displaying the schedule, is by means of a *Gantt chart*. A Gantt chart may be described as a bar chart on which time is represented on the one axis and other qualities (such as job processing) on the other axis [14].

**Example 3.3 (Example 3.1 continued)** *For the scheduling problem defined in Example 3.1, the jobs are subject to the precedence constraints as listed in Table 3.1<sup>8</sup>.*

Job $j$	Predecessors
1	-
2	-
3	$J_1$
4	$J_1, J_2$
5	$J_4$
6	$J_4$
7	-
8	$J_5$
9	$J_6, J_7$

Table 3.1: The predecessors for each of the nine jobs of Example 3.3 as required by the precedence constraints. It is unnecessary to indicate the processing time for each job, since it has been specified that all jobs have a processing time of two.

*The precedence constraints may also be illustrated using a precedence constraints graph [93]. The task-on-node approach is the more general approach, which is also adopted throughout the remainder of this thesis. For such a graph, vertices correspond to tasks and arcs correspond to precedence constraints. The precedence constraints listed in Table 3.1 are expressed as a task-on-node graph in in Figure 3.1(a). Another approach is the task-on-arc representation where arcs represent tasks and the vertices time instances. The same precedence constraints is shown as a task-on-arc graph in Figure 3.1(b).*

*Figure 3.2 shows a Gantt chart representing a feasible schedule for jobs as required by the precedence relation of the jobs and their processing times. In this graph, the processors are on the vertical axis, but it is also possible to have a Gantt chart represent the same schedule, but with the jobs on the vertical axis [14].*

*The objective of the scheduling problem for which this schedule was determined is to minimize the makespan. The respective completion times for the nine jobs may be listed as  $C_j = \{2, 2, 8, 4, 6, 6, 2, 8, 8\}$ ; therefore  $C_{max} = 8$ . ■*

The properties of the set of tasks of the job shop problem described in Example 3.2 may be displayed in table form, displaying the set route of each task through the system. In this case the processing times should also be listed. As illustrated in Example 3.3, the precedence relations may also be expressed by means of a directed graph. Another commonly used graphical

<sup>8</sup>Since the processing time for each job in Example 3.1 as given as  $p_j = 2$ , processing times are not included in Table 3.1. If the different jobs did have different processing times on specified machines, these times would also be included in the same table.

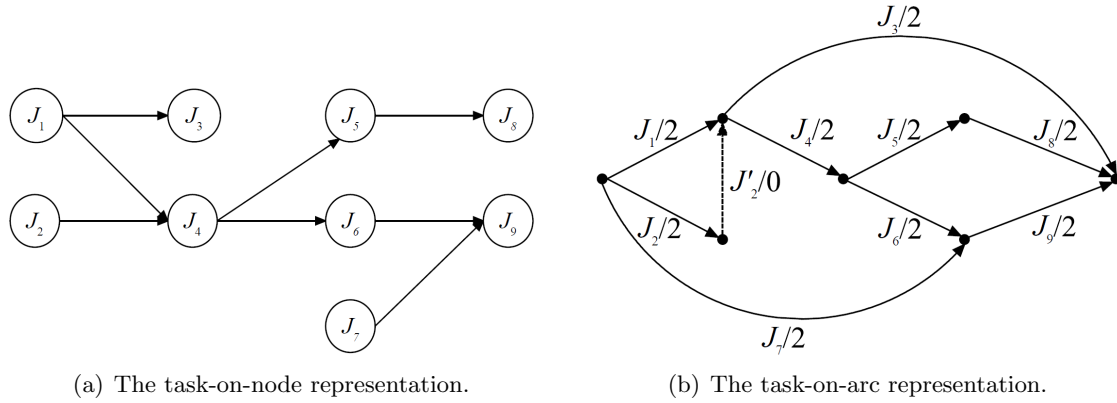


Figure 3.1: The precedence relation of the nine jobs for the scheduling problem of Example 3.3, as listed in Table 3.1, expressed as a task-on-node and a task-on-arc graph respectively.

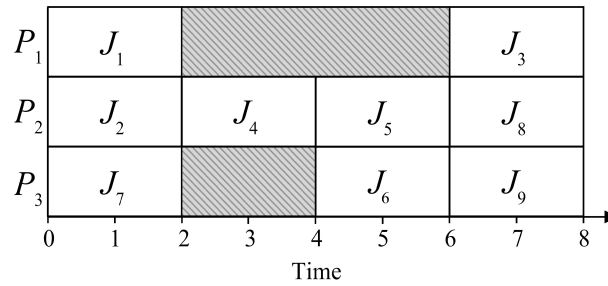


Figure 3.2: A Gantt chart representation of a solution to a scheduling problem with processing sequences as shown in Figure 3.1.

representation of a job shop schedule, proposed in 1964 by Roy and Sussman, is the *disjunctive graph* representation [18]. The disjunctive graph is a directed graph  $G = (V, A \cup D)$ . It is an *arc-weighted graph* where  $V$  denotes the set of vertices corresponding to the tasks of jobs (all  $T_{jk} \in \mathcal{T}$ ) as well as two dummy vertices denoting the start ( $T_{\text{start}}$ ) and end ( $T_{\text{end}}$ ) of the schedule. Both dummy tasks have a processing time of zero. The set  $A$  of *conjunctive arcs*<sup>9</sup> reflects the precedence constraints and initially joins every two consecutive tasks of the same job. A conjunctive arc also joins the first task  $T_{j1}$  of job  $J_j$  to its predecessor  $T_{\text{start}}$  and the last task  $T_{jk_j}$  of job  $J_j$  to its successor  $T_{\text{end}}$ . Every conjunctive arc  $(jk, \ell m)$  is labelled with the processing time  $p_{jk}$  corresponding to  $T_{jk}$ . The set  $D$  consists of *disjunctive arcs* with two opposite directions<sup>10</sup>. Such a disjunctive arc joins two mutually unordered tasks which require the same machine for their processing. For the *uncompleted* version of the disjunctive graph there are two oppositely directed arcs  $(\ell m, jk)$  and  $(jk, \ell m)$ , together referred to as a *disjunctive arc pair*, for each pair of tasks  $\{T_{\ell m}, T_{jk}\}$  that require the same processor  $P_i$  and no label. The conjunctive arcs between tasks represent the precedence constraints on tasks of the same job. The constraint that each processor can handle at most one task at a specific time interval is represented by the *disjunctive arcs*. Let  $\mathcal{E}_i$  be the set of all pairs of tasks to be processed on the same processor,  $P_i$ . Then a disjunctive arc pair,  $\{(\ell m, jk), (jk, \ell m)\}$ , is referred to as *settled* if exactly one arc of the pair has been added to a set  $\mathcal{D}_i \subset \mathcal{E}_i$ . A *completed disjunctive graph*<sup>11</sup> occurs when all disjunctive arc pairs are settled and if such a graph is acyclic, it corresponds

<sup>9</sup>When an edge in a graph has a direction, it is referred to as an arc.

<sup>10</sup>These arcs are in effect edges since a path may lead in either direction of the arc, but since this is a directed graph such arcs are marked with arrows pointing in both directions in order to avoid confusion.

<sup>11</sup>A completed graph in this sense differs from the notion of a *complete graph* in Graph Theory (a graph that contains all possible edges).

to a feasible schedule. A feasible schedule may therefore be defined by the set  $\mathcal{D}^* = \cup_{i=1}^m \mathcal{D}_i$ , where  $\mathcal{D}^* \subseteq \mathcal{E} = \cup_{i=1}^m \mathcal{E}_i$  such that

- I.  $(\ell m, jk) \in \mathcal{D}^*$  if and only if  $(jk, \ell m) \in \mathcal{E} \setminus \mathcal{D}^*$  and
- II. the graph  $G(\mathcal{D}^*) = (V, \mathcal{A} \cup \mathcal{D}^*)$  is acyclic.

The makespan,  $C_{\max}$ , of a feasible schedule is also the length of a *longest path* from the vertex  $T_{\text{start}}$  to the vertex  $T_{\text{end}}$  in a completed disjunctive graph [14, 18].

**Example 3.4 (Example 3.2 continued)** *In Table 3.2, the predetermined processing sequences for the tasks of three jobs for the four-machine job shop from Example 3.2 are listed. The table also indicates the processors associated with each of these tasks as well as their individual processing times on the corresponding processors.*

$j$	$i$	$T_{j1}$		$T_{j2}$		$T_{j3}$		$T_{j4}$	
		$p_{j1}(i)$	$i$	$p_{j2}(i)$	$i$	$p_{j3}(i)$	$i$	$p_{j4}(i)$	
1	1	1	4	4	2	2	3	1	
2	3	1	1	4	-	-	-	-	
3	4	3	2	2	3	1	-	-	

Table 3.2: The processors ( $P_i$ ) assigned to each of the tasks  $T_{jk}$  including the respective processing times ( $p_{jk}(i)$ ) for each task of the four machine job shop problem of Example 3.4.

*The constraints on the tasks, i.e. the processing sequence, may be represented by means of an uncompleted disjunctive graph. As with previous notation, when focussing on a single vertex,  $j$  denotes the particular job  $J_j$ ,  $\ell$  refers to a specific task of job  $J_j$  and  $i$  the machine  $P_i$  on which processing is required. A single vertex is illustrated in Figure 3.3.*

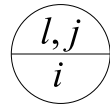


Figure 3.3: A single vertex of the disjunctive graph representation, used to illustrate processing sequences and feasible schedules, where  $j$  refers to job  $J_j$ ,  $\ell$  to a task of job  $J_j$  and  $i$  to processor  $P_i$  required to completed the associated task.

*The uncompleted disjunctive graph which serves as a means to illustrate the information contained in Table 3.2, is given in Figure 3.4. The start and end vertices are referred to as dummy vertices. Furthermore, the conjunctive arcs (solid arcs) represent the precedence constraints among tasks of the same job and the dashed arcs (with arrows at both ends) indicate the disjunctive arc pairs used to illustrate the constraints among tasks to be performed on the same processor. It is important to note that this is a representation of the processing sequences amongst tasks and their processing requirements, and not of a feasible schedule. In order to form a completed disjunctive graph, representing a feasible schedule for the problem, all disjunctive arc pairs should be settled in such a way that graph  $G(\mathcal{D}^*)$  is acyclic. A completion of the disjunctive graph of Figure 3.4 is given in Figure 3.5. The graph denotes a schedule with a makespan of  $C_{\max} = 10$  units of time. The makespan may be calculated by determining the longest path from the start to end vertices of the graph, indicated in Figure 3.5 by means of thick solid arcs.* ■

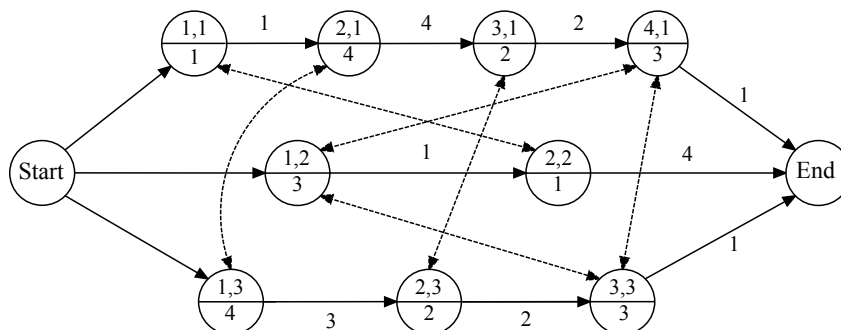


Figure 3.4: The uncompleted disjunctive graph for the processing times and processing sequences for the job shop problem from Example 3.4, as displayed in Table 3.2.

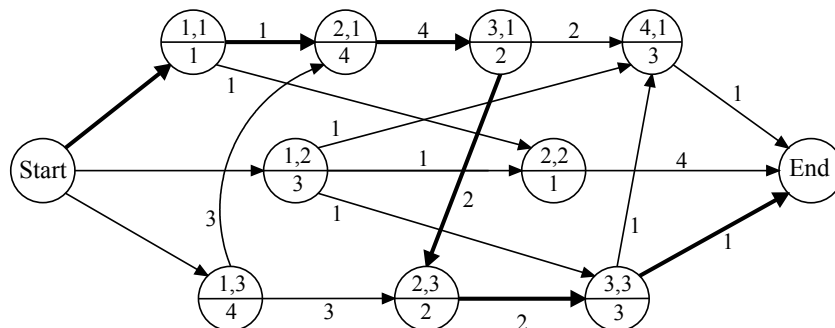


Figure 3.5: The completed disjunctive graph representing a feasible schedule that satisfies the constraints displayed in Figure 3.4 with a longest path from the start to end vertex indicated by the thicker solid arcs.

## 3.2 A concise survey of literature

Since the 1960s, scheduling problems have been the focus of a significant number of studies in the literature and a wide variety of techniques have been applied to solve problems of this nature. One of the most popular classifications of scheduling problems is without doubt the job shop scheduling problem which has become well known for being particularly difficult to solve [61]. In 1976 it was shown that the job shop scheduling problem with more than one machine is an **NP**-complete problem [14, 47]. Unfortunately, there exist far less literature focusing on the flexible job shop than on the job shop scheduling problem. This is most likely due to the fact that flexible job shop problems were first identified by Peter Brucker only in 1990 who proposed a polynomial-time algorithm to solve the assignment and scheduling problems in flexible job shops with two jobs [31]. The flexible job shop problem, as an extension of the job shop scheduling problem, may further be divided into different fields by referring to the different constraints that may be present in a specific problem. Therefore a brief literature review of job shops with some focus on the flexible job shop is conducted in §3.2.1. Furthermore, optimization methodology has been employed in different sectors of the wine industry in a variety of countries, and some of these projects are discussed in §3.2.2. Tables 3.3 and 3.4 contain references to publications on scheduling problems suggested by Bester [14], considering exact methods and heuristic methods, respectively. In order to keep the literature survey compact and to the point, focussing more on publications which inspired the ideas behind this project, only some of these publications are further discussed in §3.2.1.

Method	Author(s)
<b>Solvable in Polynomial time</b>	Johnson [67] and Williamson <i>et al.</i> [147]
<b>Mathematical Formulations</b>	
<i>Linear Programming</i>	Balas [10], Błażewics <i>et al.</i> [17], Dyer & Wolsey [40], Köppe & Weismantel [72], Serafini [109] and Wagner [136]
<i>Lagrangian Relaxation</i>	Fisher [45]
<i>Cutting Plane</i>	Applegate & Cook [4]
<b>Branch-and-bound</b>	
	Applegate & Cook [4], Ashour & Hiremath [6], Brooks & White [24], Brucker <i>et al.</i> [23], Carlier [26], Carlier & Pinson [27, 28], Charlton & Death [29], Florian <i>et al.</i> [46], Grawbowski <i>et al.</i> [53], Greenberg [56], Lageweg <i>et al.</i> and Martin & Shmoys [82]
<b>Dynamic Programming</b>	
	Kubiak & van de Velde [75]

Table 3.3: A summary of the optimization algorithms for solving the scheduling problems found in literature over the past 50 years [14].

### 3.2.1 The job shop scheduling problem

Lee and Aslanni [79] compared two approaches towards solving a single machine sequence-dependent setup time and dual criteria job shop scheduling problem. The first approach is a mixed integer linear programming model and the second approach is based on genetic programming. Their conclusion was that if the major concern of the operations scheduler is the quality of the output and if the number of jobs to be sequenced is relatively small, then mathematical programming is an appropriate methodology. However, if achieving a ‘good’ solution (as opposed to an optimal solution) is acceptable, then a genetic algorithm may be used, especially if the number of jobs to be sequenced is relatively large (since the number of jobs do not influence the performance of the genetic algorithm). On the other hand, the mathematical programming model is greatly affected and becomes very complex (even unmanageable) when the number of jobs is more than ten [79]. Genetic algorithms have been a very popular tool for generating job shop schedules. Jensen [66] showed through experiments that by using a genetic algorithm it is possible to find robust and flexible schedules with a low makespan (when the goal is to minimize the makespan), with the genetic algorithm outperforming other methods compared.

In 1989, Laarhoven, Aarts and Lenstra [130] proposed an approximation algorithm for finding the minimum makespan in a job shop, based on simulated annealing. The suggested algorithm contains a probabilistic element (the acceptance of cost-increasing transitions with a non-zero probability), rendering the approach significantly better than the classical iterative improvement search approach on which it is based, especially when applied to large problems. Along with similar implementations proposed by Nowicki and Smutnicki [90], Balas and Vazacopoulos [10], and Pezzela and Merelli [92], the simulated annealing approach of Laarhoven, Aarts and Lenstra

Method	Author(s)
<b>Solvable in Polynomial time</b>	Jansen <i>et al.</i> [64, 65] and Sevastianov [110, 111]
<b>Constructive Methods</b>	
<i>Priority Dispatch Rules</i>	Giffler & Thompson [50], Lawrence [78] and Pinedo [93]
<i>Simulation</i>	Baker [8], Chen & Chen [30] and Gere [48]
<i>Bottleneck Heuristics</i>	Adams <i>et al.</i> [2], Applegate & Cook [4], Balas <i>et al.</i> [9], Balas & Vazacopoulos [10], Dausère-Pères [35], Demirkol <i>et al.</i> [37], Mason [83], Pezzella & Mereli [92], Ramudhin & Marier [97] and Singer [113]
<i>Lagrangian Relaxation</i>	Chen & Luh [32]
<b>Iterative Methods</b>	
<i>Neural Networks</i>	Jain & Meeran [61] and Yang & Wang [157]
<i>Local Search</i>	Crawford <i>et al.</i> [34], Pirlot [95], Storer <i>et al.</i> [122], Vaessens [132] and Watson [138]
<i>Genetic Algorithms</i>	Aarts <i>et al.</i> [1], Brizuela & Sanomiya [20], Della Croce <i>et al.</i> [36], Kobayashi <i>et al.</i> [71], Storer <i>et al.</i> [122], Varela <i>et al.</i> [133], Wang & Zheng [137] and Yamanda & Nakano [155]
<i>GRASP</i>	Binato [15]
<i>Tabu Search</i>	Al-Turki <i>et al.</i> [3], Jain <i>et al.</i> [62, 63], Nowicki & Smutnicki [90], Pezella & Mereli [92], Storer <i>et al.</i> [122], Sun <i>et al.</i> [123] and Tailliard [124]
<i>Reactive Tabu Search</i>	Batiti [11] and Batiti & Tecchiolli [12]
<i>Simulated Annealing</i>	Aarts <i>et al.</i> [1], Mittenenthal <i>et al.</i> [87], Storer <i>et al.</i> [122], Steinhöfel <i>et al.</i> [120], van Laarhoven & Aarts [129], Wangh & Zengh [137] and Yamanda & Nakano [156]
<i>Beam Search</i>	Sabuncuoglu & Bayiz [106]
<i>Rolling Horizon</i>	Shafaei & Brunn [112] and Singer [113]

Table 3.4: A summary of the heuristic algorithms for solving the scheduling problems found in literature over the past 50 years [14].

[130] are considered as some of the best reported approximation algorithms for the job-shop problem with the objective of minimizing the makespan [54]. Of these, Pessela and Merelli reported that their algorithm provides better results within a reasonable amount of computer time than the results from the proposed methods of the other authors. They proposed a local search method based on a tabu search technique, using the shifting bottleneck procedure to generate the initial solution and to refine current solutions. Then, in 2005, Grabowski and Wodecki [54] developed a new, local search procedure based on the tabu search approach incorporating lower bounds to evaluate moves and perturbations to guide the search to the more promising areas of solution space. Their results were compared to those found via the

tabu search method developed by Pezzela and Merelli [92] and were found to deliver better solutions within shorter computer processing times [54].

Some heuristic (approximation algorithms) and exact solution methods have been derived for the flexible job shop scheduling problem in recent years. A branch-and-bound method for the flexible job shop scheduling problem was developed in 1992 by Jurisch [69] in his PhD dissertation. A hierarchical algorithm for the flexible job shop scheduling problem based on a tabu search approach was published by Brandimarte [19] in 1993. In 1994, Hurink, Jurisch and Thole [59] proposed a tabu search method to solve a slightly generalized flexible job shop problem with machine dependent processing times of operations, achieving excellent results for benchmark problems. In 2002, a linguistic-based meta-heuristic modelling and solution method for flexible job shop scheduling problems was proposed by Baykasoğlu [13]. The results obtained showed that the proposed algorithm is able to solve this complex problem effectively, albeit approximately, within a reasonable time frame. Machine independent capability units, referred to as Resource Elements (RE), are used to represent product processing requirements and machine processing capabilities. Based on this concept, along with a known rule-based heuristic, a simulated annealing algorithm was developed to solve the flexible job shop scheduling problem [13]. This approach simplifies the modelling process and enables usage of existing job shop scheduling algorithms for its solution. In 2007 the flexible job shop scheduling problem with sequence-dependent setup times was solved by Saidi-Mehrabad and Fattahi [107], using tabu search. They also developed a mixed integer programming model including a large number of the characteristics of the cellular scheduling problem. This mathematical programming model is considered further in §3.3 where the exact branch-and-bound solution method is discussed as a means of solving the model exactly.

### 3.2.2 Decision support in the wine industry

Decision support systems in the wine industry date back as far as 1988 when Gertioso [49] designed a decision support system for French viticultural cooperatives. She proposed a vineyard restructure and, to define the restructuring strategy, the vineyard was modelled to determine the influence of the strategy according to several economic views on the future. In 2000, simulation of the actions of certain winery workers was considered by Hansen [57]. The finished products (wine) are stored in very large stainless steel tanks. These tanks are connected by a complex set of pipes, valves and drops, and the cognitive model of Hansen focussed on storage and on possible routings between tanks. A decision support system dealing with the minimization of utility waste from the vinification processes was considered in 2005 by Musee, Lorenzen and Aldrich [88].

The interesting operations research and logistics problems occurring at wineries have also led to the founding of the Wine Supply Chain Council (WSCC) which had its fourth successful meeting in Australia in January 2009. This international wine industry research network was established in July 2006 and attempts to collaborate on issues in global wine supply chains. Members of the WSCC have been very successful in applying decision support technologies to assist their local wineries. Amongst these members, Simon Dunstall of the CSIRO has been working with Australian Orlando Wyndham Group since 2003 in order to achieve the shared supply network goal of maximizing the value that is realized from material and intellectual assets in the supply network [41]. The massive grape supply network consists of 520 growers, 3 123 blocks, 35 grape varieties, 33 areas, 104 wines, 186 harvester operators and 91 transport entities (also used by other wine companies). He approached the problem via a rule-based system comprising 250 rules for intake planning. Furthermore, Chilean member Sergio Maturana has

been involved in a project where some crucial factors of the wine making process, such as the receiving and pressing capacities of a cellar, were studied by means of simulating the receiving of grapes at the cellar [7]. Maturana, together with WSCC member Alejandro MacCawley, has also been involved in designing a practical tool for scheduling wine grape harvesting operations in Chile which adopts an optimization approach, taking into account both operational costs and grape quality [44]. A mixed integer linear programming model was formulated and solved to support harvest scheduling, labour allocation and routing decisions. A quality loss function was developed by conducting a survey amongst enologists in 2003 in order to evaluate the possible degradation of grapes harvested before or later than the optimal harvesting date. The tool developed by Maturana and MacCawley was used to solve the problem daily, implementing a rolling time horizon, and it was found that the schedule proposed by their model provides a good basis for decision makers to derive a final schedule.

### 3.3 Solving scheduling problems via mathematical programming

The aim in this section is to provide the reader with the necessary knowledge of the methods used in this thesis to solve one of the scheduling problems experienced at Wamakersvallei Winery — the wine cellar upon which the case study in later chapters is based, in an exact manner. Exact scheduling methods may be used to obtain optimal schedules (when they exist) and the subdiscipline of mathematical programming, started with the development of the simplex machine by George Dantzig in 1947, is a very popular tool for solving optimization problems [84]. The general mathematical programming problem may be portrayed by an algebraic function which is to be maximized or minimized, called the *objective function*, and a set of one or more algebraic inequalities that are to be satisfied, called the *constraints* of the problem. This section serves as a very brief introduction to mathematical programming with the focus mainly on mixed integer programming which is applied to solve one of the scheduling problems occurring at Wamakersvallei later in this thesis.

#### 3.3.1 Mixed integer programming in the context of scheduling

A linear programming model refers to the mathematical programming model where both the objective function and the set of constraints involve linear functions only. The constraints of a given problem define the set of all feasible combinations of decision variables and the set of such feasible combinations is referred to as the *feasible solution space* [121].

In a linear programming problem the objective is to

$$\begin{aligned} & \text{minimize } f(\mathbf{x}) \\ & \text{subject to } \mathbf{Ax} \geq \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned}$$

where  $\mathbf{A}$  an  $M \times N$  matrix,  $\mathbf{x}$  is an  $N$ -vector representing the decision variables,  $\mathbf{b}$  is an  $M$ -vector of constant right hand sides and  $f(\mathbf{x})$  is the objective function to be minimized [58]. In the case where some of the decision variables are required to be integer, the problem is referred to as a *mixed integer programming* problem. In order to illustrate the formulation of a mixed integer linear program when solving scheduling problems, an instance of a general flexible job shop with sequence-dependent setup times and a no-wait characteristic is considered below [107].



This generalized flexible job shop problem consists of  $n$  jobs with every job  $J_j$  having  $k_j$  tasks to be performed on  $m$  processors with the objective of minimizing the makespan of the resulting schedule. Sequence-dependent setup times refer to the fact that the setup time required for a machine depends on which two jobs are processed consecutively on the machine. A variable  $s_{ij\ell}$  denotes the setup time required in order to have machine  $P_i$  ready to process any task in job  $J_\ell$  directly after processing a task of job  $J_j$ . For the purpose of illustrating this solution method, the processing time of task  $T_{jk}$  on processor  $P_i$  is denoted by  $p_{ijk}$ . Since a task may be performed on any one of a set of machines, the variable

$$\mu_{ijk} = \begin{cases} 1 & \text{if task } T_{jk} \text{ may be performed on machine } P_i \\ 0 & \text{otherwise} \end{cases}$$

is introduced. No release dates are used, but rather the starting time,  $t_{jk}$ , indicating the time that has elapsed up until the point where processing of task  $T_{jk}$  starts. The finishing time of task  $T_{jk}$ ,  $f_{jk}$ , then indicates the overall time at which the task has been completed on the assigned machine. The variable

$$a_{ijk} = \begin{cases} 1 & \text{if task } T_{jk} \text{ is assigned to machine } P_i \\ 0 & \text{otherwise} \end{cases}$$

is used to indicate an assignment of a machine to a task. The starting time of the processing of task  $T_{jk}$  is denoted by  $t_{jk}$  and the finishing time by  $f_{jk}$ . The makespan of the schedule is referred to as  $C_{\max}$  and is the objective function to be minimized.

The first inequality,

$$t_{jk} + a_{ijk}p_{ijk} \leq f_{jk}, \quad i = 1, \dots, m, j = 1, \dots, n, k = 1, \dots, k_j,$$

ensures that the processing time of task  $T_{jk}$  is limited to the allowed  $p_{ijk}$  units of time available on the machine it is assigned to. The constraint set

$$f_{jk} \leq t_{j(k+1)}, \quad j = 1, \dots, n, k = 1, \dots, k_j - 1,$$

is included to enforce the no-wait characteristic of this problem. The makespan is included by means of the inequalities,

$$f_{jk_j} \leq C_{\max}, \quad j = 1, \dots, n,$$

thereby setting  $C_{\max}$  as the largest final task finishing time. In order to ensure that a task  $T_{jk}$  is only assigned to a processor  $P_i$  if  $\mu_{ijk} = 1$ , the constraint set

$$a_{ijk} \leq \mu_{ijk}, \quad i = 1, \dots, m, j = 1, \dots, n, k = 1, \dots, k_j,$$

is included. The decision variable  $x_{ijk\ell h}$  is equal to 1 if task  $T_{\ell h}$  follows task  $T_{ij}$  directly on processor  $P_i$ , or 0 otherwise, and may be defined by including the constraint sets

$$\begin{aligned} \sum_{j=1}^n \sum_{k=1}^{k_j} x_{ijk\ell h} &= a_{i\ell h}, \quad i = 1, \dots, m, \ell = 1, \dots, n, h = 1, \dots, k_\ell, \\ \sum_{\ell=1}^n \sum_{h=1}^{k_\ell} x_{ijk\ell h} &= a_{ijk}, \quad i = 1, \dots, m, j = 1, \dots, n, k = 1, \dots, k_j, \\ x_{ijkjk} &= 0, \quad i = 1, \dots, m, j = 1, \dots, n, k = 1, \dots, k_j, \end{aligned}$$

in the formulation. The inequalities

$$\begin{aligned} t_{jk} + p_{ijk} + s_{ij\ell} &\leq t_{\ell h} + (1 - x_{ijk\ell h}) M, & h = 1, \dots, k_\ell, i = 1, \dots, m \\ & & j = 0, \dots, n, k = 1, \dots, k_j, \ell = 1, \dots, n, \text{ and} \\ f_{jk} + s_{ij\ell} &\leq t_{j(k+1)} + (1 - x_{ilhj(k+1)}) M, & i = 1, \dots, m, j = 1, \dots, n \\ & & \ell = 1, \dots, n, k = 1, \dots, k_j - 1, h = 1, \dots, k_\ell, \end{aligned}$$

are prescribed in order to include the sequence-dependent setup times in the schedule. The value of  $M$  is taken as 10 000 and may be replaced by any number large enough to preserve the goal of this inequality. In order to achieve a feasible schedule, each task  $T_{jk}$  may only be assigned to a machine once. This rule is enforced by the inequalities

$$\sum_{i=1}^m a_{ijk} = 1, \quad j = 0, \dots, n, k = 1, \dots, k_j.$$

Finally, the trivial constraints are

$$\begin{aligned} t_{jk} &\geq 0, & j = 0, \dots, n, k = 1, \dots, k_j, \\ f_{jk} &\geq 0, & j = 0, \dots, n, k = 1, \dots, k_j, \\ a_{ijk} &\in \{0, 1\}, & i = 1, \dots, m, j = 0, \dots, n, k = 1, \dots, k_j, \\ x_{ijk\ell h} &\in \{0, 1\}, & i = 1, \dots, m, j = 0, \dots, n, \ell = 1, \dots, n, \\ & & k = 1, \dots, k_j, h = 1, \dots, k_\ell. \end{aligned}$$

Some of the variables in this formulation are defined as integer, such as  $a_{ijk}$ , and other may be integer or real, such as  $t_{jk}$ , therefore the problem formulated above is an instance of a mixed integer programming model.

### 3.3.2 The branch-and-bound method

The best known approaches in dealing with integer programming problems are cutting plane techniques, implicit enumeration methods and branch-and-bound techniques [93]. IPs are most often solved by using the branch-and-bound technique [152]. In this thesis, the focus will be more on solving scheduling problems with the use of meta-heuristics, and more specifically a tabu search approach. In order to solve the IP model of the active cellular scheduling problem experienced at Wamakersvallei, the linear programming software, **Lingo 11.0**, is employed<sup>12</sup>. **Lingo 11.0** solves IPs via the branch-and-bound technique. If, in the solution to the LP relaxation of a pure IP (as opposed to a mixed IP), all variables are integers, then this solution is also an optimal solution to the IP. However, when applying the branch-and-bound technique to a mixed IP, the branching only takes place on variables that are required to be integer and the subproblem solution only requires integer values assigned to the variables that are required to be integer. Branching refers to a partitioning of the solution space and each resulting subspace of the solution is then considered separately [93]. Bounding, on the other hand, refers to the development of lower bounds for parts of the solution space (in the case of a minimization problem. If such a lower bound on the objective function value in one part of the solution space is larger than a solution already obtained in a different part of the solution space, the corresponding part of the former solution space may be disregarded [93]. The branch-and-bound

<sup>12</sup>‘LINGO is a comprehensive tool designed to make building and solving linear, non-linear and integer optimization models faster, easier and more efficient’ [81].

technique is now considered specifically when it is used to solve mixed integer programming problems.

The branch-and-bound method starts by solving the LP relaxation of the IP problem under consideration. If all the decision variables in the optimal solution of the LP relaxation are assigned integer values, an optimal solution to the IP has been found. Otherwise, any variable that is required to have an integer value, but currently has a non-integer value in the LP relaxation solution is selected. Say, for example, that variable  $x_i$  is selected with a current (non-integer) value of  $r$ . Since  $x_i$  is not allowed to have the value of  $r$ , either  $x \leq \lfloor r \rfloor$  or  $x \geq \lceil r \rceil$  should hold. Two subproblems are formed by adding the two constraints on the value of  $x_i$  to the LP relaxation, one constraint to each subproblem. The subproblems are solved and the process is repeated if there are variables requiring integer values that are assigned non-integer values in the subproblem solutions. The working of the branch-and-bound method is illustrated by means of an example below.

**Example 3.5** Consider the mixed integer programming model in which the objective is to

$$\text{Minimize } z = 90x_1 + 60x_2 + 50x_3 + 40x_4$$

subject to the constraints

$$\begin{aligned} 7x_1 + 5x_2 + 4x_3 + 3x_4 &\geq 42 \\ x_1 + x_2 + x_3 + x_4 &\leq 8 \\ x_1, x_2, x_3, x_4 &\leq 3 \\ x_1, x_2, x_3, x_4 &\geq 0 \\ x_1, x_2, x_3 &\in \mathbb{N}_0 \end{aligned}$$

This is a mixed integer programming model due to fact that some of the variables ( $x_1, x_2$  and  $x_3$ ) are specified to assume integer values and some do not have to satisfy this restriction ( $x_4$ ). The branch-and-bound tree for this example is shown in Figure 3.6. In the tree, each node refers to a subproblem with the constraint on its incident arc included in the LP problem of its parent node. In the root node, node 0, the LP relaxation of the mixed IP problem is solved with an optimal solution found as  $\mathbf{x} = (2\frac{1}{3}, 3, 2\frac{2}{3}, 0)$  with an associated objective function value of  $z = 523\frac{1}{3}$ . Although this is a feasible solution to the LP relaxation problem, it is not a feasible solution to the mixed IP problem under consideration since  $x_1$  and  $x_3$  do not assume integer values. It is therefore necessary to branch on one of the non-integer variables. Branching on  $x_1$  takes place and because the value of  $x_1$  is currently  $2\frac{1}{3}$ , the domain of  $x_1$  is split into  $0 \leq x_1 \leq 2$  and  $x = 3$  (since  $x_1$  is not allowed to exceed 3). The constraint  $x_1 \leq 2$  is considered in node 1 of Figure 3.5. When the new constraint is included in the LP relaxation problem, it results in an infeasible solution to the LP relaxation (and hence also to the IP).

The second node of the branch-and-bound sees the addition of the constraint  $x_1 = 3$  added to the original LP relaxation. The solution  $\mathbf{x} = (3, 2\frac{3}{5}, 2, 0)$  is found with an associated objective function value of  $z = 526$ . This solution is still not a feasible IP solution, since the variable  $x_2$  does not assume an integer value. Branching therefore takes place on variable  $x_2$  resulting in nodes 3 and 4. At node 3, branching is again required, this time on variable  $x_3$  resulting in nodes 4 and 5. The solution to the subproblem at node 4 is  $\mathbf{x} = (3, 2, 2, 1)$  with an associated objective function value of  $z = 530$ . This solution is a feasible IP solution and since this is the first IP solution found, the best lower bound uncovered thus far during the process is set to 530. Since this is the best solution found, it becomes the incumbent solution. Returning to node 5, the subproblem is solved and delivers a feasible LP relaxation solution. However, the

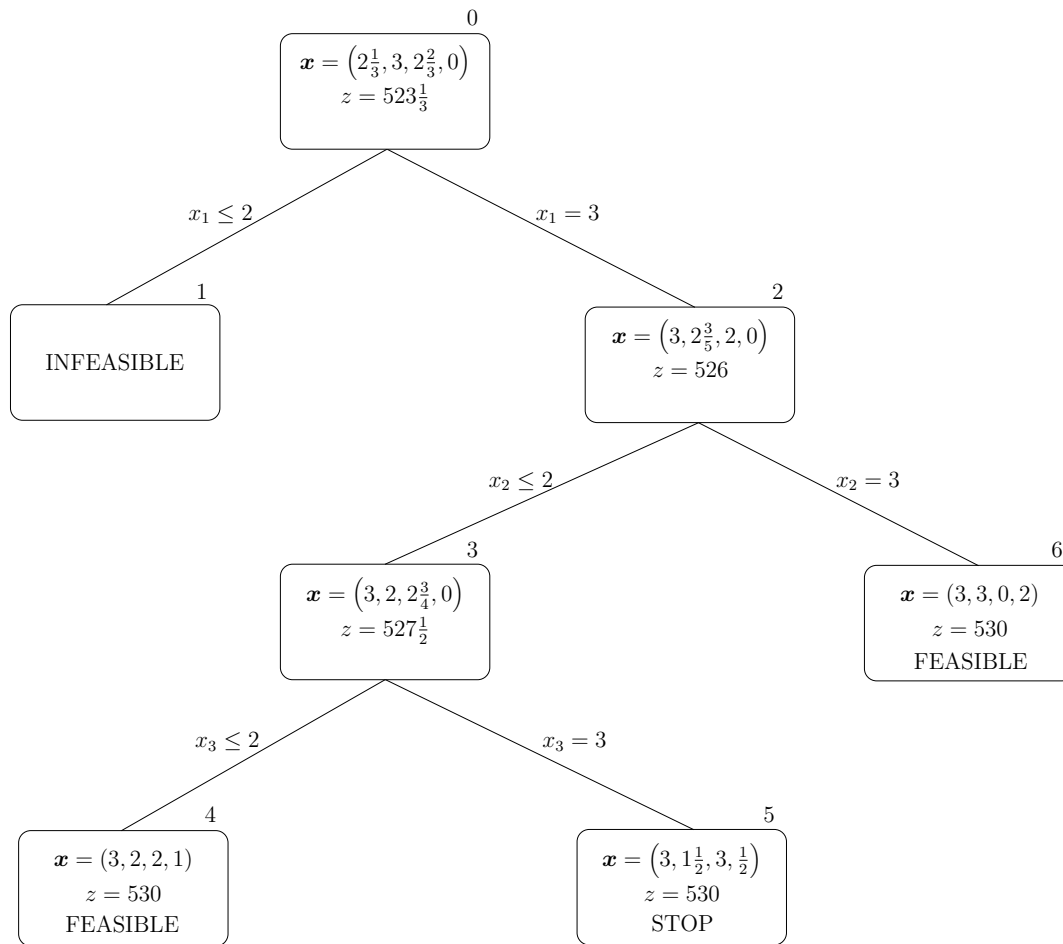


Figure 3.6: The branching tree for the branch-and-bound procedure in Example 3.5.

resulting objective function value is  $z = 530$  and since the lower bound is currently set as 530, this subproblem is not explored further. Returning to node 6, another optimal solution to the IP problem is found. However, this solution is no better than the incumbent solution. Therefore the incumbent solution  $\mathbf{x} = (3, 2, 2, 1)$  is an optimal solution to the mixed IP problem considered. ■

### 3.4 Tabu search methodology

The modern form of tabu search derives from its originator, Fred Glover. As mentioned in §3.2, the method of tabu search has achieved great success as a tool to solve scheduling problems; this is also the case for a large variety of combinatorial optimization problems such as, amongst others, character recognition problems, neural network pattern recognition problems and the well known travelling salesman problem [51].

Tabu search is classified as a meta-heuristic which is derived from the Greek prefix *meta*, meaning ‘beyond’ (here in the sense of higher level) and the word *heuristikein* which means ‘to find’ or ‘to discover’ — in this case to find the optimal solution to an optimization problem under investigation. Even so, a meta-heuristic should be described as a ‘seeking method’ rather than a ‘finding method’, mainly because it does not guarantee finding a global optimum (even if it exists). In the Collins Concise Dictionary, *tabu* or *taboo* is defined as “forbidden”, “disapproved of” or “marked off as sacred and forbidden” [33].

The goal of this section is to provide the reader with a basic understanding of the methodology of a tabu search before it is applied to the scheduling problems at Wamakersvallei Winery in later chapters. Therefore, the focus is on tabu search as a means of solving scheduling problems.

To apply a tabu search to a specific problem, one starts with an initial (possibly infeasible) solution. With the use of certain constraints and principles (also called tabu restrictions), all of which are specified in the problem investigated, as well as the underlying solution representation, one then moves to a new adjacent solution in the problem solution space. This methodology lies at the heart of the approach in a tabu search. According to Glover [51], the primary goal of these tabu restrictions is ‘to permit the method to go beyond points of local optimality while still making high quality moves at each step’.

Tabu Search methods operate by identifying *neighbourhoods* in which adjacent solutions can be reached from the current solution. Let  $\mathcal{S}$  be the finite set of all feasible solutions [52]. Then each solution  $s \in \mathcal{S}$  has an associated set of neighbours,  $\mathbf{N}(s) \subseteq \mathcal{S}$ , with  $\mathbf{N}(s)$  being referred to as the *neighbourhood* of  $s$ . Each solution  $s' \in \mathbf{N}(s)$  may be reached directly from  $s$  by performing an operation referred to as a *move*. Depending on the problem application or solution representation, such a move may be the addition or removal of an object to or from a solution, or the interchange of two objects in a solution, for example. If a solution  $s$  is better than any other solution in its neighbourhood  $\mathbf{N}(s)$ , then  $s$  is a *local optimum* with respect to this neighbourhood. A move value is associated with each move, which normally represents a change on the objective function value as a result of the move [52].

There are numerous ways of generating moves and also a variety of candidate list strategies. The *swap* is one of the most frequently used moves in combinatorial optimization problems and consists of swapping two values and then deeming any such swap as tabu. It is common to combine solution neighbourhoods which is referred to as *compound moves*. A special type of approach for generating compound moves involves applying a range of swaps. The outcome of assigning a value to a different position (thereby an element is assuming a new state, *i.e.* being assigned a new value) is that the element is ejecting another element from its current position. A chain of events is thus created, referred to as an *ejection chain strategy*. Ejection chain strategies are particularly useful in scheduling, routing, clustering, and partitioning problems [52].

A memory structure is implemented to keep track of previous solutions or moves by the means of a so-called *tabu list*. The goal of this tabu list is to avoid oscillations around local minima. Tabu restrictions are not inviolable under all circumstances. When a tabu move would result in a solution better than any solution considered thus far during the search, its tabu classification may be changed. The condition that allows this change in tabu status is referred to as an *aspiration criterion* [52]. After a move has been made (*i.e.* transferring from one solution to another), the inverse of the move is placed in a tabu list and considered tabu, or forbidden, for a certain number of future iterations of the search, unless the aspiration criterion is satisfied. This is an attempt to avoid becoming trapped at a locally optimal solution. The memory structures employed by a tabu search may be classified by four principal dimensions: recency, frequency, quality and influence. The most important of these dimensions are recency and frequency. Recency refers to how recently a specific move has been made, *i.e.* the position it has in the current tabu list for the specific problem under consideration. The frequency of a move, on the other hand, refers to how frequently it is made, *i.e.* the number of times it appears in the current tabu list for the specific problem. The third dimension, quality, refers to the ability to rate the attractiveness of alternative choices during the search and the fourth dimension, influence, considers the impact of the choices made during the search. Therefore, in a sense, quality may be regarded as a special form of influence. A general approach during a tabu search

is outlined in Algorithm 3.1 and the process of applying a tabu search to a small scheduling problem is illustrated in the following example [93].

---



---

**Algorithm 3.1:** The general framework of a tabu search

---

```

1 Choose an initial solution  $x \in \mathcal{S}$ ;
2  $best \leftarrow c(s)$ ;
3 Tabu list  $\leftarrow \phi$ ;
4 while not stop criteria do
5   Cand( $s$ )  $\leftarrow$ 
   { $s' \in \mathbf{N}(s) \mid$  the move from  $s$  to  $s'$  is not tabu OR  $s'$  satisfies the aspiration criterion};
6   Generate a solution  $\bar{s} \in \text{Cand}(s)$ ;
7   Update the tabu list;
8    $s \leftarrow \bar{s}$ ;
9   if  $c(s) < best$  then
10     $s^* \leftarrow s$ ;
11     $best \leftarrow c(s)$ ;
12  end
13 end

```

---

**Example 3.6** Consider the instance of a single-machine, total weighted tardiness problem,  $1 \parallel \sum w_j T_j$ , where the processing times, due dates and weight for each of the four jobs are indicated in Table 3.5.

jobs	$J_1$	$J_2$	$J_3$	$J_4$
$p_j$	10	10	13	4
$d_j$	4	2	1	12
$w_j$	14	12	1	12

Table 3.5: The characteristics of the single machine scheduling problem in Example 3.6. For each job  $J_j$ , the processing time  $p_j$ , the due date  $d_j$ , and the weight  $w_j$ , are listed (all times are expressed in days).

The neighbourhood of a schedule  $s \in \mathcal{S}$  is defined as a schedule that can be obtained through pairwise interchanges of jobs in the schedule  $s$ . The tabu list for this application of tabu search is of size two; a pair of jobs  $(j, \ell)$  is placed in the tabu list after being swapped and may therefore not be swapped again within the next two iterations. The first schedule is chosen as  $s_1 = \{2, 1, 4, 3\}$ . The starting time of job  $J_2$  is therefore taken as day 0, after which job  $J_1$  may be started on day 10, i.e.  $0 + p_2$ , job  $J_4$  may be started on day 20 and job  $J_3$  may be started on day 24. The total weighted tardiness of schedule  $s_1$  may be calculated as

$$\sum_{j=1}^4 w_j T_j = 12(10 - 2) + 14(20 - 4) + 12(24 - 12) + 1(37 - 1) = 500.$$

The aspiration criterion is therefore 500. Furthermore,  $\mathbf{N}(s_1) = \{ \{1, 2, 4, 3\}, \{2, 4, 1, 3\}, \{2, 1, 3, 4\} \}$ . The weighted tardiness values for each of these schedules are 480, 436 and 652, respectively. Since the tabu list is still empty, the best schedule is selected as the schedule with the lowest weighted tardiness value (436),  $s_2 = \{2, 4, 1, 3\}$ . Since the objective function value 436 is smaller than the current aspiration criterion on the value, the new aspiration criterion is the

value 436. The tabu list is updated to include as its first entry, the pair (1, 4). The neighbourhood  $\mathbf{N}(s_2)$  is shown in Table 3.6.

$\mathbf{N}(s_2)$			
$s'_j$	{4, 2, 1, 3}	{2, 1, 4, 3}	{2, 4, 3, 1}
$\sum_{j=1}^3 w_j T_j$	460	500	608

Table 3.6: The neighbourhood of schedule  $s_2$  in Example 3.6, along with the weighted tardiness values associated with the moves.

The best non-tabu schedule, is the first. This move results in a schedule with a weighted tardiness value better than that of the current schedule (which is therefore a local minimum). Schedule  $s_3$  is {4, 2, 1, 3} and the tabu list is now {{2, 4}, {1, 4}}. The neighbourhood of schedule  $s_3$  is shown in Table 3.7 along with the corresponding values of the objective function.

$\mathbf{N}(s_3)$			
$s'_j$	{2, 4, 1, 3}	{4, 1, 2, 3}	{4, 2, 3, 1}
$\sum_{j=1}^3 w_j T_j$	436	440	632

Table 3.7: The neighbourhood of schedule  $s_3$  in Example 3.6, along with the weighted tardiness values associated with the moves.

The best schedule {2, 4, 1, 3} (move (2, 4)) with the weighted tardiness value associated with the move as 436, is tabu. Therefore,  $s_4 = \{4, 1, 2, 3\}$  with the tabu list updated to {{1, 2}, {2, 4}}. The schedules  $s'_j \in \mathbf{N}(s_4)$  and the corresponding objective functions are presented in Table 3.8.

$\mathbf{N}(s_4)$			
$s'_j$	{1, 4, 2, 3}	{4, 2, 1, 3}	{4, 1, 3, 2}
$\sum_{j=1}^3 w_j T_j$	408	460	586

Table 3.8: The neighbourhood of schedule  $s_4$  in Example 3.6, along with the weighted tardiness values associated with the moves.

The schedule {1, 4, 2, 3} delivers the best objective function and is not tabu. Therefore  $s_5 = \{1, 4, 2, 3\}$  and the tabu list is updated to {{1, 4}, {1, 2}}. The aspiration criterion is updated to the value 408 (replacing the previous best of value 436). Schedule  $s_5$  is, in fact, a global minimum, i.e. an optimal schedule, but the tabu search continues until the stopping criterion is satisfied. ■

## 3.5 Chapter overview

The goal in this chapter was to provide the reader with the necessary knowledge to understand the formulation and solution methods considered when solving the scheduling problems experienced at Wamakersvallei Winery later in this thesis.

In §3.1 the notation commonly used in classical scheduling problems, and also throughout this thesis, was reviewed. This section also contained an overview of the three-field  $\alpha|\beta|\gamma$  notation

used to classify classical scheduling problems, as well as an overview of typical representations of schedules in §3.1.4.

A concise survey of literature was presented in §3.2, containing both references to the operational research literature on the scheduling problem and other applications of optimization and decision support in the wine industry.

Finally, the exact and meta-heuristic optimization methods applied to solve the scheduling problems in this thesis were described in §3.3 and §3.4, respectively. Both were illustrated by means of a small scheduling example.



---

---

## CHAPTER 4

---

# Formal problem statement

### Contents

4.1	Wamakersvallei Winery . . . . .	55
4.1.1	<i>Cellar location and layout</i> . . . . .	55
4.1.2	<i>Cellar machinery and location</i> . . . . .	57
4.1.3	<i>The staff at Wamakersvallei</i> . . . . .	61
4.1.4	<i>The product</i> . . . . .	62
4.2	From the vineyards to the press . . . . .	65
4.2.1	<i>Harvesting the grapes</i> . . . . .	65
4.2.2	<i>Scheduling the arrival of the grapes</i> . . . . .	67
4.2.3	<i>On arrival at the winery</i> . . . . .	68
4.3	Production flow and layout inside the cellar . . . . .	70
4.3.1	<i>Production flow of white grapes</i> . . . . .	71
4.3.2	<i>Production flow of red grapes</i> . . . . .	72
4.4	EzyWine . . . . .	73
4.5	Chapter overview . . . . .	74

The objective of this chapter is to provide the reader with the necessary information to fully understand the origin of the scheduling problems experienced at Wamakersvallei Winery. A clear understanding of the products, objectives and main production processes is necessary to develop appropriate scheduling models for Wamakersvallei Winery.

### 4.1 Wamakersvallei Winery

Some important aspects of Wamakersvallei Winery, such as its physical location and current cellar layout, the machinery used, products delivered and staff employed are introduced in this section.

#### 4.1.1 Cellar location and layout

Wamakervallei Winery is situated in the town of Wellington which is located 72km from Cape Town. Wellington was established in 1840 and lies in a valley on the banks of the Kromme



Figure 4.1: The view of the Wamakersvallei Winery building as seen from the parking area. In the picture the main entrance that leads toward the tasting area is visible.

River at the foot of the Groenberg with the Hawequa Mountains on its eastern border [141]. The Wellington Wine Route is one of the newest additions to the Cape wine routes even though the area has a viticultural heritage dating back to the French Huguenots of the late 1600's. The Wellington area is also the primary producer of rootstock for a number of South African vineyards [25]. In the map of the Cape winelands (Figure 4.2), Wellington is situated at the top middle of the map.

The history of Wamakersvallei Wine cellar dates back to 1941, when this central cellar was founded by a group of grape-growers in the area for the pressing of their grapes [140]. Wamakersvallei (directly translated as *valley of the wagon builders*) owes its name to the pioneers who stopped in Wellington to have their wagons serviced before venturing into northern parts of South Africa [148]. With more than half a century of experience Wamakersvallei has firmly established itself as a New World wine producer.

The Wamakersvallei building is located on Distillery Road. As seen in the floor plan (Figure 4.3), the facility may be divided into nine main areas: The administration and wine sales area, Stores A to F and the outside areas (including the tipping area and weighbridge room).

The administration area indicated in the floor plan, consists of offices for the administrative staff and some other employees as well as a tasting area. The wine sales is also indicated in the floor plan. The wine cellar itself is divided into seven main functional areas consisting of several types of tanks and other wine making machinery referred to as Stores A, B, C Front, C Back, D, E and F.

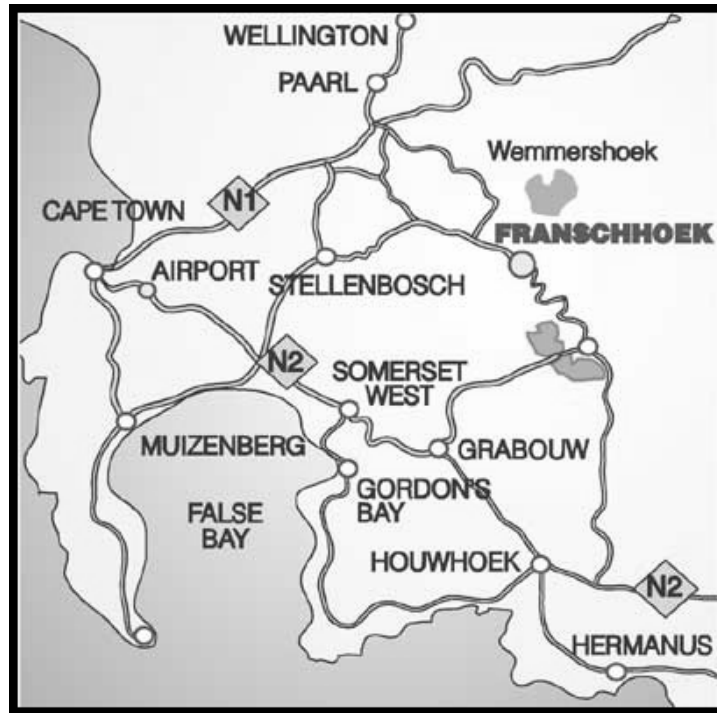


Figure 4.2: A map of the Cape winelands, with Wellington at the top, middle.

#### 4.1.2 Cellar machinery and location

The process of making red and white wine was described in §2.3.2. The majority of wine produced at Wamakersvallei is either white or red drinking wine; therefore the focus of this study is only on these wines. In this section, machinery and the different types of tanks that are used during the wine making process, as well as their different capacities and locations inside the cellar, are discussed.

#### Cellar machinery and the general processes

The sequence of the processes involved in making white and red wine are shown in Figures 2.6 and 2.7 respectively. For both processes the grapes enter the cellar at the tipping bins; tipping bin number three is shown in Figure 4.4(a). Inside the tipping bin a corkscrew shaped feed auger<sup>1</sup> is situated on top of a mechanical crusher/destemmer. The grape clusters are fed into this machine where they are systematically destemmed and transferred to the crusher as shown in Figures 4.4(b) and 4.4(c). The main component of the mechanical destemming machine is shown in Figure 4.4(d). The small circular slots in the large cylinders rotate to remove the larger chunks of stems with another multi-paddle like instrument fitted inside the cylinder to move the grapes around [146].

After crushing and destemming, various operations are performed on the grapes. These operations differ in action and sequence, depending on the desired wine and grape colour. Some of these tanks and associated equipment may now briefly be discussed — a more in-depth description of the sequence and difference in actions is deferred to §4.3.

<sup>1</sup>An auger is a device designed for moving material (or liquid) by means of a rotating helical flighting. The material is moved along the axis of rotation [146].



Figure 4.3: A floor plan of the cellar layout at Wamakersvallei, showing its seven functional areas [135].

A press is a machine which may be used to separate juice (or wine) and grape skins and can also press the juice from the skins. One half of the press consists of a sieve and the other half is impenetrable. While the crushed grapes are transported into the press, the juice is released through the sieve. The skins are pressed by an impenetrable balloon inside the press and the moisture is forced through the sieve. Once the grape skins have been successfully pressed, their remains are completely dry, usually reduced to a powdery substance. Typically, the free run-juice (from here onwards referred to as *A juice*) is of a higher quality than the juice that has been pressed (hereafter referred to as *B juice*). However, B juice is used to increase production per ton; B juice can represent between 15 and 30 percent of the total juice volume from the grapes [135, 146]. B juice is considered as the resulting juice from a pressure varying between 0.4 and 1.2 Bars.

Situated beneath the presses are *buffer tanks* which collect the juice as it is released from the press. In order to have the choice of either combining A and B juice or keeping it separate, there are more buffer tanks than presses so that A juice can go to one buffer tank and B juice to another. The pipes connecting the presses and tanks are not permanent; they may be moved around as needed.



(a) Tipping bin three as viewed from the front of the building.



(b) Grapes being dropped into the tipping bin.



(c) Grapes being loaded into the tipping bin and moved into the crusher/destemmer using the worm.



(d) The central component of the mechanical destemming machine and the multi-paddle instrument on its right.

Figure 4.4: Equipment used to receive the grapes at the cellar as well as to crush and destem the grape bunches.

Similar to these presses, the Wamakersvallei cellar also has *separators*. These machines may also be used to separate grape skins and juice after which grape skins may still be sent to the press if B juice is required, which is usually the case. A separator is therefore similar in function to a press, except that an impenetrable balloon is not used in a separator.

A *settling tank* is used to settle the murky white juice after the grapes have been pressed. The murky particles of the wine settles at the bottom of the tank and the clear juice is then drawn from above. The winemaker may also decide to rather connect a flotation device to a tank resulting in the exact opposite process whereby murky wine particles are caused to float to the top of the juice, rather than settling at the bottom. The clear juice may then be drawn from the bottom of the tank. These tanks may also be used for storage and fermentation of white or rosé wines.

With the red grapes, primary fermentation of the juice occurs together with the grape skins and this mixture is therefore fermented in a specific red wine fermentation tank. When the grape juice and skins are contained in a tank, the skins float to the top and create what is referred

to as a *skin cap*. Constant contact between the skins and juice is required in order to extract flavour and colour from the skins. In order to ensure this contact, the fermenting juice needs to be taken from the bottom of the tank and poured over the skin cap using a mechanical pump which forms part of the fermentation tank. Once the primary fermentation process has been completed, there are two taps at different heights of the tank in order to ensure that the best quality of juice is released from the tanks. The skins are then released through another opening at the bottom of the tank with the use of a comb on the inside of the tank and is then released into worms which transport the skins to a press. These tanks are again used for the second fermentation and may also be used for storage. Such a tank cannot be filled to the top when used for fermentation since space is required for the chemical process of converting the sugar into alcohol; the tank is filled to more or less three quarters of its capacity. The pressure created by this reaction is high enough to cause dents in the stainless steel tanks if a lid should be closed. Hereafter the theoretical maximum amount of fluid that can be contained in the tank at one time is referred to as its *physical capacity* even though, practically, this amount should never be reached. The actual amount of grape skins and wine that is allowed in a tank is referred to as the *actual capacity*. These red wine fermentation tanks are referred to as RT-tanks and DF-tanks. The abbreviations, RT and DF, simply serve as a means of referring to the tanks by their location and is derived from the two different suppliers of the tanks; both types are stainless steel red wine fermentation tanks. The capacities and locations of the different tanks are discussed in the following section.

### Cellar machinery and location inside the cellar

The crushers/destemmers are located inside each of the three tipping bins in the tipping area as shown in Figure 4.3.

Store A contains ten tanks suitable for storage and fermentation during the wine making process. The wall between Store A and the wine tasting area is constructed of glass; hence tasters have a view of the inside of Store A. It is therefore kept as clean as possible and is used mainly as storage area and only for fermentation purposes if the situation calls for it. Also in Store A are some of the 360 wooden barrels used for the wood maturation of the red wines. An average of thirty barrels are bought per year to replace some of the third fill barrels and each of the barrels has a capacity of 225 litres.

Store B contains 59 tanks. There are sixteen settling tanks which are used to settle the murky white juice after the grapes have been pressed and have a combined capacity of 741 kilolitres. Store B also contains forty-three fermentation tanks (with a combined capacity of 1 428 kilolitres) used for the fermentation and storage of white and rosé wines.

Store C is divided into two areas, Store C Front and Store C Back. The eight presses of the cellar are situated in Store C Front of which presses one, two, three and six are used for white grapes only; the remaining presses are used for the pressing of red grapes. There are also thirteen buffer tanks with capacities varying between 8 and 26 kilolitres. In Store C Front there are also a further ten tanks which are used for cold stabilisation (and may also be used as buffer tanks during harvesting when the cellar is ideally running at full capacity and cold stabilisation only occurs at the end of the wine making process). The remaining sixteen RT fermentation tanks are eight 80-ton RT-tanks and eight 50-ton RT-tanks. When referring to a tank as an eighty ton tank, the tank has an actual capacity of 80 tons of juice and skins that are allowed, since room is left to allow fermentation. The physical capacities of all the tanks are listed in Table A.1. Store C Back contains sixteen tanks used for both storage and fermentation and has a combined physical capacity of 1 541 kilolitres.



At the start of this project Store D contained old concrete storage tanks dating back to the start of Wamakersvallei Winery. However, in 2008 these tanks were replaced with new stainless steel storage tanks. The storage tanks are being used as of the 2009 harvest. There are ten 130 kilolitre storage tanks and also a further ten 129 kilolitre tanks, comprised of three smaller tanks (75, 34 and 20 kilolitre tanks) stacked on top of one another, and four 112 kilolitre tanks, each consisting of a 75 and a 37 kilolitre tank. The stacked tanks may either be connected and considered as one large tank or separated into the smaller tanks. One of these tanks are shown in Figure 4.5, note the three different openings for the (possible) separate use of the three tanks. In order to move these tanks into the cellar, the roof had to be removed and various moving and lifting equipment was required.



(a) One of the triple stacked stainless steel tanks to replace the old concrete tanks in Store D.



(b) In order to move the new tanks into the cellar, the roof had to be removed and various lifting and moving equipment was required.

Figure 4.5: The new stainless steel fermentation tanks replacing the old concrete tanks of Store D in 2008.

There are seven separators in Store E as well as two 11 710 litre capacity buffers. Because of the two very large buffer tanks, the winemakers attempt to limit the grapes received at the separators to a maximum of two cultivars at any one time. Once such a buffer tank reaches its maximum capacity, the transportation of juice to a press may begin. Store E also contains the ten 80-ton DF-tanks.

Store F contains fifty-five fermentation tanks with a combined physical capacity of 3 678 kilolitres; these tanks are used for both fermentation and storage.

A summary of the different stores and their functions, as well as their total physical capacities, are listed in Table 4.1, while a complete list of the physical capacities of all relevant tanks appears in Appendix A.

### 4.1.3 The staff at Wamakersvallei

During the course of this thesis, many changes have been made in accordance to the Wamakersvallei staff. Wamakersvallei Winery has seven directors of which Ernest Brink is the chairman, Jannie Bosman the vice-chairman and Johan Truter [126] the managing director [135]. In 2007, the two winemakers were Christiaan Visser [135] and Hugo Truter [125], who is also the Cellar Master, and the viticulturist was Koos van der Merwe [127]. However, both Christiaan Visser and Koos van der Merwe left Wamakersvallei at the end of 2008. A new winemaker, Pieter-Niel Rossouw, and a viticulturist, Marko Roux [101], have since joined the Wamakersvallei workforce.

Store	Use	Total capacity (liters)
A	mainly storage may be used for fermentation	266 657
B	settling fermentation of white and rosé storage of white and rosé	2 170 208
C Front	buffer tanks for presses cold stabilisation tanks fermentation of red wine	1 176 855
C Back	fermentation storage	1 541 834
D	fermentation storage	3 150 000
E	fermentation (mostly red wine) separators and buffers storage	2 062 208
F	fermentation storage	3 678 286

Table 4.1: The functions and theoretical total capacities (in litres) of the different Stores of the cellar, *i.e.* the total amount of fluid able to fit into all the tanks in a store at any one moment, as well as the different functions of the vessels contained in a store [135].

There are also another seventeen permanent employees consisting of a public relations officer, a quality co-ordinator, an accountant, a storeman who also attends to wine sales, a maintenance officer, an export co-ordinator, a cellar supervisor, two administration officers and eight general cellar workers (of which one is also the tea lady).

#### 4.1.4 The product

Wamakersvallei Winery produces a wide variety of wines under five different labels: La Cave, Bains Way, Wamakersvallei, 33° South and Phambili. The different varieties, vintages and prices (expressed in South African Rand) of the wines that are currently available for purchase at the Wamakersvallei Winery wine sales are listed in Tables 4.2 and 4.3. Other than wines, they also produce and sell bottled grape juice. In §§4.1.4–4.1.4, the different labels and wines are discussed and in Figure 4.6 two of the Wamakersvallei Winery products are shown.

#### La Cave

The La Cave label appears on the premium wines for which only red wines are produced. All the La Cave wines are made from a single vineyard with a yield of less than 10 tons per hectare and have won numerous awards. For these wines, respectively, the grapes are all fermented on the skins at 26°C after which the free-run portion is placed in new 225 litre oak barrels. Full malolactic fermentation is completed in the barrels and the wines are racked after SO<sub>2</sub> adjustments and returned to the same barrels. The Cabernet Sauvignon and Pinotage are aged in barrels, 80% French and 20% American origin, for 12 months. The Merlot is aged in similar barrels but for only 10 months and the Shiraz is aged for 12 months in barrels consisting of 70%





(a) A bottle La Cave Pinotage [115].



(b) Three bottles of Bains Way, with a 2005 Viognier in front [115].

Figure 4.6: Wine bottles displaying two of the Wamakersvallei labels.

Cultivar	La Cave		Cultivar	Bains Way	
	Vintage	Price per bottle (R)		Vintage	Price per bottle (R)
Merlot	2003	80	Chenin Blanc	2007	20
Shiraz	2003	80	Sauvignon Blanc	2007	23
Cabernet Sauvignon	2003	80	Chardonnay	2006	23
Pinotage	2005	80	Viognier	2006	23
			Brut Sparkling Wine	n.a.	35
			Merlot	2005	26
			Pinotage	2004	24

Table 4.2: A price list of the La Cave and Bains Way wines available at Wamakersvallei Winery at the end of 2007, expressed in South African Rands [115].

French oak and 30% American oak<sup>2</sup> [115].

### Bains Way

The Bains Way label is named after the famous master road builder Andrew Bain who created the gateway to the north through the Bainskloof Pass [115]. The optimum ripeness of the white grapes are from about 22° Balling<sup>3</sup> for Chenin Blanc, 21° Balling for Sauvignon Blanc, 23° Balling for Chardonnay and 25° Balling for Viognier. The main difference in the making of the different white wines is the time the juice is left on the skins after it has been crushed. Chenin Blanc juice is left on the skins for three to five hours, Sauvignon Blanc and Viognier for five to

<sup>2</sup>This information is based on descriptions of the different processes for the 2005 vintage Cabernet Sauvignon, Shiraz and Merlot as well as the 2006 Pinotage.

<sup>3</sup>German chemist, Karl Balling, developed the Balling scale and it refers to the concentration of a sucrose solution, as the weight percentage sucrose at 17.5°C. The Brix scale is a more recent scale that takes into account the effect of gravity and a reference temperature of 20°C. Balling is often still used in the South African wine industry, even though Brix is the international standard in the wining industry. The terms Balling and Brix are also often used wrongly.

Cultivar	Wamakersvallei		Cultivar	33° South	
	Vintage	Price per bottle (R)		Vintage	Price per bottle (R)
Fishermans Jerepigo	n.a.	30	Rosé	n.a.	20
Jagters Port	n.a.	34	Semi sweet	n.a.	20

Table 4.3: A price list of the Wamakersvallei dessert wines and 33° South wines available at Wamakersvallei Winery at the end of 2007, expressed in South African Rands [115].

eight hours and Chardonnay for three to five hours, after which the skins are all pressed. Only the free-run juice is used and the fermentation temperature is about 13°C. The Brut Sparkling wine is made from Sauvignon Blanc grapes and the process is the same as with Sauvignon Blanc wine. The wine is only impregnated with Carbon dioxide in the end.

All of the red grapes for the Bains Way wines are harvested at as close as possible to optimal ripeness and are then fermented dry on the skins at 28°C. Thereafter the free-run portion is placed in second and third fill oak barrels, *i.e.* oak barrels that have been used once or twice, for nine to twelve months for Merlot, Pinotage and the Shiraz/Mourvedre blend and exactly twelve months for Shiraz and Cabernet Sauvignon. Unless otherwise stated, all wines are made from one cultivar, the Shiraz/Mourvedre blend consists of 80% Shiraz, 18% Mourvedre and 2% Viognier<sup>4</sup> [115].

### Wamakersvallei

The two wines listed under the Wamakersvallei label, namely the Fishermans Jerepigo and Jagters Port, are both dessert wines and due to the small percentage produced, they do not form part of the focus area of this thesis.

### 33° South

The label 33° South is derived from the location of Wamakersvallei wine cellar which is exactly 33 degrees South of the Equator on the southern tip of Africa. Under this label, Wamakersvallei produces a classic Semi sweet wine, a Rosé and a Shiraz/Mourvedre blend.

For the Semi sweet wine Chenin Blanc grapes are picked at a ripeness of about 21° Balling or higher. After crushing, the juice is left on the skins for five to eight hours and then pressed. Only free-run juice is used and the fermentation temperature is maintained at approximately 12°C.

With the Rosé, the free-run juice is used to produce the cherry pink colour after which this free-run portion is fermented at 13°C. This Rosé is made from Pinotage grapes.

The Shiraz/Mourvedre blend consists of 60% Shiraz, 25% Mourvedre and 15% Cinsaut grapes which are fermented dry on the skins at 28°C; thereafter the free-run portion is placed in second and third-fill oak barrels for 9 to 12 months. Blending takes place after 6 months and the wine is then returned to the barrels for further maturation to ensure a homogeneous final blend.

<sup>4</sup>This information is based on a description of the different processes for the 2007 vintage white Bains Way wines and 2006 Pinotage, Shiraz, Cabernet Sauvignon and Brut Sparkling wine as well as the 2005 Shiraz/Mourvedre blend and Merlot [115].

## Phambili

Both white and red wines are produced for the fourth label. The first is a white wine blend consisting of 45% Chenin Blanc, 45% Chardonnay and 10% Viognier. The grapes are picked at a ripeness of 25° Balling or slightly higher. After which they are crushed and the juice is left on the skins for five to eight hours until pressed. Only the free-run juice is used and the fermentation temperature is maintained at approximately 13°C.

There are two red wines under the Phambili label. The Pinotage grapes are picked once mature and are fermented dry on the skins at 28° after which the free-run portion is placed in second and third-fill oak barrels and matured for nine to twelve months.

The Cabernet Sauvignon Reserve is made from grapes of a single vineyard with a yield of less than 10 tons per hectare. The grapes are once again fermented dry on the skins at 26° after which the free-run portion goes into new 225 litre oak barrels. Full malolactic fermentation is allowed in the barrels and the wine is then racked and after SO<sub>2</sub> adjustments, it is returned to the same barrels for a further 12 months. The barrels are made from 80% French Oak and 20% American Oak.

However, it is not only the quality of the wine produced under the Phambili label, that makes this label special. The Inkquebela Phambili Empowerment Trust, is a separate trading entity established by 18 formerly disadvantaged workers at Wamakersvallei. The profits from the trust activities are distributed amongst the workers and their families including long term investments in permanent housing, education and training and future business development projects. All the trustees of the Inkquebela Phambili Empowerment Trust are employees at Wamakersvallei and include the cellar manager Johan Truter.

In one of the 11 national South African languages, Xhosa (a local language in the Western Cape Province), the word Phambili means ‘moving forward’.

## 4.2 From the vineyards to the press

In this section an overview is provided of processes regarding the receiving of grapes from the suppliers until the point where they enter the cellar at the tipping bins.

### 4.2.1 Harvesting the grapes

Grapes are divided into three classes, with Class 1 the highest quality grapes. In order to produce grapes of the highest quality, the farmer should follow the instructions given by the viticulturist, Marko Roux [101], who visits the farms throughout the year to ensure that viticultural aspects, such as trellis systems, are in place.

Harvesting of the grapes at the various vineyards of the suppliers is performed as close as possible to the time of optimal ripeness of the specific cultivar. The ripeness is determined through tasting and the sugar, pH and acidity levels of a sample of the grapes. Sugar is measured in degrees Balling and 24° Balling indicates 24% sugar in the grapes. The pH of the grapes is an indication of the active acidity. A wine with a too high a pH level, say over 4.0, becomes unstable with respect to micro-organisms and a pH level that is too low inhibits micro-organism growth. Acidity is a measurement of the tartaric acid present in the wine and is given as a percentage per volume [91]. Even though total acidity and pH are related (the

higher the pH, the lower the acidity and vica versa), they represent different ways of measuring acidity in wine [91, 135]. Harvesting roughly stretches over a period of three months, starting any time from middle January and continuing until middle April. Some harvesting data of past years are listed in Appendix A.2.

A load of grapes may only contain grapes from the same vineyard block and, depending on the decision made by the supplier, these grapes may either be harvested by hand or machine. A harvesting machine is shown in Figure 4.7(b). Both methods have different influences on the quality of the harvested grapes. The harvester is a large tractor that straddles the grapevine trellises and loosens the grapes from their stems by striking the fruiting zone of the grapevine with firm plastic or rubber rods. Such a grape load typically contains fewer stems than a grape load that was harvested manually, but it may contain other impurities such as mouldy grapes, leaves, canes, metal debris, rocks and even small animals [146]. Grape loads harvested by machine also contain more juice since the grapes may split open during the process. This is not a favourable situation, since the heat causes the juice to start fermenting prematurely due to the presence of wild yeast. Wild yeast causes the juice to extract different (some undesired) flavours compared to the predictable, desired effect of the added controlled yeast [126]. Therefore sun exposure is especially harmful to grapes harvested by machine which seems to be the largest single disadvantage associated with using harvesting machines. Another disadvantage of using a harvesting machine, is that it is very expensive and with manual labour in South Africa being rather inexpensive it is not always possible for a farmer to afford such a machine. The most significant advantage of using a harvesting machine is the fact that no workers are required. Workers who do not show up for harvesting is currently one of the most serious problems affecting Wamakersvallei.



(a) The empty bunch after a harvester has removed all the grapes.



(b) A harvester as seen from the side, about to load the picked grapes into a transportation vehicle.

Figure 4.7: A typical grape harvester used to harvest grapes by loosening the grapes from their stems by striking the fruiting zone of the grapevine.

During the harvesting period, the viticulturist receives numerous samples from the different vineyard blocks<sup>5</sup>. The sugar, pH and acidity of the samples are measured and the viticulturist keeps a form up to date containing all the calculated samples together with references as to which farm and supplier the grapes are from [101]. An example of this form is shown in Figure 4.8. The form columns consist of a date, member, farm, cultivar, block number, sugar level, pH and acidity level column.

<sup>5</sup>The samples are supposed to be received once every fortnight for each vineyard block. However, this is not always the case with all the suppliers.

datum	lid	plaas	kultivar	blok no	B	pH	TA
26/01	W. Laubscher	Wamakers	Steen	15	18.4	2.89	10.43
"	"	"	Sauv. Bl.	11 50	22.3	2.95	8.25
"	"	"	"	9	19.2	3.04	8.46
"	"	"	"	35	20.8	<del>3.171</del> 6.45	<del>6.45</del> 3.171
"	"	"	Novelle	13 50	23.5	3.30	4.38
"	du Plessis	uitkyk	Sauv. Bl.	24	21.6	3.00	10.63
"	"	"	"	22	20.2	3.08	8.60
"	"	"	Chenin Bl.	19	20.7	<del>3.37</del> 7.42	7.43
"	AJ du Toit	Vleisbank	Chenin Bl.	13	17.2	2.80	11.93
"	"	"	Chardonnay	7	20.4	2.99	10.50
"	"	"	Pinotage	2	21.4	3.03	8.16
"	JC Bosman	Goedehoop	Chenin Bl.	C	18.4	<del>2.977</del> 9.73	9.73
"	JC Bosman	Goedehoop	Chenin Bl.	R	15.00	2.82	18.02
"	"	"	"	J	17.3	2.76	10.40
"	"	Leliefontein	Chenin Bl.	N	16.8	2.83	11.84
"	"	Groenfontein	Chenin Bl.	A	18.8	2.84	9.99
"	"	Goedehoop	Chenin Bl.	B	15.2	2.97	10.16
"	"	Twyfeling	Chenin Bl.	F	15.9	2.69	16.51

Figure 4.8: A filled in form consisting of the calculated sugar, pH and acidity levels of samples received.

#### 4.2.2 Scheduling the arrival of the grapes

Different grape cultivars ripen during different periods of harvesting. During these periods the sugar, pH and acidity of the grapes are constantly monitored by the viticulturist through samples received from the suppliers. According to the agreement between the grape farmers and Wamakersvallei Winery, at least two samples are required before harvesting may start [127]. From these samples, the ripeness of the grapes may be monitored and the expected class of the grape may also be derived from such a list. Table 4.4 lists the grading of the different grape cultivars according to its sugar level (in Balling). A list with the grapes closest to optimum maturity is constructed and at the end of each day, the viticulturist and the two winemakers, as well as the cellar manager, sit around a table and discuss the different options of receiving grapes the next day. An example of such a list from the 2009 Wamakersvallei harvest is shown in Figure 4.9. This list includes the cultivar, class and producer of each selected vineyard block as well as the block reference number and the expected yield of the vineyard block expressed in tonnes. This is done for every working day, since the cellar does not receive grapes over the weekend. When scheduling the grapes, they have to keep in mind which blocks are fully ripened and where there is available space in the cellar. They then manually construct a list of blocks to be received the next day and notify the different suppliers via an SMS service. For example, such an SMS may typically inform a farmer to deliver 10 tons of his class one Cabernet Sauvignon and 50 tons of his class three Cinsaut the next day.

White grape cultivars			Red grape cultivars		
Cultivars	Class	Sugar	Cultivars	Class	Sugar
Chenin Blanc	1	21 – 24	Cabernet Franc	1	24 – 26
	2	20 – 24	Cabernet Sauvignon	2	23 – 27
	3	> 18	Shiraz	3	21 – 28
	6	< 18		6	< 21 and > 28
Sauvignon Blanc	1	19 – 22	Petit Verdot	1	24 – 27
	2	18.5 – 23	Pinotage	2	23 – 27.5
	3	> 18		3	22 – 28
	6	< 18		6	< 22 and > 28
Chardonnay	1	22 – 25	Merlot	1	23.5 – 25.5
Viognier	2	21 – 25		2	23 – 26.5
	3	> 18		3	22 – 28
	6	< 18		6	< 22 and > 28
Colombar	2	18 – 23	Mourvedre	1	23 – 25
Hanepoot	3	> 18	Malbec	2	22 – 26
SA Riesling	6	< 18		3	22 – 28
Weisser Riesling				6	< 21 and > 28
			Roobernet	2	23 – 27
			Ruby Cabernet	3	21 – 28
				6	< 21 and > 28
			Cinsaut	2	22 – 26
				3	21 – 28
				6	< 21 and > 28

Table 4.4: Summary of the grape grading according to the sugar level when measured in degrees Balling [127].

When a problem occurs with the harvesting of the grapes at a farm, such as when workers do not show up or as a result of machinery failure, it influences the scheduling of the grapes at the cellar, because by then space has already been allocated for the expected grapes. Therefore a very quick rescheduling process takes place to ensure maximum use of the available space. In order to provide the reader with a better understanding of the grape intakes, a summary of the grape intakes from 2000 to 2006 is given in Table 4.5. The daily total intake of grapes also appears in Appendix A.2.

### 4.2.3 On arrival at the winery

When a load of freshly harvested grapes arrives from a farmer, the first step is to have a sample taken manually by a worker. The sugar, pH and acidity levels are then determined at the weighing station to ensure that the grapes are ready for pressing.

The block is then given a docket number and the relevant information is stored accordingly for future reference by means of software called EzyWine. The docket number enables the winemakers to determine the origin of the grapes in a specific tank at any time. At the weighing station, the transportation vehicles queue to have their loads weighed, see Figure 4.10. Since the weight of every empty vehicle is known, the nett weight is the actual amount of grapes and

WAMAKERSVALLEI WYNKELDER				
Maandag				
DATUM: 16/02/2009				
KULTIVAR	KLAS	PRODUSENT	BLOK	TON
Steen	2	Houtkloof	A	3
	2	Klein-Driefontein	3	8
	2	Jannie Louw	2	30
	2	Wouter Basson	2	8
	<del>1</del>	Bernie		7
	1	M'Fshola	A1	10
	1	Patatskloof	13,20	12
Chardonnay	1	Ladi Willemse	2	10
	1	Mike Heramb		20
	2	Byl Brink	25,11	60
	2	Jannie Louw	Klein plasie	10
Pinotage	1	Byl Brink	25,28	90
	1	Uitkyk	3	30
	1	Willie Laubser	20 & 19	20
	1	Patatskloof	19	9
	1	Bouwer Nel		5
		*	La Cave	M1
	*	Diemersfontein	M	25
Merlot	2	Welgegund		8
	2	Gawie Rossouw	2	15
	1	Lelienfontein	P	12
	1	De Rust	5-2	40
Shiraz		Far. Horizons		80
SAP		Barry Gillespe		25

Figure 4.9: An example of the list of vineyard blocks selected by the group consisting of wine-makers, the viticulturist and cellar manager to be harvested on the February 16th 2009. The list consists of the cultivar, class, supplier, block number and expected vineyard yield for each of the selected vineyard blocks.

also the weight that is entered into EzyWine. Suppliers are awarded an extra R50 for every ton of grapes weighed before 10:00 in the morning, since this ensures that the grapes have had the minimal amount of exposure to the sun. Suppliers who use machinery to harvest also receive



	Year						
	2000	2001	2002	2003	2004	2005	2006
Number of intake days	42	37	36	38	39	44	40
Highest tons per day	535	584	598	636	714	700	610
Lowest tons per day	2	3	11	7	9	5	9
Average tons per harvesting day	255	280	255	342	309	303	290
Highest tons per week	2 340	2 433	2 120	2 840	2 913	2 668	2 584
Lowest tons per week	392	331	348	622	517	740	1 018

Table 4.5: Summary of the annual grape intakes at Wamakersvallei cellar [127].

an extra 2% added to the physical weight to account for the lack of stems [126].



(a) A sample of the grapes is taken and its sugar, pH and acidity is measured at the weighing station which is then entered into the database for future reference.



(b) Transportation vehicles queuing at the weighing station during harvesting.

Figure 4.10: The weighing station is also used to control the quality grapes received during harvesting.

Vehicles then queue to load their grapes into a tipping bin where they are destemmed and crushed. Tartaric acid may also be added at this stage to lower the pH level of fermenting must (combination of juice and skins) to a level where many undesirable spoilage bacteria cannot live<sup>6</sup>. It also acts as a preservative after fermentation [143]. Once the grapes are destemmed, the stems are disposed of and the resulting must is taken up into the cellar.

### 4.3 Production flow and layout inside the cellar

The must can now be transported through the cellar in a variety of ways, depending on the grapes used and also the wine that is required. A rough illustration of the building layout is provided in Figure 4.3. The *tipping area* consists of the three tipping bins where the grapes are

<sup>6</sup>Grapes grown in warmer climates have lower acidity than grapes grown in cooler climates and the warmer the climate the higher the sugar content of the grapes. Therefore, in certain wine making regions, such as Chablis in France, sugar is added. This is prohibited in warmer regions such as South Africa and the Napa Valley of California [91, 126].



loaded at the outside of the cellar. Usually tipping bin 1 is used for white grapes, tipping bin 3 is used for red grapes whilst tipping bin 2 may be used for either.

After being destemmed and crushed the grapes are pumped to its desired location, depending on the grape colour and the available space. There are different pipes that connect the different tanks and machinery; some are stainless steel pipes (the most durable, but also the most expensive) and others are PVC pipes. The routes represented by the different pipes are colour coded and not all pipes have access to the same machinery. A layout of the machinery and necessary pipes are given in Figure 4.11; this layout excludes the storage tanks, rather focusing on the machinery and tanks required for the active wine making process. The indicated layout is an exact replica of the layout sketch used throughout production at Wamakersvallei Winery. Another means of indicating machines and the different pipes is considered in Chapter 5. From the diagram in Figure 4.11, it is clear that only certain pipes can reach certain machinery, this should be borne in mind when scheduling the grape intakes.

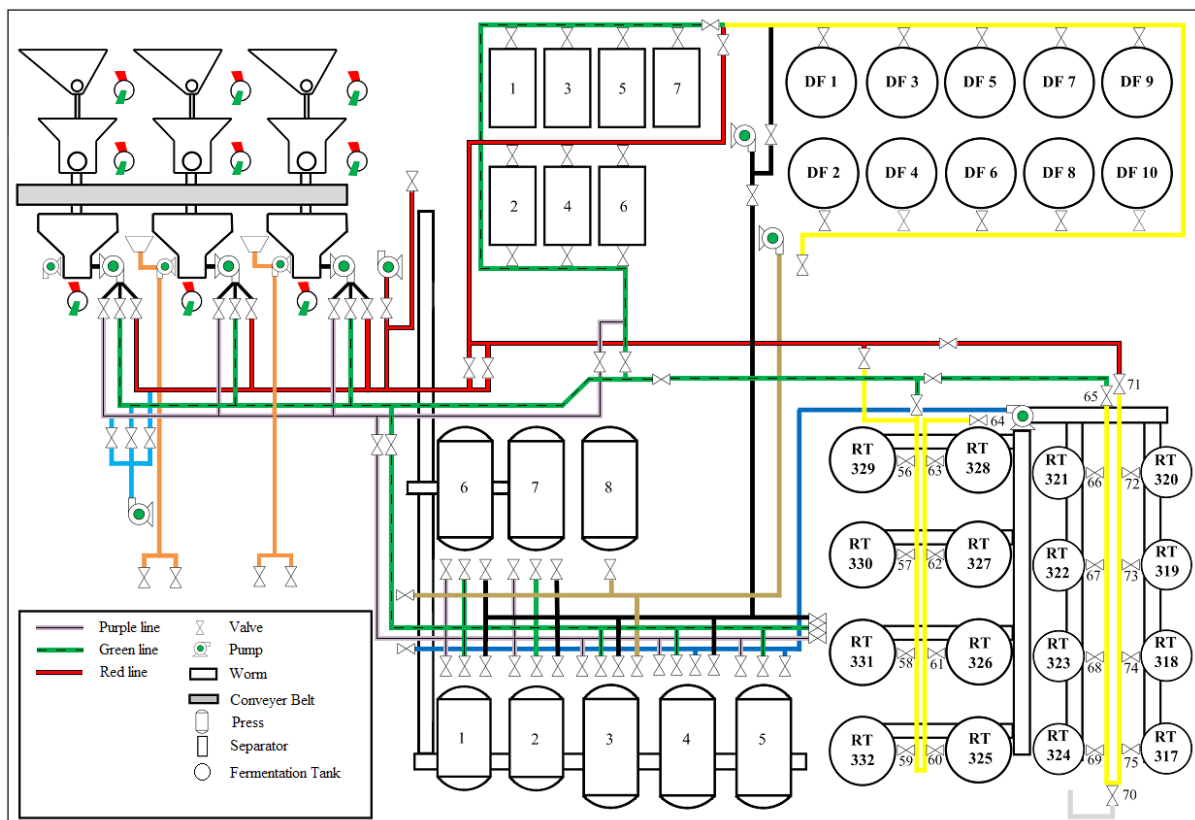


Figure 4.11: Layout of the pipes in the cellar with only the three most important pipes indicated clearly. The purple pipe is indicated with an additional single black line, the green pipe is indicated with the additional dotted black line and the red pipe is indicated with two additional black lines. This is an exact replica of the Wamakersvallei Winery graphical layout.

#### 4.3.1 Production flow of white grapes

Once white grapes have entered the cellar, they may either be sent through the separators to a press or directly to the press. At either machine, the juice may be left on the skins for more or less 30 minutes before starting the process. The separator then takes 45 minutes in order to

separate the juice and skins after which the press takes between one and a half to three hours to retrieve the B juice. Only presses one, two, three and six, as indicated in Figure 4.11, are available for the pressing of white grapes. During these processes it may be decided whether the A and B juice should be kept separate or whether they should be combined.

The juice may then be transported to Store B where the clear juice is drained and racked; this juice may then either be kept in Store B for fermentation or may be transported to Stores C or F for fermentation. Fermentation usually lasts about 14 days and since this process is very temperature sensitive, it is performed as close as possible to 14°C at all times. Once fermentation has been completed, the lees may be drained in order to obtain clearer wine and SO<sub>2</sub> may also be added. The other fining and filtering process also occurs at this stage before bottling. The possible sequences of the different processes are illustrated in Figure 4.12.

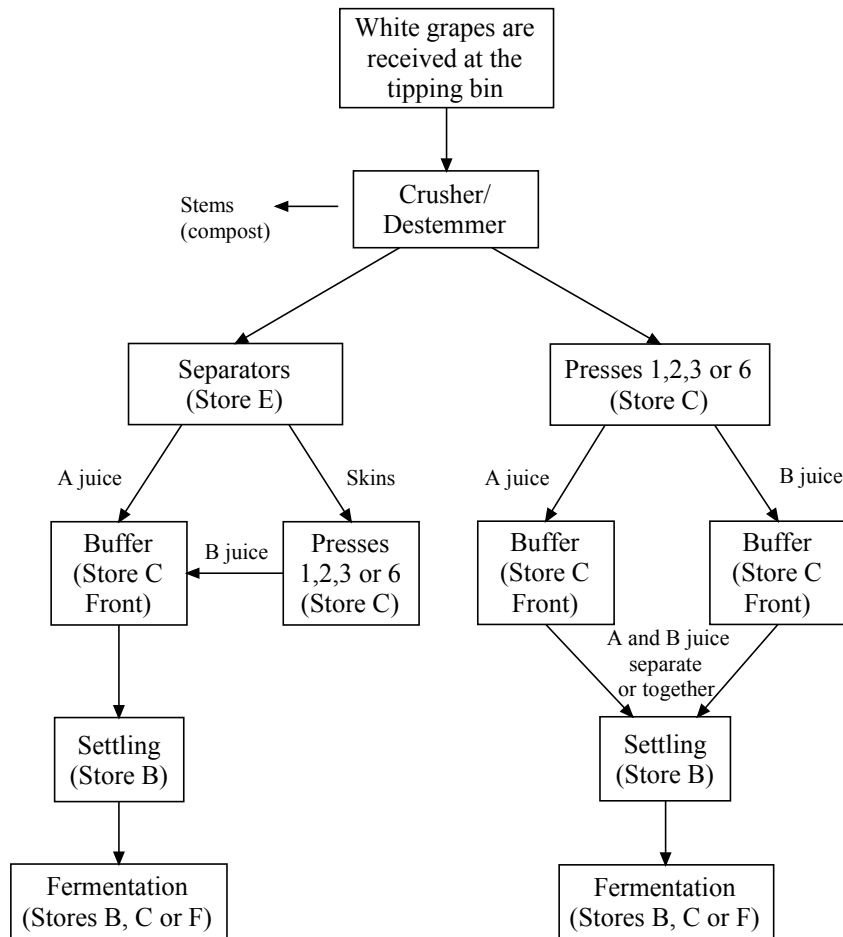


Figure 4.12: Production options for white wine at Wamakersvallei wine cellar [135].

### 4.3.2 Production flow of red grapes

Once the crushed and destemmed red grapes enter the cellar they are immediately allowed to start fermenting. They are therefore taken directly to either DF-tanks or RT-tanks for about five days at a warm 26–28°C since the colour extraction is better at a higher temperature [135]. The DF-tanks can only be reached using the brown line in Figure 4.11. Once the primary fermentation has been completed, the pomace from the DF-tanks may be transported to presses

seven or eight and the pomace from the RT-tanks may be taken to presses four or five. It is once again up to the winemaker to decide whether he wants the A and B juice to remain separate or whether he wishes to combine the two. The wine now goes back to its original tank for further fermentation. This secondary, malolactic fermentation is a process that happens spontaneously, but is catalysed by adding certain bacteria. If this is not done, the process will cause the cork to be pressed out of the bottle due to the resulting pressure from this process starting spontaneously. After secondary fermentation, the lees may be removed and the wine racked, *i.e.* moved around in the cellar, the proper fining and filtering of the wine may also take place. This process is illustrated in Figure 4.13.

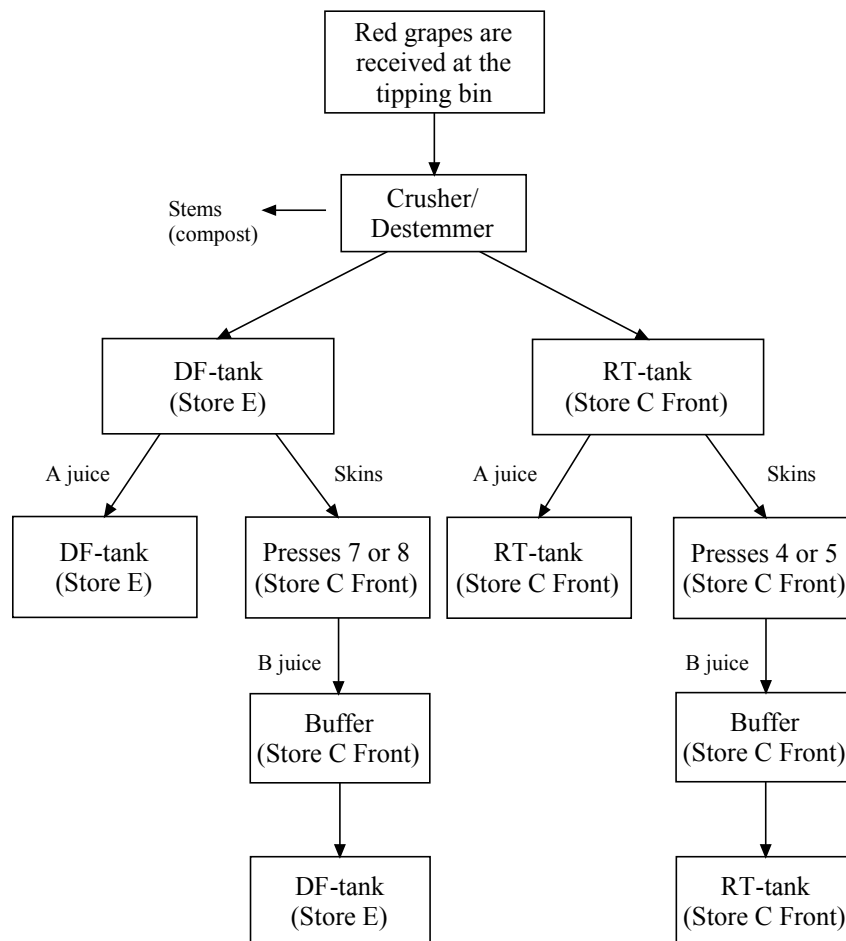


Figure 4.13: Production options for red wine at Wamakersvallei wine cellar [135].

## 4.4 EzyWine

EzyWine [43] was designed in Australia and is now also available in South Africa and New Zealand. Its South African distribution is handled by Action Technologies, a South African-owned information technology company that specialises in business management software for the wine industry. EzyWine was designed to streamline all winery procedures into one comprehensive package which may be implemented easily. Wamakersvallei is one of 69 wineries in South Africa currently using EzyWine [43].

The features of EzyWine include stock control, wine making assistance, marketing, purchasing and sales. Tools to assist in further financial aspects such as payroll management, budgeting and grower payments are also included. All these features are made accessible with a very powerful report writer.

EzyWine is also able to interface with devices such as bar-coding, weather service information, weighbridge scales, temperature control and e-mail browsers. Further services included in the EzyWine package include support and training, online help and bi-annual site visits by EzyWine technicians.

Most importantly for this project, EzyWine easily exports data to Microsoft Excel [86] spreadsheets. It is therefore possible to export sugar level indicators of the samples received to Excel files which may be used as input to the decision support system developed in this thesis.

It is clear that EzyWine is an extremely powerful tool for any winery. However, it may take some time before South African wineries take full advantage of all the features included, as it requires significant effort to fully understand and employ these features as part of the daily cellar management. There are also many features of the EzyWine software package that may be helpful to the decision support system developed as part of this study if the two may be combined in some way, or allowed to interface.

## 4.5 Chapter overview

The goal in this chapter has been to provide the reader with the necessary information to fully understand the origin of the scheduling problems experienced at Wamakersvallei Winery. Therefore, the sections of this chapter contain information regarding the products, objectives and main production processes at this winery.

In the first section of this chapter, some important aspects of Wamakersvallei were discussed, such as its physical location, its current cellar layout, the machinery used in the winery, products delivered by the winery and staff employed there.

Then, in §4.2, more information was provided on the work methods of the employees and the process of ensuring grape quality (received at the cellar). Aspects such as the harvesting process, the method of scheduling of vineyard blocks to be harvested and the required, related procedures were described.

The different production processes and orders of production were considered in §4.3. The focus in this section was mainly on the production of white and red wines, excluding the minority of cases such as Rosé and dessert wines.

Finally, a concise overview of the EzyWine [43] system, used at Wamakersvallei, was presented in §4.4.

---

---

## CHAPTER 5

---

# Mathematical formulation of the cellar scheduling problem

### Contents

5.1	Defining the workspace mathematically . . . . .	75
5.1.1	<i>Jobs and their characteristics</i> . . . . .	76
5.1.2	<i>Processors and their characteristics</i> . . . . .	76
5.1.3	<i>Further parameters and variables</i> . . . . .	77
5.2	Model formulation disregarding pipe assignment . . . . .	80
5.2.1	<i>The constraint sets</i> . . . . .	81
5.2.2	<i>Objective function</i> . . . . .	88
5.3	Chapter overview . . . . .	91

This chapter concerns the scheduling problem occurring inside the cellar, where the aim is to assign grape loads received at the cellar to specific processors for the production of wine, starting at the assignment of a suitable tipping bin. The assignment should be determined in such a manner that it complies with the different machine and process requirements and also to the preferences of the winemakers. The problem with scheduling arises in the active cellar where bottlenecks most often occur during harvesting; therefore the focus in the scheduling problem discussed in this chapter is on the processing of wine inside the active cellar. In §5.2, a mathematical programming model is derived in which assignment of the grape loads to specific pipes is not considered. The mathematical programming model presented in this chapter will also be considered in Chapter 6 when a meta-heuristic method is developed to solve the scheduling problem. However, some of the more general parameters that are required throughout this chapter, describing the jobs, workspace and some of the problem characteristics, is first described in §5.1.

## 5.1 Defining the workspace mathematically

In §5.1.1 the different types of jobs and their characteristics are considered after which parameters concerning the different processor requirements are introduced in §5.1.2, also providing a means of graphical representation of processor relationships in the form of a so-called cellar graph. In §5.1.3 the last parameters to be defined are the problem specific parameters required

for adhering to the rules of the active cellar, as well as the decision variables required to solve the scheduling problem.

### 5.1.1 Jobs and their characteristics

Throughout the remainder of this chapter, the processing of grapes within the active cellar will be referred to as a job and a specific job  $j$  is denoted by  $J_j$ . For each job a parameter,  $w_j$ , indicating the weight of the load is required to compare the load size to the relevant processor capacities.

The jobs  $J_1, \dots, J_r$  refer to the processing of red grapes (used to produce red wine) received at the cellar on the considered day, whereas  $J_{r+1}, \dots, J_w$  refer to the processing of white grapes. The models introduced in this chapter are capable of generating a schedule for one working day at a time and since fermentation of red wine takes approximately 5 to 7 days, the emptying of the fermentation tanks and the pressing of red grapes take place on another day. Jobs  $J_{w+1}, \dots, J_n$  may therefore refer to loads of red grapes currently in specific red wine fermentation tanks which need to be emptied and the contents pressed on the day for which a schedule is sought.

A job  $J_j$  is further divided into a set of tasks,  $T_{j1}, \dots, T_{jk_j}$ , according to the different phases of processing and the specific machine requirements. For the purpose of this thesis, there will be six types of tasks. The first, denoted by  $T_1$ , takes place at the tipping bin and concerns the crushing and destemming of a load of grapes. The second type of task,  $T_2$ , is concerned with separating the juice from the skins of white grapes and may be performed on the separators or presses. The skins of both white and red grapes should be pressed at some stage during the production process — this forms the third task type,  $T_3$ . The fourth task type,  $T_4$ , refers to the primary fermentation of red grapes, while task type  $T_5$  refers to the emptying of a fermentation tank. Finally, task type  $T_6$ , indicates the transportation of grapes, juice or skins between any two machines and is relevant only to the problem formulation of §B.3. Every task type can be performed on any one of a specified subset of processors.

### 5.1.2 Processors and their characteristics

Throughout this chapter, there will be a difference in definition of *processors* and *machines*. Processors refer to any equipment used for the processing of grapes in the active cellar, whereas machines only refer to physical machines, such as a red wine fermentation tank or a press, and not to pipes. In order to separate the different types of processors used, the processor set,  $\mathcal{P} = \{P_1, \dots, P_m\}$ , is further divided into subsets. The first subset  $\{P_1, \dots, P_{m_1-1}\}$  always refers to the tipping bins. Furthermore,  $\{P_{m_1}, \dots, P_{m_2-1}\}$  denotes the set of separators and  $\{P_{m_2}, \dots, P_{m_3-1}\}$  the presses. Finally, the red wine fermentation tanks are denoted by  $\{P_{m_3}, \dots, P_{m_4-1}\}$  for the DF tanks and  $\{P_{m_4}, \dots, P_{m_5-1}\}$  for the RT tanks. There exists no need to number the individual machines inside their subsets in any specific order, but when including the assignment of tasks to pipes, it is necessary to further differentiate between types of pipes. Before these subsets can be derived, the need for this division and some further concepts are first explained. From Figure 4.11 it is clear that a pipe connecting two machines is most often divided into parts, allowing pipes to join and split with the use of valves. This makes it possible to consider each section of pipe individually, together with a preceding and following processor — be it a machine or pipe. In order to understand the numbering and combination of such sections, the concept of a *cellar graph* is introduced. A cellar graph simply serves as a graphical representation of the model at hand and is used to illustrate certain concepts and

rules; it does not reflect the actual layout of the cellar and other physical characteristics. When the division of a pipe does not serve any specific goal, it is considered and illustrated in a cellar graph as one processor. The interpreted cellar graph of the active cellar at Wamakersvallei Winery is shown in Figure 5.1. The numbering of pipes may be omitted when pipe assignment is not considered.

The solid (or coloured) vertices represent machines and the thick arcs (directed edges) represent the pipes and their direction of flow. The circles allow pipes to separate and/or join. For example  $P_{67}$  and  $P_{70}$  are joined together by the circle to form the resulting processor,  $P_{72}$ . The lines are used to illustrate the connection between processors where one pipe is connected to more than one processor. For example, the green line from Figure 4.11 allows entry to all separators; therefore in Figure 5.1, the corresponding processor  $P_{72}$  is connected to  $\{P_4, \dots, P_{10}\}$  by means of lines not representing pipes. It is important to use such a method, since multiple pipes connecting each of  $P_{67}$  and  $P_{70}$  to  $\{P_4, \dots, P_{10}\}$  (replacing the single  $P_{72}$ ) will not only look cluttered, but will also create the impression that all of these parallel pipes are allowed to be used simultaneously, which is not the case.

In order to formulate a model that includes the assignment of tasks to the different pipe segments, the pipes are split into two further subsets of  $\mathcal{P}$  which are only relevant to §B.3. For the mathematical programming model in §5.2 the last machine,  $P_{m_5-1}$ , is the last processor, implying that  $m_5 - 1 = m$  for that specific problem formulation. However, the first set of pipes for a formulation including pipe assignment is  $\{P_{m_5}, \dots, P_{m_6-1}\}$  and refers to the set of *single pipes*. A so-called single pipe is a pipe that has a machine as predecessor and any processor as successor. In Figure 5.1 examples of single pipes are  $\{P_{45}, \dots, P_{53}\}$  and also  $P_{54}$  which is used to transport the grape skins from the separators to a select set of presses. In addition to the actual single pipes represented in Figure 5.1, there are also dummy pipes from every press to itself allowing the tasks of separation and pressing to follow on one another on the same press; these dummy pipes are also considered single pipes. All other pipes may be referred to as *non-single* pipes and consists of the set  $\{P_{m_6}, \dots, P_m\}$  and refers to the pipes which have only other pipes as predecessors. Because of the circles and lines used in the cellar graph, it is not always clear which pipes are single, since it may create the illusion that a pipe is, in fact, following on another pipe and not the relevant set of machines (take  $P_{54}$  as an example). In such cases the numbering of the pipes can serve as a guideline. For the Figure 5.1, the sets may be defined by the parameters  $m_1 = 4, m_2 = 11, m_3 = 19, m_4 = 29, m_5 = 45, m_6 = 65$  and  $m = 79$  with the single pipes  $P_{57}, \dots, P_{64}$  representing the dummy pipes at the presses.

For each of the red fermentation tanks, the allowed capacity of the tank is denoted by  $c_i$  and is measured in tonnes. This allowed capacity for red wine fermentation tanks, form about three quarters of its physical capacity<sup>1</sup>. Each separator also has a limited capacity of 20 tonnes and there are both 15 and 20 tonne presses. Furthermore, the red wine fermentation tanks also require a parameter  $v_i$  which indicates its volume at the start of the considered day.

### 5.1.3 Further parameters and variables

The parameters  $\mu_{1jk}, \dots, \mu_{mjk}$  are associated with each task  $T_{jk}$ , where

$$\mu_{ijk} = \begin{cases} 1 & \text{if } T_{jk} \text{ is allowed to be performed on processor } P_i, \\ 0 & \text{otherwise.} \end{cases}$$

<sup>1</sup>Exceeding three quarters of the actual capacity may cause the tank to dent, or worse, break the lid open, either way resulting in serious damage due to the gas that is omitted during fermentation.

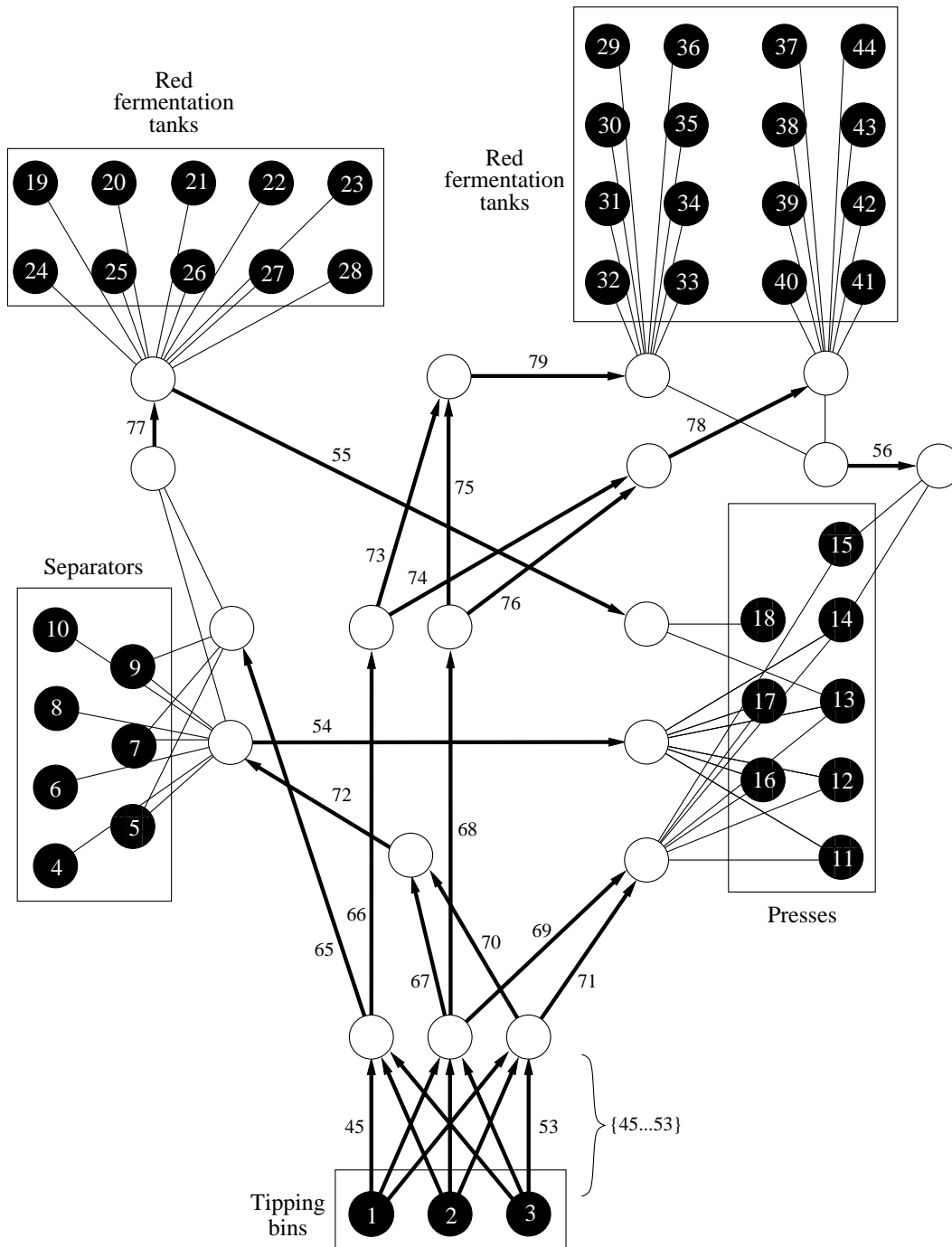


Figure 5.1: The cellar graph representation of the active cellar at Wamakersvallei Winery where the solid vertices represent the machines and the thick directed edges represent the pipes. For this cellar graph  $m_1 = 4, m_2 = 11, m_3 = 19, m_4 = 29, m_5 = 45, m_6 = 65$  and  $m = 79$ . The processors  $P_{57}, \dots, P_{64}$ , representing the dummy pipes from a press to itself, are omitted in the representation.

The duration of processing task  $T_{jk}$  on processor  $P_i$  is denoted by  $p_{ijk}$  and expressed in hours. When red wine and white wine are processed directly after each other on the same processor, further time is required for the cleaning of the machine. There are also machines, such as the presses, that always require some form of cleaning after use, regardless of the colour of successive



grape loads<sup>2</sup>. It is therefore necessary to include setup times, where  $s_{ij\ell}$  refers to the setup time required to have machine  $P_i$  ready to process any task from  $J_\ell$  directly after processing a task of  $J_j$ . The assumed time of arrival of Job  $J_j$  at the cellar, is given as  $e_j$ .

Red wine fermentation tanks have the unique characteristic that more than one job is allowed to be assigned to the tank in overlapping times. However, not just any two loads are allowed to be assigned to the same tank in this manner. The first requirements are that the two relevant jobs should involve the same cultivar and quality<sup>3</sup>. It is also important to respect the intuition of the winemaker and the quality of the final product (or wine) that he desires. Therefore, a parameter

$$q_{j\ell} = \begin{cases} 1 & \text{if grapes from } J_j \text{ and those of } J_\ell \text{ are allowed to be mixed,} \\ 0 & \text{otherwise,} \end{cases}$$

is defined, allowing the winemaker to specify when two jobs that are of the same cultivar and quality may not be mixed. If the winemaker does not give any input with respect to this matter, wines of the same cultivar and quality are allowed to be mixed. It is also still allowed to add a new load of grapes to the load already in the tank if the received load is similar to that inside the tank. This is, however, only allowed if all the loads assigned to a tank arrive over a period of two contiguous days and not longer. Therefore a tank may be suitable for a new load either if the tank is empty or if it is allowed to mix the new load with its current load. Another parameter,

$$tq_{ij} = \begin{cases} 1 & \text{if it is allowed to assign } J_j \text{ to red wine fermentation tank } P_i, \\ 0 & \text{otherwise,} \end{cases}$$

similar to  $q_{j\ell}$ , is thus required, again allowing the winemaker to intervene.

Subject to the objective function and the constraints, the first variables to determine the feasibility of a schedule are the starting and ending times of the specific tasks, denoted  $t_{jk}$  and  $f_{jk}$  for each task  $T_{jk}$ , respectively. The assignment variable,

$$a_{ijk} = \begin{cases} 1 & \text{if } T_{jk} \text{ is assigned to machine } P_i, \\ 0 & \text{otherwise,} \end{cases}$$

is required for each  $(T_{jk}, P_i)$  combination. In order to determine where the relevant setup times are required and also to schedule tasks in such a manner that their processing times do not overlap when assigned to the same processor, the variable

$$x_{ij\ell} = \begin{cases} 1 & \text{if a task of job } J_\ell \text{ follows a task of Job } J_j \text{ immediately on } P_i, \\ 0 & \text{otherwise,} \end{cases}$$

is additionally defined. Furthermore, the variable  $M$  refers to a large number set as 10 000 throughout the thesis.

The objective function to be minimized is denoted by  $C$  and will be defined in the relevant sections. The goal of each of the following approaches is to derive a feasible schedule using the parameters and variables defined to generate a constraint set.

<sup>2</sup>This refers to the removal of the remaining grape skins from the presses, rather than the actual cleaning of the whole machine.

<sup>3</sup>There are no exceptions to this rule. Even when a blended wine is desired, the different cultivars are kept separate until the final stages of production, which do not occur in the active cellar.

## 5.2 Model formulation disregarding pipe assignment

A mathematical programming model for a general multi-operational job shop with sequence dependent setup times was given in §3.3.1. Due to additional constraints, such as the no-wait characteristic, and further problem specific complications, this mathematical programming model requires further expansion to conform to the scheduling problem at Wamakersvallei Winery.

In this approach, the assignment of tasks to the pipes is omitted, since Wamakersvallei (as well as the majority of other wineries) make use of temporary pipes that can be moved around the cellar making it possible to connect any two machines for the purpose of transporting juice<sup>4</sup>. The tasks for red and white wines, their task types and the processors required are listed in Tables 5.1 and 5.2.

Red wine	First Job		Second job	
Task type	$T_1$	$T_4$	$T_5$	$T_3$
Tasks	$T_{j1}$	$T_{j2}$	$T_{j'1}$	$T_{j'2}$
Processor	Tipping bins	Fermentation tanks	Fermentation tanks	Presses
Time (hours)	0.15 tot 0.25	120 to 168	4 to 6	3

Table 5.1: The different tasks required in order to process a load of red grapes. The corresponding task types, processor requirements and approximate durations for the specific processes are also shown. The first job concerns the day of grape delivery, whereas the second job refers to the processing after primary fermentation.

White wine			
Task type	$T_1$	$T_2$	$T_3$
Tasks	$T_{j1}$	$T_{j2}$	$T_{j3}$
Processor	Tipping bins	Separators or Presses	Presses
Time (hours)	0.15 tot 0.25	0 to 4	2.5

Table 5.2: The different tasks required in order to process a load of white grapes, as well as the corresponding task types, processor requirements and production times. The great difference in duration for the separation of grapes occurs since it can be a by product of the press therefore not taking any additional time or it may spend up to 4 hours in a separator.

The durations presented in Tables 5.1 and 5.2 are based on times indicated by one of the winemakers at Wamakersvallei [135] and is more thoroughly explained in Chapter 4. It is still important to note that the production times do not include any setup times. For example, it takes approximately 30 minutes to fill up a press. For red wine the grapes are pressed on a cycle of 2.5 hours (2 hours for white wine) and after pressing the dry pressed grape skins have to be removed, which takes between 45 minutes and 1 hour and is considered as setup time. The significant variation in duration of separation of white grapes is due to the fact that it is possible to separate the grapes and skins in a press, where separation (or more accurately draining of the juice) takes place, while the press is being filled. Therefore the actual task of separation on the press requires no more time than required for the pressing of the grapes. When assigned to separators, certain white grape varieties often require skin contact and may be left on the

<sup>4</sup>A worm is used for the transportation of grape skins and can therefore not be replaced by a temporary pipe.

skins for up to 2 hours. It takes an additional 30 to 45 minutes to fill the separator, 45 minutes to drain the juice and 30 minutes to remove the skins. In this case, the 30 minutes is not considered as setup time, since the grape skins still require further processing.

### 5.2.1 The constraint sets

The constraint

$$t_{j1} \geq e_j, \quad j = 1, \dots, n$$

ensures that no job starts before its actual arrival at the cellar. In order to ensure that the starting time  $t_{jk}$  and ending time  $f_{jk}$  of all tasks allow enough time for production,  $p_{ijk}$ , on the assigned machine  $P_i$ , the inequality

$$t_{jk} + a_{ijk}p_{ijk} \leq f_{jk}, \quad i = 1, \dots, m, \quad j = 1, \dots, n, \quad k = 1, \dots, k_j, \quad (5.1)$$

is included. The value of  $t_{jk} + a_{ijk}p_{ijk}$  should ideally be equal to  $f_{jk}$ , in which case the starting time together with the production time is equal to the time at which production ends<sup>5</sup>. It is unfortunately not possible to enforce this with the current notation, since not all assignments  $a_{ijk}$  will be equal to 1; therefore the equality  $t_{jk} + p_{ijk} = f_{jk}$  will hold for some processor  $P_i$  with  $p_{ijk} \neq 0$ , as well as the equality  $t_{jk} = f_{jk}$  for all other  $a_{ijk} = 0$ . Hence it is important that  $f_{jk}$  or  $t_{jk}$  forms part of the objective function to be minimized in order to obtain the equality  $t_{jk} + p_{ijk} = f_{jk}$  for the assigned processor  $P_i$ . It is, however, imperative that once a task  $T_{jk}$  has been completed, including (possibly) waiting in a buffer tank, processing of the following task,  $T_{j(k+1)}$ , should start without delay. This is referred to as the no-wait characteristic of the schedule and is implemented by an equality constraint of the form

$$f_{jk} = t_{j(k+1)}, \quad j = 1, \dots, n, \quad k = 1, \dots, k_j - 1.$$

The additional time required to transport the grapes or juice from one machine to another is so small (approximately 30 seconds [135]) that it is negligible when compared to the processing times of the jobs ranging from 2 hours to 7 days<sup>6</sup>. However, if this level of detail is required, the starting time of a new task,  $t_{j(k+1)}$ , may be set equal to the ending time of predecessor together with the additional time required for transportation, *i.e.*  $f_{jk} + 30$ s.

Not all tasks are allowed to be processed on all machines. Therefore the inequality

$$a_{ijk} \leq \mu_{ijk}, \quad i = 1, \dots, m, \quad j = 0, \dots, n, \quad k = 1, \dots, k_j,$$

is incorporated in order to limit the assignment of each task to only machines that are predetermined as allowable. If the parameter  $\mu_{ijk}$  for a task  $T_{jk}$  is equal to 1 (therefore allowed), the assignment variable  $a_{ijk}$  is still allowed to be equal to 0 if the actual assignment to the machine is not made.

There are tasks that may be assigned to more than one machine simultaneously. This occurs, for example, when a load of white grapes larger than 20 tonnes requires separation and then

<sup>5</sup>It is possible to hold the grapes in any processor other than the tipping bins for a while longer since they all make use of buffer tanks to receive the juice. The processor remains unavailable to new loads during this time.

<sup>6</sup>These times are valid when focusing on the active cellar. Fermentation of white wine, which does not occur in the active cellar, may take up to 14 days.

need to be assigned to a set of separators with a capacity of only 20 tonnes each. However, the constraint

$$\sum_{i=1}^m a_{ij1} = 1, \quad j = 1, \dots, n$$

limits the assignment of the relevant tasks to exactly one processor (these processors are the tipping bins and red wine fermentation tanks). In the event that red grape loads exceed the maximum available tank capacity, the grape load is split up to form more than one job as input to the mathematical programming model<sup>7</sup>.

Task  $T_{j2}$  of any white grape job  $J_j$  may be assigned to either a press or a separator. An additional variable,  $y_{(j-r)}$ , is required for every white grape job  $J_j$  in order to allow the assignment of task,  $T_{j2}$ , to either a press or a separator, and not both. The variable  $y_{(j-r)}$  is then 1 if task  $T_{j2}$  is performed on a separator and 0 otherwise. Since the separators may each handle up to 20 tonnes at a time, the number of separators required for the processing of a task,  $T_{j2}$ , is  $\lceil \frac{w_j}{20} \rceil$ . The inequality

$$\sum_{i=m_1}^{m_2-1} a_{ij2}c_i \geq w_j y_{(j-r)}, \quad j = r+1, \dots, w,$$

combines the two requirements by ensuring that an assignment is made to a separator only if the assignment is not made to a press by including  $y_{j-r}$  and also assigning the task to the correct number of separators as determined by the job weight. Another characteristic of separators is that since they are connected to the presses by a single worm, only one separator may be emptied at a time. Only the last hour of separation concerns the emptying of the tank. The inequality  $|f_{j2} - f_{\ell 2}| \geq 1$  should therefore hold for all white grape jobs,  $J_j$  and  $J_\ell$ , with tasks  $T_{j2}$  and  $T_{\ell 2}$  assigned to the separators. Unfortunately, including absolute values in the constraint set results in a non-linear model, which should be avoided when possible. For this reason another decision variable,  $z_{(j-r)(\ell-r)}$  is rather created for every  $(J_j, J_\ell)$  pair where  $j, \ell = r+1, \dots, w$ . Incorporation the inequalities

$$f_{j2} - f_{\ell 2} \geq 1 - (3 - z_{(j-r)(\ell-r)} - y_{(j-r)} - y_{(\ell-r)}) M, \quad \begin{array}{l} j = r+1, \dots, w-1, \\ \ell = j+1, \dots, w \end{array} \quad (5.2)$$

and

$$f_{\ell 2} - f_{j2} \geq 1 - (z_{(j-r)(\ell-r)} + 2 - y_{(j-r)} - y_{(\ell-r)}) M, \quad \begin{array}{l} j = r+1, \dots, w-1, \\ \ell = j+1, \dots, w \end{array} \quad (5.3)$$

will enforce the rule that the ending time of any two tasks  $T_{j2}$  and  $T_{\ell 2}$  should always differ by at least 1 if they are both assigned to the separators<sup>8</sup>.

The presses have different capacities and hence the inequality

$$\sum_{i=m_2}^{m_3-1} a_{ij2}c_i \geq w_j (1 - y_{(j-r)}), \quad j = r+1, \dots, w,$$

<sup>7</sup>A non-linear constraint set is required when the splitting of red grape jobs is allowed in the programming model. This is due to the additional characteristic that a part of the split job is allowed to mix with another job and there should still be kept track of current volumes.

<sup>8</sup>The decision variable  $z_{(j-r)(\ell-r)}$  is included to ensure that only one of Equations (5.3) and (5.4) is considered, Equation (5.3) if  $z_{(j-r)(\ell-r)} = 0$  and Equation (5.4) if  $z_{(j-r)(\ell-r)} = 1$ .

is included to ensure that task  $T_{j2}$  of any white grape job should only be assigned to the presses if it is not already assigned to a separator and that it should be limited by the available capacities. It is not necessary to limit the assignment from above. If a job is assigned to more presses than necessary, the assignment will be reduced if it influences the objective function discussed in §5.2.2. However, when a task  $T_{j2}$  is assigned to a separator,  $y_{(j-r)} = 1$ , the above inequality does not ensure that  $a_{ij2} = 0$  for all presses. It is therefore required that a constraint of the form

$$a_{ij2} \leq 1 - y_{(j-r)}, \quad i = m_2, \dots, m_3 - 1, \quad j = r + 1, \dots, w,$$

is also included. When the task,  $T_{j2}$ , is assigned to a press for separation, it remains in that same press for pressing. This requirement may be expressed by the inequality

$$a_{ij2} \leq a_{ij3}, \quad i = m_2, \dots, m_3 - 1, \quad j = r + 1, \dots, w, \quad (5.4)$$

which ensures that separation is only allowed on a machine if pressing is also performed on the same machine. It is, however, allowed that only pressing is performed on press  $P_i$  ( $a_{ij3} = 1$ ) and that separation is carried out on a separator ( $a_{ij2} = 0$  for the same  $P_i$ ); in this case the capacities of the presses should be adhered to by including a constraint set of the form

$$\sum_{i=m_2}^{m_3-1} a_{ij3}c_i \geq w_j, \quad j = r + 1, \dots, w.$$

The same inequality is also included for task  $T_{j2}$  of all jobs ( $j = w + 1, \dots, n$ ) because the red grape skins also require pressing after primary fermentation.

Another set of processors requiring specialized constraints, are the red wine fermentation tanks used for primary fermentation. Other than most of the other processors used in the active cellar, these tanks typically already contain a current volume of red grapes received on a previous day. In order to ensure that the weight of the job assigned to such a tank does not exceed its available space, it is required that

$$v_i + \sum_{j=1}^r a_{ij2}w_j \leq c_i, \quad i = m_3, \dots, m. \quad (5.5)$$

Two aspects of this inequality form a very important part of the independent characteristics of the set of processors. The first is the summation over the jobs of red grapes, which is due to the characteristic that more than one load is allowed to be assigned to the same fermentation tank. The assignment of two jobs  $J_j$  and  $J_\ell$  to the same fermentation tank is limited by means of a constraint set of the form

$$a_{ij2} + a_{i\ell 2} \leq q_{j\ell} + 1, \quad i = m_3, \dots, m, \quad j = 1, \dots, r, \quad (5.6)$$

$$\ell = 1, \dots, r \text{ and } \ell \neq j.$$

The second aspect is the fact that the current volume of the tank is included in the inequality. This indicates that it is possible to add a new load of grapes to a tank already containing grapes from the previous day (only). A constraint set of the form

$$a_{ij2} \leq tq_{ij}, \quad i = m_3, \dots, m, \quad j = 1, \dots, r \quad (5.7)$$

enforces this relationship by limiting the assignment of grape loads to non-empty tanks adhering to job-specific requirements. This parameter is also used to manipulate the assignment of a task

of type  $T_5$  to the tank in which it has completed primary fermentation. The equality constraints of the form

$$a_{ij1} = tq_{ij}, \quad i = m_3, \dots, m, \quad j = w + 1, \dots, n \quad (5.8)$$

are included in order to set a time at which the emptying of this vessel should start. This time consequently determines the period during which a press should be assigned to the following task of the job. Another characteristic is that no two tanks from the same set of red wine tanks may be emptied at the same time<sup>9</sup>, because there is only one worm connecting each group of tanks to the presses, and temporary pipes cannot be used to transport the grape skins. If two tanks of the same subset are to be emptied in one day, it is of no significant importance which is emptied first. Therefore, inequalities of the form

$$f_{j1} \leq t_{\ell 1} + \left( 2 - \sum_{i=m_3}^{m_4-1} a_{ij1} - \sum_{i=m_3}^{m_4-1} a_{i\ell 1} \right) M, \quad j = w + 1, \dots, n, \quad \ell = j + 1, \dots, n$$

for the DF tanks and

$$f_{j1} \leq t_{\ell 1} + \left( 2 - \sum_{i=m_4}^m a_{ij1} - \sum_{i=m_4}^m a_{i\ell 1} \right) M, \quad j = w + 1, \dots, n, \quad \ell = j + 1, \dots, n$$

for the RT tanks limit a task  $T_{j1}$  to be performed before the task  $T_{\ell 1}$  if they are assigned to the same set of red wine tanks.

In order to include the relevant setup times required between two successive jobs following on the same processor, inequality constraints of the form

$$f_{jk} + s_{ij\ell} \leq t_{\ell h} + (3 - x_{ij\ell} - a_{ijk} - a_{i\ell h}) M, \quad h = 1, \dots, k_\ell - 1, \quad i = 1, \dots, m, \quad j = 0, \dots, n, \\ k = 1, \dots, k_j - 1, \quad \ell = 1, \dots, n, \quad \text{and } \ell \neq j$$

are included. Not only do these constraints allow time for the setup of machine  $P_i$ , should  $T_{\ell h}$  follow  $T_{jk}$  directly on machine  $P_i$ , but they also impose the constraint that no two jobs should be performed simultaneously on one machine. For all processors other than the presses, the variables  $x_{ij\ell}$  are required to satisfy

$$\sum_{j=0}^n x_{ij\ell} = \sum_{h=1}^{k_\ell} a_{i\ell h}, \quad i = 1, \dots, m_2 - 1, m_3, \dots, m, \quad \ell = 1, \dots, n$$

and

$$\sum_{\ell=0}^n x_{ij\ell} = \sum_{k=1}^{k_j} a_{ijk}, \quad i = 1, \dots, m_2 - 1, m_3, \dots, m, \quad j = 0, \dots, n.$$

Since a job involving white grapes is allowed to be processed on the presses for two consecutive tasks, the constraint set for determining the values  $x_{ij\ell}$  for presses, differs slightly. The decision variable  $x_{ijj}$  should never be allowed to equal one; hence the second task of a white wine job is not included in the summation when determining  $x_{ij\ell}$  for a press. The inequality constraints for each of the presses may be expressed as

<sup>9</sup>One tank from the DF subset and one from the RT subset may be emptied simultaneously.

$$\sum_{j=0}^n x_{ij\ell} = \begin{matrix} k_\ell (\text{with } h \neq 3, \\ \text{if } r+1 \leq \ell \leq w) \end{matrix} \sum_{h=1} a_{i\ell h}, \quad i = m_2, \dots, m_3 - 1, \ell = 1, \dots, n$$

and

$$\sum_{\ell=0}^n x_{ij\ell} = \begin{matrix} k_j (\text{with } k \neq 3, \\ \text{if } r+1 \leq j \leq w) \end{matrix} \sum_{k=1} a_{ijk}, \quad i = m_2, \dots, m_3 - 1, j = 0, \dots, n.$$

It is furthermore required that  $x_{ijj} = 0$  for all  $i = 1, \dots, m$  and  $j = 0, \dots, n$ . Further trivial constraints include

$$t_{jk} \geq 0, \quad j = 0, \dots, n, k = 1, \dots, k_j$$

and

$$f_{jk} \geq 0, \quad j = 0, \dots, n, k = 1, \dots, k_j,$$

ensuring that the starting and ending times of all tasks are positive numbers. The starting and ending times may also be limited to business hours by including inequalities ensuring that the relevant  $t_{jk}$  and  $f_{jk}$  values are smaller than the desired number of hours contained in one working day. The variables  $a_{ijk}$ ,  $x_{ij\ell}$ ,  $y$  and  $z$  should be binary variables, requiring constraints of the form

$$\begin{aligned} a_{ijk} &\in \{0, 1\}, & i = 1, \dots, m, j = 0, \dots, n, k = 1, \dots, k_j, \\ x_{ij\ell} &\in \{0, 1\}, & i = 1, \dots, m, j = 0, \dots, n, \ell = 1, \dots, n, \\ y_{(j-r)} &\in \{0, 1\}, & j = r + 1, \dots, w, \\ z_{(j-r)(\ell-r)} &\in \{0, 1\}, & j = r + 1, \dots, w, \ell = j + 1, \dots, w. \end{aligned}$$

The cellar scheduling problem at Wamakersvallei Winery comprises a relatively large number of processors and therefore also a large number of constraints and variables. A smaller (fictional) active cellar is used to illustrate the different characteristics, such as the allocation of processor and setup times, and also the adjustment of input parameters in order to suit the wishes of the winemakers. This example is continued later on in order to justify the chosen objective function.

**Example 5.1** *In order to illustrate the working of the mathematical programming model disregarding pipe assignments, a smaller, fictitious cellar is considered. The cellar graph for this active cellar is shown in Figure 5.2 and is constructed in such a manner that specific characteristics from the active cellar at Wamakersvallei Winery are present.*

*Further characteristics of the workspace such as the capacities of the relevant machines and also the different starting volumes are presented in Table 5.3. Included in Table 5.3 are the current contents of each of the non-empty red fermentation tanks in order to determine which tanks are still allowed to be assigned to further jobs. When assignment to a tank is still possible, it is marked as an open tank in Table 5.3.*

*The mathematical programming model is applied in order to schedule one day of processing where the different grape loads that require processing are two loads of Cabernet Sauvignon of*

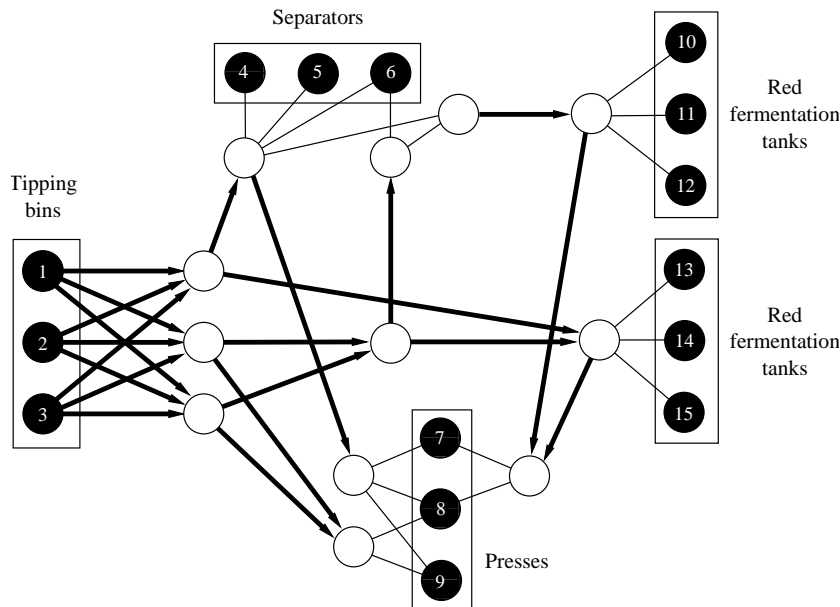


Figure 5.2: The cellar graph for the active cellar used in Example 5.1 to illustrate the working of the mathematical programming model.

$P_i$	$v_i$	$c_i$	Cultivar	Class	Open
$P_4$		20			
$P_5$		20			
$P_6$		20			
$P_7$		20			
$P_8$		20			
$P_9$		15			
$P_{10}$	55	80	Merlot	1	No
$P_{11}$	45	80	Pinotage	2	Yes
$P_{12}$	0	80			Yes
$P_{13}$	40	50	Pinotage	1	No
$P_{14}$	40	50	Shiraz	1	No
$P_{15}$	50	50	Shiraz	1	No

Table 5.3: The allowed capacities  $c_i$  of the machines ( $P_4, \dots, P_{15}$ ) as well as the relevant volumes  $v_i$  together with the classification of the contents of each red fermentation tank (when applicable) for Example 5.1.

class 1, weighing 40 and 25 tonnes respectively, followed by a third load consisting of 20 tonnes of class 2 Pinotage. Three loads of white grapes are also expected, namely a class 1 batch of Chardonnay weighing 40 tonnes and one of 30 tonnes, as well as a 20 tonne load of class 1 Sauvignon Blanc. Suppose the winemaker has requested that both loads of Chardonnay should lie on their skins for 1 hour in the separators. Assume further that, in addition to these grape loads to be received, there is a load of Merlot currently in red wine fermentation tank  $P_{10}$  which requires pressing. The subsequent jobs are listed in Table 5.4. Also contained in the Table is the allowable grape and tank combinations. This derivation of allowable red wine fermentation tanks rely heavily on whether or not the tank is still open to receive further grapes as shown in Table 5.3. For example, from this table it may be deduced that  $t_{q_{10,6}} = 1$  since the Merlot of



job  $J_6$  is allowed to be processed on  $P_{10}$ , and that  $t_{q_{13,j}} = 0$  for all jobs  $J_j$  since the tank is not open. Job  $J_3$  is allowed to be added to the contents of  $P_{11}$  since the Pinotage in  $P_{11}$  matches the new load and was received only one day before the current scheduling day. The dummy job  $J_0$  should always be allowed on all machines that are open for production on the day of scheduling, since the dummy job is used in determining the various  $x_{ij\ell}$  values for all active machines and serves as the first and last tasks to be processed on a machine.

$J_j$	Cultivar	$w_j$ (tonnes)	Allowed red fermentation tanks
$J_0$	Dummy job	0	$P_{10}, \dots, P_{15}$
$J_1$	Cabernet Sauvignon	40	$P_{12}$
$J_2$	Cabernet Sauvignon	25	$P_{12}$
$J_3$	Pinotage	15	$P_{11}$ and $P_{12}$
$J_4$	Chardonnay	40	
$J_5$	Chardonnay	30	
$J_6$	Sauvignon Blanc	20	
$J_7$	Merlot	50	$P_{10}$

Table 5.4: The jobs to be scheduled and the allowed red fermentation tanks for the loads of red grapes of Example 5.1.

For the parameter set of  $q_{j\ell}$ , the only non-zero parameter is  $q_{1,2}$  (or  $q_{2,1}$ ) since  $J_1$  and  $J_2$  are the only two jobs where mixing is allowed. The durations of the different tasks on the various machines are listed in Table 5.5. All  $\mu_{ijk}$  values may be derived from Table 5.5 — these values are listed in Table B.1.

Job $j$	Task $k$	Allowed $P_i$	Corresponding $p_{ijk}$
0	1	$\mathcal{P}$	0
1	1	$P_1, P_2, P_3$	0.2
	2	$P_{10}, \dots, P_{15}$	144
2	1	$P_1, P_2, P_3$	0.25
	2	$P_{10}, \dots, P_{15}$	144
3	1	$P_1, P_2, P_3$	0.2
	2	$P_{10}, \dots, P_{15}$	120
4	1	$P_1, P_2, P_3$	0.25
	2	$P_4, P_5, P_6$	3.0
	3	$P_7, P_8, P_9$	2.5
5	1	$P_1, P_2, P_3$	0.2
	2	$P_4, P_5, P_6$	3.0
	3	$P_7, P_8, P_9$	2.5
6	1	$P_1, P_2, P_3$	0.2
	2	$P_4, P_5, P_6$	1.75
		$P_7, P_8, P_9$	0
	3	$P_7, P_8, P_9$	2.5
7	1	$P_{10}, \dots, P_{15}$	5
	2	$P_7, P_8, P_9$	3

Table 5.5: The duration of processing  $p_{ijk}$  for task  $T_{jk}$  on an allowed set of processors (all processors for which  $\mu_{ijk} = 1$ ) in Example 5.1.

Setup times for this example are incurred when a load of white grapes follows a load of red grapes in the tipping bins (1.5 hours) or when a load of red follows on a load of white (0.5 hours). Furthermore, a setup time is required on the presses when two separate white grape jobs follow one another (1 hour), when a load of red grape skins from a fermentation tank follow a load of white grapes (1.5 hours) or when a load of white grapes follows on a load of red (2 hours). All non-zero setup times are shown in Table B.2.

Furthermore, assume that the arrival times of the jobs are given as  $e = \{5, 0, 3, 2, 0.5, 0, 2\}$ . Even though the last job does not really arrive at the cellar, it is assigned an earliest starting time. Now that the environment for the example has been described and all relevant parameters have been defined, various objective functions should be considered before the example problem may be solved. ■

### 5.2.2 Objective function

In Chapter 3 possible objective functions for scheduling problems were considered. The most commonly used objective function is the makespan and is applied to minimize the longest completion time of any job. Since the handling of red grapes ends at the start of primary fermentation, the starting time of such tasks is minimized by including an inequality set of the form

$$t_{jk_j} \leq C, \quad j = 1, \dots, r.$$

All other jobs require processing up to the point where the last task leaves the active cellar. Therefore, inequality constraints of the form

$$f_{jk_j} \leq C, \quad j = r + 1, \dots, n$$

are incorporated when the makespan serves as the objective function to be minimized and is then subject to the constraints described in §5.2.1.

**Example 5.2 (Example 5.1 continued)** *Lingo 11.0* [81] may be applied in order to solve the scheduling problem described in Example 5.1 with the makespan serving as the objective function to be minimized. The Gantt chart representation of the resulting schedule is shown in Figure 5.3. In addition to the normally assigned tasks (indicated by the solid coloured areas), setup times are also indicated by white rectangles. Furthermore, the striped areas represent time that was unnecessarily allocated to the specific task as a result of Equation (5.1). The ending time is not defined as the starting time plus allowed processing time, but is rather bounded from below by it. For example, the schedule states that  $t_{71} = 2$  and  $f_{71} = 11.2$ , but the time required for processing of  $T_{71}$  on  $P_{10}$  is, in fact, presented as 5 hours in Table 5.5. Therefore, during the first 4.2 hours the task  $T_{71}$  is still assigned to  $P_{10}$ , but the actual processing only needs to start after 4.2 hours. Even though this is not ideal, the solution is completely feasible, since it is possible for the grapes to await processing in most processors. For the specific example of  $T_{71}$ , the early assignment makes no difference whatsoever, since the red wine is already in  $P_{10}$  even before  $t = 0$ . The makespan determined by this schedule refers to the ending time of  $T_{7,2}$  and is 14.2 hours. ■

When the adapted makespan is used as objective function to solve the example problem of Example 5.1, it seems that time is often allocated unnecessarily. Take for example  $T_{1,1}$  where an additional 9 hours is assigned. This occurs due to the fact that only the ending times of

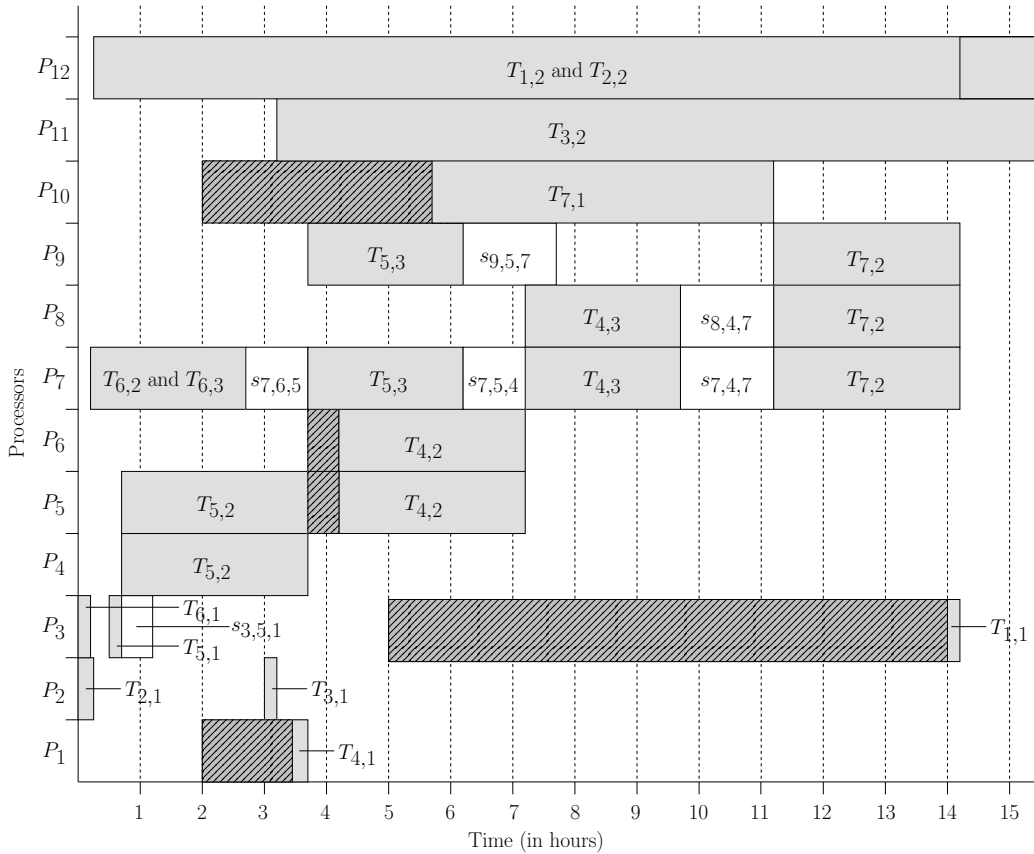


Figure 5.3: Gantt chart representing a solution to the scheduling problem of Example 5.1 when the makespan is used as objective function. The coloured areas indicate time assigned to the processing of a task, the striped areas are unnecessarily assigned and the white rectangles indicate setup times.

jobs ending at the presses are considered. The unnecessary assignment of  $T_{1,1}$  has no influence on the objective function since its successor task,  $T_{1,2}$ , is not included when determining the makespan. However, simply including such tasks in this objective function will only worsen the effect, since the makespan will most definitely be the ending time of one of the fermentation processes. Resulting in a makespan of 5 to 7 days, serving as the upper bound on Equation (5.1). In this case, even less focus is on the earliest ending time of the presses, which is the busiest part of the active cellar. However, this schedule determines the earliest possible ending time, which is all that is required in the end.

Another commonly used objective function is the total weighted completion time, as defined in Chapter 3. This usually gives an indication of the total holding or inventory costs incurred by the schedule. This objective function (using equal weights) may be adopted by incorporating the constraint

$$\sum_{j=1}^r t_{jk_j} + \sum_{j=r+1}^n f_{jk_j} \leq C$$

into the existing constraints described in §5.2.1 requiring that  $C$  be minimized. If a certain job requires immediate processing, it is possible to assign weights to the different jobs so that the completion time of the job contributes more to the objective function.

**Example 5.3 (Example 5.1 continued)** *The example problem described in Example 5.1 may be solved with **Lingo 11.0** adopting the total completion time (with equal weights) as objective function. The Gantt chart representation of the resulting schedule is shown in Figure 5.4. The solid coloured bars indicate the required processing time of the specified task on the relevant processor, while striped areas indicate time allocated unnecessarily. Finally, setup times are indicated by white rectangles. This schedule also results in an optimal makespan of 14.2, but with unnecessarily assigned time limited to only  $T_{j,1}$ . Furthermore, this schedule avoids setup times at the tipping bins, as opposed to the schedule determined by the makespan as objective function.* ■

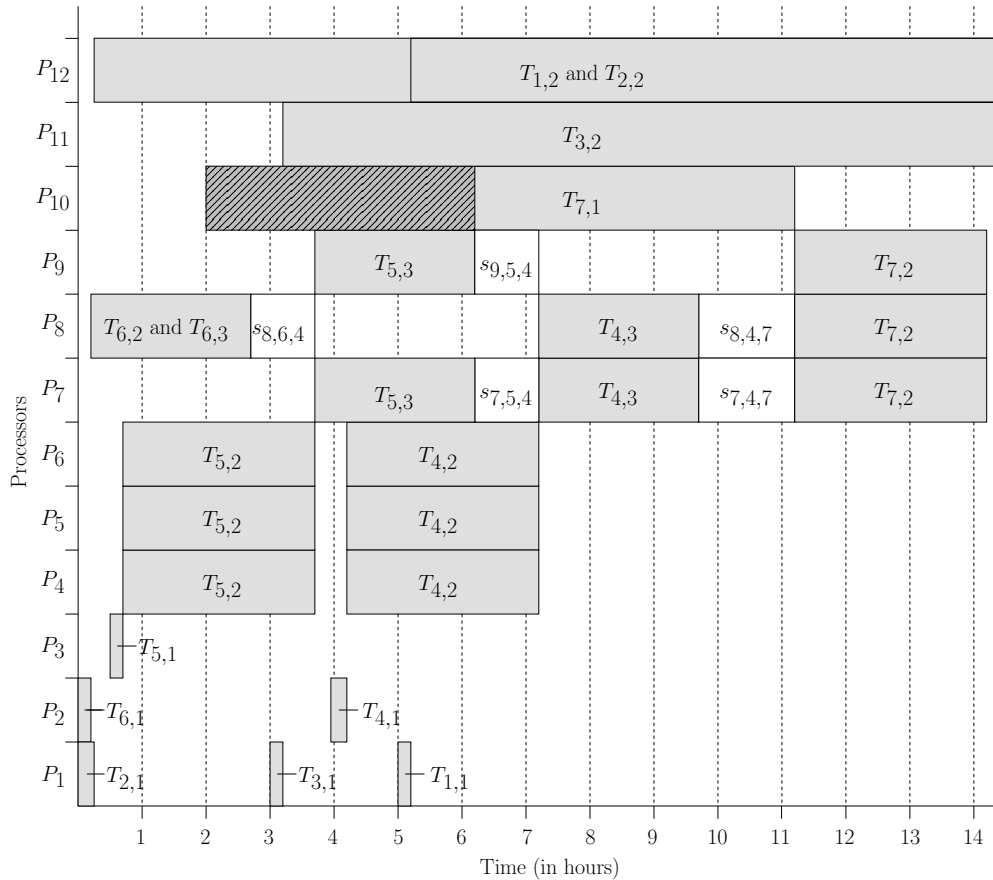


Figure 5.4: Gantt chart representing a solution to the scheduling problem of Example 5.1 when the total completion time is used as objective function. The coloured areas indicate time assigned to the processing of a task, the striped areas are unnecessarily assigned and the white rectangles indicate setup times.

Both the objective functions resulted in a schedule with a makespan of 14.2 hours. However, the distribution of the assignments when using the total completion time as objective function resulted in a more balanced schedule. Furthermore, the solution expressed in Figure 5.3 required 1 minute and 5 seconds of running time when solved with **Lingo 11.0**, whereas the second schedule, expressed in Figure 5.4, required only 30 seconds. The **Lingo 11.0** model is included in Appendix B.2 with the objective function taken as the total weighted completion time. The makespan is referred to as the completion time from hereon, not to be confused with the ending time of a task.

## 5.3 Chapter overview

The purpose of this chapter is to supply the necessary mathematical framework in order to devise a mathematical programming model with which to find an exact solution to the cellar scheduling problem. In §5.1 the cellar workspace is defined mathematically according to the job and processor characteristics. Further parameters required to create the mathematical programming model are also defined.

The model formulation is then fully explained in §5.2 and two objective functions are considered. To illustrate the working of the mathematical programming model, a fictitious cellar is considered and the optimal solution to the cellar scheduling problem is found according to the two objective functions considered.



---

---

## CHAPTER 6

---

# Tabu Search solution of the cellar scheduling problem

### Contents

6.1	The initial solution . . . . .	95
6.1.1	Assignment of jobs to the tipping bins . . . . .	95
6.1.2	Further assignment of Type I jobs . . . . .	96
6.1.3	Further assignment of Type II jobs . . . . .	99
6.1.4	Further assignment of Type III jobs . . . . .	100
6.2	Evaluating a solution . . . . .	102
6.2.1	The cellar packing algorithm . . . . .	102
6.2.2	Evaluating an assignment to the red fermentation tanks . . . . .	107
6.3	Generating candidate moves and selecting the best move . . . . .	109
6.3.1	The general ejection chain move and further communal move aspects . . . . .	110
6.3.2	Move Type A: Tipping bins . . . . .	115
6.3.3	Move Type B: separators . . . . .	116
6.3.4	Move Type C: presses . . . . .	117
6.3.5	Move Type D: separator or press assignment of $T_{j2}$ . . . . .	118
6.3.6	Move Type E: red fermentation tanks . . . . .	119
6.4	Solving the cellar scheduling problem with a tabu search . . . . .	121
6.5	Chapter overview . . . . .	127

The aim of this study is to create a good schedule for the harvesting of grapes based on the expected volume, quality and optimal ripeness of grapes, also keeping in mind available cellar space. Therefore, the eventual goal of the cellar scheduling problem is not to create an actual schedule for activities within the cellar, but rather to use it as a means of evaluating the possible harvesting schedule, thereby including the aspect of available cellar space and time in the evaluation of a suggested harvesting schedule. Certain constraints in the cellar dictate when vineyard blocks are scheduled for harvesting. These constraints include the availability of machines and their capacities. Solving the cellar scheduling problem in Chapter 5, using **Lingo 11.0**, requires at least 30 seconds of processing time and may take up to 15 minutes. A large number of possible scenarios are considered which implies that a significant number of possible solutions may be visited. If the evaluation of a solution of a simple example problem requires

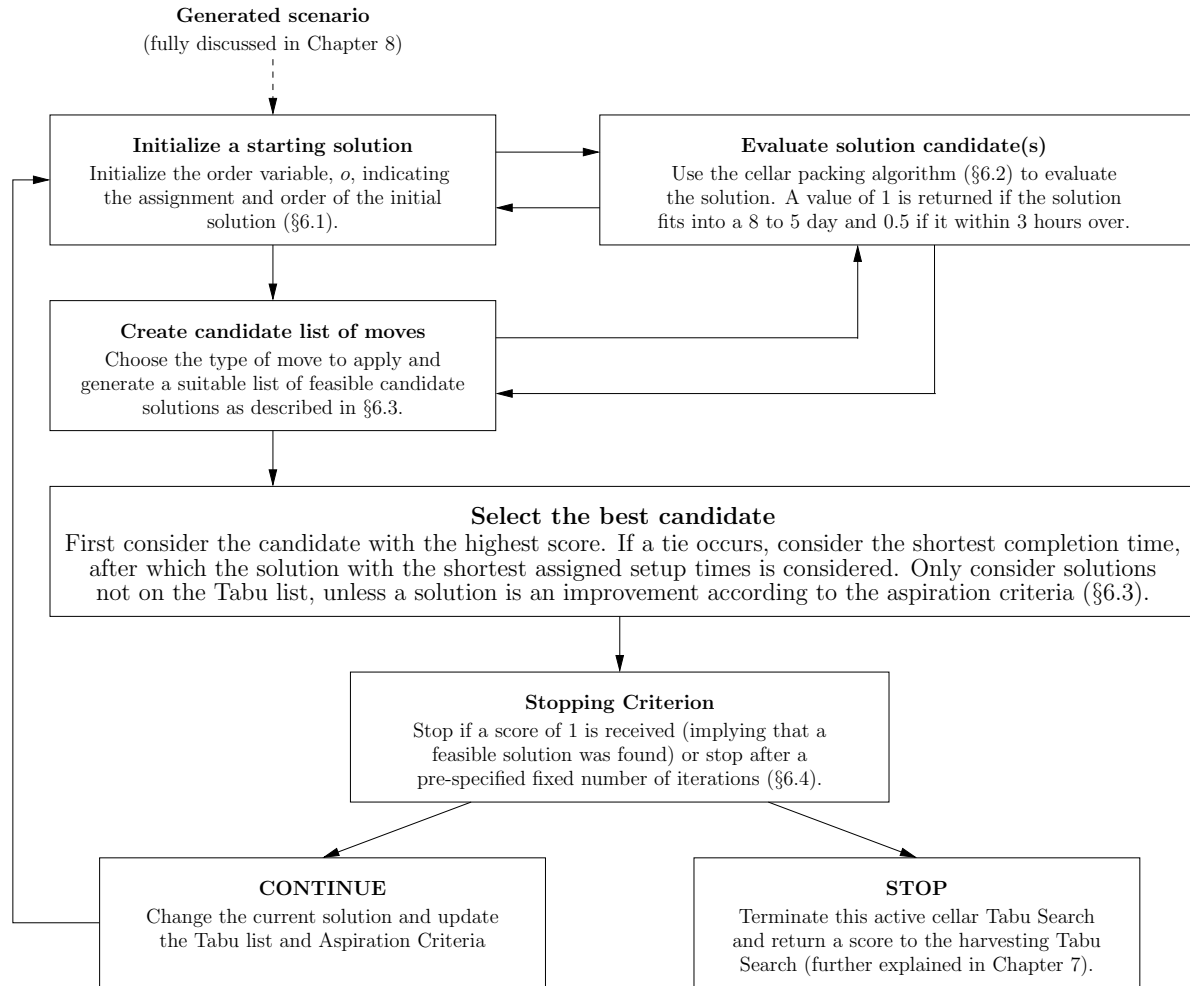


Figure 6.1: An outline of the Tabu Search applied to the active cellar scheduling problem.

30 seconds when solved exactly, solving the real life harvest scheduling problem is expected to be an extremely lengthy process. Therefore, a heuristic approach is now considered.

The application of the cellar scheduling problem to evaluate changes in the harvesting schedule, is fully explained in Chapter 7. The goal of this chapter is to apply a Tabu Search to the cellar scheduling problem. The output of the Tabu Search will not result in a cellar schedule as mentioned above, but will rather be an indication of whether or not a feasible production schedule may be found for the specific harvesting day. A solution is feasible if it adheres to the constraint set defined in §5.2.1 and if the solution may be carried out during the business hours of the cellar.

A detailed description of the Tabu Search methodology was given in §3.4. The Tabu Search application to the active cellar scheduling problem is illustrated in Figure 6.1 and the different aspects seen in this figure, are described in §§6.1–6.4. However, the approach discussed in this chapter should be considered as the generic approach. Each cellar for which the cellar scheduling problem will be attempted to solve using the tabu search developed in this chapter, might require some minor changes to the tabu search method in order to achieve the best results. The adaptation of this proposed active cellar scheduling solution method to best suit the requirements of Wamakersvallei winery, is considered in Chapter 8.



## 6.1 The initial solution

Creating a sensible initial solution based on problem specific characteristics can have a significant influence on the success of a Tabu Search. Therefore, a great deal of effort has gone into ensuring such an initial solution for this specific Tabu Search application. Each job type has specific requirements on the processors. In order to easily distinguish between the three job types, the red grapes scheduled to arrive at the cellar (previously described as the first job of Table 5.1) are from here on referred to as Type I jobs. Type II jobs refer to the white grapes arriving at the cellar (described in Table 5.2) and Type III to the red grape juice and skins that require pressing after primary fermentation (described in Table 5.1 as the Second job).

The ordering and assignment of the tasks to the relevant processors for the initial solution is described in four parts. The first part is concerned with the assignment of both Type I and II jobs to the tipping bins (not considering assignments to any other processor types). The second part deals with the assignment of further tasks associated with the processing of Type I jobs (as described in Table 5.1). The third part then concerns the further assignment of Type II jobs (described in Table 5.2) and the last part the further assignment of Type III jobs (also described in Table 5.1).

The jobs are considered in their order of estimated arrival times,  $e_j$ , at the cellar. Even though Type III jobs do not really arrive at the cellar, they are assigned an earliest starting time. Since it may take approximately 5 hours to empty the fermentation tank and get the grapes to the press, all jobs of this type are penalized with a further 5 hours when the ordered list,  $L$ , of jobs is calculated.

Further variables are required in addition to variables used in §5.2. When referring to the *assigned weight*, denoted by  $a$ , the maximum volume (measured in tonnes) possibly already assigned to a machine is implied. Therefore, a job has been *fully assigned* if its entire weight has been assigned. The order in which jobs are processed is indicated by the matrix denoted  $\mathbf{o}$ . The entry  $\mathbf{o}(i, p)$  is  $j$  only if job  $J_j$  has been assigned to processor  $P_i$  in position  $p$ . Job  $J_0$  is always assigned to position  $p = 0$ .

### 6.1.1 Assignment of jobs to the tipping bins

Jobs are assigned to the tipping bins depending on the number of Type I or Type II jobs received. If only Type I (Type II) jobs are scheduled for intake at the cellar, then all three tipping bins are made available to the reception of Type I (Type II) jobs. If a comparable number of Type I and II jobs are scheduled, one tipping bin receives Type I jobs during the first half of the ordered jobs and Type II from the second half. If there is a majority of Type I (Type II) jobs, two of the three tipping bins are used for the processing of Type I (Type II) jobs. The exact distributions are listed in Table 6.1.

The ordered list of jobs is traversed and jobs are assigned to the tipping bins in that order. Algorithm 6.1 is used for this assignment and is described further with Example 6.1.

**Example 6.1** Consider the same fictitious cellar as in Figure 5.2 with the processing and setup times as listed in Tables 5.5 and B.2 respectively. Also, consider the same situation in the cellar in terms of content and volume of the red fermentation tanks (Table 5.3) and the same set of expected jobs (Table 5.4). In addition to these specifics, take the estimated time of arrival (in hours) for jobs  $J_1, \dots, J_7$  to be  $e = \{5, 0, 3, 2, 0.5, 0, 2\}$ . This results in an ordered list of jobs of  $L = \{J_0, J_2, J_6, J_5, J_4, J_3, J_1, J_7\}$  with  $J_7$  being a Type III job and therefore penalized with 5

State	% Type I jobs	% Type II jobs	Tipping bin distribution		
			I	I and II	II
1	I = 0	II = 100	—	—	$P_1, P_2, P_3$
2	$0 < I \leq 40$	$60 \leq II < 100$	$P_1$	—	$P_2, P_3$
3	$40 < I \leq 60$	$40 \leq II < 60$	$P_1$	$P_2$	$P_3$
4	$60 < I < 100$	$0 < II < 40$	$P_1, P_2$	—	$P_3$
5	I = 100	II = 0	$P_1, P_2, P_3$	—	—

Table 6.1: The distribution of assigning Type I or Type II jobs between the three tipping bins. Here % Type I (Type II) jobs refers to its percentage of the total number of Type I and Type II jobs (thus excluding Type III jobs and the dummy job  $J_0$ ).

hours<sup>1</sup>. Since there is a comparable number of Type I and Type II jobs, State 3 of Table 6.1 is applicable. Therefore, the second tipping bin ( $P_2$ ) may receive white grapes during the first part of the ordered list and red during the second part. The first job of list  $L$ ,  $J_2$ , is considered and assigned to  $P_1$ . The second job,  $J_6$ , is of Type II and since  $2 < \frac{n}{2}$ , it is assigned to tipping bin  $P_2$ . The next job, job  $J_5$ , is also of Type II and is therefore assigned to the next available Type II tipping bin,  $P_3$ . After the assignment,  $i_w$  is increased to 4. Therefore when the next Type II job,  $L_4 = J_4$ , is considered for assignment it is also assigned to  $P_3$  since  $4 > \frac{n}{2}$ . Tipping bin  $P_2$  is now available for Type I jobs. This process is continued as described in Algorithm 6.1 and results in an assignment of  $J_2$  and  $J_1$  to tipping bin  $P_1$  (in that order), while tipping bin  $P_2$  receives job  $J_6$ , then  $J_3$  and tipping bin  $P_3$  is assigned jobs  $J_5$  and  $J_4$ . ■

After assigning the jobs to tipping bins, the ordered list is again traversed, starting at  $L_1$ . The relevant algorithm is chosen depending on the job type. The next job in the ordered list is then considered. The circumstances and algorithms for the different job types are now described<sup>2</sup>.

### 6.1.2 Further assignment of Type I jobs

As seen in Table 5.1, Type I jobs are only assigned to a tipping bin and then to suitable fermentation tanks. The process of assigning the jobs and keeping track of the current tank volumes is given in pseudo-code form in Algorithm 6.2.

In this algorithm the variable *starting tank* is used to keep track of the current tank when the algorithm is called<sup>3</sup>. The first time when Algorithm 6.2 is called in the traversal of list  $L$ , the value  $i_{r,f}$  is set to  $m_3$ . The currently assigned weight,  $a$ , is set to 0 with every call of the algorithm. The volume of a tank  $P_{i_{r,f}}$  is denoted by  $v_{i_{r,f}}$ . Changes to this value only changes the actual volume of the tank once a new solution has been chosen as the final solution — then the new volumes are carried over to the next day for further assignments. When a job is assigned to a tank, the current volume is either increased until the capacity of the tank is reached or by the actual volume of the job (line 8). This depends on the currently unassigned volume of the job.

<sup>1</sup>In the ordered list (as well as in the order matrices), job  $J_0$  is always in position 0 of the list (or row). This dummy job is required for later use.

<sup>2</sup>All of the algorithms described in the remainder of this section forms part of the larger traversal of the list of ordered jobs. Therefore, uninitiated parameters are assumed to be global to the list traversal.

<sup>3</sup>This should not be confused with the the physical first red fermentation tank of the cellar denoted by  $P_{m_3}$  as defined in §5.1.2.

**Algorithm 6.1:** Initial active cellar assignment for the tipping bins**Input:** A list of jobs and their specifications**Output:** An updated version of the assignment matrix  $o$  in terms of the tipping bins

```

14 if States 1 or 5 occur then
15     for  $z = 1$  to  $n$  do
16          $j \leftarrow L_z$ ;
17          $i \leftarrow z \bmod 3 + 1$ ;
18          $p_i \leftarrow \lfloor \frac{z}{3} \rfloor + 1$ ;
19          $o(i, p_i) \leftarrow j$ ;
20     end
21 else if State 2 occurs then
22      $i_r \leftarrow 1$ ;  $i_w \leftarrow 3$ ;
23     for  $z = 1$  to  $n$  do
24          $j \leftarrow L_z$ ;
25         switch Type of job  $j$  do
26             case Type I
27                 if  $i_r = 3$  then  $i_r \leftarrow 1$ ;
28                  $o(i_r, p_{i_r}) \leftarrow j$ ;
29                  $i_r \leftarrow i_r + 1$ ;
30                  $p_{i_r} \leftarrow p_{i_r} + 1$ ;
31             case Type II
32                  $o(i_w, p_{i_w}) \leftarrow j$ ;
33                  $p_{i_w} \leftarrow p_{i_w} + 1$ ;
34         end
35     end
36 else if State 3 occurs then
37      $i_r \leftarrow 1$ ;  $i_w \leftarrow 2$ ;
38     for  $z = 1$  to  $n$  do
39          $j \leftarrow L_z$ ;
40         switch Type of job  $j$  do
41             case Type I
42                 if  $z \leq \frac{n}{2}$  then if  $i_r = 2$  then  $i_r \leftarrow 2$ ;
43                 if  $i_r = 3$  then  $i_r \leftarrow 1$ ;
44                  $o(i_r, p_{i_r}) \leftarrow j$ ;
45                  $i_r \leftarrow i_r + 1$ ;
46                  $p_{i_r} \leftarrow p_{i_r} + 1$ ;
47             case Type II
48                 if  $i_w = 4$  then
49                     if  $z \leq \frac{n}{2}$  then
50                          $i_w \leftarrow 2$ ;
51                     else  $i_w \leftarrow 3$ 
52                 end
53                  $o(i_w, p_{i_w}) \leftarrow j$ ;
54                  $i_w \leftarrow i_w + 1$ ;
55                  $p_{i_w} \leftarrow p_{i_w} + 1$ ;
56         end
57     end

```

58 Continues on next page;

---

```

59 else if State 4 occurs then
60    $i_r \leftarrow 1; i_w \leftarrow 2;$ 
61   for  $z = 1$  to  $n$  do
62      $j \leftarrow L_z;$ 
63     switch Type of job j do
64       case Type I
65         if  $i_r = 3$  then  $i_r \leftarrow 1;$ 
66          $\mathbf{o}(i_r, p_{i_r}) \leftarrow j;$ 
67          $p_{i_r} \leftarrow p_{i_r} + 1;$ 
68       case Type II
69         if  $i_r = 4$  then  $i_r \leftarrow 2;$ 
70          $\mathbf{o}(i_w, p_{i_w}) \leftarrow j;$ 
71          $i_w \leftarrow i_w + 1;$ 
72          $p_{i_w} \leftarrow p_{i_w} + 1;$ 
73     end
74   end
75 end

```

---

As with the mathematical programming model, an assignment of job  $J_j$  to machine  $P_{i_{rf}}$  is allowed when it is an available allowable processor. The quality of the grapes already contained in the tank should also match the grape quality of the current job (line 4). Once all the possible processors for the job have been considered and the job has not been fully assigned, the excess weight is assigned to the variable  $u$  referring to all unassigned weight. A solution is only feasible if the unassigned weight is 0.

---



---

**Algorithm 6.2:** Further initial active cellar assignment for Type I jobs

---

**Input:** The current job  $j$ , the current red fermentation tank  $i_{rf}$  and a list of positions  $p$   
**Output:** The updated version of the assignment matrix  $\mathbf{o}$  and the total weight of unassigned jobs  $u$

```

1  $a \leftarrow 0;$ 
2 starting tank  $\leftarrow i_{rf};$ 
3 while  $a < w_j$  do
4   if Assignment  $a_{i_{rf},j} = 1$  is allowed then
5      $\mathbf{o}(i_{rf}, p_{i_{rf}}) \leftarrow j;$ 
6      $p_{i_{rf}} \leftarrow p_{i_{rf}} + 1;$ 
7      $a \leftarrow a + (c_{i_{rf}} - v_{i_{rf}});$ 
8      $v_{i_{rf}} \leftarrow v_{i_{rf}} + \min \{c_{i_{rf}} - v_{i_{rf}}, w_j - a + c_{i_{rf}} - v_{i_{rf}}\};$ 
9      $i_{rf} \leftarrow i_{rf} + 1;$ 
10  else
11     $i_{rf} \leftarrow i_{rf} + 1;$ 
12  end
13  if  $i_{rf} = m + 1$  then  $i_{rf} \leftarrow m_3;$ 
14  if (starting tank) and ( $w_j - a > 0$ ) then
15     $u \leftarrow u + w_j - a;$ 
16     $a \leftarrow w_j;$ 
17  end
18 end

```

---

**Example 6.2 (Example 6.1 continued)** *At the start of the first iteration,  $i_{rf} = 10$ . The first job of list  $L$ , job  $J_2$ , is of Type I. From Table 5.4 it may be seen that job  $J_1$  refers to 40 Tonnes of Cabernet Sauvignon. The only available tank that is allowable for  $J_1$  is  $P_{12}$ . Job  $J_1$  is fully assigned to  $P_{12}$  with its current volume,  $v_{12}$ , increased from 0 to 40 tonnes. Algorithm 6.2 is repeated for the remainder of the Type I jobs. Job  $J_3$  is assigned to  $P_{11}$  and  $J_2$  is added to  $J_1$  in  $P_{12}$ . ■*

### 6.1.3 Further assignment of Type II jobs

Algorithm 6.3 is used for the assignment of Type II jobs. When a Type II job is reached during the traversal of the list  $L$ , assignment to a separator is first considered. Even though it is always faster to skip processing at the separators and separate skins from juice at the presses, the separators still play an important role in the cellar. The first is when some additional juice and skin contact is required (this is not possible in a press). When the grape intake is at its peak, the separator may be used to ‘store’ the grapes (while starting the separating process at

---



---

#### Algorithm 6.3: Further initial active cellar assignment for Type II jobs

---

**Input:** The current job  $j$ , the current separator  $i_s$  and press  $i_p$  and a list of positions  $p$   
**Output:** The updated version of the assignment matrix  $\mathbf{o}$

```

1 if Assigning job  $J_j$  to a separator is allowed then
2    $a \leftarrow 0$ ;
3   while  $a < w_j$  do
4      $\mathbf{o}(i_s, p_{i_s}) \leftarrow j$ ;
5      $a \leftarrow a + c_{i_s}$ ;
6      $p_{i_s} \leftarrow p_{i_s} + 1$ ;
7      $i_s \leftarrow i_s + 1$ ;
8     if  $i_s = m_2$  then
9        $i_s \leftarrow m_1$ ;
10    end
11  end
12 end
13  $a \leftarrow 0$ ;
14 while  $a < w_j$  do
15   if  $\mathbf{o}(i_p, p_{i_p} - 1) \neq j$  then
16      $\mathbf{o}(i_p, p_{i_p}) \leftarrow j$ ;
17      $a \leftarrow a + c_{i_p}$ ;
18      $p_{i_p} \leftarrow p_{i_p} + 1$ ;
19   end
20    $i_p \leftarrow i_p + 1$ ;
21   if  $i_p \geq \text{first red press}$  then
22     if  $a < w_j$  and  $\mathbf{o}(m_2, p_{m_2} - 1) \neq j$  then
23        $i_p \leftarrow m_2$ ;
24     else if  $a \geq w_j$  then
25        $i_p \leftarrow m_2$ ;
26     end
27   end
28 end

```

---

the same time) since bottlenecks always occur at the presses. Assigning a job  $J_j$  to a separator is therefore allowable (line 1) if it is specified that job  $J_j$  must be assigned to a separator. An additional rule used for the further assignment of Type II jobs in the initial solution, allows additional Type II jobs to be assigned to the separators if less than 25 % of the Type II jobs must be assigned to a separator. A further limitation in creating the initial solution is that no more than 20 % of Type II jobs not intentionally assigned to the separators, may be assigned to a separator. The assignment to separators is considered in lines 1–12 of the algorithm.

The set of presses in a cellar is partitioned into two smaller sets, one for the pressing of Type II jobs and one for the pressing of Type III jobs. The number of reserved Type III presses is determined by the number of Type III jobs as well as by the number of red fermentation tanks that may be drained during the timespan of one day (this is further discussed in the following section). The first press in the set of Type III presses is referred to as the *first red press*. However, a Type II job is allowed to be processed on a Type III press if it is too large to fit into the designated Type II presses. This assignment occurs in lines 13–28 of Algorithm 6.3. Example 6.2 is now continued to illustrate the further assignment of Type II jobs.

**Example 6.3 (Example 6.2 continued)** *At the start of the first call of Algorithm 6.3,  $i_s = 4$  and  $i_p = 7$ . The first red press is processor  $P_9$ . Since two of the three Type II jobs must be processed on the separators (more than 25% of the Type II jobs), there is no need to assign any Type II jobs other than  $J_4$  and  $J_5$  to the separators. The first Type II job in the list  $L$  is job  $J_6$ . Since assigning this job to a separator is not allowed, the presses are considered next. This is a job of weight  $w = 20$  and is fully assigned to the first available press,  $P_7$ , after which the value of  $i_p$  is updated to 8. The second Type II job in the list  $L$  is job  $J_5$  with  $w = 30$ . This job should be assigned to a separator and is assigned to  $P_4$  and  $P_5$  after which the value of the current separator,  $i_s$ , is updated to 6. Since the current press is  $P_8$ , this press is filled with job  $J_5$  still leaving a further 10 tonnes to be assigned. Since  $P_9$  is considered as a red press in this example, the remainder of  $J_5$  is assigned to  $P_7$ . The last Type II job,  $J_4$ , is assigned to separator  $P_6$  and then to  $P_4$ ; it is also assigned to presses  $P_7$  and  $P_8$ . ■*

#### 6.1.4 Further assignment of Type III jobs

Type III jobs move from the already assigned fermentation tank to a number of presses. As with the case of Type II jobs, it is only allowed to assign Type III jobs to a processor outside of the designated set if there is not sufficient capacity within in the Type III press set. The number of processors set aside for Type III jobs is determined by dividing the number of Type III jobs by the number of tanks that may be drained in one day and taking the floor of this number after adding 1. With the assignment of Type III jobs, the list of suitable processors are traversed from the last to the first — therefore the decrease in press number in line 8 of Algorithm 6.4. This assignment method is further discussed in Example 6.4.

**Example 6.4 (Example 6.3 continued)** *During the first iteration (and only iteration for this example) the first red press is set as  $i_{rp} = 9$ . Since there is only one Type III job, only one press is set aside for the use of Type III jobs. However, the pressing of  $J_7$  requires a capacity of 50 Tonnes. Therefore the list of presses is traversed from the last to the first and  $J_7$  is only fully assigned once it has been assigned to all presses. The final corresponding entries of the matrix  $\mathbf{o}$  are given in Table 6.2. ■*

**Algorithm 6.4:** Further initial active cellar assignment for Type III jobs

---

**Input:** The current job  $j$ , current red press  $i_{rp}$ , list of positions  $p$   
**Output:** The updated version of the assignment matrix  $o$

```

1  $a \leftarrow 0$ ;
2 while  $a < w_j$  do
3   if  $o(i_{rp}, p_{i_{rp}-1}) \neq j$  then
4      $o(i_{rp}, p_{i_{rp}}) \leftarrow j$ ;
5      $a \leftarrow a + c_{i_{rp}}$ ;
6      $p_{i_{rp}} \leftarrow p_{i_{rp}} + 1$ ;
7   end
8    $i_{rp} \leftarrow i_{rp} - 1$ ;
9   if  $i_{rp} < \text{first Red press}$  then
10    if  $a < w_j$  and  $o(m_3 - 1, p_{m_3-1} - 1) \neq j$  then
11       $i_{rp} \leftarrow m_3 - 1$ ;
12    else if  $a \geq w_j$  then
13       $i_{rp} \leftarrow \text{first Red press}$ ;
14    end
15  end
16  for  $t = m_3$  to  $m$  do
17    if  $\mu_{tj} = 1$  then
18       $o(t, p_t) \leftarrow j$ ;
19      break;
20    end
21  end
22 end

```

---

In order to evaluate the solution suggested by order matrix  $o$ , the cellar packing algorithm is used. Furthermore, a method to evaluate the assignment of jobs to the red fermentation tanks is also created. These evaluation techniques are discussed in the following section.

Machine	Job	Job	Job	Job	Job
$P_1$	0	2	1	—	—
$P_2$	0	6	3	—	—
$P_3$	0	5	4	—	—
$P_4$	0	5	4	—	—
$P_5$	0	5	—	—	—
$P_6$	0	4	—	—	—
$P_7$	0	6	5	4	7
$P_8$	0	5	4	7	—
$P_9$	0	7	—	—	—
$P_{10}$	0	7	—	—	—
$P_{11}$	0	3	—	—	—
$P_{12}$	0	1	2	—	—
$P_{13}$	0	—	—	—	—
$P_{14}$	0	—	—	—	—
$P_{15}$	0	—	—	—	—

Table 6.2: The final order matrix  $o$  in Example 6.3 in table form.

## 6.2 Evaluating a solution

A suggested solution should be evaluated on two levels. The first concerns the processing time required to execute the move and whether it will fit into one business day of the cellar, *i.e.* minimizing the makespan of the schedule. This is achieved by applying the cellar packing algorithm, discussed in §6.2.1. However, this algorithm does not take into account the assignments to the red fermentation times, since such an assignment has no influence on the makespan of the suggested schedule. Therefore a further evaluation of the suggested solution is necessary, specifically focusing on the assignment to the red fermentation tanks. This approach is discussed in §6.2.2.

### 6.2.1 The cellar packing algorithm

The goal of the cellar packing algorithm (Algorithm 6.5) is to evaluate a candidate solution by determining its completion time. The *completion time* refers to the ending time of the last task on any processor other than a red fermentation tank. For the active cellar, all jobs that do not finish at the fermentation tanks, finish at the presses. Therefore, the completion time is, in fact, the ending time of the last task performed on a press for the day under consideration. The order in which tasks are processed on machines, as well as the assignment of tasks to machines, is determined by the Tabu Search. To find the completion time, the tasks are all packed into feasible time intervals as explained in this section; this process is referred to as the *cellar packing algorithm*.

In order to identify the order in which a job travels between processors, an additional matrix,  $\mathbf{o}_m$ , is derived from order matrix  $\mathbf{o}$ . The entry  $\mathbf{o}(i, p)$  **refers to the job** processed on  $P_i$  after processing job  $J_j$  (with  $j = \mathbf{o}(i, p - 1)$ ) on  $P_i$ . Now the entry  $\mathbf{o}_m(j, q)$  **refers to the processor** on which job  $J_j$  is processed after processing on  $P_i$  (with  $i = \mathbf{o}_m(j, q - 1)$ )<sup>4</sup>. The *machine order matrix* derived from the order matrix of Example 6.3 (given in Table 6.2), is

$$\mathbf{o}_m = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & \dots \\ 1 & 12 & -1 & -1 & -1 & -1 & \dots \\ 1 & 12 & -1 & -1 & -1 & -1 & \dots \\ 2 & 11 & -1 & -1 & -1 & -1 & \dots \\ 3 & 4 & 6 & 7 & 8 & -1 & \dots \\ 3 & 4 & 5 & 7 & 8 & -1 & \dots \\ 2 & 7 & -1 & -1 & -1 & -1 & \dots \\ 10 & 9 & -1 & -1 & -1 & -1 & \dots \end{bmatrix}.$$

From this matrix it may, for example, be seen that job  $J_3$  is processed first on  $P_2$  (since  $\mathbf{o}_m(3, 0) = 2$ ) and then on  $P_{11}$  (since  $\mathbf{o}_m(3, 1) = 11$ ). As with the order matrix  $\mathbf{o}$ , open entries are assigned  $-1$ . In the following algorithm an entry  $\mathbf{o}(i, p)$  or  $\mathbf{o}_m(j, q)$  is said to exist if it is not equal to  $-1$  and the indexes are not out of bounds, *i.e.*  $\mathbf{o}_m(1, 10)$  does not exist and neither does  $\mathbf{o}_m(2, 3)$ . Furthermore, the starting and ending times of a job correspond with the order matrix,  $\mathbf{o}$ . Therefore the starting time  $\mathbf{t}(i, p)$  (or ending time  $\mathbf{f}(i, p)$ ) refers to the start (or end) of job  $J_j$  on processor  $P_i$  where  $j = \mathbf{o}(i, p)$ .

The simplified method of evaluating a candidate solution is captured in pseudo-code form in Algorithm 6.5, with further required procedures listed below. In Algorithm 6.5, the variable

<sup>4</sup>Therefore, row  $j$  of the matrix refers to job  $J_j$ , the columns are an indication of order and the entries refer to processors.



$p_{\text{last}}$  (lines 6 and 7) is used to refer to the last position where the corresponding order matrix entry is not yet  $-1$ .

---



---

**Algorithm 6.5:** Evaluating a possible solution
 

---

**Input:** The machine order matrix  $\mathbf{o}_m$   
**Output:** The corresponding completion time

```

1 completion time  $\leftarrow$  0;
2 Initialize times( $\mathbf{o}, \mathbf{t}, \mathbf{f}, m$ );
3 Update with arrival times( $\mathbf{o}, e$ );
4 Apply forward move( $\mathbf{o}, \mathbf{o}_m, \mathbf{t}, \mathbf{f}, m_1$ );
5 for  $i = 1$  to  $m_3 - 1$  do
6   | if  $\mathbf{f}(i, p_{\text{last}}) > \text{complete}$  then
7   |   | completion time  $\leftarrow \mathbf{f}(i, p_{\text{last}})$ ;
8   | end
9 end
```

---

When considering the layout of a Gantt chart, initializing the times may be considered as the process of packing all the jobs in the right order next to their assigned processors. The only constraint considered is that the processing times and setup times be adhered to. This process is outlined in Procedure **Initialize times** and is further explained in Example 6.5.

---



---

**Procedure Initialize times( $\mathbf{o}, \mathbf{t}, \mathbf{f}, m$ )**


---

```

1 for  $i = 1$  to  $m$  do
2   |  $\mathbf{t}(i, o) \leftarrow 0$ ;
3   |  $\mathbf{f}(i, o) \leftarrow 0$ ;
4   |  $p \leftarrow 1$ ;
5   | while  $\mathbf{o}(i, p)$  exist do
6   |   |  $\mathbf{t}(i, p) \leftarrow \mathbf{f}(i, p - 1) + \text{setup time}$ ;
7   |   |  $\mathbf{f}(i, p) \leftarrow \mathbf{t}(i, p) + \text{processing time}$ ;
8   | end
9   |  $p \leftarrow p + 1$ ;
10 end
```

---

**Example 6.5 (Example 6.3 continued)** Consider the initial solution expressed in Table 6.2. For each machine, the starting and ending times of the first job are initialized to 0. Thereafter the corresponding starting time,  $\mathbf{t}(i, p)$ , and ending time,  $\mathbf{f}(i, p)$ , for each existing  $\mathbf{o}(i, p)$  are updated according to its processing times and the required setup times. The initial times returned by Procedure **Initialize times** are shown in the Gantt chart in Figure 6.2. ■

The first step in updating the starting and ending times is to update these times at the tipping bins according to their relevant arrival times. For this purpose, Procedure **Update with arrival times** is introduced.

Starting at the first tipping bin,  $P_1$ , (line 1) and continuing until the last tipping bin  $P_{m_1-1}$ , each job assigned to a tipping bin is considered. If the starting time,  $\mathbf{t}(i, p)$ , of job  $J_j$  (where  $j$  refers to the entry  $\mathbf{o}(i, p)$  in the order matrix) is already as least as large as the estimated time of arrival,  $e_j$ , then no update is necessary (line 5). However, if this is not the case, the starting and ending times of job  $J_j$  should be updated to  $e_j$ . Furthermore, the successors of job  $J_j$  on processor  $P_i$  should also be updated with the same amount of time (referred to as the *offset*

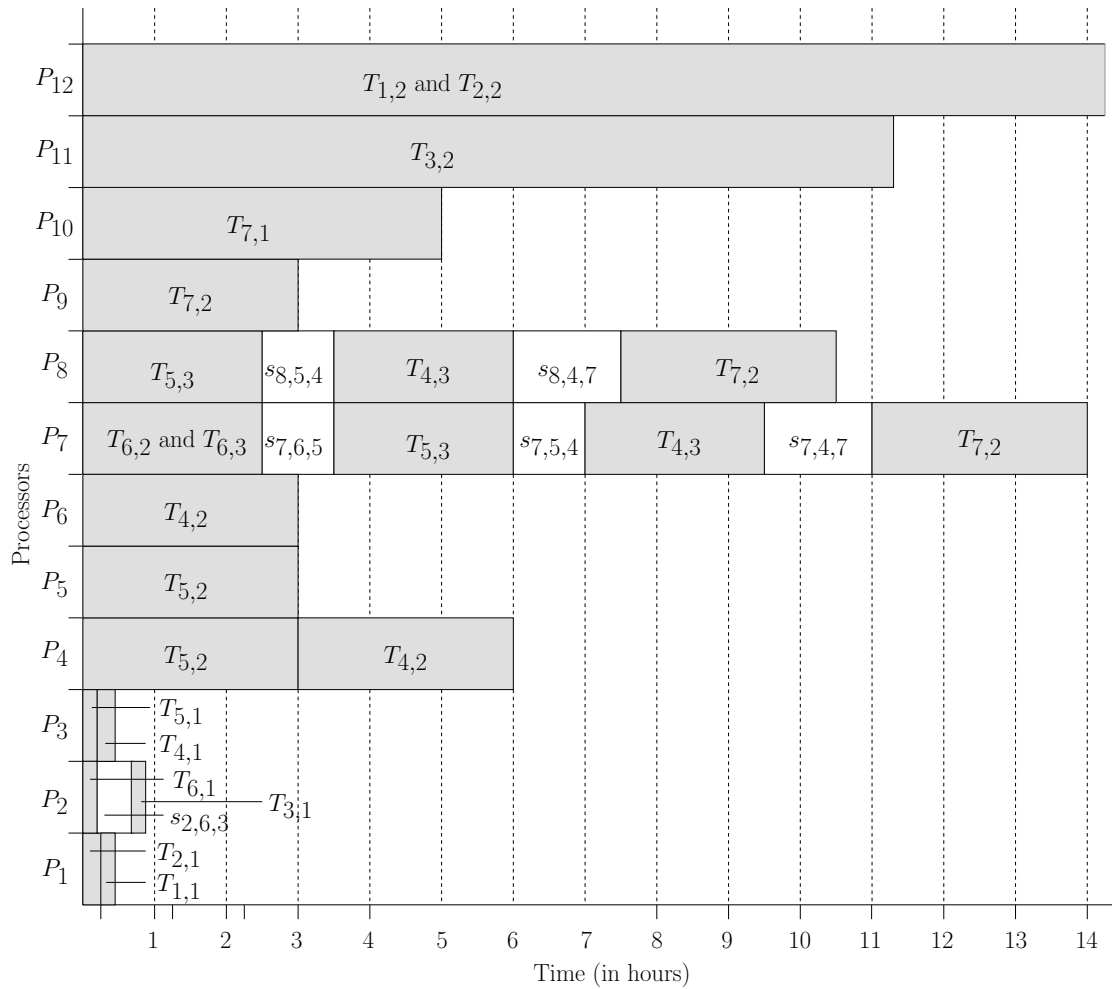


Figure 6.2: The cellar graph for the active cellular situation in Example 6.5 illustrating the initialization of the starting and ending times.

— line 6). This function is named `updateTimes` and is used throughout the algorithms in this section. When Procedure `Update with arrival times` is applied to the situation in Figure 6.2, the new cellar graph in Figure 6.3 is obtained.

---



---

**Procedure Update with arrival times( $o, e$ )**

---

```

1  $i \leftarrow 1$ ;
2 while  $i < m_1$  do
3   for  $p = 1$  to  $p_{last}$  do
4      $j \leftarrow o(i, p)$ ;
5     if  $t(i, p) \leq e_j$  then
6        $offset \leftarrow e_j - t(i, p)$ ;
7       updateTimes( $i, p, offset$ );
8     end
9   end
10   $i \leftarrow i + 1$ ;
11 end

```

---

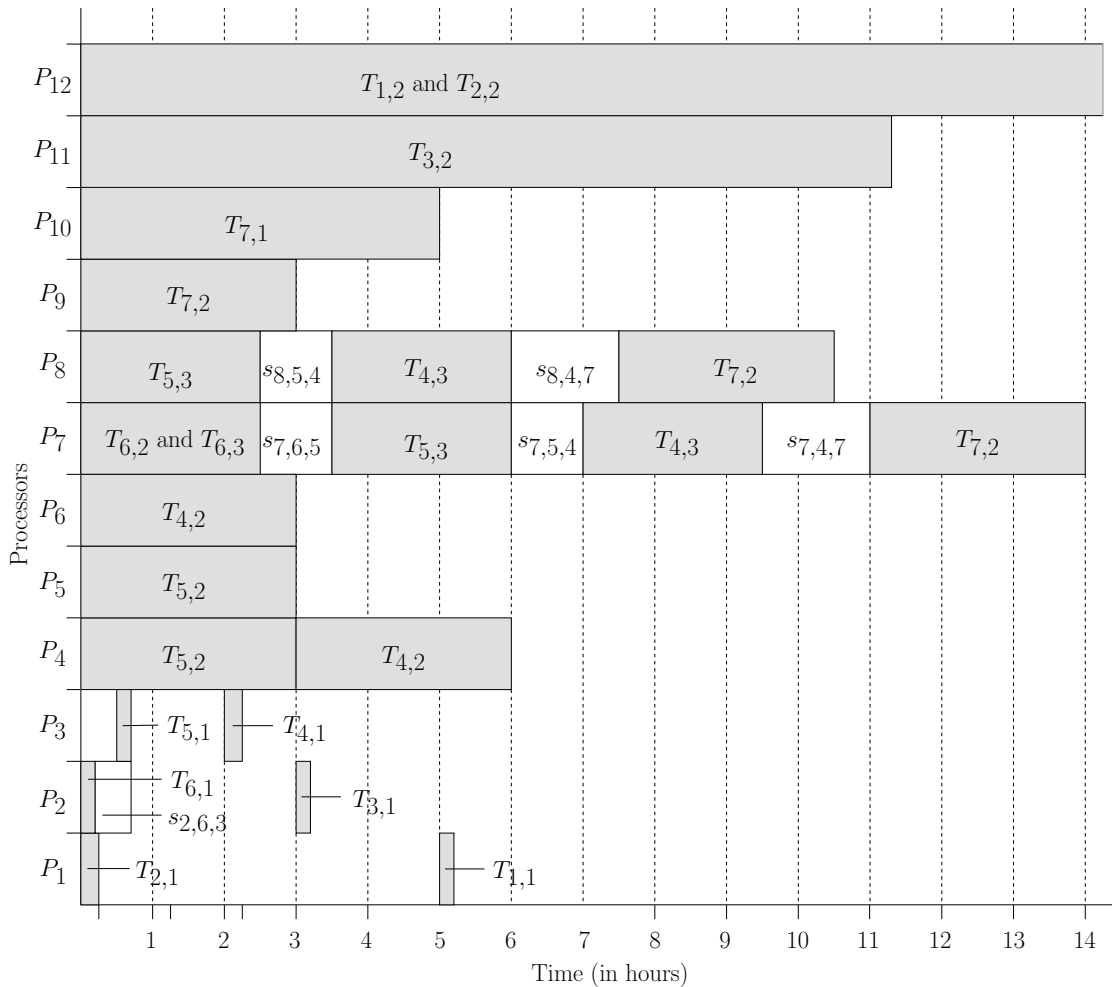


Figure 6.3: The cellular graph for the active cellar from Figure 6.2, after Procedure Updates with arrival times has been applied, with the arrival times from Example 6.1.

The next step in determining the completion time, is to apply a *forward move*. This refers to the update of task times so as to ensure that the starting time of a task,  $T_{j,k}$ , is at least as large as the ending time of its predecessor task,  $T_{j(k-1)}$ . The process is outlined in Procedure Apply forward move.

The variables  $p_o$  and  $p_{o_m}$  are indicators of position when traversing  $o$  and  $o_m$  respectively. In line 4,  $p_o$  is set to 1 since job  $J_0$  is always in position 1 and does not require direct consideration in this algorithm. In line 6,  $p_{o_m}$  is also set to 1. This is done in order to ignore the assignment of jobs to the tipping bins, since their times have already been determined by the application of Procedure Update arrival times. Furthermore, the variable  $p(j)$  is used to indicate the position in which job  $j$  is performed on the relevant machine for this pseudo-code. For example, consider the order matrix of Table 6.2. When the machine considered is  $P_7$ ,  $p(4) = 3$ . Starting at tipping bin  $P_1$ , all the jobs assigned to tipping bins are considered. The outside loop (starting in line 5) therefore determines the job,  $J_j$ , currently under consideration. The second loop (starting in line 9) traverses the machines receiving further tasks of job  $J_j$ . The function `ifSameSetUpdate` determines whether two processors,  $o_m(j, p_{o_m} - 1)$  and  $o_m(j, p_{o_m})$ , used to process tasks of job  $J_j$ , are from the same set. If they are, it implies that it is one task of job  $J_j$

that is assigned to more than one machine. Their times will be updated in order to have such a task performed on parallel processors, starting at the same time on both and the variable *same* is returned as true. If the sets differ, no update is performed at this stage and the value of *same* remains false.

---



---

**Procedure Apply forward move( $\mathbf{o}, \mathbf{o}_m, \mathbf{t}, \mathbf{f}, m_1$ )**


---

```

1   $i \leftarrow 1$ ;
2   $j \leftarrow 0$ ;
3  while  $i < m_1$  do
4       $p_o \leftarrow 1$ ;
5      while  $\mathbf{o}(i, p_o)$  exists do
6           $p_{o_m} \leftarrow 1$ ;
7           $\text{same} \leftarrow \text{false}$ ;
8           $j \leftarrow \mathbf{o}(i, p_o)$ ;
9          while  $\mathbf{o}_m(j, p_{o_m})$  exists do
10              $\text{same} \leftarrow \text{false}$ ;
11              $\text{same} \leftarrow \text{ifSameSetUpdate}(j, p_{o_m}, p_{o_m} - 1)$ ;
12              $i_1 \leftarrow \mathbf{o}_m(j, p_{o_m} - 1)$ ;
13              $i_2 \leftarrow \mathbf{o}_m(j, p_{o_m})$ ;
14             if ( $\text{same} = \text{false}$ ) and ( $\mathbf{t}(i_2, p(j)) < \mathbf{f}(i_1, p(j))$ ) then
15                  $\text{offset} \leftarrow$ 
16                      $\max\{\mathbf{f}(i_1, p(j)) - \mathbf{t}(i_2, p(j)), \mathbf{t}(i_2, p(j)) - \mathbf{f}(i_2, p(j) - 1) + \text{setup time}\}$ ;
17                 updateTimes( $i_2, p(j), \text{offset}$ );
18             end
19              $p_{o_m} \leftarrow p_{o_m} + 1$ ;
20         end
21          $p_o \leftarrow p_o + 1$ ;
22     end
23      $i \leftarrow i + 1$ ;
24 end

```

---

The offset determined in line 15 of the Algorithm, determines the time with which a job should be updated. The offset can be the difference between the starting time of task  $T_{jk}$  and the ending time of its predecessor  $T_{j(k-1)}$ . It can refer to the difference between the end of task  $T_{\ell h}$  that was processed on the same processor  $P_i$  directly before  $T_{jk}$  and the starting time of  $T_{jk}$ . The setup time,  $s_{i\ell j}$ , is then also added to the difference.

Applying the algorithm to the situation in Figure 6.3 results in the cellar graph presented in Figure 6.4 with a completion time of 14.2 hours. The striped blocks still refer to time that was assigned unnecessarily, but also to instances where a task should be moved forward in order for the following task of the same job to follow directly. For example,  $T_{4,1}$  ends at 2.25 hours. However the following task,  $T_{4,2}$ , only starts at 3.7 hours. Therefore, the ending time of task  $T_{4,1}$  should, in fact be 3.7 and the starting time 3.45 hours. These shortcomings may be rectified by applying a similar *backwards move*, but since the completion time is already known, the backwards move is not required.

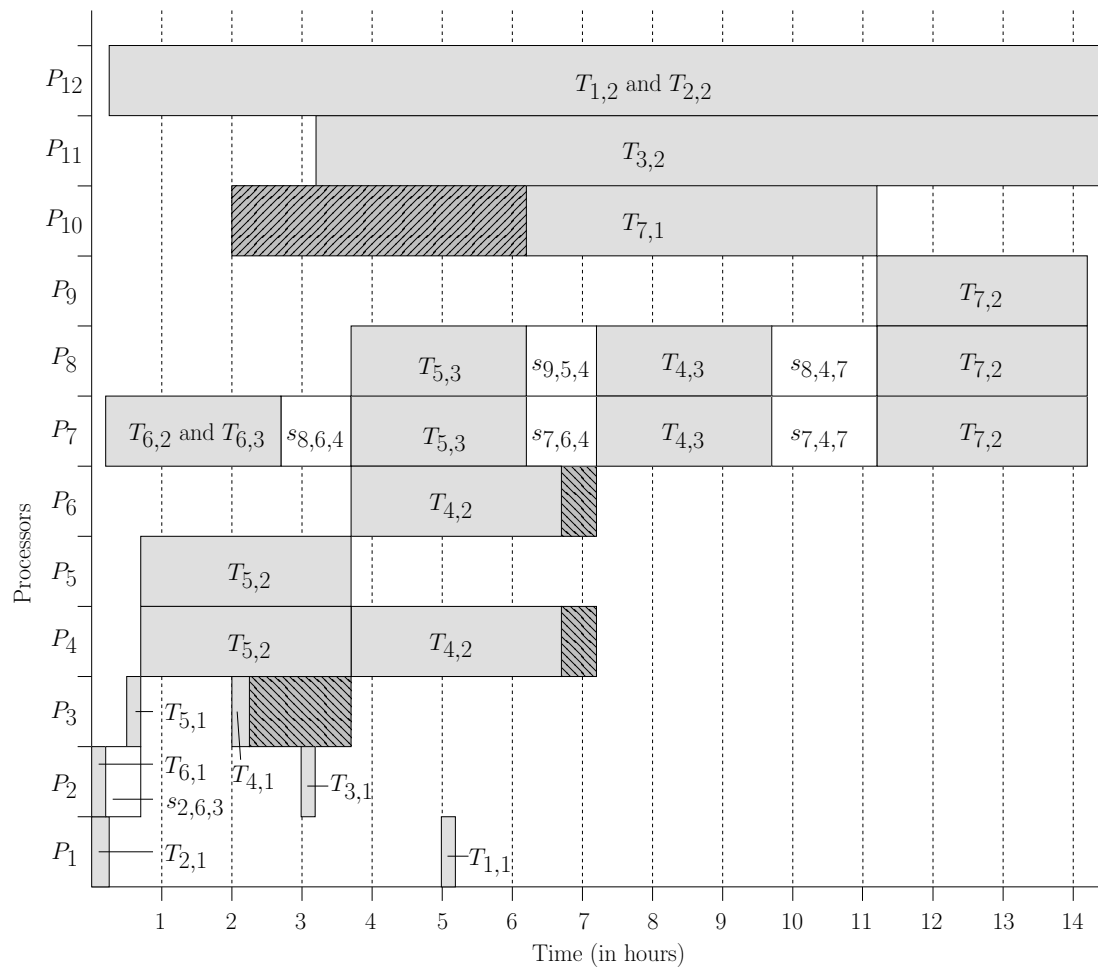


Figure 6.4: The cellular graph for the active cellar, after applying the full packing algorithm to the initial solution generated in Example 6.4, resulting in a completion time of 14.2 hours.

### 6.2.2 Evaluating an assignment to the red fermentation tanks

When assigning jobs to the red fermentation tanks, the order in which these jobs are assigned is not of concern. The jobs assigned to the same tank are mixed and do not follow one another. Furthermore, the fermentation process lasts more than a day. Therefore, processing time is not included in the evaluation of red fermentation tank assignments. However, there are two criteria of evaluation to consider. The first is to minimize the total weight of unassigned jobs, thereby ensuring a superior solution is one in which as many as possible scheduled jobs are processed on the desired day. This is the most important of the two evaluation criteria. The other criterion involves minimizing wasted space, *i.e.* the sum of the volumes that are still available in each of the assigned tanks.

Algorithm 6.9 utilizes a combination of these two criteria to form the final evaluation score,  $v$ , for a specific assignment. The tank order matrix,  $\mathbf{o}_t$ , employed in the algorithm is similar to the normal order matrix,  $\mathbf{o}_m$ , in that  $\mathbf{o}_t(j, q)$  refers to the tank on which job  $J_j$  is processed after processing on processor  $P_i$  (with  $i = \mathbf{o}_t(j, q - 1)$ )<sup>5</sup>. However, the tank order matrix only

<sup>5</sup>Therefore, row  $j$  of the matrix  $\mathbf{o}_t$  refers to job  $J_j$  and the entries correspond to tanks.

concerns red fermentation tanks and jobs of Type I. The derivation of the tank order matrix is explained fully in the next section of this chapter.

---

**Algorithm 6.9:** Evaluate a red fermentation tank assignment

---

**Input:** The tank order matrix  $\mathbf{o}_t$   
**Output:** Evaluation score  $v$

- 1  $v_1 \leftarrow \mathbf{determineUnassignedJobs}(\mathbf{o}_t)$ ;
- 2  $v_2 \leftarrow \mathbf{determineWastedSpace}(\mathbf{o}_t)$ ;
- 3 **if**  $v_2 \geq 0$  **then**
- 4   |  $v \leftarrow \text{priority}(1) \times v_1 + \text{priority}(2) \times v_2$ ;
- 5 **else**
- 6   |  $v \leftarrow M$ ;
- 7 **end**

---

Since the importance of the unassigned jobs outweighs the wasted space contribution, an importance index is required. This is referred to as *priority* in Algorithm 6.9 and is currently set as  $\{5, 1\}$  indicating that the unassigned jobs is five times as important as the wasted space criteria.

Determining the total weight of the unassigned jobs is a simple task and as the name suggests, it is simply the sum of the weights of all unassigned Type I jobs. This is the value returned by the function  $\mathbf{determineUnassignedJobs}(\mathbf{o}_t)$  in Algorithm 6.9. The function  $\mathbf{determineWastedSpace}(\mathbf{o}_t)$  determines the total space still available within the tanks considered in the assignment. The working of this function may be illustrated by means of the following example. Consider the simple tank order matrix

$$\mathbf{o}_{t_1} = \begin{bmatrix} 10 & 11 & 12 & \dots \\ 10 & -1 & -1 & \dots \\ 11 & -1 & -1 & \dots \\ 12 & -1 & -1 & \dots \end{bmatrix}$$

for some fictitious cellar, with the rows of the matrix corresponding to jobs  $J_0, \dots, J_3$ . The wasted space is easily determined as

$$(c_{10} - v_{10} - w_1) + (c_{11} - v_{11} - w_2) + (c_{12} - v_{12} - w_3) \quad (6.1)$$

since each job is only assigned to one tank and no mixing occurs. However, consider the situation where  $J_1$  and  $J_2$  are mixed and both are assigned to only  $P_{10}$  as indicated in

$$\mathbf{o}_{t_2} = \begin{bmatrix} 10 & 11 & 12 & \dots \\ 10 & -1 & -1 & \dots \\ 10 & -1 & -1 & \dots \\ 12 & -1 & -1 & \dots \end{bmatrix}.$$

Simply substituting  $(c_{11} - v_{11} - w_2)$  with  $(c_{10} - v_{10} - w_2)$  in (6.1) will not do, since the capacity and volume of  $P_{11}$  is then counted twice in the expression. The wasted space should rather be expressed as

$$(c_{10} - v_{10} - w_1 - w_2) + (c_{12} - v_{12} - w_3).$$

If the assignment indicated by

$$\mathbf{o}_{t_3} = \begin{bmatrix} 10 & 11 & 12 & \dots \\ 10 & 11 & 12 & \dots \\ 10 & 11 & -1 & \dots \\ 12 & 13 & -1 & \dots \end{bmatrix}$$

occurs, however unlikely this specific assignment might be, the wasted space is expressed as

$$\sum_{i=10}^{13} (c_i - v_i) - \sum_{j=1}^3 w_j.$$

The total wasted space may therefore be expressed as the sum of all  $(c_i - v_i)$  values where  $i \in \mathbf{o}_t$ , subtracting the total weight of all jobs that have been assigned.

**Example 6.6 (Example 6.3 continued)** *When the initial solution from Example 6.3 is considered, the derived tank order matrix may be expressed as*

$$\mathbf{o}_t = \begin{bmatrix} 10 & 11 & \cdots \\ 12 & -1 & \cdots \\ 12 & -1 & \cdots \\ 11 & -1 & \cdots \end{bmatrix}.$$

*The capacities and volumes are listed in Table 5.3 as  $c_{11} = 80, v_{11} = 45, c_{12} = 80, v_{12} = 0$ . Since the weights for jobs  $J_1, J_2$  and  $J_3$  are listed as 40, 25 and 15 respectively and these are the only jobs of Type I, the unassigned weight,  $v_1 = 0$ . Furthermore, the wasted space may be calculated as  $v_2 = 35$ . Therefore the evaluation score for this red tank assignment is derived as 35. ■*

## 6.3 Generating candidate moves and selecting the best move

Each set of machines exhibits different characteristics and requirements in terms of scheduling. These machine sets therefore require different sets of rules guiding the generation of a list of candidate moves. Some also require different criteria for the selection of the best candidate move. In the tabu search implementation described here, five different move types are considered. At the start of each tabu search iteration, one of these move types is randomly selected.

The first, Move Type A, applies to the assignment and order of the tipping bins. A tipping bin move is unique in two respects: First, each job of Types I and II are assigned to a tipping bin exactly once, regardless of its weight. Furthermore, tabu search moves involving the tipping bins are employed only as diversification moves. The aspiration criterion for the tabu search refers to the best move evaluation score up to date. Therefore a separate aspiration criterion exists for moves applied to the red fermentation tanks and moves applied to the rest of the cellar since the two have different means of evaluation (as described in §§6.2.1 and 6.2.2). Move Type A is applied only if the aspiration criterion has been unimproved on, *i.e.* the relevant best move score has been unchanged, for a set number of iterations. Hence, Move Type A is not considered when a move type is selected randomly at the start of the iteration.

Move Types B and C refer to tabu search moves involving separators and presses, respectively. In both these move types, the order and number of assignments are considered. However, a different approach to generating the list of candidate solutions is considered in Move Type C.

The assignment of task  $T_{j2}$  of a Type II job  $J_j$  may be assigned to either a separator or a press. Move Type D is concerned with the assignment of this task. A variable  $b$  is introduced where  $b_j$  equals 1 if task  $T_{j2}$  is assigned to a separator rather than a press.

The last move type, Move Type E, requires a slightly different approach. This move is concerned with the assignment of Type I jobs to the red fermentation tanks. Jobs assigned to the same

tank are mixed and do not follow one another as with any of the previously mentioned move types. Therefore, the relevant machine availability and starting volumes should be kept in mind.

The move types mentioned above are described in more detail in this section. First, the general ejection chain move for this tabu list application is considered in §6.3.1. This is followed by descriptions of the five move types in §§6.3.2–6.3.6, respectively.

### 6.3.1 The general ejection chain move and further communal move aspects

All the move types described above, save Move Type D, make use of ejection chain moves applied to the order matrix,  $\mathbf{o}$ , of the current solution. Different types of moves within the ejection chain were considered. In most of these cases, a set of rules was considered that would bring about a type of swap within the ejection chain move. For example, only applying horizontal swaps (swaps occurring within rows only) when an assignment is feasible and the assignment order is infeasible. After considering various options for the ejection chain structure, it was found that the simplest approach delivered the best results on a set of test problems. Even though different requirements have to be considered for each processor set, a general ejection chain move for the cellar scheduling problem is included in Algorithm 6.10.

The most significant difference between the ejection chains for the various processor sets lie in the procedure **checkMoveFeasibility**( $\mathbf{o}_a, P_f, P_\ell, \mathbf{L}, \text{lastCol}$ ), since different requirements have to be met for a solution to be feasible with respect to each job type and processor set combination.

For every new move  $a$ , the ejection chain starts with a copy,  $\mathbf{o}_a$ , of the current assignment order matrix,  $\mathbf{o}$ . Swaps are applied to the order matrix,  $\mathbf{o}_a$ , until at least *minNrOfSwaps* have been made and the current assignment order is considered feasible. For each processor set considered, the swaps are only applied to the specified section of the assignment order. This section is determined by the first and last processors of the set considered, for example,  $P_{m_1}$  and  $P_{m_2-1}$  when the separators are considered. The section considered is also limited by the last column to move on. This last column, expressed as *lastCol* in Algorithm 6.10, is determined as the maximum of all first positions (one per row considered) of the current assignment order matrix,  $\mathbf{o}$ , with entries of  $-1$ . For the assignment order expressed in Table 6.2 this last column would therefore be 5. This allows the number of jobs assigned to a single processor to vary while still discouraging a randomly generated move where all jobs are assigned to only a few processors of a specific set.

In line 2, the starting position for the ejection chain move is chosen. This may be any cell of  $\mathbf{o}_a$  between rows  $P_f$  and  $P_\ell$  and between columns 0 and *lastCol* with an entry not equal to  $-1$ . The cell to consider is referred to by its row,  $i$ , and the position within that row,  $p$ . The current job,  $J_{j_c}$ , is determined as the entry in the chosen cell<sup>6</sup>. This entry is then replaced by the old job,  $J_{j_o}$ , whose value is set to  $-1$  for the first iteration. This way, the feasibility of the move may be considered at any stage when the current job value is  $-1$ , indicating that no real job has been displaced. It is therefore an open ejection chain move, since the last entry to be changed is not necessarily in the original starting position.

After the replacement  $j_o \leftarrow j_c$ , a new current job may be considered. The loop that starts at line 8 is repeated until a feasible move has been created. A new target cell is repeatedly considered by means of a simple function **verticalMove**( $i, p, P_f, P_\ell$ ). This function returns

<sup>6</sup>The variable  $j_c$  (or  $j_o$ ) only refers to a value indicating a job  $J_{j_c}$  (or  $J_{j_o}$ ). The variable  $j_c$  (or  $j_o$ ) is allowed to be  $-1$  even though no job  $J_{-1}$  exists.



a new row  $P_f \leq i \leq P_\ell$  that is not equal to the previously considered row. The new job  $J_{j_c}$  is determined by the entry in the new row  $i$  and column  $p$  of  $\mathbf{o}_a$ . If  $j_o = -1$ , then the new  $j_c$  should not also have a value of  $-1$ , since such a replacement will result in no difference to the assignment order.

---

**Algorithm 6.10:** The general ejection chain move

---

**Input:** An current order matrix  $\mathbf{o}$ , the first and last processors to apply the move to,  $P_f$  and  $P_\ell$ , and the last column to consider

**Output:** The list of considered solutions,  $\mathbf{L}$ , containing the considered order matrices after each feasible ejection chain move has been successfully applied

```

1 for  $a = 0$  to last move do
2    $[i, p] \leftarrow \text{generateFirstCell}(\mathbf{o}_a, P_f, P_\ell, \text{lastCol});$ 
3    $j_c \leftarrow \mathbf{o}_a(i, p);$ 
4    $j_o \leftarrow -1;$ 
5    $\mathbf{o}_a(i, p) \leftarrow j_o;$ 
6    $\text{isFeasibleMove} \leftarrow \text{false};$ 
7   counter  $\leftarrow 0;$ 
8   while  $\text{isFeasibleMove} = \text{false}$  do
9      $j_o \leftarrow j_c;$ 
10     $i \leftarrow \text{verticalMove}(i, p, P_f, P_\ell);$ 
11     $p \leftarrow \text{rand}(1, \text{lastCol});$ 
12    if  $j_o \neq -1$  then
13      while  $j_o \in \mathbf{o}_a(i, :)$  do
14         $i \leftarrow \text{verticalMove}(i, p, P_f, P_\ell);$ 
15      end
16    else
17      while  $\mathbf{o}_a(i, p) = -1$  do
18         $p \leftarrow \text{rand}(1, \text{lastCol});$ 
19         $i \leftarrow \text{verticalMove}(i, p, P_f, P_\ell);$ 
20      end
21    end
22     $j_c \leftarrow \mathbf{o}_a(i, p);$ 
23     $\mathbf{o}_a(i, p) \leftarrow j_o;$ 
24    counter  $\leftarrow$  counter + 1;
25    if  $j_c = -1$  then
26      if counter  $\geq \text{minNrOfSwaps}$  then
27         $\text{isFeasibleMove} \leftarrow \text{checkMoveFeasibility}(\mathbf{o}_a, P_f, P_\ell, \mathbf{L}, \text{lastCol});$ 
28      end
29    end
30  end
31   $\mathbf{L}(a) \leftarrow \mathbf{o}_a;$ 
32 end

```

---

Therefore the function  $\text{verticalMove}(i, p, P_f, P_\ell)$  is applied together with a new randomly generated column  $p$ , until the resulting variable  $j_c$  does not also have a value of  $-1$ . On the other hand, if  $j_o \neq -1$ , the prerequisite is simply that the new row  $i$  for the placement of the value of  $j_o$  should not already contain a cell with the value of  $j_o$ , *i.e.* a value other than  $-1$  may only be placed in a row if the value is not already present somewhere in that row. Finally, in lines 22 and 23 the displacement is made, *i.e.* the new job value is set as the entry  $\mathbf{o}_a(i, p)$  after

which the old job value,  $j_o$ , replaces  $j_c$  as the value of  $\mathbf{o}_a(i, p)$ . The process is then repeated until a feasible move is found. A move is considered feasible within the ejection chain if  $j_c = -1$ , the number of iterations is at least as large as the minimum number of swaps and the move is determined to be feasible by the function **checkMoveFeasibility**( $\mathbf{o}_a, P_f, P_\ell, L, \text{lastCol}$ ).

One of the requirements that may be expected for a move  $\mathbf{o}_a$  to be considered feasible, depending on the move type, is that the order in which moves are processed on the different processors in the same set, should be consistent. This requirement should be tested in all move types where a job is allowed to be assigned to more than one processor, except for move type E. Algorithm 6.11 may be used to determine whether such an order has remained consistent.

---



---

**Algorithm 6.11:** Check correct order
 

---

**Input:** The order matrix,  $\mathbf{o}_a$ , the first processor to form part of the considered section,  $P_f$ , and the last processor,  $P_\ell$ , and the value of  $\text{lastCol}$

**Output:** The boolean, *feasible*, indicating the feasibility of the considered order

```

1 feasible  $\leftarrow$  true;  $i \leftarrow P_f$ ;
2 while ( $i \leq P_\ell$ ) and (feasible = true) do
3    $p_1 \leftarrow 1$ ;
4   while ( $p_1 \leq \text{lastCol} - 1$ ) and (feasible = true) do
5      $j_1 \leftarrow \mathbf{o}_a(i, p_1)$ ;
6     if  $j_1 \neq -1$  then
7        $p_2 \leftarrow p_1 + 1$ ;
8       while ( $p_2 \leq \text{lastCol}$ ) and (feasible = true) do
9          $j_2 \leftarrow \mathbf{o}_a(i, p_2)$ ;
10        if  $j_2 \neq -1$  then
11          if  $j_1 = j_2$  then
12            feasible  $\leftarrow$  false;
13            break;
14          end
15          pairs  $\leftarrow$  checkAndAddPair(pairs,  $j_1, j_2$ );
16          if pairs(0,0) = -1 then
17            feasible  $\leftarrow$  false;
18            break;
19          end
20        end
21         $p_2 \leftarrow p_2 + 1$ ;
22      end
23    end
24     $p_1 \leftarrow p_1 + 1$ ;
25  end
26   $i \leftarrow i + 1$ ;
27 end
```

---

After a move type has been selected and applied, it is sometimes possible that the remainder of the order matrix may require updating. For example, when a move is applied to the separators, the order of the jobs on this set of processors may have changed. However, since the separated skins move on to the presses, the same new order has to be maintained there — otherwise the considered order will be infeasible. Algorithm 6.13 is used to apply these updates to the order matrices considered on the list of moves to preserve the order feasibility.

The function **createPairs**, refers to the creation of a pairs matrix similar to the one generated in Algorithm 6.11. The only difference is that it is not necessary to test for order feasibility, simply to create the pairs matrix. Say, for example, that a move on a separator was made. Then the pairs matrix is created considering the order of the jobs assigned to the separators. The updates to be made, are made in the form of altered assignments to the presses, *i.e.*  $P_s = P_{m_2}$  and  $P_e = P_{m_3-1}$ . All the assignment pairs are considered per press and if a considered assigned pair  $(j_1, j_2)$  is infeasible (when  $\text{pairs}(j_2, j_1) = 1$ ), the order is changed. More specifically, job  $J_{j_2}$  is moved into the position of job  $J_{j_1}$  and the remaining jobs are moved one position to the right until the original position of job  $J_{j_2}$  is reached. This process is repeated for each order considered in the list of moves until the order of each move has been updated.

After a list of candidate solutions has been generated, the next step is to select the best solution from this list and to update the Tabu list. Algorithm 6.14 is used to find the best solution in the list and then to update the necessary tabu search criteria. The first step in selecting the most favourable move, is to sort all the currently considered candidate solutions according to the evaluation scores. This is achieved by applying the function **sortMoveValues**( $L, v$ ) and generating the array  $\beta$  to contain the move numbers of the best moves in order of decreasing favourability.

---



---

**Procedure** `checkAndAddPair`( $\text{pairs}, j_1, j_2$ )

---

```

1 feasible  $\leftarrow$  true;
2 if  $\text{pairs}(j_2, j_1) = 1$  then
3   |  $\text{pairs}(0,0) \leftarrow -1$ ;
4   | feasible  $\leftarrow$  false;
5 else if  $\text{pairs}(j_1, j_2) \neq 1$  then
6   |  $\text{pairs}(j_1, j_2) \leftarrow 1$ ;
7   | for  $j = j_r + 1$  to  $n$  do
8     | if  $\text{pairs}(j_2, j) = 1$  then
9       | if  $\text{pairs}(j_1, j) \neq 1$  then
10        |  $\text{pairs} \leftarrow \text{CheckAndAddPair}(\text{pairs}, j_1, j)$ ;
11        | end
12        | end
13        | if  $\text{pairs}(j, j_1) = 1$  then
14          | if  $\text{pairs}(j, j_2) \neq 1$  then
15            |  $\text{pairs} \leftarrow \text{CheckAndAddPair}(\text{pairs}, j, j_2)$ ;
16            | end
17          | end
18        | end
19 end
20 return  $\text{pairs}$ ;
```

---

Until a solution is found, the list of possible solutions is traversed in the order suggested by  $\beta$ . To test whether a move is allowed, the corresponding tabu list is first considered. Each move type has its own tabu list. Each tabu list consists of previously chosen solutions. However, only the section of the order matrix applying to the processor set of the specific move type is placed on the tabu list. This section refers to the rows corresponding to the current set of processors of the current move type. A move may only be chosen if it is not equal to another move already in the relevant tabu list or if it improves on the best move found according to the aspiration criterion,  $\alpha$  (line 8). The equality of two moves does not necessarily refer to a

direct equality, two moves are considered equal if they have the same meaning inside the cellar as a move already on the list. For example, consider two identical processors in an assignment order matrix. If the two rows corresponding to these processors are swapped (with no change to the order of the values in each row), it results in an order matrix implying the same solution. These two solutions are therefore considered as equal.

---



---

**Algorithm 6.13:** Update influenced machine
 

---

**Input:** The list of considered moves,  $\mathbf{L}$ , and the first and last processors of the processor set on which the updates should be made

**Output:** The updated list of moves,  $\mathbf{L}$ , with no conflicting orders

```

1 for  $a = 0$  to last move do
2   pairs  $\leftarrow$  createPairs();
3   for  $i = P_f$  to  $P_\ell$  do
4      $p_1 \leftarrow 1$ ;
5     while  $(p_1 \leq n - 1)$  and  $(\mathbf{L}(a)(i, p_1) \neq -1)$  do
6        $j_1 \leftarrow o_a(i, p_1)$ ;
7        $p_2 \leftarrow p_1 + 1$ ;
8       while  $(p_2 \leq n)$  and  $(\mathbf{L}(a)(i, p_2) \neq -1)$  do
9          $j_2 \leftarrow \mathbf{L}(a)(i, p_2)$ ;
10        if  $(j_1 \neq -1)$  and  $(j_2 \neq -1)$  and  $(pairs(j_2, j_1) = 1)$  then
11           $p \leftarrow p_1$ ;
12           $j \leftarrow j_2$ ;
13          newPrevious  $\leftarrow 0$ ;
14          previous  $\leftarrow j_1$ ;
15          while  $p \leq p_2 - 1$  do
16            newPrevious  $\leftarrow \mathbf{L}(a)(i, p + 1)$ ;
17             $\mathbf{L}(a)(i, p + 1) \leftarrow$  previous;
18            previous  $\leftarrow$  newPrevious;
19             $p \leftarrow p + 1$ ;
20          end
21           $\mathbf{L}(a)(i, p_1) \leftarrow j$ ;
22           $p_2 \leftarrow p_1$ ;
23        end
24       $p_2 \leftarrow p_2 + 1$ ;
25    end
26     $p_1 \leftarrow p_1 + 1$ ;
27  end
28 end
29  $a \leftarrow a + 1$ ;
30 end

```

---

Once a suitable move has been selected, the relevant order matrix section of the move is added to the corresponding tabu list. In the common tabu search application, it is rather the inverse of a move that is added to the tabu list to stop the move from being undone. However, the goal of the tabu list for this tabu search, is rather to prevent a solution and its similar solutions from being revisited to often. The tabu tenure, size of the tabu list, is taken as the number of processors in the set relevant to the move type considered. For example, the tabu list size for move type A is taken as  $m_1 - 1$  where  $P_{m_1-1}$  refers to the last tipping bin.

**Algorithm 6.14:** Selecting the best candidate solution

---

**Input:** The considered list of order matrices,  $\mathbf{L}$ , and the evaluation scores of each  $\mathbf{v}$   
**Output:** The position,  $\phi$ , of the best selected move on list  $\mathbf{L}$

```

1  $\beta \leftarrow \text{sortMoveValues}(\mathbf{L}, \mathbf{v});$ 
2  $a \leftarrow 0;$ 
3  $\text{solutionFound} \leftarrow \text{false};$ 
4  $\text{alreadyOnList} \leftarrow \text{false};$ 
5 while  $\text{solutionFound} = \text{false}$  do
6    $\text{alreadyOnList} \leftarrow \text{isOnTabuList}(\text{moveType}, \text{tabuList}, \mathbf{L}(\beta(a)));$ 
7   if  $\text{alreadyOnList} = \text{true}$  then
8     if  $\mathbf{v}(\beta(a)) < \alpha$  then
9        $\alpha \leftarrow \mathbf{v}(\beta(a));$ 
10       $\text{solutionFound} \leftarrow \text{true};$ 
11    end
12  else
13    if  $\mathbf{v}(\beta(a)) < \alpha$  then
14       $\alpha \leftarrow \mathbf{v}(\beta(a));$ 
15    end
16     $\text{solutionFound} \leftarrow \text{true};$ 
17  end
18   $a \leftarrow a + 1;$ 
19 end
20  $\phi \leftarrow \beta(a - 1);$ 
21  $\text{addToTabuList}(\text{moveType}, \mathbf{L}(\phi));$ 

```

---

**6.3.2 Move Type A: Tipping bins**

As previously mentioned, Move Type A is only applied as a diversification move. It is only considered if no improvement on the evaluation score has occurred for a set number of iterations. This move is outlined in Algorithm 6.15.

It is implemented by means of a set of ejection chains on the order matrix of the current solution. The function `ejectionChainMove( $\mathbf{o}$ , 0,  $m_1 - 1$ , minNrOfSwaps)` is called on line 2 and is applied on the processor set of the tipping bins following the guidelines set out in the general ejection chain algorithm in Algorithm 6.10.

**Algorithm 6.15:** Move on tipping bins

---

**Input:** The current order matrix,  $\mathbf{o}$   
**Output:** The updated order matrix,  $\mathbf{o}$ , after the best move is selected from the generated list,  $\mathbf{L}$

```

1  $|\mathbf{L}| \leftarrow 2n(m_1 - 1);$ 
2  $\mathbf{L} \leftarrow \text{ejectionChainMove}(\mathbf{o}, 0, m_1 - 1, \text{minNrOfSwaps});$ 
3 for  $a = 0$  to last move do
4    $\text{createNewOrderFromTippingBin}(\mathbf{L}(a));$ 
5    $\mathbf{v}(a) \leftarrow \text{evaluateCandidateSolution}(\mathbf{L}(a));$ 
6 end
7  $\phi \leftarrow \text{selectMoveOnTippingBin}(\mathbf{L}, \mathbf{v});$ 
8  $\mathbf{o} \leftarrow \mathbf{L}(\phi);$ 

```

---

All moves generated by the general ejection chain are already feasible since each job can be assigned only once to the tipping bins and hence no order of job assignment should result in an infeasibility. The procedure **checkMoveFeasibility**( $\mathbf{o}_a, P_f, P_\ell, \mathbf{L}, \text{lastCol}$ ) in Algorithm 6.10 therefore only tests whether the move considered is already in the list of moves. For each of the five move types, the generated ejection chain list has a different size,  $|\mathbf{L}|$ , to fit the requirements of the move type. The size of the list is determined experimentally by means of solving a set of test problems. This is true for all move types. For move type A, this list size is calculated as twice the product of the number of jobs and the number of tipping bins. This is one of the largest list sizes since the diversification move is not applied as often as the remaining move types. The fact that the move type is seldom selected, just emphasizes the importance of having a good selection of candidate solutions.

Each candidate solution in the returned list,  $\mathbf{L}$ , is updated. In this case, rather than applying Algorithm 6.13 twice to update the current assignment of jobs to the separators and presses, a new order matrix is created for each of the solutions considered. The function **createNewOrderFromTippingBin**( $\mathbf{L}(\mathbf{a})$ ) is used for this purpose. The new tipping bin assignment and order is considered and full order matrices are generated according to the method used to assign the initial solution.

Each of the new moves are then evaluated in order to eventually select the best feasible move to apply, as outlined in Algorithm 6.14.

### 6.3.3 Move Type B: separators

The process of creating a list of moves involving separators and selecting the best candidate move from this list is outlined in Algorithm 6.16.

The list of moves is generated by means of an ejection chain as described in Algorithm 6.10. When determining the list size,  $\delta_j$  refers to the number of assignments required to feasibly assign job  $J_j$  to the separators. A move generated is feasible if it is not already in the move list,  $\mathbf{L}$ . The jobs should also have a consistent order in which they are processed, as described in Algorithm 6.11. Once the moves have been made on the separator section within the order matrices, rows  $m_2, \dots, m_3 - 1$  are updated to ensure that the same consistent processing order is applied on the presses. Algorithm 6.13 is used for updating the order matrices in the list. The most favourable order matrix in the list is selected and the current order matrix of the tabu search is replaced.

---

**Algorithm 6.16:** Move on separators

---

**Input:** The current order matrix,  $\mathbf{o}$

**Output:** The updated order matrix,  $\mathbf{o}$ , after the best move is selected from the generated list,  $\mathbf{L}$

- 1  $|\mathbf{L}| \leftarrow \sum_{j=1}^n \delta_j \mathbf{b}(j)$ ;
  - 2  $\mathbf{L} \leftarrow \text{ejectionChainMove}(\mathbf{o}, m_1, m_2 - 1, \text{minNrOfSwaps})$ ;
  - 3  $\mathbf{L} \leftarrow \text{updateInfluencedMachine}(\mathbf{L}, m_1, m_2 - 1, m_2, m_3 - 1)$ ;
  - 4 **for**  $a = 0$  **to** *last move* **do**
  - 5    $v(a) \leftarrow \text{evaluateSolution}(\mathbf{L}(a))$ ;
  - 6 **end**
  - 7  $\phi \leftarrow \text{selectMoveOnSeparator}(\mathbf{L}, \mathbf{v})$ ;
  - 8  $\mathbf{o} \leftarrow \mathbf{L}(\phi)$ ;
-

### 6.3.4 Move Type C: presses

In Algorithm 6.17 it is clear that a move of type C differs from the previous two move types described above. Perhaps the most significant difference is the fact that two lists are generated when performing a move of type C.

---



---

**Algorithm 6.17:** Move on presses

---

**Input:** The current order matrix,  $\mathbf{o}$   
**Output:** The updated order matrix,  $\mathbf{o}$ , after the best move is selected from the second generated list,  $\mathbf{L}_r$

- 1  $\mathbf{o} \leftarrow \text{createMaximumAssignmentOrder}(m_2, m_3 - 1)$ ;
- 2  $|\mathbf{L}_e| - 1 \leftarrow \sum_{j=r+1}^n \delta_j$ ;
- 3  $\mathbf{L}_e \leftarrow \text{ejectionChainMove}(\mathbf{o}, m_2, m_3 - 1, \text{minNrOfSwaps})$ ;
- 4  $\mathbf{L}_r \leftarrow \text{updateToRealListOfMoves}(\mathbf{L}_e)$ ;
- 5  $\mathbf{L}_r \leftarrow \text{updateInfluencedMachine}(\mathbf{L}_r, m_2, m_3 - 1, m_1, m_2 - 1)$ ;
- 6 **for**  $a = 0$  **to** *last move* **do**
- 7  $v(a) \leftarrow \text{evaluateSolution}(\mathbf{L}_r(a))$ ;
- 8 **end**
- 9  $\phi \leftarrow \text{selectMoveOnPress}(\mathbf{L}_r, v)$ ;
- 10  $\mathbf{o} \leftarrow \mathbf{L}_r(\phi)$ ;

---

Unlike the tipping bins and separators, not all the presses have the same capacity. Therefore, each of the jobs have a maximum number of assignments and a minimum with respect to their weights and the different weights of the processors. For example, consider a set of three presses with capacities 20, 15 and 15 tonnes. A job of weight 20 tonnes may either be assigned to the one 20 ton tank or if it is assigned to either of the 15 ton tanks, it should also be assigned to the other 15 ton tank. This job then has a maximum possible assignment value of 2 and a minimum of 1.

The current order matrix,  $\mathbf{o}$ , is updated to a *maximum assignment order* in line 1 where further feasible assignments (not necessarily in feasible order) are added until the maximum number of assignments have been made per press.

The first list,  $\mathbf{L}_e$ , is generated by means of the ejection chain move described in Algorithm 6.10. The size of this list is taken as the sum of all the maximum assignment values of Type II and Type II jobs. A move generated here is considered feasible if the processing order of the jobs are consistent on the presses and also when it is not already present in the list. However, these assignment orders generated by the ejection chain move, are maximum assignment orders and most are therefore likely not even near optimal.

In line 4, the real list of moves,  $\mathbf{L}_r$ , is generated by means of the function **updateToRealListOfMoves**( $\mathbf{L}_e$ ). This procedure uses a recursive function to keep removing assignments until a complete list of possibilities has been generated and returns this list as  $\mathbf{L}_r$ . It is then this list of moves for which the separator assignments are updated to fit the processing order of the jobs specified by assignments to the presses. Finally,  $\mathbf{L}_r$  is evaluated, the best solution in the list is selected and the current order matrix is updated.

### 6.3.5 Move Type D: separator or press assignment of $T_{j2}$

This move type is outlined in Algorithm 6.18 for a one dimensional array,  $\mathbf{b}$ , unlike the two dimensional order matrix considered previously. Applying an ejection chain move will therefore not work; hence simple swaps are considered instead.

---



---

**Algorithm 6.18:** Move on  $b$  of Type II jobs

---

**Input:** The current order matrix,  $\mathbf{o}$ , and the  $T_{j2}$  assignment specification,  $\mathbf{b}$   
**Output:** The updated  $T_{j2}$  assignment specification,  $\mathbf{b}$ , and the corresponding order matrix,  $\mathbf{o}$ , after the best move is selected from the generated list,  $\mathbf{S}$

```

1  $|\mathbf{S}| \leftarrow 1 + (w - r) - \sum_{j=r+1}^w \bar{\mathbf{b}}(j)$ ;
2  $a \leftarrow 0$ ;
3 while  $a < \text{last move}$  do
4   feasible  $\leftarrow$  false;
5   while  $\text{feasible} = \text{false}$  do
6      $\mathbf{S}(a) \leftarrow \mathbf{b}$ ;
7      $\mathbf{S}(a) \leftarrow \text{Swap}(\mathbf{S}(a), r + 1, w)$ ;
8      $p \leftarrow a - 1$ ;
9     isEqual  $\leftarrow$  false;
10    feasible  $\leftarrow$  true;
11    while  $(\text{feasible} = \text{true})$  and  $(p \geq 0)$  do
12      if  $\mathbf{S}(a) = \mathbf{S}(p)$  then
13        feasible  $\leftarrow$  false;
14      end
15       $p \leftarrow p - 1$ ;
16    end
17  end
18   $a \leftarrow a + 1$ ;
19 end
20 for  $a = 0$  to  $\text{last move}$  do
21    $\mathbf{L}(a) \leftarrow \text{updateInfluencedMachine}(\mathbf{o}, m_2, m_3 - 1, m_1, m_2 - 1)$ ;
22    $\mathbf{v}(a) \leftarrow \text{evaluateSolution}(\mathbf{L}(a), \mathbf{S}(a))$ ;
23 end
24  $\phi \leftarrow \text{selectMoveOnPressOrSep}(\mathbf{S}, \mathbf{v})$ ;
25  $\mathbf{b} \leftarrow \mathbf{S}(\phi)$ ;
26  $\mathbf{o} \leftarrow \mathbf{L}(\phi)$ ;

```

---

The move list,  $\mathbf{S}$ , is constructed in lines 2–18. The move list size is determined as one more than the number of Type II jobs ( $w - r$ ) without the jobs that is forced to be processed on the separators (by the request of the winemaker).

When applying the procedure  $\text{Swap}(\mathbf{S}(a), r + 1, w)$ , either one or two swaps are applied. In this procedure a swap does not refer to exchanging the values between two entries; since  $\mathbf{b}$  is a binary vector, it refers to swapping the value of the chosen cell between 0 and 1.

One aspect of Algorithm 6.13 not previously mentioned, is that when the processing order on the separators is considered, only assignments of job  $J_j$  is considered for which  $\mathbf{b}(j) = 1$ . Therefore, if the values of  $\mathbf{b}$  change, an update of the processing orders on the separators and presses is required. Once the updates have been made, the combination of the two lists,  $\mathbf{S}$  and  $\mathbf{L}$ , are considered together in order to evaluate the list of moves.



Once the move values have been determined, the best solution may be selected as outlined in Algorithm 6.14.

### 6.3.6 Move Type E: red fermentation tanks

When generating a feasible list for moves of type E, it is important to note that not every job may be assigned to any tank, unlike the previous move types where the only possible differences between a set of machines was their capacity.

In Chapter 5, the variable  $\mu_{ijk}$  was defined to be  $\mu_{ijk} = 1$  if task  $T_{jk}$  may be performed on processor  $P_i$ .

The procedure **createTankSpecifics()** in line 1 of Algorithm 6.19 is used to generate a new assignment matrix,  $\mathbf{o}_r$ , only applicable to the allowed tanks for the set of jobs considered.

---

**Algorithm 6.19:** Move on red fermentation tanks

---

**Input:** The cellar requirements and job information.  
**Output:** The updated order matrix after the selected move has been applied to the red fermentation tanks.

- 1 **createTankSpecifics()**;
- 2  $\mathbf{o}_r \leftarrow \mathbf{updateMaximumRedOrder}(\ell)$ ;
- 3  $|\mathbf{L}_e| \leftarrow r - \sum_{j=1}^r \bar{\delta}_j$ ;
- 4  $\mathbf{L}_e \leftarrow \mathbf{ejectionChainMove}(\mathbf{o}_r, \text{minNrOfSwaps})$ ;
- 5 **reassignJobs()**;
- 6  $\mathbf{L}_f \leftarrow \mathbf{createFeasibleList}(\mathbf{L}_e)$ ;
- 7  $\mathbf{L}_r, \mathbf{v} \leftarrow \mathbf{createRealListAndEvaluate}(\mathbf{L}_f)$ ;
- 8  $\phi \leftarrow \mathbf{selectBestRedTankMove}(\mathbf{L}_r, \mathbf{v})$ ;
- 9 **addToTabuList**(E,  $\mathbf{L}_r(\phi)$ );
- 10  $\mathbf{o} \leftarrow \mathbf{updateCurrentOrderMatrix}(\mathbf{o}, \mathbf{L}_r(\phi))$ ;

---

In the new assignment matrix,  $\mathbf{o}_r$ , the jobs are also assigned to a maximum number of tanks. Since the jobs may be mixed, the maximum allowed values are estimated. Let the maximum assignment value for a job  $J_j$  be denoted by  $\delta_j$ . If  $\delta_j = x$  for job  $J_j$  and some value  $x$ , it indicates that in the maximum assignment order, the job may be assigned to exactly  $x$  processors and in the final assignment matrix it may be assigned to no more than  $x$  processors. If the weight of the job is less than 10 tonnes,  $w_j < 10$ ,  $\delta_j$  is 1. If the weight of job  $J_j$  falls within the interval,  $10 \leq w_j < 80$ , the maximum assignment value,  $\delta_j$ , is 2. For any job  $J_j$  with a weight greater than 80 tonnes,  $\delta_j$  is 3. A single vineyard block seldom yields more than 100 tonnes of grapes.

However, if only  $y$  tanks are allowed for job  $J_j$ , then the assignments are also limited by the restriction  $\delta_j \leq y$ . Let  $\bar{\delta}_j$  be 1 only if  $\delta_j = y$  and 0 otherwise. Then the size of list  $\mathbf{L}_e$  is taken as the number of type I jobs with  $\bar{\delta}_j = 0$ .

The matrix entries of the new order matrix still correspond to the jobs assigned to the tank considered, which is indicated by the row index. By eliminating all the unnecessary tanks when applying the ejection chain move, processing time is reduced.

Furthermore, **createTankSpecifics()** is also used to eliminate all jobs that are assigned to all of the allowed tanks. These jobs are reassigned in **reassignJobs()**. This is only possible because the order of the assignment is not of importance.

When the ejection chain move is performed, a move is considered feasible if all jobs are assigned to allowable machines.

If a second criterion were to be added, ensuring that two jobs are only allowed to be assigned to the same machine if their qualities match, feasible moves become scarce and the ejection chain requires significant processing time. This is where the procedure `createFeasibleList( $L_e$ )` is applied.

---



---

**Procedure createFeasibleList( $L_e$ )**


---

```

1   $p \leftarrow 0$ ;
2  for  $a = 0$  to last move on  $L_e$  do
3       $p_c \leftarrow p$ ;
4       $L_f(p_c) \leftarrow L_e(a)$ ;
5       $p \leftarrow p + 1$ ;
6      while  $p_c < p$  do
7           $\rho \leftarrow \text{jobQualitiesMatch}(L_f(p_c))$ ;
8          while  $\rho(0) \neq -1$  do
9               $j_1 \leftarrow L_f(p_c)(\rho(0), \rho(1))$ ;
10              $j_2 \leftarrow L_f(p_c)(\rho(0), \rho(2))$ ;
11             stop  $\leftarrow p$ ;
12             for  $i = p_c$  to stop do
13                  $L_f \leftarrow \text{splitMove}(L_f, i, p, \rho(0), j_1, j_2)$ ;
14                 if  $L_f(p)(0,1) \neq 0$  then
15                      $p \leftarrow p + 1$ ;
16                 end
17             end
18              $\rho(0) \leftarrow \text{jobQualitiesMatch}(L_f(p_c))$ ;
19         end
20          $p_c \leftarrow p_c + 1$ ;
21     end
22 end

```

---

This procedure considers each move in the list generated by the ejection chain move,  $L_e$ , and in turn applies the function `jobQualitiesMatch( $L_f(p_c)$ )`. This function returns  $\rho(0) = \{i, x_1, x_2\}$  where  $i$  denotes the row (or tank), and  $x_1$  and  $x_2$  the positions of the two jobs that are not allowed to be assigned to the same tank represented by row  $i$ . If no such assignment is made,  $\rho(0) = -1$ . However, if there is such a clash, the current move is split into two moves, where each move has one of the two jobs removed.

After the feasible list has been generated, the various removals of the jobs are considered. Rather than using an algorithm similar to the one used to create such a list for moves of type C, the heuristic `createRealListAndEvaluate( $L_f$ )` is used. For each move in list  $L_f$ , different removals are considered and a new removal is only added to the final list of moves,  $L_r$ , if it improves the evaluation score of the previous move. If the currently considered removals remain unimproved for a the set number,  $r$  (referring to the number of Type I jobs) of iterations, the next move in the list  $L_r$  is considered.

The function `fixTankOrder( $o_t, j, p_2$ )` removes the  $-1$  replacing  $o_t(j, p_2)$  in line 19 by shifting the following entries to the left. Otherwise an infeasible tank order matrix would have been considered.

**Procedure createRealListAndEvaluate( $L_f$ )**


---

```

1   $p \leftarrow 0$ ;
2  for  $a = 0$  to  $p$  do
3       $j \leftarrow 0$ ;
4       $p_1 \leftarrow 0$ ;  $p_2 \leftarrow 0$ ;
5       $\omega \leftarrow \text{evaluateMoveOnRedTanks}(\mathbf{o}_t, M)$ ;
6       $L_r \leftarrow \text{addMoveToRealList}(L_r, p, L_f(a), \mathbf{o}_t)$ ;
7       $v(p) \leftarrow \omega$ ;
8       $p \leftarrow p + 1$ ;
9       $\text{unimproved} \leftarrow 0$ ;
10     while  $\text{unimproved} \leq r$  do
11          $\bar{\omega} \leftarrow \omega$ ;
12          $j \leftarrow \text{rand}(1, r)$ ;
13          $p_1 \leftarrow \text{returnFirstPosition}(\mathbf{o}_t(j, :), -1) - 1$ ;
14         while  $p_1 = -1$  do
15              $j \leftarrow \text{rand}(1, r)$ ;
16              $p_1 \leftarrow \text{returnFirstPosition}(\mathbf{o}_t(j, :), -1) - 1$ ;
17         end
18          $p_2 \leftarrow \text{rand}(1, p_1 - 1)$ ;
19          $\text{temp} \leftarrow \mathbf{o}_t(j, p_2)$ ;
20          $\mathbf{o}_t(j, p_2) \leftarrow -1$ ;
21          $\omega \leftarrow \text{evaluateMoveOnRedTanks}(\mathbf{o}_t, M)$ ;
22         if  $\omega = M$  then
23              $\mathbf{o}_t(j, p_2) \leftarrow \text{temp}$ ;
24              $\omega \leftarrow \bar{\omega}$ ;
25              $\text{unimproved} \leftarrow \text{unimproved} + 1$ ;
26         else if  $\omega < \bar{\omega}$  then
27              $\mathbf{o}_t \leftarrow \text{fixTankOrder}(\mathbf{o}_t, j, p_2)$ ;
28              $L_r \leftarrow \text{addMoveToRealList}(L_r, p, L_f(a), \mathbf{o}_t)$ ;
29              $v(p) \leftarrow \omega$ ;
30              $p \leftarrow p + 1$ ;
31              $\text{unimproved} \leftarrow 0$ ;
32         else
33              $\mathbf{o}_t = \text{fixTankOrder}(\mathbf{o}_t, j, p_2)$ ;
34              $\text{unimproved} \leftarrow \text{unimproved} + 1$ ;
35         end
36     end
37 end

```

---

## 6.4 Solving the cellar scheduling problem with a tabu search

A tabu search customized for the cellar scheduling problem is outlined in Algorithm 6.22. The function `generateMoveType(previousType, firstMoveType, lastMoveType)` is used to generate the random move type, choosing between types B, C, D or E. Once either of the stopping criteria in lines 35 and 38 has been fulfilled, the move types that may be used in the tabu search are updated accordingly.

---



---

**Algorithm 6.22:** The cellar scheduling tabu search

---

**Input:** The created scenario for which to solve the cellar scheduling problem  
**Output:** The best solution found

```

1 createInitialSolution(J, P, w);
2  $[\alpha, \epsilon] \leftarrow \text{evaluateFirstSolution}(o)$ ;
3 asp  $\leftarrow \alpha$ ;
4 solutionFound  $\leftarrow$  false;
5 firstMoveType  $\leftarrow$  B;
6 lastMoveType  $\leftarrow$  E;
7 previousType  $\leftarrow$  -1;
8 unchanged  $\leftarrow$  -1;
9 while (solutionFound = false) and (counter  $\leq$  counterMaximum) do
10   moveType  $\leftarrow$  generateMoveType(previousType, firstMoveType, lastMoveType);
11   if asp =  $\alpha$  then
12     unchanged  $\leftarrow$  unchanged + 1;
13     if unchanged  $\geq$  10 then
14       moveType  $\leftarrow$  A;
15       unchanged  $\leftarrow$  0;
16     end
17   else
18     asp  $\leftarrow \alpha$ ;
19     unchanged  $\leftarrow$  0;
20   end
21   previousType = moveType;
22   switch moveType do
23     case A
24       | moveOnTippingBins();
25     case B
26       | moveOnSeparators();
27     case C
28       | moveOnPresses();
29     case D
30       | moveOnTj2Assignment();
31     case E
32       | moveOnRedTanks();
33     end
34   end
35   if  $\epsilon \leq \epsilon_{max}$  then
36     lastMoveType  $\leftarrow$  D;
37   end
38   if  $\alpha \leq \alpha_{max}$  then
39     firstMoveType  $\leftarrow$  E;
40   end
41   if (firstMoveType = E) and (lastMoveType = D) then
42     foundSolution = true;
43   end
44   counter  $\leftarrow$  counter + 1;
45 end

```

---

The variable  $\alpha$  is used to store the shortest makespan achieved during the tabu search and  $\epsilon$  denotes the best red fermentation evaluation score. The tabu search may terminate once both the makespan goal,  $\alpha_{\max}$ , and the red tank assignment goal,  $\epsilon_{\max}$ , have been achieved. If neither of the stopping criteria has been satisfied within *counterMaximum* iterations, the tabu search is terminated and the current best scores,  $\alpha$  and  $\epsilon$ , are considered as approximately optimal values. Once either of the stopping criteria has been reached, it is not necessary to consider moves to further improve the best encountered solution score. Therefore, whenever either criteria is satisfied, the list of moves is shortened in order to focus on the outstanding stopping criteria.

The variables *asp* and *unchanged* are employed in order to record the number of iterations during which the value of  $\alpha$  has remained unchanged. Whenever the number of iterations during which the value of  $\alpha$  remains unchanged reaches 10, a move of type A replaces the move generated by **generateMoveType**(previousType, firstMoveType, lastMoveType).

In order to illustrate the tabu search heuristic developed in this chapter, a small hypothetical example problem is considered.

**Example 6.7** *The initial solution generated in Example 6.4 is already an optimal solution for the specific problem set and, therefore, not the ideal example to illustrate the working of the cellar scheduling tabu search. Consider the same cellar environment and job characteristics as in Example 6.4, but suppose the arrival times are  $e = \{0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$  instead of  $e = \{5, 0, 3, 2, 0.5, 0, 2\}$ . Furthermore, suppose no task  $T_{j2}$  separator assignment is forced, i.e. no specific request is added that  $T_{j2}$  must be processed on a separator rather than on a press. When adopting the mathematical programming formulation in Chapter 5 with either of the objective functions described in §5.2.2, a makespan of  $C_{\max} = 12.25$  is achieved. Using the makespan as objective function results in a running time of 5 minutes when the example problem is solved in **Lingo 11.0** or a running time of 1 minute and 3 seconds when using the total completion time as objective function. Since the optimal makespan as well as the feasible red tank assignment (when the optimal makespan is considered) are already known,  $\epsilon_{\max}$  is taken as 35 (the evaluation score calculated from the known optimal solution) and  $\alpha_{\max}$  as 12.25. Therefore this tabu search will terminate once the optimal solution has been found or if the maximum number of iterations is reached. Normally the optimal values will not be known in advance, in which case a suitable bound will be used.*

When applying the algorithms illustrated in §6.1 to generate the initial solution for the example problem, the first assignment order matrix is

$$o = \begin{bmatrix} 0 & 1 & 2 & 3 & -1 & \dots \\ 0 & 4 & -1 & -1 & -1 & \dots \\ 0 & 5 & 6 & -1 & -1 & \dots \\ 0 & 4 & 6 & 5 & -1 & \dots \\ 0 & 4 & -1 & -1 & -1 & \dots \\ 0 & 5 & -1 & -1 & -1 & \dots \\ 0 & 4 & 5 & 6 & 7 & \dots \\ 0 & 4 & 5 & 7 & -1 & \dots \\ 0 & 7 & -1 & -1 & -1 & \dots \\ 0 & 7 & -1 & -1 & -1 & \dots \\ 0 & 3 & -1 & -1 & -1 & \dots \\ 0 & 1 & 2 & -1 & -1 & \dots \\ & & & \vdots & & \end{bmatrix}.$$

On closer inspection it seems that this order matrix is infeasible due to the order of jobs  $J_5$  and  $J_6$ , with  $J_5$  processed first on the presses (and tipping bins), but second on the separators. However, the order of these two jobs on the separators may be ignored for now due to the assignment of  $T_{52}$  and  $T_{62}$  only to the presses rather than to both the separators and the presses. This is not clear from the order matrix itself, but is determined by the vector  $\mathbf{b} = [1, 0, 0, 0, 1, 0, 0, 0]$ , specifying which jobs are processed on the separators.

The makespan of the initial solution is 18.75 and therefore both  $\alpha$  and  $asp$  are taken as 18.75 (lines 2 and 3, respectively, of Algorithm 6.22). Furthermore, the evaluation score,  $\epsilon$ , returned after the evaluation of the initial solution, is 35.

During the first iteration of the tabu search algorithm, move type  $D$  is selected. Since the generation of the move type is based on random selection, it will not necessarily be the first move type selected when reapplying the tabu search. When following the approach presented in Algorithm 6.18, a list of moves,

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

is first generated. As mentioned above, the order of assignment on the first separator,  $P_4$ , is currently infeasible if both  $\mathbf{b}(5)$  and  $\mathbf{b}(6)$  have a value of 1. Therefore, the only assignment order matrix that requires updating is at move  $a = 0$ . As expressed in line 20 of Algorithm 6.18, the order of the job assignments to the presses is preserved and the order of job assignments to the separators is updated. The updated rows 4 to 6 of the order matrix added, is

$$\mathbf{L}(0) = \begin{bmatrix} 0 & 4 & 5 & 6 & -1 & \dots \\ 0 & 4 & -1 & -1 & -1 & \dots \\ 0 & 5 & -1 & -1 & -1 & \dots \end{bmatrix}.$$

All other matrices  $\mathbf{L}(a)$  are equal to the current order matrix,  $\mathbf{o}$ .

The total amount of setup time assigned in a solution is found when sorting the list of moves ranging from the most favourable to the least favourable. However, the setup time is not considered when the aspiration and stopping criteria are applied. For all the candidate solutions resulting from the combination of lists  $\mathbf{S}$  and  $\mathbf{L}$ , total setup time is 6 hours. Furthermore, moves 0, 1 and 3 result in a makespan of 18.75, while evaluating move 2 delivers a makespan of 15.75. When selecting the best move following the general approach outlined in Algorithm 6.14, the moves are first sorted from most favourable to least favourable as  $\beta = [2, 0, 1, 3]$ . Since all the candidate solutions exhibit equal setup times, it has no influence on the move arrangement. Move  $\beta(0) = 2$  is not already in the tabu list (currently containing only  $\mathbf{b} = [1, 0, 0, 0, 1, 0, 0, 0]$  of the initial solution), the best move,  $\phi$ , is the combination  $\mathbf{S}(2)$  and  $\mathbf{L}(2)$ . Furthermore, the best makespan has been updated to  $\alpha = 15.75$  since  $\mathbf{v}(\phi) < \alpha$ . The selected move is added in position 0 of the current tabu list and remaining moves in the list are shifted up one position.

Currently,  $\alpha = 15.75$  and  $\epsilon = 35$ , indicating that an optimal assignment to the fermentation tanks has been found, but the assignments to the remaining processors are not yet optimal. Therefore, a new iteration is initiated during which a selection between move types  $B$  and  $C$  is considered — moves of type  $E$  are no longer considered and the previously selected move type is  $D$ , which is removed from consideration for this iteration only.

The move type chosen for iteration two, is move type  $C$ . The presses are influenced by this move type; therefore rows 7 to 9 of order matrix  $\mathbf{o}$  are considered. Following Algorithm 6.17

to generate and evaluate the list of moves, the first step is to update the order matrix to a maximum assignment matrix. The maximum assignment values for jobs  $J_4$  to  $J_7$  are 3, 2, 2 and 3, respectively. Rows 7 to 9 of the updated order matrix are therefore

$$\mathbf{o} = \begin{bmatrix} 0 & 4 & 5 & 6 & 7 & \dots \\ 0 & 4 & 5 & 7 & 6 & \dots \\ 0 & 7 & 4 & -1 & -1 & \dots \end{bmatrix}.$$

Adding job  $J_4$  to the assignment row for processor  $P_9$  and job  $J_6$  to the assignment row for processor  $P_8$  were required, since the other jobs already occurred at their maximum assignment values.

The size of the list  $\mathbf{L}_e$  is determined as 9. Rows 7 to 9 of the assignment order matrices in  $\mathbf{L}_e$ , are generated as

$$\begin{array}{l} 0: \begin{bmatrix} 0 & 5 & 4 & 6 & 7 & \dots \\ 0 & 4 & 6 & 7 & -1 & \dots \\ 0 & 5 & 4 & 7 & -1 & \dots \end{bmatrix}, \quad 1: \begin{bmatrix} 0 & 4 & 5 & 6 & 7 & \dots \\ 0 & 4 & 6 & 7 & -1 & \dots \\ 0 & 4 & 5 & 7 & -1 & \dots \end{bmatrix}, \quad 2: \begin{bmatrix} 0 & 7 & 4 & 6 & -1 & \dots \\ 0 & 7 & 5 & 4 & 6 & \dots \\ 0 & 7 & 5 & 4 & -1 & \dots \end{bmatrix}, \\ 3: \begin{bmatrix} 0 & 7 & 4 & 5 & 6 & \dots \\ 0 & 7 & 4 & 5 & -1 & \dots \\ 0 & 7 & 4 & 6 & -1 & \dots \end{bmatrix}, \quad 4: \begin{bmatrix} 0 & 5 & 7 & 4 & -1 & \dots \\ 0 & 5 & 7 & 4 & 6 & \dots \\ 0 & 7 & 4 & 6 & -1 & \dots \end{bmatrix}, \quad 5: \begin{bmatrix} 0 & 5 & 7 & 4 & -1 & \dots \\ 0 & 5 & 6 & 7 & 4 & \dots \\ 0 & 6 & 7 & 4 & -1 & \dots \end{bmatrix}, \\ 6: \begin{bmatrix} 0 & 7 & 5 & 6 & 4 & \dots \\ 0 & 7 & 5 & 4 & -1 & \dots \\ 0 & 7 & 6 & 4 & -1 & \dots \end{bmatrix}, \quad 7: \begin{bmatrix} 0 & 4 & 5 & 7 & -1 & \dots \\ 0 & 4 & 5 & 7 & 6 & \dots \\ 0 & 4 & 7 & 6 & -1 & \dots \end{bmatrix}, \quad 8: \begin{bmatrix} 0 & 4 & 7 & 5 & -1 & \dots \\ 0 & 4 & 7 & 6 & -1 & \dots \\ 0 & 4 & 7 & 5 & 6 & \dots \end{bmatrix}.$$

The next list of moves,  $\mathbf{L}_r$ , contains 42 assignment order matrices all derived from  $\mathbf{L}_e$ . For example,

$$\begin{bmatrix} 0 & 5 & 4 & 7 & -1 & \dots \\ 0 & 4 & 6 & 7 & 6 & \dots \\ 0 & 5 & 4 & 7 & -1 & \dots \end{bmatrix}.$$

shows rows 7 to 9 of  $\mathbf{L}_r(2)$  derived from  $\mathbf{L}_e(0)$ . List  $\mathbf{L}_r$  requires no updating, since only one job ( $J_6$ ) is actively assigned to the separators. Therefore, there exists no order on the separators. When selecting the best move, both the makespan  $\mathbf{v}(a)$  and the total setup time obtained from the order matrix are considered. These two values are listed in Table 6.3 for each of the forty-two lists  $\mathbf{L}_e(a)$ .

The shortest makespan appearing in Table 6.3, is 12.25. However, two moves results in the shortest makespan, namely  $\mathbf{v}(8)$  and  $\mathbf{v}(9)$ . On further inspection,  $\mathbf{v}(a)$  has a smaller total setup time and is therefore the best solution to consider from the list  $\mathbf{L}_r$ . Applying this move results in the order matrix

$$\mathbf{o} = \begin{bmatrix} 0 & 1 & 2 & 3 & -1 & \dots \\ 0 & 4 & -1 & -1 & -1 & \dots \\ 0 & 5 & 6 & -1 & -1 & \dots \\ 0 & 4 & 6 & 5 & -1 & \dots \\ 0 & 4 & -1 & -1 & -1 & \dots \\ 0 & 5 & -1 & -1 & -1 & \dots \\ 0 & 4 & 5 & 7 & -1 & \dots \\ 0 & 4 & 6 & 7 & -1 & \dots \\ 0 & 5 & 7 & -1 & -1 & \dots \\ 0 & 7 & -1 & -1 & -1 & \dots \\ 0 & 3 & -1 & -1 & -1 & \dots \\ 0 & 1 & 2 & -1 & -1 & \dots \end{bmatrix}.$$

$a$	$v(a)$	$\sum s_{ijk}$	$a$	$v(a)$	$\sum s_{ijk}$	$a$	$v(a)$	$\sum s_{ijk}$
0	16.20	8.5	1	16.20	7.5	2	16.20	7.5
3	16.20	6.5	4	16.20	7.5	5	16.20	6.5
6	15.75	8.5	7	15.75	7.5	8	12.25	7.5
9	12.25	6.5	10	15.75	7.5	11	15.75	6.5
12	22.50	10.0	13	22.50	9.0	14	22.50	9.0
15	22.50	8.0	16	22.50	9.0	17	22.50	8.0
18	22.50	10.0	19	22.50	9.0	20	22.50	9.0
21	22.50	7.0	22	20.20	11.0	23	20.20	10.0
24	20.20	10.0	25	20.20	8.0	26	17.20	11.5
27	17.20	9.5	28	17.20	10.0	29	17.20	8.0
30	22.50	10.0	31	22.50	9.0	32	22.50	9.0
33	22.50	7.0	34	16.75	10.5	35	16.75	9.0
36	16.75	8.5	37	16.75	7.0	38	19.75	10.0
39	19.75	10.0	40	16.25	10.5	41	16.25	9.0

Table 6.3: The makespan,  $v(a)$ , for every move  $a$  in list  $L_r$  from Example 6.7 and the total setup time required,  $\sum s_{ijk}$ , is listed. The highlighted cells indicate the occurrence of the shortest makespan, 12.25, and smallest total setup time, 6.5.

and also fulfils the stopping criteria. ■

For this example problem an optimal solution was found in only 2 iterations. However, the success of the tabu search may vary from one application to the next. In order to evaluate other possibilities and also obtain a better indication of processing times, the tabu search described above was applied 10 000 times to the test problem in Example 6.7 with varying maximum numbers of iterations. The outcome of this experiment are listed in Table 6.4.

Maximum iterations	Optimal solutions found		
	Number	Percentage	Average processing time
200	10 000	100%	0.07
100	9 940	99%	0.06
50	9 566	96%	0.06
25	8 812	88%	0.05
10	7 148	71%	0.04

Table 6.4: The number and percentage of optimal moves found when running a Java implementation of the tabu search 10 000 times to repeatedly solve the example problem from Example 6.7 for different considered maximum iteration values, as well as the average processing time per run of the Java implementation.

When the maximum number of allowed iterations is set to 200, an optimal solution (as determined by the exact solution of the same problem) is found for each of the 10 000 applications. As the maximum number of allowed iterations decreases, the reliability of the heuristic decreases in the sense that an optimal solution is found less frequently, even though the majority of the solutions are close to optimal. For example, if the optimal makespan of 12.25 is replaced by 13.35 as the stopping criteria,  $\alpha_{\max}$ , it is found that 93% of the 10 000 solution makespans are smaller than 13.35 when only allowing 10 iterations. Furthermore, the average processing time also decreases with the decrease in the maximum number of iterations, as expected.



The proposed tabu search may therefore be viewed as a very strong competitor to the exact solution method described in Chapter 5. The efficiency of this tabu search approach may now be further explored to determine whether the heuristic reliability and processor time is still a suitable solution technique when applied to the actual cellar scheduling problem experienced at Wamakersvallei Winery.

## 6.5 Chapter overview

In this Chapter, a tabu search approach towards solving the cellar scheduling problem was developed, the goal of the tabu search being to indicate whether or not a feasible production schedule may be found for a specific harvesting day in order to comply with thesis objective I expressed in Chapter 1. A solution is feasible if it adheres to the constraint set defined in §5.2.1 and if the solution may be carried out during the business hours of the cellar.

In §6.1 the process of generating the initial solution for the tabu search was considered. The focus is first on the assignment of the jobs to the tipping bins. Then the assignment of all further tasks is considered in three sections covering jobs of Types I, II and III.

Evaluating a suggested solution is performed by two evaluation techniques discussed in §6.2. The first technique, referred to as the cellar packing algorithm, is described in §6.2.1 and is used to approximate the makespan of the suggested solution. In the case where two solutions have the same makespan, the total setup time required to process jobs in the suggested order is used to distinguish between a good solution and an even better one. Since the fermentation process in jobs of Type I is not considered when computing the makespan, another evaluation technique is described in §6.2.2. This technique is used to evaluate the assignment of Type I jobs to the red fermentation tanks by producing a solution score consisting of the total wasted space caused by the assignment and, more importantly, the total weight of all Type I jobs not assigned to a fermentation tank.

Further tabu search specifications concerned with the generation and selection of moves were considered in §6.3. In §6.3.1, an ejection chain move used to generate at least a part of the list of moves for four out of the five move types was proposed. Further techniques required to identify feasible moves were also considered. The five move types were described in some detail in §§6.3.2–6.3.6.

The tabu search developed was finally applied to a small, hypothetical example problem in §6.4. The tabu search was implemented in Java and the solution uncovered by the heuristic was compared to that of the exact method from Chapter 5. It was found, based on the processing time and the reliability of the heuristic, that it is definitely a better technique for the small cellar scheduling problem considered compared to the exact method previously considered. For the example problem considered, the processing time of the heuristic was approximately 1 000 times as fast as computing an exact solution, with an optimal solution found for all 10 000 repetitions of the problem considered in §6.4.



---

---

## CHAPTER 7

---

# The harvest scheduling problem at Wamakersvallei Winery

### Contents

7.1	Defining the harvest scheduling attributes . . . . .	129
7.2	The initial harvest schedule . . . . .	131
7.3	Evaluating a harvest schedule . . . . .	133
7.3.1	<i>Creating cellar scheduling scenarios</i> . . . . .	133
7.3.2	<i>Calculating the harvest evaluation score</i> . . . . .	137
7.4	Generating candidate moves and selecting the best move . . . . .	141
7.5	Solving the harvest scheduling problem with the tabu search . . . . .	145
7.6	Chapter overview . . . . .	145

In this chapter the scheduling problem concerned with the harvesting of vineyard blocks is considered. The goal in this chapter is to find a generic solution method to the harvest scheduling problem with grapes being harvested as close as possible to their optimal ripeness. Depending on the wine cellar and job characteristics of the winery where the harvest scheduling problem occurs, some minor changes is to be expected in order to have the developed decision support system perform as well as possible. As an example, these changes are noted at the start of Chapter 8 when the decision support system is developed for and adjusted to fit the scheduling problems at Wamakersvallei Winery. The cellar restrictions outlined in Chapters 5 and 6 are also considered in order to achieve a realistic solution. The harvest scheduling tabu search application is discussed in §§7.2–7.5, starting with the process of generating an initial solution in §7.2. Then, a means of evaluating harvest schedules is considered in §7.3, after which the generation of candidate moves as well as the selection of the most suitable move is discussed in §7.4. The final tabu search application is described in §7.5. First, some harvest scheduling attributes are defined and expressed mathematically in §7.1.

### 7.1 Defining the harvest scheduling attributes

The job set,  $\mathcal{J}$ , employed in the previous chapters concerning the active cellar scheduling refers to the individual grape loads as they are received at the cellar and not to the vineyard blocks as a whole. Instead, the set  $\mathcal{B} = \{B_1, \dots, B_N\}$  is used to refer to each of the vineyard blocks

considered. The total number of blocks considered for the harvesting scheduling problem over a specified period of time,  $N$ , is further divided into the batches (truck loads) received at the cellar<sup>1</sup>. It is important to note that the value of  $N$  does not refer to the total number of vineyard blocks from which grapes are received, but rather the smaller selection of blocks considered for harvesting within a specific time frame of the harvest scheduling problem. Only blocks that are within a certain ripeness interval are considered, otherwise the scheduling problem becomes unnecessarily complex.

A harvesting schedule is generated for a specified number of days,  $D$ . The schedule is displayed in the form of a set of vectors, referred to as  $\mathbf{H}$ , where  $\mathbf{H}(d)$  refers to the vineyard blocks to be harvested on day  $d$  of the harvesting schedule. This vector does not indicate the order in which these blocks are harvested or received at the cellar. For example, the schedule

$$\mathbf{H} = \left\{ \begin{array}{l} [ B_{b_1} \quad B_{b_2} \quad -1 \quad \cdots ], \\ [ B_{b_3} \quad -1 \quad -1 \quad \cdots ], \\ \vdots \\ [ B_{N-1} \quad B_N \quad -1 \quad \cdots ] \end{array} \right\}$$

indicates that the blocks currently numbered as blocks  $B_{b_1}$  and  $B_{b_2}$  should be harvested on the first day of the harvest schedule and blocks  $B_{N-1}$  and  $B_N$  on the last, day  $D$ .

A block class (determined as outlined in Chapter 4) is associated with each of the vineyard blocks. These block classes range from 1 to 3, with 1 being the best. Each vineyard block,  $B_b$ , is assumed to yield an expected grape volume,  $W_b$ , expressed in tonnes. Since a vineyard block is most often too large to deliver to the cellar in one truckload, the blocks arrive at the cellar in smaller batches.

The most significant attribute of a good harvesting schedule, is that all grapes should be received at the cellar when they are as close as possible to their optimal ripeness. The ripeness sugar intervals considered in Table 4.4 may be employed to arrange the blocks in order of urgency to be harvested. Consider the variable  $\varsigma_{\chi,\kappa} = [a, b]$ , where  $\chi$  indicates the *sugar class* of the grapes. This sugar class does not refer to the actual expected class of the block, but rather to the number of the sugar level interval, as expressed in Table 4.4. The cultivar is indicated by  $\kappa$ , with cultivars numbered in their order of appearance in Table 4.4. Here  $\kappa \in \{1 \dots 11\}$  denotes the red grape varieties, Cabernet Franc, Cabernet Sauvignon, Shiraz, Petit Verdot, Pinotage, Merlot, Mourvedre, Malbec, Roobernet, Ruby Cabernet, and Cinsaut. The white grape varieties are denoted by  $\kappa \in \{12 \dots 19\}$  indicating Chenin Blanc, Sauvignon Blanc, Chardonnay, Viognier, Colombar, Hanepoot, SA Riesling or Weisser Riesling. The interval  $[a, b]$  refers to the given interval of the sugar levels. For example, for Cabernet Sauvignon of Class 2,  $\varsigma_{2,2} = [23, 27]$ . This variable holds for the classes  $\chi \in \{1, 2, 3\}$  and any block with a sugar level outside the interval ( $\varsigma_{3,\kappa}$ ) is considered the worst block class<sup>2</sup>.

Grape samples are received on a bi-weekly<sup>3</sup> basis from each of the vineyard blocks. A list of recent samples and sugar levels is maintained by the viticulturist [101]. The most recent sugar level of block  $B_b$  is referred to as  $\zeta_b$ . The initial harvest schedule of the tabu search application conceived to solve the harvest scheduling problem may now be described.

<sup>1</sup>The number of jobs,  $n$ , is equal to the number of jobs batches arriving at the cellar and adding the number of red fermentation tanks to be emptied (Type III jobs).

<sup>2</sup>In Table 4.4, the worst class is listed as class 6. In this chapter, the worst class is considered class 4 (rather than jumping to the number 6) in order to achieve a more compact representation of algorithms to follow.

<sup>3</sup>Receiving samples from every vineyard block at least once every two weeks is the desired minimum number of samples. However, in reality the samples are not always received as consistently as desired [127].

## 7.2 The initial harvest schedule

As mentioned in Chapter 6, it is essential to have a good initial solution for a tabu search to be as successful as possible. Two aspects are of concern when evaluating a harvesting schedule. The first (and most important) is the ripeness of the grapes received. The second aspect is concerned with the cellar capacity and processing time. From the list of most recent sample sugar levels of vineyard blocks, a reduced list of blocks containing all blocks with sugar levels above  $\varsigma_{3,\kappa}(1)$  for the relevant value of  $\kappa$  may be deduced. In the case where the required harvesting schedule period falls outside an acceptable range, for example  $D = 100$ , the full range of blocks may be added to the list of considered blocks. However, this is further discussed in the next section. An example is used to illustrate the list characteristics.

**Example 7.1** Consider the same fictitious cellar used in the examples of Chapters 5 and 6. This cellar has a total of 15 vineyard blocks from which grapes are received. In Table 7.1, the vineyard blocks for which sugar samples have been received are listed together with the expected yield,  $W_b$ , and class of each vineyard block  $B_b$ . The sugar level,  $\zeta_b$ , determined from the most recent sample of each block and the value of  $\varsigma_{3,\kappa}(1)$  are listed to indicate how far the grapes are from reaching maturity.

$B_b$	Cultivar	$W_b$	Class	$\zeta_b$	$\varsigma_{3,\kappa}(1)$
$B_{b_1}$	Cabernet Sauvignon	50	1	25	21
$B_{b_2}$	Cabernet Sauvignon	40	1	24	21
$B_{b_3}$	Cabernet Sauvignon	25	1	26	21
$B_{b_4}$	Cabernet Sauvignon	25	2	23	21
$B_{b_5}$	Cabernet Sauvignon	10	2	27	21
$B_{b_6}$	Merlot	50	1	20	22
$B_{b_7}$	Merlot	30	1	21	22
$B_{b_8}$	Pinotage	40	1	23	22
$B_{b_9}$	Pinotage	20	3	24	22
$B_{b_{10}}$	Pinotage	10	1	23	22
$B_{b_{11}}$	Chardonnay	30	1	24	18
$B_{b_{12}}$	Chardonnay	15	1	25	18
$B_{b_{13}}$	Sauvignon Blanc	45	1	17	18
$B_{b_{14}}$	Sauvignon Blanc	20	1	17	18
$B_{b_{15}}$	Sauvignon Blanc	15	2	16	18

Table 7.1: The 15 vineyard blocks and the most recent sample sugar levels of the cellar considered in Example 7.1.

Suppose the required harvest schedule should span a period of 5 days ( $D = 5$ ). Comparing the sugar levels listed in Table 7.1 with the lower bound of the acceptable intervals also listed (when  $\zeta_b \geq \varsigma_{3,\kappa}(1)$ ), results in the list of ten blocks  $\{B_{b_1}, B_{b_2}, B_{b_3}, B_{b_4}, B_{b_5}, B_{b_8}, B_{b_9}, B_{b_{10}}, B_{b_{11}}, B_{b_{12}}\}$  to consider for the scheduling period. These blocks are renumbered as  $B_1, \dots, B_{10}$  in the order listed and, therefore,  $N = 10$ . ■

The blocks considered for the harvest scheduling period are now ordered in a list according to the optimal ripeness levels. Algorithm 7.1 outlines the ordering process for each of the  $N$  blocks considered. Not only is the  $\chi$  value considered, but in cases where  $\chi \neq 1$ , the value by which a better class is missed, is also considered. When the list of blocks is sorted according to the

two vectors calculated, the vector *sampleClass* is used to sort the list in ascending order and *difference* is used as a tiebreaker, also sorting the list in ascending order of difference. Algorithm 7.1 is further explained in the following example.

**Example 7.2 (Example 7.1 continued)** For the first block,  $B_1$ ,  $\kappa = 2$  since Cabernet Sauvignon is cultivar number 2. Block  $B_1$  is considered to be in sugar class interval 1 since  $\zeta_1 = 25$  and the relevant interval is expressed as  $\varsigma_{1,2} = [24, 26]$ . Then, the value of  $\text{sampleClass}(1)$  is 1 and  $\text{difference}(b) = 26 - 25 = 1$ . This process is continued until  $\text{sampleClass} = [1, 1, 1, 2, 2, 2, 1, 2, 1, 1]$  and  $\text{difference} = [1, 2, 0, 1, -1, 1, 3, 1, 1, 0]$ .

When the list of blocks are arranged in ascending order primarily considering *sampleClass* with *difference* applied as a tiebreaker, the list of blocks in scheduling order is  $\{B_3, B_{10}, B_9, B_1, B_2, B_7, B_5, B_4, B_6, B_8\}$ . ■

---



---

**Algorithm 7.1:** Generating the order of assignment for the initial harvesting schedule

---

**Input:** The set of blocks to consider,  $\mathcal{B}$ , their sample sugar levels,  $\zeta_b$  and all the sugar intervals  $\varsigma_{\chi,\kappa}$ .

**Output:** The ordered list of blocks expressed in *blockOrder*.

```

1 for  $b = 1$  to  $N$  do
2    $\kappa \leftarrow \kappa_b$ ;  $\chi \leftarrow 1$ ;
3    $\text{classFound} \leftarrow \text{false}$ ;
4   while  $\text{classFound} = \text{false}$  do
5     if  $(\zeta_b - \varsigma_{\chi,\kappa}(1) \geq 0)$  and  $(\zeta_b - \varsigma_{\chi,\kappa}(2) \leq 0)$  then
6       |  $\text{classFound} \leftarrow \text{true}$ ;
7     else
8       |  $\chi \leftarrow \chi + 1$ ;
9       | if  $\chi \geq 4$  then
10        |    $\text{classFound} \leftarrow \text{true}$ ;
11        |   end
12        | end
13      end
14      if  $(\chi > 1)$  and  $(\varsigma_{\chi-1,\kappa}(1) \geq 0)$  then
15        |  $d \leftarrow \varsigma_{\chi-1,\kappa}(1) - \zeta_b$ ;
16        | if  $d < 0$  then
17          |    $\text{difference}(b) \leftarrow \varsigma_{\chi-1,\kappa}(2) - \zeta_b$ ;
18          | else  $\text{difference}(b) \leftarrow d$ ;
19        | else if  $\chi = 1$  then
20          |    $\text{difference}(b) \leftarrow \varsigma_{\chi,\kappa}(2) - \zeta_b$ ;
21        | end
22         $\text{sampleClass}(b) \leftarrow \chi$ ;
23      end
24  $\text{blockOrder} \leftarrow \text{sortWithDoubleCriteria}(\text{sampleClass}, \text{difference});$ 

```

---

In order to generate the initial solution, the total weight of the expected yield of all the blocks in the list is considered. This weight is compared to the average yield tonnage per harvesting day of previous years, referred to as  $\bar{x}$  in order to estimate the number of blocks to be scheduled for harvesting per day. For each of the first  $D - 1$  days, the total weight of the expected yield for the scheduled blocks should be at least as large as  $\bar{x}$  with all remaining blocks scheduled for

harvesting on day  $D$ . However, when  $\bar{W}$  is defined as

$$\bar{W} = \frac{1}{D} \sum_{b=1}^N W_b,$$

and  $\bar{W} > \bar{x}$ , the total expected weight should be at least as large as  $\bar{W}$  (rather than  $\bar{x}$ ) per day until all the blocks are assigned. The generation of the initial solution is illustrated in the following example.

**Example 7.3 (Example 7.2 continued)** *Suppose the average daily intake of grapes of the last harvesting season for the hypothetical cellar, is 80 tonnes setting the value of  $\bar{x}$  as 80. A harvesting schedule is generated for  $D = 5$  days. Therefore,*

$$\bar{W} = \frac{1}{5} \sum_{b=1}^{10} W_b,$$

*results in  $\bar{W} = 53$  and since  $\bar{W} < \bar{x}$ , the average intake over past seasons,  $\bar{x}$ , is employed to determine the number of blocks harvested each day. The initial harvesting schedule for this example problem is therefore*

$$\mathbf{H} = \left\{ \begin{bmatrix} B_3 & B_{10} & B_9 & B_1 \end{bmatrix}, \begin{bmatrix} B_2 & B_7 & B_5 & B_4 \end{bmatrix}, \begin{bmatrix} B_6 & B_8 & -1 & -1 \end{bmatrix}, \begin{bmatrix} -1 & -1 & -1 & -1 \end{bmatrix}, \begin{bmatrix} -1 & -1 & -1 & -1 \end{bmatrix} \right\}.$$

*The grape intake per day is then  $W_3 + W_{10} + W_9 + W_1 = 120$  for day 1,  $W_2 + W_7 + W_5 + W_4 = 95$  for day 2 and  $W_6 + W_8 = 50$  for day 3. Days 4 and 5 are not assigned any blocks. ■*

## 7.3 Evaluating a harvest schedule

As mentioned in the previous section, a good harvest schedule will not only have grapes scheduled for harvesting at the right time, but will also consider the restrictions imposed by processor availability and capacity. In order to consider these restrictions, a number of likely *cellar scenarios* are generated and evaluated based on the tabu search presented in Chapter 6. The generation of these scenarios and the evaluation thereof is presented in §7.3.1, after which the evaluation score of a harvest schedule is discussed in §7.3.2. Both these sections contain the generic approach to evaluating the harvesting schedule. However, when applying the developed model to a winery, each winery will likely contain a set of changes to the evaluation method in order to better suit the expectancies of the scheduler and also to consider the best approach for the winery size and characteristics. The changes required in order for the developed decision support system to best suit the Wamakersvallei cellar, is discussed in Chapter 8

### 7.3.1 Creating cellar scheduling scenarios

Each determined scenario will rely on three attributes. First, the number of batches,  $s_b$ , a vineyard block is split into when delivered to the cellar, then the order in which these batches arrive at the cellar and lastly, the distribution of their arrival times.

In order to achieve an approximation to the number of batches a single vineyard block,  $B_b$ , is split into,  $s_b$ , the harvesting data for 2007 are considered. The general distribution of these batches were studied without considering the specific grape cultivar, only considering the yield of the vineyard block and the number of batches in which it arrives at the cellar. The frequencies with which vineyard blocks within a specified size interval are split into different batches are shown in Tables 7.2–7.4.

Datapoints	Number of splits									
	1	2	3	4	5	6	7	8	9	
(0,3]	20	100.0								
(3,6]	56	57.1	39.3	3.6						
(6,9]	60	30.0	45.0	16.7	5.0	3.3				
(9,12]	39	5.1	38.5	23.1	28.2	5.1				
(12,15]	40		30.0	20.0	25.0	17.5	7.5			
(15,18]	24		12.5	29.2	16.7	25.0	8.3	8.3		
(18,21]	34		8.8	35.3	20.6	11.8	8.8	2.9	5.9	5.9

Table 7.2: The frequency, expressed as a percentage of the datapoints, of the occurrence of a block split into  $s_b$  batches for vineyard blocks smaller than 21 tonnes.

Datapoints	Number of splits											
	2	3	4	5	6	7	8	9	10	11	12	
(21,31]	53	1.9	13.2	24.5	22.6	15.1	15.1	3.8	3.8			
(31,41]	28			7.1	25.0	39.3	3.6	7.1	14.3	0	3.6	
(41,51]	20				5.0	45.0	10.0	10.0	5.0	5.0	10.0	10.0

Table 7.3: The frequency, expressed as a percentage of the datapoints, of the occurrence of a block split into  $s_b$  batches for vineyard blocks between 21 and 51 tonnes.

Datapoints	Number of splits								
	6	7	8	9	10	11	12	13	
(51,71]	31	6.5	9.7	29.0	12.9	25.8	9.7	3.2	3.2
(71,91]	20		5.0			15.0	20.0	35.0	25.0

Table 7.4: The frequency, expressed as a percentage of the datapoints, of the occurrence of a block split into  $s_b$  bathes for vineyard blocks between 51 and 71 tonnes.

The intervals considered in Tables 7.2–7.4 vary over a range of different sizes with the smaller blocks divided into intervals of width three tonnes as listed in Table 7.2. The reason for choosing intervals to be of width three tonnes, is simply due to the fact that all vineyard blocks smaller than three tonnes are delivered in a single batch. In Table 7.3, the medium-sized blocks are divided into intervals of width 10 tonnes and in Table 7.4, the larger blocks are divided into intervals of width 25 tonnes.

There is also a last distribution for vineyard blocks with a yield larger than 91 tonnes. However, this distribution is based on the average and standard deviation of the size of each of the individual batches. In the 2007 harvesting data, there are only 24 datapoints for vineyards with a yield greater than 91 tonnes. Furthermore, these datapoints are spread over vineyard blocks sized up to 215 tonnes. Hence the distribution of these datapoints are too sparse to facilitate



the construction of a similar frequency table and so a different approach is considered for large vineyard blocks.

When a vineyard block has an expected yield larger than 91 tonnes, the focus shifts to the distribution of the different sized bathes rather than the number of batches a vineyard block is split into. These frequencies are listed in Table 7.5 and is used to determine the number of batches a vineyard block with yield larger than 91 tonnes is split into.

Interval	Frequency
(4,5]	1
(5,6]	1
(6,7]	6
(7,8]	12
(8,9]	2
(9,10]	1

Table 7.5: The frequency of a batch size occurring within the specified interval for vineyard blocks larger than 91 tonnes, based on the 24 datapoints from the 2007 harvesting data.

When generating the batch distribution of a specific scenario, a number is generated randomly for each vineyard block. Depending on the size of the vineyard block and the percentages expressed in Tables 7.2–7.4, the number of batches this block will be split into is calculated accordingly. When a vineyard block with yield larger than 91 tonnes is considered, the random number,  $r$ , is generated in the interval  $(0,23]$ . From the random number, an interval for the batch size is calculated with the use of the list  $\{0, 1, 2, 8, 20, 22, 23\}$ . For example, if  $0 < r \leq 1$ , the first interval is chosen. Once the interval has been determined, another random number is generated indicating the exact batch size within the interval after which the number of batches the vineyard block is split into, may be deduced. For the sake of simplicity, the arrival batches of one vineyard block are chosen to be of equal size. Although this is not necessarily the case, it will not impact the accuracy of the model. Machinery inside the cellar is set to a specified cycle and the batch size has no influence on the cycle time.

The matrix

$$\mathbf{s} = \begin{bmatrix} 0 & 3 & 100 & - & - & - & - & - & - & - & - & - & - & - & - \\ 3 & 6 & 57.1 & 96.4 & 100 & - & - & - & - & - & - & - & - & - & - \\ 6 & 9 & 30 & 75 & 91.7 & 96.7 & 100 & - & - & - & - & - & - & - & - \\ 9 & 12 & 5.1 & 43.6 & 66.7 & 94.9 & 100 & - & - & - & - & - & - & - & - \\ 12 & 15 & 0 & 30 & 50 & 75 & 92.5 & 100 & - & - & - & - & - & - & - \\ 15 & 18 & 0 & 12.5 & 41.7 & 58.4 & 83.4 & 91.7 & 100 & - & - & - & - & - & - \\ 18 & 21 & 0 & 8.8 & 44.1 & 64.7 & 76.5 & 85.3 & 88.2 & 94.1 & 100 & - & - & - & - \\ 21 & 31 & 0 & 1.9 & 15.1 & 39.6 & 62.2 & 77.3 & 92.4 & 96.2 & 100 & - & - & - & - \\ 31 & 41 & 0 & 0 & 0 & 7.1 & 32.1 & 71.4 & 75 & 82.1 & 95.4 & 95.4 & 100 & - & - \\ 41 & 51 & 0 & 0 & 0 & 0 & 5 & 50 & 60 & 70 & 75 & 80 & 90 & 100 & - \\ 51 & 71 & 0 & 0 & 0 & 0 & 0 & 6.5 & 16.2 & 45.2 & 58.1 & 83.9 & 93.6 & 96.8 & 100 \\ 71 & 91 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 5 & 5 & 20 & 40 & 75 & 100 \end{bmatrix}$$

was constructed from Tables 7.2–7.4 for the purpose of generating the scenarios. If  $\mathbf{s}(\text{row}, 1) < W_b \leq \mathbf{s}(\text{row}, 2)$  holds for weight  $W_b$  of block  $B_b$  and  $r$  is a randomly generated number between 0 and 100, then the number of batches that block  $B_b$  is split into is calculated as the largest value of  $x$  for which  $x \leq \mathbf{s}(\text{row}, x + 2)$ . These batches are the jobs considered in Chapters 5 and 6.

**Example 7.4 (Example 7.3 continued)** Consider the list of blocks  $\{B_1, \dots, B_{10}\}$  in Example 7.3 and consider the following list,  $\mathbf{r} = \{33.9, 87.9, 32.1, 48.9, 7.0, 67.0, 68.4, 5.7, 10.3, 86.2\}$ , of 10 random numbers. First consider block  $B_1$ , which has weight  $W_1 = 50$ . This places block  $B_1$  in row 10 and since  $\mathbf{r}(1) = 33.9$  lies between  $\mathbf{s}(10, 7)$  and  $\mathbf{s}(10, 8)$ , block  $B_1$  is split into 6 batches. Applying the same technique for the remaining blocks, blocks  $B_2$  to  $B_{10}$  are divided into 9, 4, 5, 2, 6, 5, 2, 3 and 5 batches respectively. The resulting jobs are shown in Table 7.6. ■

Jobs	Cultivar	Weight $w_j$	Class
$J_1, \dots, J_6$	Cabernet Sauvignon	8.3	1
$J_7, \dots, J_{15}$	Cabernet Sauvignon	4.4	1
$J_{16}, \dots, J_{19}$	Cabernet Sauvignon	6.3	1
$J_{20}, \dots, J_{24}$	Cabernet Sauvignon	5	2
$J_{25}, J_{26}$	Cabernet Sauvignon	5	2
$J_{27}, \dots, J_{32}$	Pinotage	6.7	1
$J_{33}, \dots, J_{37}$	Pinotage	4	3
$J_{38}, \dots, J_{39}$	Pinotage	5	1
$J_{40}, \dots, J_{44}$	Chardonnay	10	1
$J_{45}, \dots, J_{49}$	Chardonnay	3	1

Table 7.6: The resulting jobs after the 10 blocks considered in Example 7.4 have been split into batches.

The next attribute of the scenario is the order and timing of the arriving jobs. The arrival of the trucks at the cellar start early in the morning, with the exact starting time depending on how far into the harvesting season they are, and then steadily picks up until the majority of the grapes have been received by approximately 14h00. The frequency of the arriving truckloads then steadily drops until the end of the day [101]. Consider the vector,  $\mathbf{a} = \{0.05, 0.10, 0.15, 0.35, 0.60, 0.70, 0.85, 1.00, 1.00, 1.00\}$  indicating the distribution of scheduled jobs received over time. Let  $b$  denote the number of business hours during which the cellar is open. No more than  $\mathbf{a}(x)$  of jobs scheduled for arrival at the cellar have been received by  $\frac{1}{10}xb$  hours. The fact that no more than  $\mathbf{a}(x)$  of the jobs is received by the set time, means that even all jobs are supposed to have been received, some jobs may still arrive at the cellar. For this reason and in order to fit the arrival pattern discussed with the viticulturist [101], the last three entries of  $\mathbf{a}$  is set as 1.0. For example, consider  $b = 12$ , then the business day is divided into parts of 1.2 hours ( $\frac{1}{10} \times 12$ ) each. If  $x = 1$ , then a proportion no more than  $\mathbf{a}(1) = 0.05$  of all scheduled arriving jobs have been received before 1.2 hours ( $\frac{1}{10} \times 12$ ) into the day. This deduction may be made for  $x = 2, \dots, 10$  until, finally, all jobs have been received after  $b = 12$  hours. The process of determining the arrival times from vector  $\mathbf{a}$  is outlined in Algorithm 7.2.

The new variable,  $n'$ , refers to the number of jobs of Types I and II collectively. The function  $\mathbf{rand}(0, z)$  generates a random integer between 0 and  $z$ ; therefore the constant division by 10 or 100 in order to include decimal numbers in the selection. The variable,  $x$  may be considered as the number of arrival times that must be generated in one iteration of the for-loop, while  $y$  refers to the total number of arrival times generated so far. The final arrival times of the  $n'$  jobs is then returned as the vector  $\mathbf{e}$ . A random permutation is applied to  $\mathbf{e}$  in order to consider the different orders in which the jobs may arrive.

**Example 7.5 (Example 7.4 continued)** Consider the  $n' = 49$  jobs generated in Example 7.4 and listed in Table 7.6. Algorithm 7.2 is applied to generate the individual arrival times

when the length of one business day is considered as 8 hours. Starting at  $p = 0$ , the value of  $x$  is 2 and the value of  $z$  is 4 (referring to the first 0.4 hours). The value of  $r$  is generated as 2 indicating that the first arrival time to be assigned in position 1 of  $\mathbf{e}$ , is 0.2 hours. The value of  $x$  is decreased, indicating that one fewer arrival time still needs to be generated in this for-loop iteration, and  $y$  is increased since the number of arrival times generated increases by 1. Since  $x > 0$  still holds, another arrival time is generated as  $\mathbf{e}(2) = 0.1$ , after which  $p = 1$  is considered. The algorithm continues in this fashion until the vector  $\mathbf{e}$  has been filled with the generated arrival times. ■

---



---

**Algorithm 7.2:** Generating arrival times

---

**Input:** The set business hours and the list of jobs  $J_1, \dots, J_n$ .  
**Output:** The arrival times,  $\mathbf{e}$ , for jobs  $J_1, \dots, J_{n'}$ .

```

1  $\mathbf{a} \leftarrow \{0.05, 0.10, 0.15, 0.35, 0.60, 0.70, 0.85, 1.00, 1.00, 1.00\}$ ;
2  $n' \leftarrow (n - |\text{Type III jobs}|)$ ;
3  $x \leftarrow 0$ ;
4  $y \leftarrow 0$ ;
5 for  $p = 0$  to 9 do
6    $x \leftarrow \lfloor (\mathbf{a}(p) n') \rfloor - y$ ;
7    $z \leftarrow \lfloor (\mathbf{a}(p) b) / 10 \rfloor$ ;
8   while  $x > 0$  do
9      $r \leftarrow \text{rand}(0, z)$ ;
10     $y \leftarrow y + 1$ ;
11     $x \leftarrow x - 1$ ;
12     $\mathbf{e}(y) \leftarrow r/10$ ;
13  end
14 end
15 while  $y < n'$  do
16    $r \leftarrow \text{rand}(0, 10b)$ ;
17    $\mathbf{e}(y - 1) \leftarrow r/10$ ;
18    $y \leftarrow y + 1$ ;
19 end
20  $\mathbf{e} \leftarrow \text{applyRandomPermutation}(\mathbf{e})$ ;
21  $\mathbf{e}(0) \leftarrow -1$ ;
```

---

The created scenarios may now be used in order to evaluate a harvesting schedule, by means of calculating a harvest evaluation score.

### 7.3.2 Calculating the harvest evaluation score

There are three main factors to be taken into account in order to successfully evaluate a harvesting schedule. First, there is the (expected) ripeness of the grapes from the selected vineyard block on the day it is scheduled for harvesting. Then there are the two factors inherited from the cellar scheduling problem, namely the total processing time and the lack of space and wasted space of job assignment to the red fermentation tanks. In order to generate a harvest evaluation score based on a sensible, weighted combination of these three factors, the importance of the three factors are considered. However, before the different contributions or any further aspects of the generation of a harvest evaluation score may be discussed, the concept of a harvesting schedule,  $\mathbf{H}$ , requires some alteration.

As described earlier in this chapter, the harvesting schedule,  $\mathbf{H}$ , contains the blocks to be harvested on each day of the harvesting period under consideration. These blocks are chosen from the set of  $N$  blocks,  $\mathcal{B}$ , which is determined by considering the ripeness of each of the blocks. However, it should be possible for a block to appear in this short-list of blocks and not be selected for harvesting during the period of time under consideration. Denote the set of blocks scheduled for harvesting on day  $d$  by set  $\mathcal{B}_d$ . It should therefore be possible for a vineyard block to appear in the set  $\mathcal{B}$  and to not be selected for harvesting, *i.e.* the block should not appear in any one of the sets  $\mathcal{B}_1, \dots, \mathcal{B}_D$  (that is,  $\bigcup_{d=1}^D \mathcal{B}_d \neq \mathcal{B}$ ).

Since a tabu search is applied to the harvest scheduling problem, moves are applied to the initial harvesting schedule, as discussed in §7.2. It is important that any vineyard block not selected for harvesting in the current harvesting schedule, should be eligible for harvesting in a following harvesting schedule. Therefore, the vineyard block not selected for harvesting still appears in the harvest scheduling matrix,  $\mathbf{H}$ , but the value of a new variable,  $u$ , is employed to distinguish between blocks selected for harvesting and blocks that are not selected. All blocks appearing in entries  $\mathbf{H}(d, p)$  with  $p \geq u$ , are not considered part of the harvesting schedule for the duration of the harvesting period. For example, in the harvesting schedule

$$\mathbf{H} = \left\{ \begin{array}{cccc|cc} & & & & u & & \\ & & & & \downarrow & & \\ \left[ \begin{array}{cccc|ccc} B_{b_1} & B_{b_2} & -1 & \cdots & B_{b_y} & \cdots & -1 \\ B_{b_3} & B_{b_4} & B_{b_5} & \cdots & -1 & \cdots & -1 \end{array} \right], & & & & & & \\ & & & & \vdots & & \\ \left[ \begin{array}{cccc|ccc} B_{b_x} & -1 & -1 & \cdots & B_N & \cdots & -1 \end{array} \right] \end{array} \right\},$$

vineyard blocks  $B_{b_y}$  and  $B_N$  are not selected for harvesting. However, all the vineyard blocks appearing to the left of the divider are harvested. Furthermore, the set  $\mathcal{B}_1 = \{B_{b_1}, B_{b_2}\}$  does not include vineyard block  $B_{b_y}$  even though  $B_{b_y} \in \mathcal{B}$ .

The process of calculating the harvest evaluation score is outlined in Algorithm 7.3. The evaluation score is a combination of the *average cellar scheduling score*,  $\Omega_{\bar{x}}$  and the *sugar level score*,  $\Omega_s$ . It is possible for the sugar level score to have a value of 0. Therefore, in order to never exclude the average cellar scheduling score, the harvest evaluation score,  $\Omega$ , is taken as  $(\Omega_h + 1)\Omega_{\bar{x}}$ .

Lines 2–20 of Algorithm 7.3 are responsible for generating the average cellar scheduling score,  $\Omega_{\bar{x}}$ , which is the average of the five cellar scheduling scores,  $\Omega_c$ , each determined by the daily generated scenarios. The scenarios are generated from the different sets  $\mathcal{B}_1, \dots, \mathcal{B}_D$  of the harvesting schedule,  $\mathbf{H}_x$ , being evaluated. The first step in forming a complete harvesting scenario, starts with the generation of the daily scenario, as explained in §7.3.1, for the first day of harvesting. If at least one block is scheduled for harvesting on the first day, the scenario (or generated cellar scheduling problem) is solved via the tabu search described in Chapter 6. If the best completion time found,  $\alpha$ , is not within the required business hours, the current cellar scheduling score is severely penalized by setting  $\alpha$  equal to  $10\,000\alpha$ . The reason for such a drastic penalization is that the best completion time of the cellar scheduling problem may, in fact, cause the harvesting schedule to be infeasible when the cellar scheduling solution suggests that the best completion time exceeds the chosen business hours. However, rather than considering such a harvesting schedule as infeasible and excluding any such schedules from consideration, a very poor harvest evaluation score is generated, causing it to rather be a highly unfavourable harvesting schedule than an infeasible schedule.

Each harvesting schedule spans  $D$  days and it was initially considered to multiply the sum of the best completion time and the best red fermentation tank assignment score by  $(D - d)$ . This was considered in order to allow the first day to have the largest influence on the eventual cellar scheduling score. With the argument in mind that the closer a day is to the start of the harvesting period, the larger influence it has on the cellar scheduling score, not because the first day in the scheduling period is more important than the following days, but rather because the information available for the first day is more accurate than that of any of the following days. However, after applying this evaluation method to an example problem, it was soon realized that by allowing the day of the schedule to influence the score in such a way, fewer and fewer blocks are assigned to the first day of harvesting. The day of harvesting is included when the block scores are calculated. Aside from the fact that some predictions are made in terms of sugar levels, the individual suppliers and the restrictions from their farms may also cause a forced change in the suggested schedule. If  $\alpha_d$  and  $\epsilon_d$  refer to the values of  $\alpha$  and  $\epsilon$  calculated on day  $d$ , then the cellar scheduling score may be expressed as

$$\Omega_c = \sum_{d=1}^D (\alpha_d + \epsilon_d).$$

---



---

**Algorithm 7.3:** Evaluating a harvest schedule

---

**Input:** The harvesting schedule to be evaluated,  $\mathbf{H}_x$

**Output:** The harvest evaluation score,  $\Omega$ , of  $\mathbf{H}_x$

```

1  $\Omega_\Sigma \leftarrow 0$ ;
2 for scenario = 1 to 5 do
3   generateDailyScenario( $\mathcal{B}_1$ );
4    $\Omega_c \leftarrow 0$ ;
5   for  $d = 1$  to  $D$  do
6     if  $|\mathcal{B}_d| \geq 1$  then
7       solveScenario();
8       if  $(\alpha - \alpha_{max}) > 0$  then
9          $\alpha \leftarrow 100\alpha$ ;
10      end
11       $\Omega_c \leftarrow \Omega_c + (\alpha + \epsilon)$ ;
12      updateMachineVolumes();
13    end
14    if  $(d + 1 \leq D)$  and  $|\mathcal{B}_{d+1}| \geq 1$  then
15      | generateDailyScenario( $\mathcal{B}_{d+1}$ );
16    end
17  end
18   $\Omega_\Sigma = \Omega_\Sigma + \Omega_c$ ;
19 end
20  $\Omega_{\bar{x}} \leftarrow \text{round}(\Omega_\Sigma/5)$ ;
21  $\Omega_s \leftarrow \text{determineSugarLevelScore}(\mathbf{H}_x)$ ;
22  $\Omega \leftarrow (\Omega_s + 1)\Omega_{\bar{x}}$ ;

```

---

After the cellar scheduling problem of the first day in the harvesting schedule has been generated and solved, the red fermentation tank volumes require updating since grape batches have (possibly) been assigned to them. The new volumes are then considered when the contribution to the cellar scheduling score of the second day is calculated. This process is repeated for each day of the harvest scheduling period, until the first cellar scheduling score has been calculated.

The whole process is repeated until five individual cellar scheduling scores have been determined. For each of the cellar scheduling scores, the original machine volumes are considered for the first day and then updated after each instance of the daily cellar scheduling problem has been solved. The average cellar scheduling score may then be calculated from the five cellar scheduling scores collectively as indicated in line 20.

The sugar level score,  $\Omega_s$ , is determined by the function `determineSugarLevelScore( $H_x$ )`. The calculation of the sugar level score, may be split into two parts, the first part concerning blocks assigned to positions  $H_x(d, p)$  where  $p < u$  and the second part to blocks assigned to positions where  $p \geq u$  in the harvesting schedule,  $H_x$ , under evaluation.

---



---

**Procedure determineSugarLevelScore( $H_x$ )**


---

```

1 for  $d = 1$  to  $D$  do
2   for  $p = 0$  to  $u$  do
3      $b \leftarrow H_x(d, p)$ ;
4     if  $b > 0$  then  $\Omega_b(b) \leftarrow d\rho(d, b)$ ;
5   end
6 end
7 for  $d = 1$  to  $D$  do
8   for  $p = u$  to  $|H_x|$  do
9      $b \leftarrow H_x(d, p)$ ;
10    for  $d_2 = 1$  to  $D$  do
11      if  $b > 0$  then
12        if  $\rho(d_2, p) = 0$  then  $\Omega_b(b) \leftarrow M/2$ ;
13        if  $(\zeta_b + \frac{1}{2}d_2) \geq \varsigma_{0,\kappa}(2)$  then
14           $\Omega_b(b) \leftarrow M$ ;
15          break;
16        end
17      end
18    end
19  end
20 end
21  $\Omega_s \leftarrow \sum_{b=1}^N \Omega_b(b)$ ;
22 return  $\Omega_h$ ;

```

---

The matrix  $\rho$  refers to a penalty matrix generated at the start of the tabu search. This penalty matrix is based on the sugar level of each block and the sugar class intervals described earlier in this chapter. Unfortunately, not enough vineyard data are available to make good predictions in terms of the expected increase in the sugar levels of the grapes as the harvesting period progresses. However, this factor, along with a motivation for the method considered for estimating sugar levels, is further discussed in Chapter 9. For now, it is assumed that once the sugar level of the grapes enters the interval  $\varsigma_{3,\kappa}$  and is considered in the shortened set of blocks,  $\mathcal{B}$ , the increase in sugar level each day assumes a linear relationship with the number of days since the measurement of the sample level. Furthermore, as indicated in the limited data made available by the cellar, this daily increase in sugar level value is considered to be 0.5. The entry  $\rho(d, b) = c$  indicates that on day  $d$  of the harvesting schedule, block  $B_b$  is considered to be in sugar level class  $\chi = c + 1$ . If the samples were measured one day before the start of the harvest scheduling period, the predicted sugar level value used to determine the sugar level class on day  $d$ , is  $\zeta_b + \frac{1}{2}d$ .

All the blocks with entries  $\mathbf{H}_x(d, p)$  for which  $p < u$  are assigned the block score,  $\Omega_b(b) = (d - 1) \rho(d, b)$ . However, the case where blocks have entries  $\mathbf{H}_x(d, p)$  for which  $p \geq u$ , require special precaution. These are the blocks considered as unassigned to any of the days in the harvesting schedule. Omitting a block from the harvesting schedule that should have been harvested may have even more drastic consequences than simply harvesting a block on the wrong day. Two such cases are identified. The first occurs when a vineyard block reaches maturity, *i.e.* the block has a sugar level class of 1 sometime during the harvesting period, and is never assigned to be harvested. The block score is then set as 5 000. An even worse case, is when a block reaches maturity and is then left to over-ripen, in which case the block score is set as 10 000. The sum of the individual block scores are then returned as the sugar level score  $\Omega_s$ , and the harvest evaluation score  $\Omega$  is calculated.

## 7.4 Generating candidate moves and selecting the best move

Generating moves for the harvesting schedule is not as complicated as the move types considered for the cellar scheduling tabu search. This is due to the fact that there are not as many different characteristics and requirements present within the harvesting schedule as there are within the cellar scheduling problem. The types of moves considered, are swaps within the harvesting schedule list  $\mathbf{H}$ . Two entries in the harvesting schedule list are selected (following a set of rules) and their values are exchanged. The process of generating the list of swap moves, evaluating the moves and selecting and applying the best move, is outlined in Algorithm 7.5.

---

**Algorithm 7.5:** Generating the list of swap moves

---

**Input:** The current harvesting schedule list,  $\mathbf{H}$

**Output:** The best move is selected and applied and the tabu lists are updated.

- 1  $[d, p] \leftarrow \text{chooseBlockToGenerateMoves}();$
  - 2  $b \leftarrow \mathbf{H}(d, p);$
  - 3  $\mathbf{L} \leftarrow \text{createListOfSwapMoves}(b, d, p, \mathbf{H});$
  - 4  $\Omega \leftarrow \text{evaluateSwaps}(\mathbf{L});$
  - 5  $\text{chooseAndApplyBestMove}(\mathbf{L}, \mathbf{v}, b);$
- 

Whenever the sugar level score,  $\Omega_s$ , is determined, the list  $\Omega_b(b)$  of individual block scores is first calculated. These scores are considered in the function `chooseBlockToGenerateMoves()` in order to select a block with which to generate the list of moves. The block is chosen to be the vineyard block with the largest contribution to the sugar level score, *i.e.* the block,  $B_b$ , with the largest block score,  $\Omega_b(b)$ . In order to avoid the same block from being selected repeatedly, a small tabu list is created in order to limit the frequency with which each block is selected.

For the remainder of this chapter, all harvesting schedule entries of the form  $\mathbf{H}(d, p)$  that are not equal to  $-1$  and for which  $p < u$ , are considered to be entries of type 0. In the case where  $p < u$  still holds, but  $\mathbf{H}(d, p) = -1$ , entries are considered to be of type 1. Types 2 and 3 refer to the entries  $\mathbf{H}(d, p)$  for which  $p \geq u$ , where entries of type 2 refer to block numbers, and entries of type 3 have a value of  $-1$ . Table 7.7 contains a summary of the entry type characteristics.

When a swap is performed, two entries in  $\mathbf{H}$  are switched. There are some restrictions on which swaps are allowed in order to avoid duplicate harvesting schedules in the list of moves,  $\mathbf{L}$ . Consider blocks  $B_{b_1}$  and  $B_{b_2}$  currently in positions  $\mathbf{H}(d_1, p_1)$  and  $\mathbf{H}(d_2, p_2)$ , respectively. Both entries are of type 0, therefore no difference will occur in the harvesting schedule if  $d_1 = d_2$ , since the blocks selected for harvesting on day  $d_1$  ( $d_2$ ) remain the same after a swap. Whenever

Type number	Entry restriction	Position restriction
0	$\mathbf{H}(d, p) > 0$	$p < u$
1	$\mathbf{H}(d, p) = -1$	$p < u$
2	$\mathbf{H}(d, p) > 0$	$p \geq u$
3	$\mathbf{H}(d, p) = -1$	$p \geq u$

Table 7.7: The entry types 0, 1, 2 and 3 as determined by corresponding harvesting schedule entry and position.

$d_2 \neq d_1$  the harvesting schedule is influenced upon application of a swap. Therefore, a swap between two entries of type 0 (referred to as a type (0,0) swap) is only allowed if  $d_2 \neq d_1$ . The type of block  $B_{b_1}$  is always the first entry when listing the swap type as type  $(i, j)$ , where block  $B_{b_1}$  refers to the block selected in the function **chooseBlockToGenerateMoves()**. Entries of types 1 and 3 will never be selected by the function **chooseBlockToGenerateMoves()**, since both these types have  $\mathbf{H}(d, p) = -1$  and are therefore only place-holding entries, not referring to any block. A summary of the restrictions on all possible swaps is listed in Table 7.8.

Type $i$	Type $j$	Swap move restrictions
0	0	allowed for all, $d_j \neq d_i$
0	1	only one move per $d_j$ , $d_j \neq d_i$
0	2	allowed
0	3	only one move allowed
2	0	allowed
2	1	only one move per $d_j$ , including $d_j = d_i$
2	2	never allowed
2	3	never allowed

Table 7.8: The swap move restrictions, where Type  $i$  refers to the entry type of the selected entry  $\mathbf{H}(d_i, p_i)$  and Type  $j$  refers to all entries  $\mathbf{H}(d_j, p_j)$  available for the swap.

None of the swaps that are considered *not allowed* will necessarily cause an infeasible harvesting schedule. They are simply avoided since they will have no influence on the harvesting schedule. The process of generating a list of swap moves is applied in the procedure **createListOfSwapMoves( $b, d, p, \mathbf{H}$ )**. When a swap move is selected as the best move and its inverse is added to the relevant tabu list, the entry types, as well as the positions of the entries in the harvest scheduling list are also relevant. Therefore, the list of moves includes more information than only the two entries of the harvest scheduling list being switched. The list entry,  $\mathbf{L}(a, 1)$ , contains the current entry type<sup>4</sup> of the block  $B_b$  chosen and in  $\mathbf{L}(a, 2)$  the entry type of the other entry required for the swap is added, where  $a$  refers to the move number. In positions 3, 4, 5 and 6, the values of  $d_b$ ,  $p_b$ ,  $d_2$  and  $p_2$  are assigned, where  $d_b$  and  $p_b$  refer to the day and position of  $B_b$  (the selected block) in the current harvesting schedule, and  $d_2$  and  $p_2$  refer to the day and position of the other vineyard block selected for the swap.

<sup>4</sup>Referred to as current entry type, since the entry type does not depend on the block number or any other vineyard block characteristics, but rather on the position in the harvesting schedule.



---



---

```

Procedure createListOfSwapMoves( $b, d, p, \mathbf{H}$ )
1 if  $p < u$  then
2   |   blockType  $\leftarrow 0$ ;
3   |   both  $\leftarrow$  true;
4 else
5   |   blockType  $\leftarrow 2$ ;
6 end
7  $a \leftarrow 0$ ;
8 for  $d_2 = 1$  to  $D$  do
9   |    $p_2 \leftarrow 0$ ;
10  |   added  $\leftarrow$  false;
11  |   if  $d_2 \neq d$  then
12  |     |   while  $p_2 < u$  do
13  |       |   if ( $added = false$ ) and ( $\mathbf{H}(d_2, p_2) \leq 0$ ) then
14  |         |   added  $\leftarrow$  true;
15  |         |    $L \leftarrow$  addSwapToList( $a, blockType, 1, d, p, d_2, p_2$ );
16  |         |    $a \leftarrow a + 1$ ;
17  |         |   else if ( $\mathbf{H}(d_2, p_2) \neq -1$ ) and ( $\mathbf{H}_x(d_2, p_2) \neq b$ ) then
18  |         |    $L \leftarrow$  addSwapToList( $a, blockType, 0, d, p, d_2, p_2$ );
19  |         |    $a \leftarrow a + 1$ ;
20  |         |   end
21  |         |    $p_2 \leftarrow p_2 + 1$ ;
22  |       |   end
23  |     |   end
24  |     |    $p_2 \leftarrow u$ ;
25  |     |   added  $\leftarrow$  false;
26  |     |   while ( $both = true$ ) and ( $p_2 \leq |\mathbf{H}(d)|$ ) do
27  |       |   if ( $added = false$ ) and ( $\mathbf{H}(d_2, p_2) \leq 0$ ) then
28  |         |   added  $\leftarrow$  true;
29  |         |    $L \leftarrow$  addSwapToList( $a, blockType, 3, d, p, d_2, p_2$ );
30  |         |    $a \leftarrow a + 1$ ;
31  |         |   else if ( $\mathbf{H}(d_2, p_2) \neq -1$ ) and ( $\mathbf{H}_x(d_2, p_2) \neq b$ ) then
32  |         |   added  $\leftarrow$  true;
33  |         |    $L \leftarrow$  addSwapToList( $a, blockType, 2, d, p, d_2, p_2$ );
34  |         |    $a \leftarrow a + 1$ ;
35  |         |   end
36  |         |    $p_2 \leftarrow p_2 + 1$ ;
37  |       |   end
38 end
39 return  $L$ ;

```

---

The function **addSwapToList**(move, blockType,  $x, d, p, d_2, p_2$ ) is utilized to add the swap move to the list of candidate moves by placing the values of the current entry type of block  $B_b$  in  $L(\text{move}, 1)$  and  $x, d, p, d_2$  and  $p_2$  in positions 2 to 6, for each move  $a$ .

In Algorithm 7.5, the function **evaluateSwaps**( $L$ ), is employed to evaluate each of the moves in the list  $L$ . The vector  $\Omega$  is generated when applying moves; here  $\Omega(a)$  refers to the harvest evaluation score calculated when applying move  $a$ .

---



---

```

Procedure chooseAndApplyBestMove( $\mathbf{L}, \mathbf{\Omega}, b$ )
1 bestMoveNumbers  $\leftarrow$  sortMoveValues( $\mathbf{\Omega}$ );
2 found  $\leftarrow$  false;
3 moveNumber  $\leftarrow$  0;
4 while found = false do
5    $a \leftarrow$  bestMoveNumbers(moveNumber);
6    $d_b \leftarrow \mathbf{L}(a, 3)$ ;
7    $p_b \leftarrow \mathbf{L}(a, 4)$ ;
8    $d_2 \leftarrow \mathbf{L}(a, 5)$ ;
9    $p_2 \leftarrow \mathbf{L}(a, 6)$ ;
10  swap  $\leftarrow [b, \mathbf{H}(d_2, p_2)]$ ;
11  currentMove  $\leftarrow [b, \mathbf{L}(a, 1), \mathbf{L}(a, 2), d_b, d_2]$ ;
12  isInSwapList  $\leftarrow$  isInSwapList(swap);
13  isInTypeList  $\leftarrow$  isInTypeList(currentMove);
14  if (isInSwapList = true) or (isInTypeList = true) then
15    if  $\mathbf{\Omega}(a) < \vartheta$  then
16       $\vartheta \leftarrow \mathbf{\Omega}(a)$ ;
17      found  $\leftarrow$  true;
18    else
19      found  $\leftarrow$  false;
20    end
21  else
22    if  $\mathbf{\Omega}(a) < \vartheta$  then  $\vartheta \leftarrow \mathbf{\Omega}(a)$ ;
23    found  $\leftarrow$  true;
24  end
25  if found = true then
26     $\iota \leftarrow [b, \mathbf{L}(a, 2), \mathbf{L}(a, 1), d_2, d_b]$ ;
27    addToTabuLists( $\iota$ , swap);
28     $\mathbf{H}(d_b, p_b) \leftarrow \mathbf{H}(d_2, p_2)$ ;
29     $\mathbf{H}(d_2, p_2) \leftarrow b$ ;
30  else
31    moveNumber  $\leftarrow$  moveNumber + 1;
32  end
33 end

```

---

The procedure `chooseAndApplyBestMove( $\mathbf{L}, \mathbf{\Omega}, b$ )` is applied to select the best suitable move from the list of candidate moves and then to update the tabu lists. There are two tabu lists. The first list limits the two entry values, *i.e.* the two blocks exchanging positions in the harvesting schedule, from being exchanged again. The second tabu list limits the block  $B_b$  from moving back to its previous harvesting day directly from the new day with the entry type also reverting back to its previous state. For example, the best move in list  $\mathbf{L}$  is represented by  $\mathbf{L}(\phi) = [b, e_1, e_2, d, p, d_2, p_2]$  where  $\mathbf{H}(d_2, p_2) = b_2$ . Then  $(b, b_2)$  is placed in the first tabu list so that the exchange of the two values,  $b$  and  $b_2$ , is not allowed to be applied again as long as it is in the tabu list. Furthermore, the vector  $[b, e_2, e_1, d_2, d]$  (the inverse of move  $\mathbf{L}(\phi)$ ) is added to the second tabu list disallowing a block to be moved to its previous harvesting day unless it is swapped with a different entry type. For example, the move  $\mathbf{L}(a) = [b, e_2, e_1, d_2, p_2, d_x, p_x]$  is allowed only if  $d_x \neq d$ .

The moves are sorted from the highest harvest evaluation score to the lowest. Starting at the first move in the list, each move is considered in the harvest evaluation score order until a suitable move has been found. Such a move is found either if the move is not in any tabu list or if the aspiration criterion is satisfied. The value of the best harvest evaluation score is denoted by  $\vartheta$  and is updated if the selected move results in a smaller harvest evaluation score.

## 7.5 Solving the harvest scheduling problem with the tabu search

As mentioned previously, the tabu search application developed to solve the harvest scheduling problem is a much simpler process than that of the cellar scheduling tabu search. The harvest scheduling tabu search is outlined in Algorithm 7.8.

---



---

**Algorithm 7.8:** The harvest scheduling tabu search

---

**Input:** The list of blocks,  $\beta$ , and their sugar levels,  $s_b$ .  
**Output:** The harvesting schedule,  $H$ .

- 1  $H \leftarrow \text{generateInitialSolution}(\beta, s_b)$ ;
- 2  $\vartheta \leftarrow \text{evaluate}(H)$ ;
- 3 found  $\leftarrow$  false;
- 4 counter  $\leftarrow$  0;
- 5 **while** (*found = false*) and (*counter*  $\leq$  50) **do**
- 6     **applySwapMove**();
- 7     **if**  $\vartheta \leq \vartheta_{max}$  **then** found  $\leftarrow$  true;
- 8 **end**

---

First, the initial harvesting schedule  $H$  is created, as described in §7.2. The harvest evaluation score of the first harvesting schedule is then calculated and set as the best score,  $\vartheta$ . The candidate swap moves are then generated, evaluated and the best move applied until a satisfactory harvesting schedule is found.

The harvest scheduling tabu search is explained further in the following chapter, when the harvest scheduling tabu search is applied to real data and the full-size cellar at Wamakersvallei.

## 7.6 Chapter overview

In this chapter the tabu search method developed in order to solve the harvest scheduling problem was discussed. A good harvesting schedule should ensure that vineyard blocks are harvested as close to their optimal maturity dates as possible. Furthermore, the restrictions imposed by the physical capacity of the cellar should be used in order to further distinguish between harvesting schedules. The contents of this chapter therefore achieves thesis objective II expressed in Chapter 1.

In §7.1, the different harvest scheduling attributes were described. These include notations for the set of vineyard blocks and their characteristics as well as a means of representing a harvesting schedule mathematically.

The generation of the initial solution was discussed in §7.2, as well as a suggested order of vineyard blocks to represent urgency with respect to harvesting, as indicated by the sugar levels derived from samples.

In §7.3 a means of evaluating harvesting schedules was considered — first, by the generation of realistic cellar scheduling scenarios (in §7.3.1) and then by the calculation of the harvest evaluation score (as explained in §7.3.2).

The generation of a list of candidate moves is described in 7.4. The moves considered are swaps and only certain harvesting schedule entries may be swapped with one another, as explained in this section. The evaluation of such a list of candidate moves was also discussed, as well as the selection and application of the best move.

Finally, the previous sections were brought together in §7.5 where a concise outline of the harvest scheduling tabu search was presented. In the next chapter, the required changes to the harvest and cellar scheduling problems is considered in order to successfully develop the decision support system suitable for the Wamakersvallei Winery characteristics.

---

---

## CHAPTER 8

---

# The Wamakersvallei decision support system

### Contents

8.1	Required changes to the tabu searches . . . . .	148
8.1.1	<i>A new generation approach for candidate moves on the presses</i> . . . . .	148
8.1.2	<i>Job generation for the active cellar scheduling problem</i> . . . . .	149
8.1.3	<i>Generating moves in the harvest scheduling problem</i> . . . . .	150
8.1.4	<i>Evaluation a harvesting schedule</i> . . . . .	150
8.2	The decision support system applied to Wamakersvallei . . . . .	152
8.2.1	<i>Importing data</i> . . . . .	152
8.2.2	<i>Solving the harvest scheduling problem with VinDSS</i> . . . . .	154
8.2.3	<i>Generating the candidate list of moves</i> . . . . .	158
8.3	A short analysis of the performed tabu search . . . . .	159
8.4	The suggested schedule vs Wamakersvallei selection . . . . .	161
8.5	Chapter overview . . . . .	163

In Chapters 5–7, the scheduling problems occurring at a winery, often focussing on Wamakersvallei Winery, was defined mathematically and tabu search applications were developed to solve the cellar scheduling and harvest scheduling problems. The goal of this chapter is to apply the developed methods to solve the scheduling problem at Wamakersvallei using Wamakersvallei 2009 harvesting data and comparing it with the Wamakersvallei approach where possible. In §8.1, the required changes are applied to the developed tabu searches in order to successfully solve the larger scheduling problem occurring at Wamakersvallei. Such a customized approach based on the generic approach defined in previous chapters, is required to better suit the cellar or winery at hand, in this case, Wamakersvallei Winery. A means of displaying and calculating the schedules by importing the necessary data is presented as the software solution VINDSS. The working of VINDSS is illustrated by means of an example consisting of the application to real Wamakersvallei 2009 harvesting data. VINDSS in its current state is a basic tool assisting the staff at a winery to better schedule the harvesting of their grapes. However, the VINDSS programs has much potential in becoming a more sophisticated and refined tool with possible improvements discussed in Chapter 9. In §8.3, a concise analysis of the tabu search is presented after which the harvesting schedule suggested by VINDSS may be compared to the harvest as it occurred at Wamakersvallei, which is discussed in §8.4.

## 8.1 Required changes to the tabu searches

The harvest scheduling approach considered in the previous chapter delivered intuitively good harvesting schedules in a relatively short period of time when applied to the fictitious cellar considered in the various examples in that chapter. The cellar scheduling tabu search approach performed splendidly when applied to the fictitious cellar and scheduled grape loads, rendering optimal solutions in 100% of the trial runs when the number of allowed iterations was set high enough. However, when applying the combined harvest and cellar tabu search approach to solve a typical cellar scheduling problem as it occurs at Wamakersvallei some problems arise. For example, it becomes a lengthy process simply to find a feasible ordering on the presses, which is where the majority of bottlenecks occur in the cellar. A few simple changes are therefore required.

Perhaps the main problem area when applying the developed tabu searches to the Wamakersvallei scheduling problem, is that an enormous amount of processing time is required in order to achieve a good schedule. On further inspection, it was found that both an increase in processor choices and an increase in the volume of grapes received at the real cellar influence the computer processing time required to implement the tabu search when compared to the fictitious cellar of Chapters 5–7. This led to an increase in the number of iterations required to find feasible press assignments when the ejection chain move was applied. Furthermore, the process of removing the unnecessarily assigned jobs (forming the maximum assignment order) considered following the ejection chain move where every single possible removal combination of the additional job assignments have to be considered. Some further problem areas also came to light on closer inspection. The changes made to the two scheduling solution approaches are outlined in the following subsections.

### 8.1.1 A new generation approach for candidate moves on the presses

In order to contain the first problem area, the ejection chain used when generating the first list of candidate moves for assignments made to the presses is replaced by a more reliable means of generating candidate assignment order matrices. Rather than randomly moving values in the matrix until a feasible order matrix is found, the order is first considered separately. Two types of orders are considered: the first concerns the white grapes arriving at the cellar, *i.e.* Type II jobs. A desired order is generated by repeatedly selecting a random number between 1 and 3 indicating the tipping bin to consider. Furthermore, for each of the three tipping bins there is a current position indicator,  $p_1$ ,  $p_2$  and  $p_3$  respectively, each set to 1 at the start of the selection process. If, for example, the first tipping bin to be selected is tipping bin 1, row 1 of the current harvesting schedule is considered, *i.e.* the assignments made to tipping bin 1. Starting at position  $p_1$  the position indicator is increased until a Type II job is found in  $H(1, p_1)$  and then selected to be placed in the first position of the order under construction,  $p_1$  is increased by a value of 1 thereby pointing to the next job assigned to  $P_1$ . During each iteration during which a tipping bin,  $P_i$ , is selected the next Type II job assigned to the tipping bin, starting at the current  $p_i$ , is selected, *i.e.* no job is selected more than once. This process is repeated until all the Type II jobs to be received have been added to the order. A new order is generated for each time that a new move is considered. The second type of order, functioning completely separate from the previous type, consists of two orders and is concerned with the emptying of the red fermentation tanks. These two orders are simply generated by considering the generated ‘arrival times’, *i.e.* the time assigned to start emptying the tanks, for each of the two sets of red fermentation tanks, DF tanks and RT tanks. Therefore, these two sets are only

changed at the start of a new cellar scheduling problem and not during each move considered as with the previous type of order.

When applying the new modified ejection chain move, these orders are kept in mind when moving the jobs. For example, consider the generated order  $J_{j_1}, J_{j_2}, \dots, J_{j_x}$  and a job  $J_{j_6}$  selected to be moved to a new row,  $i$ , with the jobs  $J_{j_2}, J_{j_3}, J_{j_7}$  and  $J_{j_8}$  assigned in this order. By obeying the order, job  $J_{j_6}$  may be placed in positions  $p = 1, 2$  or  $3$ , but not positions  $4$  or  $5$ . From the range of allowed positions, one is chosen at random. If  $p = 2$  is selected the new row  $i$  will be  $J_{j_2}, J_{j_6}, J_{j_3}, J_{j_7}$  and  $J_{j_8}$  in this order. A new job is now selected from this row to be placed in another row in the same fashion, any job other than the last placed job, job  $J_{j_6}$ , may be selected. By considering this new ejection chain approach, a larger variety of sensible moves may be considered in a shorter time period.

The second aspect of the press order assignments from the previous chapters which causes a very slow generation of candidate moves, is the process of considering the removal of certain unnecessarily assigned jobs in order to form the maximum assignment order, as explained in §6.3.4. In this case, rather than considering all the possible removals, a method similar to that used when removing unnecessary assignments made to the red fermentation tanks, is used.

By applying both of these improvements, a major reduction in processing time is achieved without any loss to the quality of solutions.

### 8.1.2 Job generation for the active cellar scheduling problem

Even at the start of the harvesting season, there is an enormous variety of blocks to consider for harvesting. Therefore, rather than simply allowing all vineyard blocks with sample levels within sugar level class 3, only vineyard blocks expected to reach maturity during the harvesting period considered, are taken into account.

However, this alteration only leads to a minor, but still helpful, decrease in problem size. In order to further lessen the processing time required for each application of the active cellar scheduling problem, the construction of the set of jobs,  $\mathcal{J}$ , is considered. For example, when a vineyard block containing red grapes, is split into the calculated number of batches, the batches are received at the tipping bins and then added together again at the red fermentation tanks. However, since a large number of different combinations is then considered when applying the ejection chain move to the Type I jobs, unnecessarily large processing time and a possible loss in quality occurs. Therefore, a vineyard block containing red grapes,  $B_r$ , with an expected yield of  $w$  tonnes is split into the number of batches calculated,  $x$ . Job  $J_1$  is assigned the full weight of the vineyard block,  $w$ , and jobs  $J_2, \dots, J_x$  are considered to be of weight 0. These jobs are still considered at the tipping bins in order to keep the schedule realistic, but are not assigned to the red fermentation tanks. This is completely feasible since weight is not taken into account at the tipping bins and this approach dramatically reduces the number of jobs to consider when assignments are made to the red fermentation tanks.

In the same sense, two grapeloads of Type II grapes from the same vineyard block, that are received at the cellar within 2.5 hours of one another, is combined to be one job. Again, the split of a larger Type II job at the separators or the presses is less time consuming than increasing the number of jobs to assign. The waiting time is the same as the processing time on the presses; if the arrival time difference between two such jobs are greater than the processing time of one job, it no longer makes sense to wait. This construction directly imitates what happens at Wamakersvallei, where such grapeloads are added to the same separator or press if they arrive close enough to each other. In this case, the first grapeload to arrive at the cellar is immediately

received at the separator or press where it awaits the second grapeload. Therefore, two Type II jobs, jobs  $J_{j_1}$  and  $J_{j_2}$ , may be combined when they are from the same vineyard block and  $|e_{j_1} - e_{j_2}| \leq 2.5$ . The processing time of the new job  $J_j$ , is considered to be the normal press or separator cycle time together with the added waiting time,  $p_{ij} + |e_{j_1} - e_{j_2}|$ , and the arrival time is taken as  $\min\{e_{j_1}, e_{j_2}\}$ .

### 8.1.3 Generating moves in the harvest scheduling problem

It has already been established that one of the most important criteria considered during re-modelling of the scheduling problems, is the processing time. One of the simplest alterations is to simply change the number of cellar scheduling problems solved for each day in the harvesting schedule, as outlined in Algorithm 7.3. This number is then changed from the five considered in Chapter 7 to only two which was found to be equally efficient. In order to reduce processor time further, the size of the list of moves has been reduced. For each of the candidate moves in the list of harvesting moves,  $\mathbf{L}_H$ , the cellar scheduling tabu search is still applied  $2D$  times. This results in a number around 900 when the harvesting schedule is calculated from the 2009 harvesting data at Wamakersvallei. Furthermore, in order to generate an efficient harvesting schedule, at least 100 iterations of the harvest scheduling tabu search is required. Therefore, the process of generating and evaluating the list of moves has been reconstructed so that this list contains no more than 7 moves. Rather than considering each day in the harvesting schedule as a possible placement for the selected block, only the two days closest to the block's current day,  $d$ , of harvesting is considered, days  $d_1$  and  $d_2$  say. Furthermore, for each of the days determined only three moves are added to the list. For the first move, the swap occurs between the selected block scheduled for harvesting on day  $d$  and a<sup>1</sup> block with the largest contribution to the sugar level score,  $\Omega_s$ , of all the vineyard blocks considered for scheduling for each of days  $d_1$  and  $d_2$ . Then for each of days  $d_1$  and  $d_2$  another two moves are added to the list, the swap occurs between the selected block and a randomly selected position in the harvesting schedule, one position is selected with a value of  $-1$  and the second position has  $\mathbf{H}(d_i, p) \neq -1$ . The last move is then added to the list of harvesting moves as a swap between the selected block and  $\mathbf{H}(d_1, p_1)$  or  $\mathbf{H}(d_2, p_2)$ , where  $p_1, p_2 \geq u$ . However, the last move is only added to the list if the selected block of harvesting day  $d$  is of block type 0 (assigned to a position  $\mathbf{H}(d, p)$ , where  $p < u$ ).

### 8.1.4 Evaluation a harvesting schedule

After ordering the vineyard blocks considered for scheduling during the harvesting period under consideration, they are placed in the harvest schedule matrix in that order, as described in Chapter 7. However, vineyard blocks are only placed in positions  $\mathbf{H}(d, p)$ , for which  $p < u$ , until the average yearly intake has been matched in terms of the expected yield of the scheduled blocks. Thereafter, the remaining vineyard blocks are placed in positions where  $p \geq u$ .

The last aspect of the harvest scheduling tabu search proposed in Chapter 7 to be amended, is the evaluation of a harvesting schedule under consideration. However, the method of evaluation is not changed; only the construction of the harvest scheduling score,  $\Omega_s$ . The first change occurs with the strict penalization of assigning a move to the position  $\mathbf{H}(d, p)$  where  $p \geq u$ . The penalization discussed in the previous chapter, is far too strict and screws the relationship between the sugar level score  $\Omega_s$  and the average cellar scheduling score  $\Omega_{\bar{x}}$ , thereby disregarding the cellar

<sup>1</sup>More than one job may have the same contribution; simply one of these is selected.



harvesting score in some cases. The penalization is therefore reduced to only apply to blocks that will overripen if they are not harvested during the harvest scheduling period. Furthermore, rather than penalizing all such cases equally, the penalization now occurs depending on the quality of the grapes considered. Procedure `determineSecondSugarLevelScore( $\mathbf{H}_x$ )` may now be used to replace the procedure `determineSugarLevelScore( $\mathbf{H}_x$ )` described in Chapter 7, with the new penalization expression shown in line 15.

The individual block sugar level score  $\Omega_b$  is also reconsidered. It is important that these block sugar level scores are representative of the harvesting schedule as a whole by including influences from the cellar scheduling solution found. A new matrix  $\alpha$  is introduced in order to include the effect of evaluating the cellar scheduling problem. Whenever a generated scenario for a specific evaluation has a completion time larger than the set business hours, the entry  $\alpha(b)$  for each block  $B_b$  scheduled to be harvested on the particular day is increased. By including the matrix  $\alpha$  in the block sugar level score  $\Omega_b(b)$  for all blocks  $B_b \in \mathcal{B}_d$ , unfavourable blocks indicated by the cellar scheduling algorithm also stand a chance of being selected as blocks on which to apply swaps.

Furthermore, the block sugar level scores are further adjusted by adding the *deviation matrix*  $\delta$ . The deviation matrix is employed to influence the exact placing of a block in the harvesting schedule. The ideal day for harvesting any block is considered as the day where the sugar level,  $\zeta_b$ , of block  $B_b$  lies exactly in the middle of its first sugar level class, *i.e.* when  $\zeta_b = \frac{1}{2}(\varsigma_{1,\kappa}(2) + \varsigma_{1,\kappa}(2))$ . However, the urgency with which a vineyard block should be considered for harvesting when it lies in the second half of the first sugar class interval should be higher than the urgency with which a block lying in the first half is considered for harvesting. In order to avoid using negative values in the score, which may (possibly) result in two terms of the block sugar level score cancelling one another out, the following method is considered to generate the deviation matrix: First the middle of the first sugar level class is determined as

$$c_{\frac{1}{2}} = \frac{1}{2}(\varsigma_{1,\kappa}(2) + \varsigma_{1,\kappa}(2))$$

and then the true deviation of the sugar level,  $\zeta_b$ , from the middle of day  $d$  is computed as

$$x_{bd} = \left( \zeta_b + \frac{1}{4}d \right) - c_{\frac{1}{2}}.$$

The value of  $x$  is therefore positive if  $\zeta_b \geq c_{\frac{1}{2}}$  and negative otherwise. In order to emphasize the importance of overripe grapes being scheduled for harvesting when compared to under ripened grapes, the deviation matrix entry for a block  $B_b$  scheduled for harvesting on day  $d$  is

$$\delta(d, b) = \begin{cases} 2x_{bd} & \text{if } x_{bd} \geq 0, \\ -x_{bd} & \text{otherwise.} \end{cases}$$

The final change to the evaluation score of a harvesting schedule is that, rather than multiplying the sugar level score and the average cellar scheduling score, the two are now added together.

The majority of the changes are easily justified by the decrease in processing time. However, this last alteration had to be deduced intuitively after considering many more such alterations to different instances of the scheduling problem. The performance of the modified tabu searches may now be considered when the decision support system, VINDSS, employing the tabu searches to solve the harvest scheduling problem, is applied to solve the harvest scheduling problem occurring at Wamakersvallei.

---



---

**Procedure** determineSecondSugarLevelScore( $H_x$ )
 

---

```

1 for  $d = 1$  to  $D$  do
2   for  $p = 0$  to  $u$  do
3      $b \leftarrow H_x(d, p)$ ;
4     if  $b > 0$  then
5        $\Omega_b(b) \leftarrow \alpha(b) + \delta(d, b) + d\rho(d, b)$ ;
6     end
7   end
8 end
9 for  $d = 1$  to  $D$  do
10  for  $p = u$  to  $|H_x|$  do
11     $b \leftarrow H_x(d, p)$ ;
12    for  $d_2 = 1$  to  $D$  do
13      if  $b > 0$  then
14        if  $(\zeta_b + \frac{1}{4}d_2) \geq \varsigma_{1,\kappa}(2)$  then
15           $\Omega_b(b) \leftarrow \frac{1}{2}M/\rho(d_2, p)$ ;
16          break;
17        end
18      end
19    end
20  end
21 end
22  $\Omega_s \leftarrow \sum_{b=1}^N \Omega_b(b)$ ;
23 return  $\Omega_h$ ;

```

---

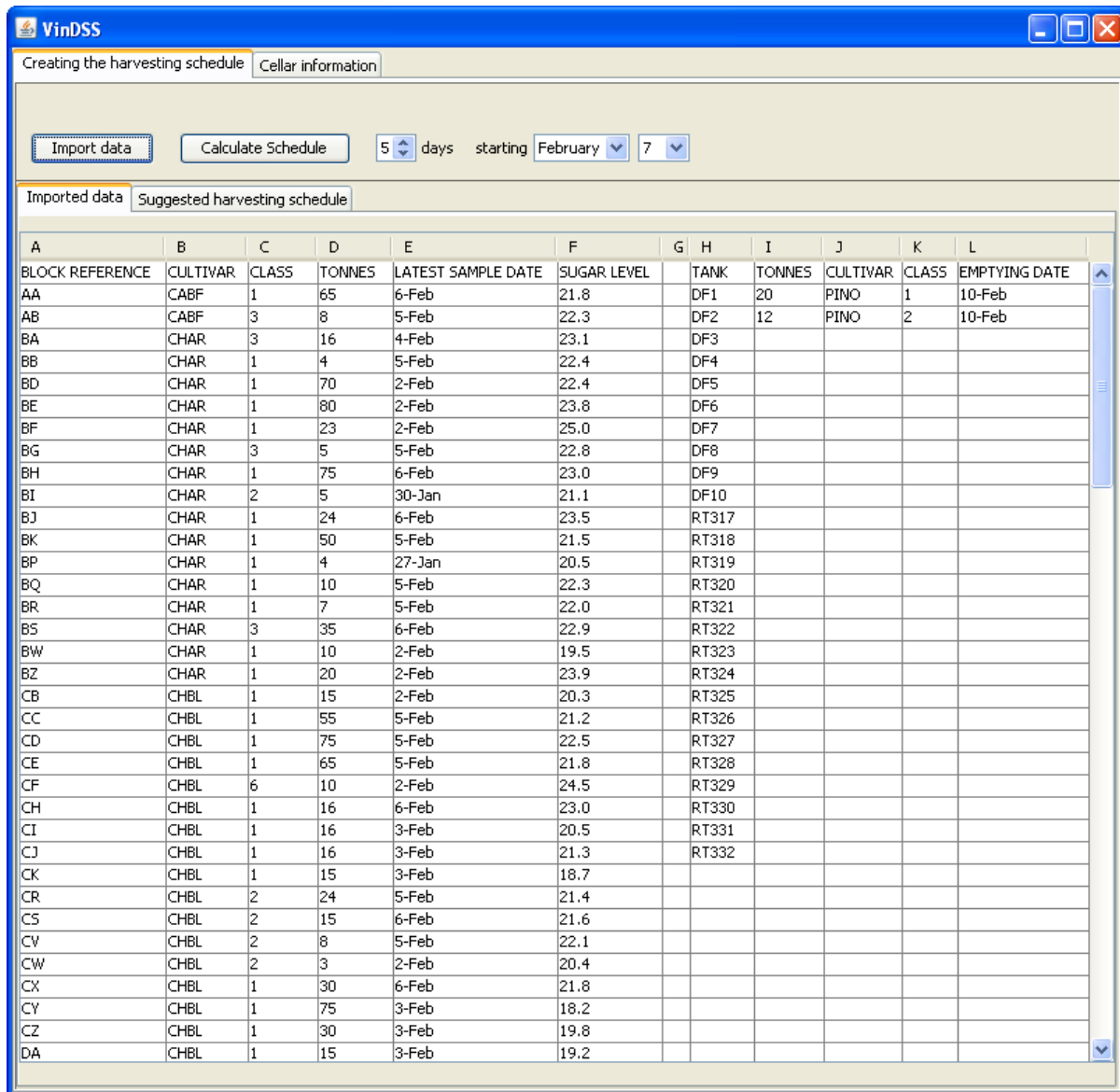
## 8.2 The decision support system applied to Wamakersvallei

The working of the decision support system, VINDSS, is now illustrated by applying the tabu searches developed in Chapters 6 and 7 to solve the harvest scheduling problem experienced at Wamakersvallei Winery. The data set used during the following sections was composed by considering the sugar level forms filled in by the viticulturist (as illustrated in Figure 4.8) during the Wamakervallei 2009 harvesting period. In order to find the expected yield for each of the vineyard blocks, the information contained in the form was cross-referenced with both the harvesting sheets (as illustrated in Figure 4.9) and the harvesting data recorded during previous years. Unfortunately, for some of the vineyard blocks further information was not available — for example, instances where a vineyard block was received for harvesting with no sugar sample being noted was rather common. However, this shortcoming is taken into account when the harvesting schedule generated is compared to the daily schedules suggested by the Wamakersvallei team.

### 8.2.1 Importing data

When VINDSS is started, the user is given the option to import data from a Microsoft Excel [86] file. The exact format of the file is very important; however, such technical aspects are discussed in the concise user manual in Appendix D. The imported information includes a

reference name for each block, the expected yield, cultivar, class, most recent sample sugar level as well as the date on which the sample was taken (in order to calculate the expected sugar level on the scheduled day of harvest). Figure 8.1 shows the user interface of VINDSS once the data has been imported from Excel.



The screenshot shows the VINDSS application window with the title "Creating the harvesting schedule" and a sub-tab "Cellar information". Below the title bar, there are buttons for "Import data" and "Calculate Schedule", followed by a dropdown menu set to "5" days, and a "starting" dropdown menu set to "February" with a "7" dropdown menu. The main area is divided into two tabs: "Imported data" (selected) and "Suggested harvesting schedule". The "Imported data" tab displays a table with the following columns: BLOCK REFERENCE, CULTIVAR, CLASS, TONNES, LATEST SAMPLE DATE, SUGAR LEVEL, TANK, TONNES, CULTIVAR, CLASS, and EMPTYING DATE. The table contains 26 rows of data, labeled from AA to DA.

A	B	C	D	E	F	G	H	I	J	K	L
BLOCK REFERENCE	CULTIVAR	CLASS	TONNES	LATEST SAMPLE DATE	SUGAR LEVEL	TANK	TONNES	CULTIVAR	CLASS	EMPTYING DATE	
AA	CABF	1	65	6-Feb	21.8	DF1	20	PINO	1	10-Feb	
AB	CABF	3	8	5-Feb	22.3	DF2	12	PINO	2	10-Feb	
BA	CHAR	3	16	4-Feb	23.1	DF3					
BB	CHAR	1	4	5-Feb	22.4	DF4					
BD	CHAR	1	70	2-Feb	22.4	DF5					
BE	CHAR	1	80	2-Feb	23.8	DF6					
BF	CHAR	1	23	2-Feb	25.0	DF7					
BG	CHAR	3	5	5-Feb	22.8	DF8					
BH	CHAR	1	75	6-Feb	23.0	DF9					
BI	CHAR	2	5	30-Jan	21.1	DF10					
BJ	CHAR	1	24	6-Feb	23.5	RT317					
BK	CHAR	1	50	5-Feb	21.5	RT318					
BP	CHAR	1	4	27-Jan	20.5	RT319					
BQ	CHAR	1	10	5-Feb	22.3	RT320					
BR	CHAR	1	7	5-Feb	22.0	RT321					
BS	CHAR	3	35	6-Feb	22.9	RT322					
BW	CHAR	1	10	2-Feb	19.5	RT323					
BZ	CHAR	1	20	2-Feb	23.9	RT324					
CB	CHBL	1	15	2-Feb	20.3	RT325					
CC	CHBL	1	55	5-Feb	21.2	RT326					
CD	CHBL	1	75	5-Feb	22.5	RT327					
CE	CHBL	1	65	5-Feb	21.8	RT328					
CF	CHBL	6	10	2-Feb	24.5	RT329					
CH	CHBL	1	16	6-Feb	23.0	RT330					
CI	CHBL	1	16	3-Feb	20.5	RT331					
CJ	CHBL	1	16	3-Feb	21.3	RT332					
CK	CHBL	1	15	3-Feb	18.7						
CR	CHBL	2	24	5-Feb	21.4						
CS	CHBL	2	15	6-Feb	21.6						
CV	CHBL	2	8	5-Feb	22.1						
CW	CHBL	2	3	2-Feb	20.4						
CX	CHBL	1	30	6-Feb	21.8						
CY	CHBL	1	75	3-Feb	18.2						
CZ	CHBL	1	30	3-Feb	19.8						
DA	CHBL	1	15	3-Feb	19.2						

Figure 8.1: A screen shot of the user interface of VINDSS after data have been imported from Microsoft Excel [86].

The harvesting period for which this harvesting schedule is to be generated is the five days starting on Monday, 9 February 2009 to Friday, 13 February 2009. The reasoning behind choosing such an early period is due to the large amount of non-electronic data that requires manual processing. Even though this period is towards the start of the 2009 harvesting season, spanning 29 January to 24 March, the acquired data indicate that an average of 373 tonnes of grapes was already being received per day, with a daily maximum of 561 tonnes received on Wednesday, 11 February 2009.

Therefore, the sample sugar levels concerned are from the first arrival of samples on 26 January 2009 until Friday, 6 February. All the sample sugar levels received are added to a list of vineyard blocks containing information regarding the contact person for the farm, the supplying farm, the cultivar, the block number and finally the sugar levels, pH and acidity, as mentioned in Chapter 4. The sugar levels received are listed by their cultivar and each block has been assigned a block name in order to easily refer to a specific vineyard block. The necessary information regarding the blocks and their calculated sugar levels is listed in Appendix C.

Furthermore, the imported data should also contain information regarding the contents of each non-empty red fermentation tank. The weight of the contents, the cultivar, class and an expected emptying date is required. The emptying date refers to the day on which the fermentation tank is emptied and the skins are transported to the presses.

After these data have been imported, they are presented graphically to the user in order to ensure that the correct set of data has been imported. A button labelled ‘Calculate Schedule’ appears as well as drop down menus to specify the harvesting period to consider. Further information regarding the process and specifics of importing the data is explained in §D.1.

## 8.2.2 Solving the harvest scheduling problem with VinDSS

For this instance of the scheduling problem at Wamakersvallei, a timespan of five days was selected from the drop down menu and the starting date was set as 9 February with the day of execution of the scheduling software taken as Friday, 7 February. The day of execution has no influence on the final harvesting schedule, since the date on which the sample was received until the start of the set harvesting period is the considered period for forecasting. Once the ‘Calculate Schedule’ button is pressed, the harvest scheduling tabu search is applied to the imported data shown in Figure 8.1. The first step when applying the harvesting tabu search to the data, is to generate the list of blocks based on the known sample sugar levels. The required vineyard block information regarding the construction of the list of blocks,  $\mathcal{B}$ , is listed in Appendix C. The resulting list of blocks, generated from the sample sugar levels is listed in Table 8.1. This list contains all the vineyard blocks that may fully ripen during the harvesting time period under consideration.

### The initial harvesting schedule

From the list of blocks, presented in Table 8.1, it is clear that during the start of the harvesting season, the main focus is on the harvesting of white grapes with only 33 of the 103 vineyard blocks received containing red grapes. Furthermore, this list contains 103 of the 450 vineyard blocks listed in Table C.1. The blocks are ordered according to their estimated urgency, as outlined in Algorithm 7.1 to be harvested as  $\{B_{64}, B_{39}, B_{30}, B_{60}, B_{57}, B_{59}, B_{69}, B_{36}, B_{18}, B_{14}, B_{24}, B_{45}, B_{63}, B_{33}, B_{62}, B_{37}, B_{41}, B_{20}, B_{47}, B_{25}, B_5, B_{31}, B_2, B_{12}, B_{50}, B_{22}, B_3, B_{43}, B_{19}, B_{44}, B_{58}, B_{35}, B_{46}, B_{51}, B_{52}, B_{54}, B_{11}, B_{34}, B_{17}, B_{40}, B_{13}, B_{65}, B_{10}, B_{71}, B_{55}, B_{15}, B_{48}, B_{72}, B_{73}, B_{74}, B_{75}, B_{76}, B_{77}, B_{80}, B_{81}, B_{82}, B_{83}, B_{84}, B_{85}, B_{86}, B_{89}, B_{90}, B_{91}, B_{92}, B_{93}, B_{94}, B_{95}, B_{96}, B_{97}, B_{100}, B_1, B_4, B_6, B_7, B_8, B_9, B_{16}, B_{21}, B_{23}, B_{26}, B_{27}, B_{29}, B_{32}, B_{42}, B_{49}, B_{53}, B_{56}, B_{61}, B_{102}, B_{79}, B_{78}, B_{98}, B_{101}, B_{28}, B_{38}, B_{66}, B_{67}, B_{68}, B_{70}, B_{87}, B_{88}, B_{103}, B_{99}\}$ .

As discussed in Chapter 7, the blocks are arranged in an initial harvesting schedule with an average daily grape intake taken as 290 tonnes and the maximum daily intake as 610 tonnes (from the Wamakersvallei 2006 data in Table 4.5). The initial harvesting solution, with its

Block list number	Block name	Cultivar	Class	Tonnes	Sample date received	Sample sugar level	Expected block sugar level
1	AB	CABF	3	8	05-Feb	22.3	23.3
2	IM	SHIR	1	15	03-Feb	23.0	24.5
3	IQ	SHIR	2	30	05-Feb	23.5	24.5
4	IR	SHIR	2	30	05-Feb	22.3	23.3
5	GH	PINO	2	10	02-Feb	23.8	25.6
6	GI	PINO	1	15	05-Feb	22.5	23.5
7	GL	PINO	1	5	02-Feb	25.7	27.5
8	GO	PINO	2	120	05-Feb	22.6	23.6
9	GQ	PINO	1	10	27-Jan	21.7	23.5
10	GV	PINO	1	5	02-Feb	23.2	25.0
11	GW	PINO	1	5	02-Feb	23.3	25.1
12	GY	PINO	1	9	05-Feb	24.5	25.5
13	HD	PINO	1	20	02-Feb	23.2	25.0
14	HE	PINO	2	11	05-Feb	25.2	26.2
15	HF	PINO	1	10	27-Jan	22.6	24.4
16	HI	PINO	1	40	06-Feb	23.0	23.8
17	FH	MERL	1	5	03-Feb	22.0	23.5
18	FJ	MERL	1	80	06-Feb	24.0	24.8
19	FK	MERL	1	70	06-Feb	23.1	23.9
20	FL	MERL	1	80	06-Feb	23.5	24.3
21	FM	MERL	2	40	05-Feb	25.3	26.3
22	FN	MERL	2	40	05-Feb	23.0	24.0
23	FO	MERL	2	40	05-Feb	24.8	25.8
24	FQ	MERL	1	40	05-Feb	23.7	24.7
25	FX	MERL	1	16	06-Feb	23.3	24.1
26	FZ	MERL	1	12	06-Feb	22.6	23.4
27	GB	MERL	1	10	06-Feb	22.6	23.4
28	GE	MERL	1	25	02-Feb	24.8	26.6
29	GF	MERL	1	35	06-Feb	25.5	26.3
30	GG	MERL	1	25	06-Feb	24.5	25.3
31	EZ	MALB	1	120	06-Feb	22.8	23.6
32	FA	MALB	1	70	06-Feb	22.1	22.9
33	FE	MALB	2	4	06-Feb	23.2	24.0
34	CB	CHBL	1	15	02-Feb	20.3	22.1
35	CC	CHBL	1	55	05-Feb	21.2	22.2
36	CD	CHBL	1	75	05-Feb	22.5	23.5
37	CE	CHBL	1	65	05-Feb	21.8	22.8
38	CF	CHBL	6	10	02-Feb	24.5	26.3
39	CH	CHBL	1	16	06-Feb	23.0	23.8
40	CI	CHBL	1	16	03-Feb	20.5	22.0

Table 8.1: The first 40 vineyard blocks of the ordered list of blocks to be considered for harvesting when applying the harvesting schedule to the sample sugar levels in Table C.1. The sugar levels of the grapes in each block expected on 9 February when the schedule period under consideration starts, are shown in the last column.

Block list number	Block name	Cultivar	Class	Tonnes	Sample date received	Sample sugar level	Expected block sugar level
41	CJ	CHBL	1	16	03-Feb	21.3	22.8
42	CK	CHBL	1	15	03-Feb	18.7	20.2
43	CR	CHBL	2	24	05-Feb	21.4	22.4
44	CS	CHBL	2	15	06-Feb	21.6	22.4
45	CV	CHBL	2	8	05-Feb	22.1	23.1
46	CW	CHBL	2	3	02-Feb	20.4	22.2
47	CX	CHBL	1	30	06-Feb	21.8	22.6
48	CZ	CHBL	1	30	03-Feb	19.8	21.3
49	DA	CHBL	1	15	03-Feb	19.2	20.7
50	DB	CHBL	2	15	03-Feb	21.0	22.5
51	DF	CHBL	2	6	06-Feb	21.4	22.2
52	DG	CHBL	2	6	06-Feb	21.4	22.2
53	DH	CHBL	1	65	03-Feb	18.8	20.3
54	DM	CHBL	2	10	06-Feb	21.4	22.2
55	DN	CHBL	1	5	06-Feb	21.1	21.9
56	DO	CHBL	1	5	06-Feb	19.8	20.6
57	DP	CHBL	1	10	06-Feb	22.9	23.7
58	DQ	CHBL	1	10	06-Feb	21.5	22.3
59	DR	CHBL	1	4	06-Feb	22.8	23.6
60	DS	CHBL	1	4	06-Feb	22.9	23.7
61	DU	CHBL	1	4	30-Jan	18.7	20.5
62	DV	CHBL	2	2	06-Feb	22.1	22.9
63	DW	CHBL	2	2	06-Feb	22.3	23.1
64	DX	CHBL	2	2	06-Feb	23.2	24.0
65	DZ	CHBL	1	8	02-Feb	20.2	22.0
66	EA	CHBL	3	4	06-Feb	24.2	25.0
67	EB	CHBL	3	4	02-Feb	22.7	24.5
68	EC	CHBL	3	4	06-Feb	23.6	24.4
69	EE	CHBL	1	40	06-Feb	22.8	23.6
70	EF	CHBL	1	20	03-Feb	18.4	19.9
71	EG	CHBL	1	2	06-Feb	21.2	22.0
72	EH	CHBL	1	20	03-Feb	20.7	22.2
73	EI	CHBL	1	17	06-Feb	20.4	21.2
74	EJ	CHBL	1	6	06-Feb	21.6	22.4
75	EL	CHBL	1	16	30-Jan	19.4	21.2
76	EM	CHBL	1	16	30-Jan	21.1	22.9
77	EN	CHBL	1	12	02-Feb	20.0	21.8
78	EQ	CHBL	2	12	26-Jan	18.4	20.2
79	ER	CHBL	1	15	26-Jan	18.7	20.5
80	ES	CHBL	1	18	06-Feb	22.3	23.1

Table 8.1 (continued): Vineyard blocks  $B_{41}, \dots, B_{80}$  of the ordered list of blocks to be considered for harvesting when applying the harvesting schedule to the sample sugar levels in Table C.1. The sugar levels of the grapes in each block expected on 9 February when the schedule period under consideration starts, are shown in the last column.

Block list number	Block name	Cultivar	Class	Tonnes	Sample date received	Sample sugar level	Expected block sugar level
81	EU	CHBL	1	25	06-Feb	23.1	23.9
82	EV	CHBL	1	6	02-Feb	19.9	21.7
83	HV	SAUV	1	35	02-Feb	19.0	20.8
84	BA	CHAR	3	16	04-Feb	23.1	24.4
85	BB	CHAR	1	4	05-Feb	22.4	23.4
86	BD	CHAR	1	70	02-Feb	22.4	24.2
87	BE	CHAR	1	80	02-Feb	23.8	25.6
88	BF	CHAR	1	23	02-Feb	25.0	26.8
89	BG	CHAR	3	5	05-Feb	22.8	23.8
90	BH	CHAR	1	75	06-Feb	23.0	23.8
91	BI	CHAR	2	5	30-Jan	21.1	22.9
92	BJ	CHAR	1	24	06-Feb	23.5	24.3
93	BK	CHAR	1	50	05-Feb	21.5	22.5
94	BP	CHAR	1	4	27-Jan	20.5	22.3
95	BQ	CHAR	1	10	05-Feb	22.3	23.3
96	BR	CHAR	1	7	05-Feb	22.0	23.0
97	BS	CHAR	3	35	06-Feb	22.9	23.7
98	BW	CHAR	1	10	02-Feb	19.5	21.3
99	BZ	CHAR	1	20	02-Feb	23.9	25.7
100	IW	VIOG	1	0	02-Feb	21.8	23.6
101	JA	VIOG	1	12	02-Feb	19.9	21.7
102	JC	VIOG	1	3	02-Feb	22.0	23.8
103	HK	RIES	1	5	02-Feb	22.8	24.6

Table 8.1 (continued): Vineyard blocks  $B_{81}, \dots, B_{103}$  of the ordered list of blocks to be considered for harvesting when applying the harvesting schedule to the sample sugar levels in Table C.1. The sugar levels of the grapes in each block expected on 9 February when the schedule period under consideration starts, are shown in the last column.

transpose arranged in table format, is presented in Table 8.2 along with the total expected yield of the vineyard blocks selected for harvesting expressed in tonnes for each harvesting day.

Thirty-four vineyard blocks were not selected for harvesting during the five day period considered. These blocks, in no particular order, are  $B_{100}, B_1, B_4, B_6, B_7, B_8, B_9, B_{16}, B_{21}, B_{23}, B_{26}, B_{27}, B_{29}, B_{32}, B_{42}, B_{49}, B_{53}, B_{56}, B_{61}, B_{102}, B_{79}, B_{78}, B_{98}, B_{101}, B_{28}, B_{38}, B_{66}, B_{67}, B_{68}, B_{70}, B_{87}, B_{88}, B_{103}, B_{99}$ . The first unconsidered position in the harvesting schedule, is position  $u = 73$ .

### Evaluating the harvesting schedules

The initial harvesting schedule,  $H_I$ , is taken as the best schedule to date and is evaluated in order to calculate the (best) current harvest evaluation score  $\vartheta$ . Consider the evaluation process outlined in Chapter 7, together with the suggested updates explained in §8.1.4 as indicated in Algorithm 7.3. The starting day of the selected harvesting period, *i.e.* day 1, is 9 February. Since  $|\mathcal{B}_1| > 0$ , the cellar scheduling scenario for day 1 may be generated. The generation of a scenario is now consider as it occurred during this application of the heuristic. Each of the vineyard blocks in the set  $\mathcal{B}_1 = \{ B_{64}, B_{39}, B_{30}, B_{60}, B_{57}, B_{59}, B_{69}, B_{36}, B_{18}, B_{14}, B_{24} \}$  is

$p$	$\mathcal{B}_1$	$\mathcal{B}_2$	$\mathcal{B}_3$	$\mathcal{B}_4$	$\mathcal{B}_5$
1	$B_{64}$	$B_{45}$	$B_2$	$B_{52}$	$B_{84}$
2	$B_{39}$	$B_{63}$	$B_{12}$	$B_{54}$	$B_{85}$
3	$B_{30}$	$B_{33}$	$B_{50}$	$B_{11}$	$B_{86}$
4	$B_{60}$	$B_{62}$	$B_{22}$	$B_{34}$	$B_{89}$
5	$B_{57}$	$B_{37}$	$B_3$	$B_{17}$	$B_{90}$
6	$B_{59}$	$B_{41}$	$B_{43}$	$B_{40}$	$B_{91}$
7	$B_{69}$	$B_{20}$	$B_{19}$	$B_{13}$	$B_{92}$
8	$B_{36}$	$B_{47}$	$B_{44}$	$B_{65}$	$B_{93}$
9	$B_{18}$	$B_{25}$	$B_{58}$	$B_{10}$	$B_{94}$
10	$B_{14}$	$B_5$	$B_{35}$	$B_{71}$	$B_{95}$
11	$B_{24}$	$B_{31}$	$B_{46}$	$B_{55}$	$B_{96}$
12	-1	-1	$B_{51}$	$B_{15}$	$B_{97}$
13	-1	-1	-1	$B_{48}$	-1
14	-1	-1	-1	$B_{72}$	-1
15	-1	-1	-1	$B_{73}$	-1
16	-1	-1	-1	$B_{74}$	-1
17	-1	-1	-1	$B_{75}$	-1
18	-1	-1	-1	$B_{76}$	-1
19	-1	-1	-1	$B_{77}$	-1
20	-1	-1	-1	$B_{80}$	-1
21	-1	-1	-1	$B_{81}$	-1
22	-1	-1	-1	$B_{82}$	-1
23	-1	-1	-1	$B_{83}$	-1
$\sum W_b$	307	353	292	306	305

Table 8.2: The initial harvesting schedule,  $\mathbf{H}_I$ , selected from the list of blocks presented in Table 8.1. The total expected weight of grapes scheduled for harvesting during each day is also included.

split into truckloads, as described in Chapter 7. During the next step in the generation of the scenario the arrival times are assigned to the expected grapeloads. The loads expected to arrive within 2.5 hours of one another are combined as discussed. The list of jobs generated consist of four Type I jobs, seven Type II and twenty-nine Type IV jobs, where Type IV jobs now refer to the red grapeloads that are ‘only’ assigned to the tipping bins with weight 0, as discussed in §8.1.2. The full list of jobs are presented in Table C.3 in Appendix C.

A scenario is generated for each of the sets  $\mathcal{B}_1, \dots, \mathcal{B}_5$  and solved using the cellar scheduling tabu search. Not one of the days are resolved within the set amount of time, with  $(\alpha - \alpha_{\max})$  ranging between 3 and 15. The average cellar scheduling score is calculated as 7625 and due to many vineyard blocks being assigned to positions  $\mathbf{H}_I(d, p)$ , with  $p \geq u$ , the total block scores add up to a further 50261 resulting in a harvest evaluation score of 57886, which is set as the current best evaluation score.

### 8.2.3 Generating the candidate list of moves

The first step in generating the list of tabu search candidate moves, is to select the vineyard block to which the swaps should be applied. The vineyard block sugar level scores,  $\Omega_b$ , are



considered. The maximum value contained in vector  $\Omega_{\mathbf{b}}$  is 5000. It is clear that such a large value is due to vineyard blocks with a grape quality of class 1 which are left to overripen. The vineyard blocks with a block score of 5000 are blocks  $B_7$ ,  $B_{21}$ ,  $B_{23}$  and  $B_{29}$ . Since this is the first iteration, none of the vineyard block candidates are present in the block tabu list. Therefore, a block is randomly selected from the four candidates and in this application of VINDSS, vineyard block  $B_{23}$  ( $\mathbf{H}_I(1, 78)$ ) was selected. Since the vineyard block is currently contained in the first row of the initial harvesting row, the two rows closest to row 1 are rows 2 and 3, referring to the assignment of the block to days 2 and 3, respectively. In Chapter 7 a move on the harvesting schedule was defined as an array containing first the block type of the selected block, then the block type of the block selected to be swapped with and then the four integers describing the row and column of each of the two blocks involved. Since vineyard block  $B_{23}$  is unassigned to a specific day for harvesting, the block is of block type 2 and the candidate list of moves therefore consists of 6 moves. The list,

$$\mathbf{L} = \left\{ \begin{array}{l} \left[ \begin{array}{cccccc} 2 & 0 & 1 & 78 & 2 & 2 \end{array} \right], \\ \left[ \begin{array}{cccccc} 2 & 1 & 1 & 78 & 2 & 11 \end{array} \right], \\ \left[ \begin{array}{cccccc} 2 & 0 & 1 & 78 & 2 & 4 \end{array} \right], \\ \left[ \begin{array}{cccccc} 2 & 0 & 1 & 78 & 3 & 11 \end{array} \right], \\ \left[ \begin{array}{cccccc} 2 & 1 & 1 & 78 & 3 & 12 \end{array} \right], \\ \left[ \begin{array}{cccccc} 2 & 0 & 1 & 78 & 3 & 7 \end{array} \right] \end{array} \right\}$$

is generated accordingly and evaluation of the moves results in the harvesting scores, 53122, 53014, 52960, 53040, 53162 and 53067 for each of the respective moves. Since this is the first iteration, all tabu lists are empty and therefore the move with the lowest harvesting score,  $\mathbf{L}(3)$ , is selected for application. Since  $52960 < \vartheta$ , the new best harvesting score is set as  $\vartheta = 52960$  and the new recorded best harvesting schedule is set to  $\mathbf{H}_1$  referring to the harvesting schedule after iteration 1 has been completed.

The inverse of the move being added to the move type tabu list, is  $[23, 0, 2, 2, 1]$ , thereby not allowing block  $B_{23}$  to be moved back from row 2 to a position in row 1 (resulting in a move type of 2). The swap move  $(23, 62)$  is placed in the swap tabu list, thereby also not allowing block  $B_{23}$  to be swapped with  $B_{62}$  for as long as the move is in the swap tabu list.

After the completion of the first iteration, the block score of block  $B_{23}$  has now improved to  $\Omega_{\mathbf{b}}(23) = 5$ . Blocks  $B_7$ ,  $B_{21}$  and  $B_{29}$  still have block scores of 5000 and therefore a block is randomly selected from these three blocks. In this instance of the tabu search, the vineyard block  $B_{29}$  was selected and the process of generating a list of candidate moves, evaluating each move and selecting the best move was repeated.

The process of generating the list of moves as described, comparing the moves to the tabu lists and finally selecting and applying the best move, is continued until the stopping criterion is satisfied. The stopping criterion is satisfied once a set number of iterations have been completed. The exact number of iterations required is considered in the next section.

### 8.3 A short analysis of the performed tabu search

Before the best solution may be compared to the harvest as it occurs at Wamakersvallei Winery, it is first necessary to consider whether a lower harvest evaluation score necessarily implies a better harvesting schedule. Two harvest schedules are now considered and compared, the first is the best schedule found by the tabu search described above after 50 iterations, and the second



varieties are received at the cellar, they are required to remain in the active cellar (possibly) being moved from the separators to the presses, which may last up to 6 hours. Therefore, the 559 tonnes of mostly white grape varieties received on day 2 of  $H_{50}$  is definitely not ideal. The maximum number of tonnes scheduled for one day in  $H_{100}$ , is 427 tonnes which seems much more reasonable for this time of the harvesting season.

Both  $H_{50}$  and  $H_{100}$  exhibit a day where the number of tonnes does not seem to fit the general pattern of the assigned weights throughout the schedule. For  $H_{50}$ , this is day 2 and for  $H_{100}$ , this is day 5, during which only 117 tonnes of grapes are scheduled for intake. However, as discussed in the previous section, assigning too much grapes to one day is definitely a worse strategy than assigning too little. Assigning too little grapes may possibly result in a large volume of grapes to be scheduled on day 1 of the next 5 day harvesting period. Furthermore, in order to fully measure the spread of the vineyard blocks over the harvesting period, the standard deviation in the assigned daily weight,  $\alpha$  is also included in the table. In this respect  $H_{100}$  is again the better solution when compared to  $H_{50}$ .

The analysis above does not yet say much about the quality of either of the harvesting schedules. However, this comparison shows that a decrease in the harvest evaluation score produces a better harvesting schedule. The final harvesting schedule is taken as the harvesting schedule  $H_{100}$  achieved after 100 iterations. A screen shot of the user interface of VINDSS is shown in Figure 8.2.

## 8.4 The suggested schedule vs Wamakersvallei selection

There are a few problem areas when attempting to verify the schedule suggested by VINDSS by comparing it to the Wamakersvallei harvest. Most of the data were not received electronically resulting in a painstaking process of piecing together when each vineyard block was harvested. Furthermore, different referencing names are used in the different sets of data when referring to the same vineyard block. Another aspect making it almost impossible to piece together the Wamakersvallei 2009 harvest, is that the list of sugar levels seems to be incomplete and also contains a few nonsensical entries. Some vineyard blocks arrived at the cellar for which no sugar level entry can be found. This might be due to the personal relationship between the Wamakersvallei team and some of the suppliers. For example, if a certain trusted supplier calls and says that he has a certain volume of grapes that is perfectly ripe and ready to be harvested, this vineyard block may be scheduled for harvesting without generating the necessary paperwork required to recreate the harvest.

The first step in comparing the Wamakersvallei scenario with the suggested schedule, is to consider the total weight of the vineyard blocks received during each day of the harvesting schedule and to compare this with the corresponding value for the schedule  $H_{100}$ . This comparison is presented in Table 8.3.

Two aspects stand out regarding the Wamakersvallei daily grape weights received. The first is the very small volume of grapes received at the cellar during the last day of harvesting. On closer inspection, it was found that the 23 tonnes of grapes harvested on day 5 are all from vineyard blocks where harvesting have started but not finished. There exists a variety of reasons for starting the process of harvesting a specific vineyard block and not finishing the process. However, the most common would be that there was no space or time to receive grapes at the cellar. The second aspect is the enormous volume of grapes being harvested on day 3 of the Wamakersvallei harvesting schedule, which is not at all typical of the start of the harvesting

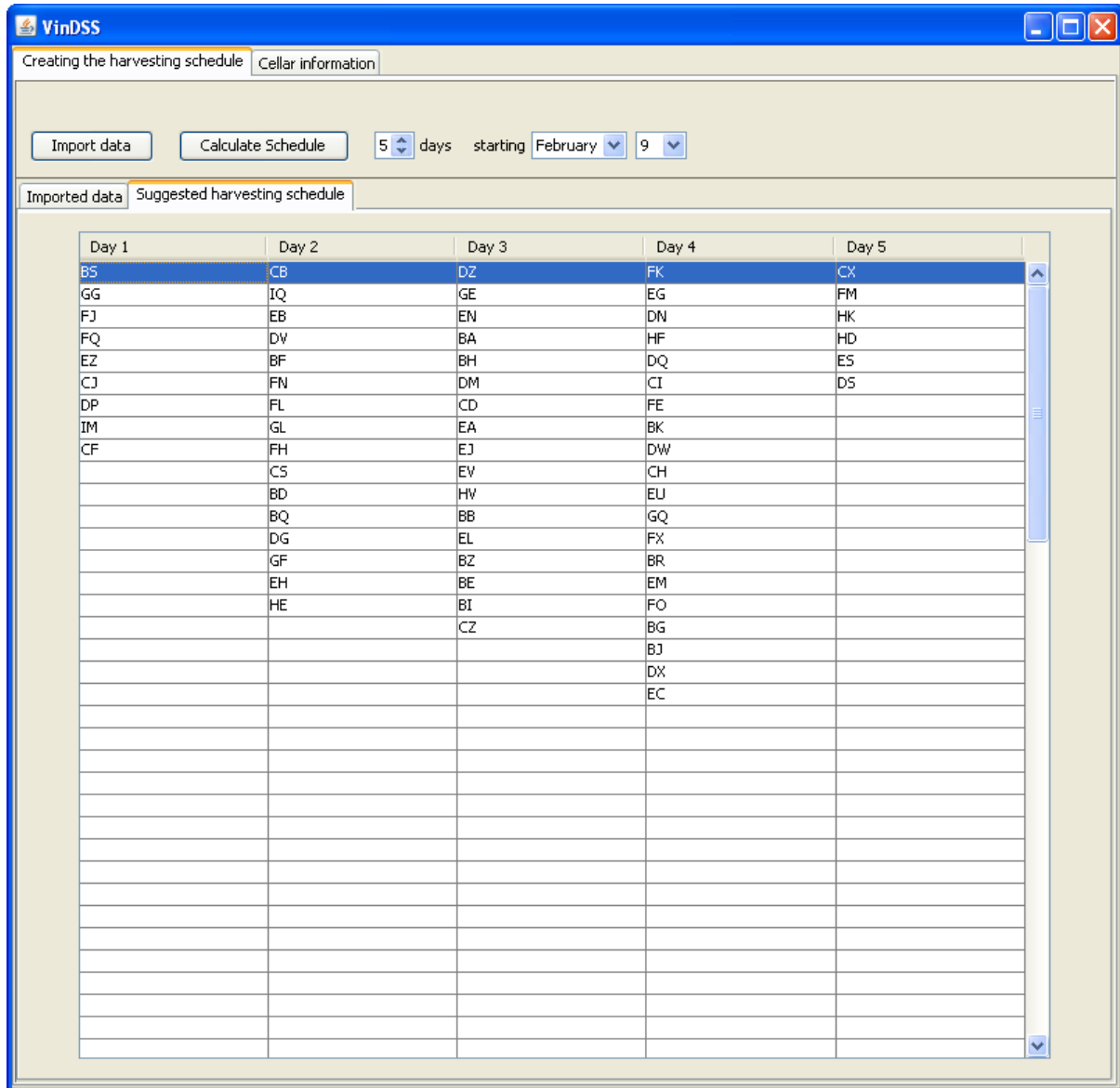


Figure 8.2: A screen shot of the user interface of VINDSS after generating the harvesting schedule.

season. During the five day harvesting period considered, a total of over 400 tonnes of grapes were received at the cellar without any trace of samples being received. This definitely influences the outcome of the schedule, since the 408 tonnes were most likely not scheduled and received from a supplier who had a large amount of excess grapes<sup>3</sup>. If this is indeed the case, and it very well may be, the scheduled volume of grapes received by Wamakersvallei and the volumes in harvesting schedule  $H_{100}$  are in very close proximity to one another. Furthermore, 120 tonnes of the 408 tonnes was received on day 3, possibly explaining the unexpected large volume of grapes harvested on this day. The harvesting of vineyard blocks in the schedule generated by VINDSS is better distributed over the harvesting period than that of Wamakersvallei.

<sup>3</sup>This often happens when a grape supplier supplies grapes to more than one winery. The staff at Wamakersvallei seem inclined to help out in such cases by taking in the additional grapes.

Property	Vineyard block set	$\mathbf{H}_{100}$	Wamakersvallei harvest
$\sum W_b$	$\mathcal{B}_1$	351	393
$\sum W_b$	$\mathcal{B}_2$	371	497
$\sum W_b$	$\mathcal{B}_3$	427	561
$\sum W_b$	$\mathcal{B}_4$	329	390
$\sum W_b$	$\mathcal{B}_5$	117	23
Total		1 595	1 865
$\bar{W}$		319	373
$\alpha$		118.6	208.6

Table 8.4: An analysis of weight of grapes scheduled for intake during each day in harvesting schedules  $\mathbf{H}_{100}$  and the actual 2009 harvest at Wamakersvallei, where  $\bar{W}$  denotes the average weight assigned per day and  $\alpha$  denotes the standard deviation of the daily weights per harvesting schedule.

## 8.5 Chapter overview

The goal in this chapter was to illustrate the working of the solution process of the harvest scheduling problem as part of the decision support system, VINDSS. As mentioned earlier in this thesis, each winery will require its own set of minor alterations to the generic cellar and harvest scheduling tabu searches considered in Chapters 6 and 7 and incorporated into VINDSS, thereby completing thesis object III expressed in Chapter 1. The required changes to the tabu searches in order to best fit the Wamakersvallei cellar were outlined in §8.1.

In §8.2, the working of the decision support system, VINDSS, was illustrated by applying the tabu searches to solve the harvest scheduling problem experienced at Wamakersvallei Winery. The data set used was composed by considering the sugar level forms filled in by the viticulturist (as illustrated in Figure 4.8) during the Wamakervallei 2009 harvesting period. In order to find the expected yield for each of the vineyard blocks, the information contained in the form was cross-referenced with both the harvesting sheets (as illustrated in Figure 4.9) and the harvesting data recorded during previous years. Unfortunately, for some of the vineyard blocks further information was not available and instances where a vineyard block was received at the cellar without a sugar sample being noted was also rather common. However, this shortcoming is taken into account when the harvesting schedule generated by VINDSS was compared to the daily schedules actually implemented by the Wamakersvallei team.

Two harvesting schedules uncovered at different iterations of the harvest scheduling problem tabu search, namely  $\mathbf{H}_{50}$  and  $\mathbf{H}_{100}$  taken after 50 and 100 iterations, respectively, were considered in §8.3. The two schedules were compared to one another and it was determined that harvesting schedule  $\mathbf{H}_{100}$  is a better schedule than harvesting schedule  $\mathbf{H}_{50}$ . This is to be expected, since the tabu search was allowed to explore the solution neighbourhood for 50 more iterations resulting in a better harvest evaluation score found for harvesting schedule  $\mathbf{H}_{100}$  than for harvesting schedule  $\mathbf{H}_{50}$ .

Finally, the final harvesting schedule generated after 100 iterations of the harvest scheduling tabu search,  $\mathbf{H}_{100}$ , was compared to the actual Wamakersvallei 2009 harvest in §8.4. It was found that a similar grape volume was harvested during both harvesting schedules. However, harvesting schedule  $\mathbf{H}_{100}$  contained a better distribution of harvested vineyard blocks over the five day period considered.



---

---

## CHAPTER 9

---

# Conclusion

### Contents

9.1 Thesis summary . . . . .	165
9.2 Suggestions and recommendations . . . . .	166
9.3 Possible future work . . . . .	167
9.3.1 <i>Improving the cellar and harvest scheduling problems</i> . . . . .	167
9.3.2 <i>Improving the mathematical representation of winery characteristics</i> . . . . .	169
9.3.3 <i>Improving the functionality of VINDSS</i> . . . . .	169

This chapter consists of three sections. The first section, §9.1, contains a brief summary of work contained in this thesis. In §9.2, a suggestion is made with respect to any winery interested in considering the VINDSS approach towards harvest scheduling, in terms of data that would be required. Suggestions for possible work to further this study is considered in §9.3.

### 9.1 Thesis summary

In the introduction to this thesis, Chapter 1, a short overview of wine making in South Africa was presented with some focus on the history of how the wine industry came to be. An informal introduction to the scheduling problems considered in this thesis was presented, along with the objectives pursued for this thesis.

The aim in the second chapter of this thesis was to provide the reader with an understanding of wine and the South African wine industry, with the development of the South African wine industry organisational structure as well as its current state forming an integral part of this chapter. Some statistics were given on wine production and the import and export of wine in South Africa. Furthermore, viticultural practices and a basic overview of wine making methods were considered.

The necessary background information on scheduling problems considered in this thesis and possible solution methods were presented in Chapter 3 with a view to properly identify the scheduling problems experienced at a winery. The chapter also contains a concise survey of literature regarding the job shop scheduling problem with some focus on the flexible job shop scheduling problem. However, this subproblem was only identified in 1994 and comparatively little research is available on this problem. A mixed integer programming model for the flexible job shop scheduling problem with sequence-dependent setup times and a no-wait characteristic

was reviewed and a brief description was included of the branch-and-bound method of solving integer programming problems. An introduction to the tabu search method used to solve both the cellar and harvest scheduling problems was also outlined in this chapter.

In order to fully understand the origin of the scheduling problems experienced at Wamakersvallei, some important aspects of the case study winery is required. These aspects were discussed in Chapter 4, including information such as the physical layout and location of the cellar, the machinery in the cellar, the wines of Wamakersvallei and the staff employed there. The current order of processing and receiving grapes at the cellar was described with a more in-depth look at the different requirements of producing red or white wine.

The active cellar scheduling problem experienced at Wamakersvallei winery deals with the assignment and ordering of certain processes required in the production of wine. The active cellar refers to the problem area where bottlenecks often occur during the busy harvesting season. The purpose of Chapter 5 was to derive a mathematical programming model in order to consider an exact solution approach to the active cellar programming problem. However, after solving the resulting mathematical programming model for a small, fictitious cellar, it was found that this method of solving the active cellar scheduling problem is too time consuming when applied to the larger scheduling problem occurring at Wamakersvallei.

However, in Chapter 6, an alternative tabu search approach towards solving the cellar scheduling problem was developed. The tabu search solved the small fictitious cellar optimally, 100% of the time, when the problem was solved a thousand times. The computer processing time associated with the tabu search approach also constituted a significant improvement over that of the branch-and-bound approach, the former being up to 1 000 times faster.

The goal of Chapter 7 was to develop a generic tabu search method which may be applied to solve the harvesting schedule for a general winery. This schedule is found by referring to the active cellar scheduling tabu search in order to verify the impact that harvesting moves have on activities in the active cellar. One harvesting schedule is considered a better solution than another when it has a lower harvest evaluation score, which is determined by the placement of the vineyard blocks in the schedule.

Some minor changes to the active cellar and harvest scheduling tabu searches were applied to the general tabu searches presented in Chapters 6 and 7. Chapter 8 contains an overview of the resulting decision support system, VINDSS, based on the tabu search approaches of Chapters 6 and 7. It was found that a harvesting schedule delivered by VINDSS is a good schedule when compared to a five day period of the 2009 harvest at Wamakersvallei. However, the processing time required when solving the harvest scheduling problem at Wamakersvallei requires between two and three hours to deliver a good harvesting schedule.

## 9.2 Suggestions and recommendations

In order to apply forecasting methods to estimate the sugar levels of vineyard blocks, significant volumes of data are required. It is therefore suggested that any winery considering applying such forecasting techniques should keep sugar level data of samples received in a disciplined fashion, electronically. Each vineyard block should be assigned a unique reference number or name and the data should be kept in a database allowing for easy extraction thereof. The weather reported for each day during the harvesting schedule should be included in the database, specifically the minimum and maximum temperatures, as well as the rainfall. This would make it possible to properly investigate the relationship between these circumstances and the sugar content of the



grapes. It is expected that the sugar content should drop in rainy weather, since the grapes absorb excess water, and also that the grapes would have much higher sugar levels during warm weather, since more moisture is lost from the grapes. Both these situations may result in an inaccurate representation of the maturity of the grapes and is therefore a very important place to start if further research is attempted in this field.

Furthermore, data as to the exact route each grape load takes inside the active cellar would be a great help in evaluating any application of a solution to the cellar scheduling problem. Such a data set should contain, for each truckload, all processors it was assigned to (with processor numbers included, *i.e.* not just ‘press’ but ‘press  $x$ ’, for example) and also, very importantly, the time the processing on each of the machines lasted. This will lead to better processing time approximations that may be based on the cultivar of the grape (if enough data are made available). This set of data should also include the time of the arrival of the truckload at the cellar in order to better simulate the arrival of the delivery trucks.

### 9.3 Possible future work

As with most projects, there is never enough time to follow through with all the ideas conceived during the research period. This section therefore contains some suggestions with respect to possible future work. The suggestions may be classified into three classes; the first is considered in §9.3.1 and is concerned with possible improvements with respect to the cellar and harvest scheduling meta-heuristics developed in this thesis. Then, in §9.3.2, possible improvements are suggested with respect to the mathematical approximation of the winery characteristics. In order to successfully research these suggestions, some of the data described in §9.2 may be required. Finally, in §9.3.3, some suggestions are made regarding increased functionality of VINDSS. This section mostly contains further areas where the scheduling solution methods developed in this thesis might be applied to solve a (possibly) different problem.

#### 9.3.1 Improving the cellar and harvest scheduling problems

Even though the general tabu searches described in Chapters 6 and 7 were adapted to better solve the Wamakersvallei scheduling problems, some improvement may still be made. This section contains some alternate methods to parts of the cellar and harvest scheduling problems that are yet to be explored.

**Suggestion 9.1** *Consider a third-level, nested tabu search within the active cellar scheduling problem to replace the ejection chain move to generate a list of moves.* ■

This suggestion would be particularly interesting to consider as a means of generating a list of moves for the set of presses when a large active cellar scheduling problem is considered. The reason why the ejection chain method developed in Chapter 6 was replaced by the separate order and assignment method, was due to the increased processing time and space that is required to apply the ejection chain move to the set of processors. The number of combinations to consider just becomes too large and with the random generation of moves it requires thousands of iterations before an assignment preserving a order throughout the presses is found.

This third tabu search may consider a means of evaluating the moving of a job from one row of the order to another, rather than randomly selecting the moves. There are numerous ways in which such moves may be evaluated. Consider, for example, Algorithm 6.11. Rather

than breaking the loop if an infeasible solution occurs, the *infeasibility count* may be increased and the process continued. The lower the infeasibility count, the better (or closer to feasible) the solution. This infeasibility count may be combined with an *estimated move impact score* referring to the expected impact that a move might have on the assignment order generated by the move. For example, moving a Type III job from a position where it follows a Type II job to a position where it precedes a Type II job, should be considered a bad move since it incurs a higher setup time. By including the estimated move impact score, the assignment orders generated from the moves should be of a higher quality.

**Suggestion 9.2** *Consider a more representative means of evaluating an order matrix in the active cellar scheduling problem than the completion time currently considered.* ■

Currently the main evaluation index of the order matrix is the completion time — the tabu search strives to find a solution with the smallest completion time. The total setup time required by the suggested order matrix is only used as a tiebreaker. A more representative means of evaluating the cellar schedule may be to combine the completion time workforce requirements. For example, in order to wash a tipping bin after red grapes have been received may take only one hour. However, stopping the intake of grapes and removing a worker from the cellar floor to clean the tipping bin in the middle of the day is more expensive than the cost incurred by a delay of one hour. Another example where considering the setup time as part of the evaluation may be desired, is when a Type II job is assigned to a press directly after a Type III job has been pressed on the same press. The Type II job is the last job to be scheduled on the specific press, but another press will be active for a couple more hours. In this case, focussing only on the completion time, the unnecessarily bad assignment is not penalized. Therefore, an evaluation method directly penalizing setup times should result in a more representative schedule with assignments guaranteed to be spread more evenly over the processors.

**Suggestion 9.3** *Find a better method of generating an initial solution to the harvest scheduling problem.* ■

During the first couple of iterations of the harvest scheduling tabu search, major improvements are made to the initial harvesting schedule, mainly due to blocks that were placed in positions  $H_I(d, p)$ , with  $p \geq u$ , which overripen during the harvesting period under consideration. A simple, yet (possibly) effective method of improving the initial harvest scheduling would be to ensure that no block remains unassigned if it is expected to exceed the upper bound on the first-class sugar level during the harvesting period.

**Suggestion 9.4** *Add an ejection chain move as a diversification move to the current harvest scheduling problem.* ■

When the best harvest evaluation score for each iteration is considered over a large number of iterations, it sometimes happens that the best harvest evaluation score remains unchanged for a very large number of iterations. Furthermore, it is possible that some vineyard blocks are not subjected to moves and it is not clear whether this is due to the fact that blocks have been assigned to their ‘perfect’ positions or whether their assigned positions are just never bad enough. Both of these situations may be avoided by applying a randomly generated ejection chain move. A possible list of ejection chain moves may be generated easily using the general ejection chain of Algorithm 6.10. Since the order in which the moves are assigned is irrelevant, all moves generated are feasible. The move with the best harvest evaluation score may then be selected and the tabu lists cleared.

### 9.3.2 Improving the mathematical representation of winery characteristics

In this section, some improvements in the formulation of the winery characteristics are suggested. However, in order to apply any of these suggestions, further data are required (as mentioned in §9.2).

**Suggestion 9.5** *Consider a more efficient measure of optimal grape ripeness.* ■

The reliability of considering only the sugar level of grapes as an indicator of the ripeness of the grapes has been doubted for a long time [131]. Studies have reported a block of grapes harvested at the same sugar level for four years in a row (Pinotage at 22°B) still result in fluctuations in grape quality [131]. However, it has been determined that an index of sugar multiplied by the pH level may serve as a much better yardstick for the ripeness of certain red cultivars [131]. Finding the best index for each cultivar might require processing large quantities of data.

**Suggestion 9.6** *Reconsider the forecasting technique used to calculate the expected sugar levels of the vineyard blocks.* ■

The credibility of the harvest scheduling solution suggested will increase significantly if more research can be done concerning the forecasting of optimal grape ripeness. The data made available for this study were not sufficient to derive proper forecasting techniques. It is very important to accommodate the daily rainfall in order to fully understand grape maturation, since the sugar levels in the grapes may be expected to decrease during a rainy week as a result of grapes absorbing water. The relationship between rainfall and sugar levels of grapes should also be explored in order to find a good approach when compared to forecasting grape maturation dates. Researching and developing such a model may yield a drastic improvement when compared to the current approach.

### 9.3.3 Improving the functionality of VinDSS

VINDSS is currently a very basic, yet efficient, decision support system that may be used to determine good harvesting schedules. However, the solution methods incorporated into the system for the active cellar scheduling and harvest scheduling problems may be applied in a wider context than is currently the case if some alterations are made.

**Suggestion 9.7** *Include the option of finding a solution to only the active cellar scheduling problem when the list of vineyard blocks for harvesting on a specific day has already been selected. The cellar scheduling problem should then be solved for the full cellar by adding the storage tanks to the problem definition.* ■

This functionality will be particularly helpful to smaller estate wineries where the vineyard blocks are all owned by the cellar — even more so if it is an estate only producing red (or white) wines, since this implies that all the vineyard blocks should be nearing optimal harvesting dates around the same time. It is to be expected that the size of such a cellar is much smaller than that of the case study winery. An optimal assignment of grape loads to the machinery inside the cellar is very important. In a smaller cellar it should also be possible to include the rest of the cellar in the scheduling problem (as opposed to only focussing on the active cellar).

The addition of another set of processors is easily accommodated by increasing the number of machine sets and generating an additional move for assignments made to these machines. These moves should follow a similar process to the moves applied to the red fermentation tanks, seeing that the rest of the cellar also consists of tanks and the problem when assigning jobs to the tanks is similar to that of assigning jobs to the red fermentation tanks.

When the cellar scheduling problem is adapted to fit a whole cellar, it may be applied to determine the influence of decisions at the tipping bins on later stages of processing or storage facilities (for example, what would be to gain or to lose if only white or only red grapes were to be accepted during one particular day). This could very well yield surprisingly good results since it will eliminate most setup times. The question is whether this will result in bottlenecks in other parts of the cellar.

**Suggestion 9.8** *Rather than fixing the cellar information, such as the number of tipping bins, allow the user to change these settings.* ■

By including this simple functionality, the cellar scheduling solution method may be applied by cellar management to determine which processors may possibly be unnecessary or scheduled for maintenance. It would also be helpful when the cellar is expanded. By adding different machine types in different quantities and sizes, the delivered schedule may be analysed in order to support a decision of whether to expand in certain areas rather than others. This added feature should easily answer questions such as: How would adding an extra tipping bin influence the working inside the cellar? Will this just lead to bottlenecks occurring at the presses or would more varieties be able to enter the cellar without too much setup time being wasted. The user may thus analyse the optimality of the physical cellar.

---

## References

- [1] AARTS EHL, VAN LAARHOVEN PJM, LENSTRA, JK & ULDER NLJ, 1994, *A computational study of local search algorithms for job shop scheduling*, Operations Research Society of America Journal of Computing, **6**(2), pp. 118–125.
- [2] ADAMS J, BALAS E & ZAWACK D, 1988, *The shifting bottleneck procedure for job shop scheduling*, Management Science, **34**, pp. 391–401.
- [3] AL-TURKI U, FEDJKI C & ANIJANI A, 2001, *Tabu search for a class of single-machine scheduling problems*, Computers & Operations Research, **28**, pp. 1223–1230.
- [4] APPLGATE D & COOK W, 1991, *A computational study of the job shop scheduling problem*, Operations Research Society of America Journal on Computing, **3**(2), pp. 149–156.
- [5] ARKELL J, 2003, *Wine — A comprehensive guide to drinking and appreciating wine*, New Holland Publishers (UK) Ltd, London.
- [6] ASHOUR S & HIREMATH SR, 1973, *A branch-and-bound approach to the job-shop scheduling problem*, International Journal of Production Research, **11**(1), pp. 47–58.
- [7] AUGER A, FERRER J, MATURANA S & VERA J, 2003, *Simulation of the grape reception at a winery*, [Online], [cited January 31st, 2007], Available from [http://www.gepuc.cl/publicaciones/Simulation\\_of\\_the\\_Grape-AA-JCF-SM-JV.pdf](http://www.gepuc.cl/publicaciones/Simulation_of_the_Grape-AA-JCF-SM-JV.pdf)
- [8] BAKER KR, 1984, *Sequencing rules and due date assignments in a job shop*, Management Science, **30**, pp. 1093–1104.
- [9] BALAS E, LENSTRA JK, VAZACOPOULOS A, 1995, *The one-machine problem with delayed precedence constraints and its use in job shop scheduling*, Management Science, **41**(1), pp. 94–109.
- [10] BALAS E & VAZACOPOULOS A, 1998, *Guided local search with shifting bottleneck for job-shop scheduling*, Management Science, **44**(2), pp. 262–275.
- [11] BATITI R, 1996, *Reactive search: Toward self-tuning heuristics*, pp. 61–83 in RAYWARD-SMITH VF, OSMAN IH, REEVES CR & SMITH GD (EDS), *Modern heuristic search methods*, John Wiley and Sons Ltd, Chichester.
- [12] BATITI R & TECCHIOLLI G, 1994, *The reactive tabu search*, Operations Research Society of America Journal on Computing, **7**, pp. 126–140.
- [13] BAYKASOĞLU A, 2002, *Linguistic-based meta-heuristic optimization model for flexible job shop scheduling*, International Journal of Production Research, **40**(17), pp. 4523–4543.

- [14] BESTER MJ, 2005, *Design of an automated decision support system for scheduling tasks in a generalized job-shop*, MSc Thesis, University of Stellenbosch, Stellenbosch.
- [15] BINATO S, HENRY WJ, LOEWENSTERN DM & RESENDE MGC, 2000, *A grasp for job shop scheduling*, AT&T Labs Research Technical Report: 00.6.1, pp. 1–17.
- [16] BŁAZĘWICS J, ECKER KH, PESCH E, SCHMIDT G & WEGLARZ J, 2001, *Scheduling computer and manufacturing processes*, 2<sup>nd</sup> Edition, Springer-Verlag, Berlin.
- [17] BŁAZĘWICS J, DROR M & WEGLARZ J, 1991, *Mathematical programming formulations for machine scheduling: a survey*, European Journal of Operational Research, **51**, pp. 283–300.
- [18] BŁAZĘWICS J, PESCH E & STERNA M, 2000, *The disjunctive graph machine representation of the job shop scheduling problem*, European Journal of Operational Research, **127**, pp. 317–331.
- [19] BRANDIMARTE P, 1993, *Routing and scheduling in a flexible job shop by tabu search*, Annals of Operations Research, **41**(3), pp. 157–183.
- [20] BRIZUELA CA & SANNOMIYA N, 2000, *From the classical job shop to a real problem: A genetic algorithm approach*, Proceedings of the 39th IEEE Conference on Decision and Control, IEEE, Sydney, pp. 4174–4180.
- [21] BRUCKER P, 2001, *Scheduling algorithms*, 3<sup>rd</sup> Edition, Springer-Verlag, Berlin.
- [22] BRUCKER P, JURISCH B & KRÄMER A, 1997, *Complexity of scheduling problems with multi-purpose machines*, Annals of Operations Research, **70**, pp. 57–73.
- [23] BRUCKER P, JURISCH B & SIEVERS B, 1994, *A branch and bound algorithm for the job-shop scheduling problem*, Discrete Applied Mathematics, **49**, pp. 109–127.
- [24] BROOKS GH & WHITE CR, 1965, *An algorithm for finding optimal or near-optimal solutions to the production scheduling problem*, The Journal of Industrial Engineering, **16**, pp. 34–40.
- [25] CAPE TOWN ROUTES UNLIMITED, 2007, *Wellington*, [Online], [cited February 20th, 2007], Available from [http://www.tourismcapetown.co.za/xx1/\\_lang/en/\\_site/visit-travel/\\_area/westerncape/\\_subArea/355760/\\_subArea2/358119/\\_subArea3/364754/\\_articleId/364773/index.html](http://www.tourismcapetown.co.za/xx1/_lang/en/_site/visit-travel/_area/westerncape/_subArea/355760/_subArea2/358119/_subArea3/364754/_articleId/364773/index.html)
- [26] CARLIER J, 1982, *The one-machine sequencing problem*, European Journal of Operational Research, **11**, pp. 42–27.
- [27] CARLIER J & PINSON E, 1989, *An algorithm for solving the job-shop problem*, Management Science, **35**, pp. 164–176.
- [28] CARLIER J & PINSON E, 1994, *Adjustment of heads and tails for the job-shop problem*, European Journal of Operational Research, **78**, pp. 146–161.
- [29] CHARLTON JM & DEATH CC, 1970, *A generalized machine-scheduling algorithm*, Operational Research Quarterly, **21**(1), pp. 127–134.
- [30] CHEN J & CHEN FF, 2003, *Adoptive scheduling in random flexible manufacturing systems subject to machine breakdowns*, International Journal of Production Research, **41**, pp. 1927–1951.

- [31] CHEN JC, CHEN KH, WU JJ & CHEN CW, 2007, *A study of the flexible job shop scheduling problem with parallel machines and reentrant process*, The International Journal of Advanced Manufacturing Technology, Springer-Verlag London Limited 2007, [Online], [cited March 11th, 2008], Available from <http://www.springerlink.com/content/317358276103j4k1/>
- [32] CHEN H & LUH PB, 2003, *An alternative framework to Lagrangian relaxation approach for job shop scheduling*, European Journal of Operational Research, **149**, pp. 499–512.
- [33] COLLINS CONCISE DICTIONARY 21ST CENTURY EDITION, 5<sup>th</sup> Edition, HarperCollins Publishers, Glasgow.
- [34] CRAWFORD JM, DALAL M & WALSER JP, 1998, *Abstract local search*, [Online], [cited June 9th, 2009], Available from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.2287>
- [35] DAUZERE-PERES S & LASSERRE JB, 1993, *A modified shifting bottleneck procedure for job shop scheduling*, International Journal for Production Research, **31**, pp. 11–39.
- [36] DELL CROCE F, TADEI R & VOLTA G, 1995, *A genetic algorithm for the job shop problem*, Computers and Operations Research, **22**, pp. 15–24.
- [37] DEMIRKOL E, MEHTA S & UZSOY R, 1997, *A computational study of shifting bottleneck procedures for shop scheduling problems*, Journal of Heuristics, **3**(2), pp. 111–137.
- [38] DEPARTMENT OF LOGISTICS, UNIVERSITY OF STELLENBOSCH, 2000, *Optimal viticulture practice*, Report prepared for Winetech Vision 2020, Stellenbosch.
- [39] DEPARTMENT OF INDUSTRIAL ENGINEERING, UNIVERSITY OF STELLENBOSCH, 2000, *Best practise study on the wine cellar and wine making environment*, Report prepared for Winetech Vision 2020, Stellenbosch.
- [40] DEYER ME & WOLSEY LA, 1990, *Formulating the single machine sequencing problem with release dates as a mixed integer program*, Discrete Applied Mathematics, **26**, pp. 255–270.
- [41] DUNSTALL S & JOHNSTONE R, 2005, *Applying innovative decision support technologies to achieve harmony and adaptability in an Australian wine supply network*, Media release, SMART 2005 Conference, Australia.
- [42] EWINEPLANET.COM, 2007, *South Africa*, [Online], [cited May 2nd, 2007], Available from <http://ewineplanet.com/country.asp?id=23>
- [43] EZY SYSTEMS, 2004, *Business software you can depend on*, [Online], [cited February 5th, 2008], Available from <http://www.ezysys.com.au>
- [44] FERRER JC, MACCAWLEY A, MATURANA S & VERA J, 2008, *An optimization approach for scheduling wine grape harvest operations*, International Journal of Production Economics, **112**, pp. 985–999.
- [45] FISHER ML, 1973, *Optimal solution of scheduling problems using Lagrange multipliers: Part I*, Operations Research, **21**, pp. 1114–1127.
- [46] FLORIAN M, TREPANT P & MCMAHON G, 1971, *An implicit enumeration algorithm for the machine sequencing problem*, Management Science, **17**, pp. B-789–B-792.

- [47] GAREY MR, JOHNSON DS & SETHI R, 1976, *The complexity of flow shop and job shop scheduling*, Mathematics of Operations Research, **1**(2), pp. 117–129.
- [48] GERE WS, *Heuristics in job shop scheduling*, 1996, Management Science, **13**, pp. 167–190.
- [49] GERTIOSO C, 1988, *A decision support system for wine-making cooperatives*, European Journal of Operational Research, **33**, pp. 273–278.
- [50] GIFFLER B & THOMPSON GL, 1960, *Algorithms for solving production scheduling problems*, Operations Research, **8**, pp. 487–503.
- [51] GLOVER F, 1990, *Tabu search: A tutorial*, Interfaces, **20**, pp. 74–94.
- [52] GLOVER F & LAGUNA M, 1993, *Tabu search*, pp. 70–150 in REEVES CR (ED), *Modern heuristic techniques for combinatorial problems*, Blackwell Scientific Publications, Oxford.
- [53] GRABOWSKI J, NOWICKI E & ZDRZLKA S, 1986, *A block approach for single-machine scheduling with release dates and due dates*, European Journal of Operational Research, **26**, pp. 278–285.
- [54] GRABOWSKI J & WODECKI M, 2005, *A very fast tabu search algorithm for job shop problem*, pp. 117–144 in REGO C & ALIDAEE B (EDS), *Metaheuristic optimization via memory and evolution: Tabu search and scatter search*, Kluwer Academic Publishers, Hingham (MA).
- [55] GRAPE, 2006, *New chair of SA Wine Industry Council takes over*, Grape News, 17 October 2006, [Online], [cited May 8th, 2007], Available from <http://www.grape.co.za/News/061011Asmal.htm>
- [56] GREENBERG HH, 1968, *A branch-bound solution to the general scheduling problem*, Operations Research, **16**, pp. 353–361.
- [57] HANSEN G, 2000, *Cognitive process simulation in wine making: Determining the best route for wine transfers*, Wine Business Monthly, October 2000, [Online], [cited June 5th, 2009], Available from <http://www.winebusiness.com/wbm/?go=getArticle&dataId=3563>
- [58] HOOKER JN, 2007, *Integrated methods for optimization*, Springer, New York (NY).
- [59] HURINK J, JURISCH B & THOLE M, 1994, *Tabu search for the job-shop scheduling problem with multi-purpose machines*, OR Spectrum, **15**, pp. 205–215.
- [60] INMAN RA, 2006, *Layout*, [Online], [cited July 26th, 2007], Available from <http://www.referenceforbusiness.com/management/Int-Loc/Layout.html>
- [61] JAIN AS & MEERAN S, 1998, *A state-of-the-art review of job shop scheduling techniques*, Technical Report, Department of Applied Physics, Electronics and Mechanical Engineering, University of Dundee, Dundee.
- [62] JAIN AS, RANGASWAMY B & MEERAN S, 1998, *Job shop neighbourhoods and move evaluation strategies*, [Online], [cited June 7th, 2009], Available from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.790>
- [63] JAIN AS, RANGASWAMY B & MEERAN S, 2000, *New and “stronger” job-shop neighbourhoods: A focus on the method of Nowicki and Smutnicki*, Journal of Heuristics, **6**(4), pp. 1–29.



- [64] JANSEN K, SOLIS-OBA R & SVIRIDENKO MI, 1999, *A linear time approximation scheme for job shop scheduling*, [Online], [cited June 9th, 2009], Available from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.8172>
- [65] JANSEN K, SOLIS-OBA R & SVIRIDENKO MI, 1999, *Makespan minimization in job shops: a polynomial time approximation scheme*, Proceedings of the 31st Annual ACM Symposium on the Theory of Computing (STOC'99).
- [66] JENSEN MT, 2003, *Generating robust and flexible job shop schedules using genetic algorithms*, IEEE Transactions on Evolutionary Computation, **7**(3), pp. 275–288.
- [67] JOHNSON SM, 1954, *Optimal two-and three-stage production schedules with setup times included*, Naval Research Logistics Quarterly, **1**, pp. 61–68.
- [68] JOUBERT E, 2007, *SAWIT restructures SAWB — Makes way for new representative wine industry muscle*, [Online], [cited May 8th, 2007], Available from [www.sawit.co.za/news/news\\_articles\\_07.asp](http://www.sawit.co.za/news/news_articles_07.asp)
- [69] JURISCH B, 1992, *Scheduling jobs in shops with multi-purpose machines*, PhD Dissertation, Fachbereich Mathematik/Informatik, Universität Osnabrück, Osnabrück.
- [70] KACEM I, HAMMADI S & BORNE P, 2002, *Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems*, IEEE Transactions on Systems, Man and Cybernetics — Part C: Applications and Reviews, **32**(1), pp. 1–13.
- [71] KOBAYASHI S, ONO I & YAMAMURA, 1995, *An efficient genetic algorithm for job shop scheduling problems*, Proceedings of International Conference on Genetic algorithms, pp. 506–511.
- [72] KÖPPE M & WEISMANTEL R, 2003, *An algorithm for mixed integer optimization*, Mathematical Subject Classification, Springer-Verlag, New York (NY).
- [73] KRAJEWSKI LJ & RITZMAN LP, 1990, *Operations management: Strategy and analysis*, 2<sup>nd</sup>, Addison-Wesley Publishing Company, Inc, Reading (MA).
- [74] KRUGER B, 2000, *A logistic strategy for the RSA wine industry: Part I*, Report prepared for Winetech 2020, Winetech, Stellenbosch.
- [75] KUBIAK W & VAN DE VELDE S, 1998, *Scheduling deteriorating jobs to minimize makespan*, Naval Research Logistics, **45**, pp. 511–523.
- [76] KWV, 2007, *KWV*, [Online], [cited May 2nd, 2007], Available from <http://www.kwv.co.za/>
- [77] LAGEWEG BJ, LENSTRA JK & RINNOOY-KAN AHG, 1977, *Job-shop scheduling by implicit enumeration*, Management Science, **24**, pp. 441–450.
- [78] LAWRENCE S, 1984, *Supplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques*, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh (PA).
- [79] LEE SM & ASLLANI AA, 2004, *Job scheduling with dual criteria and sequence-dependent setups: Mathematical versus genetic programming*, The International Journal of Management Science, **32**, pp. 145–153.

- [80] LEUNG JYT (ED), 2004, *Handbook of scheduling: Algorithms, models and performance analysis*, CRC Press, Boca Raton (FL).
- [81] LINDO SYSTEMS, 2008, *Lindo Systems*, [Online], [cited August 25th, 2008], Available from <http://www.lindo.com/>
- [82] MARTIN P & SHMOYS DB, 1996, *A new approach to computing optimal schedules for the job shop scheduling problem*, [Online], [cited June 9th, 2009], Available from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.695>
- [83] MASON SJ, FOWLER JW & CARLYLE WM, 2002, *A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops*, *Journal of Scheduling*, **5**, pp. 247–262.
- [84] MCMILLAN C, 1970, *Mathematical programming: An introduction to the design and application of optimal decision machines*, John Wiley & Sons, Inc, New York (NY).
- [85] MEDIAVISION, ON BEHALF OF SA WINE COUNCIL, 2008, *Wine Council flexes muscles on social transformation in the wine industry*, Media Release, [Online], [cited March 19th, 2009], Available from <http://www.wine.co.za/news/news.aspx?NEWSID=11442&Source=News>
- [86] MICROSOFT OFFICE, 2003, *Excel*, [Online], [cited June 7th, 2009], Information available from <http://office.microsoft.com/en-us/excel/default.aspx>
- [87] MITTENTHAL J, RAGHAVACHARI M & RANA AI, 1993, *A hybrid simulated annealing approach for single machine scheduling problems with non-regular penalty functions*, *Computers and Operations Research*, **20**, pp. 103–111.
- [88] MUSEE N, LORENZEN L & ALDRICH C, 2005, *Decision support for waste minimization in wine making processes*, *Environmental Progress*, **25**(1), pp. 56–63.
- [89] NAHMIA S, 1997, *Production and operations analysis*, 3<sup>rd</sup> Edition, The McGraw-Hill Companies, Inc, Hightstown (NJ).
- [90] NOWICKI E & SMUTNICKI C, 1996, *A fast tabu search algorithm for the permutation flow-shop problem*, *European Journal of Operational Research*, **91**, pp. 160–175.
- [91] PANDELL AJ, 1999, *The acidity of wine*, [Online], [cited November 21st, 2007], Available from [http://www.wineperspective.com/the\\_acidity\\_of\\_wine.htm](http://www.wineperspective.com/the_acidity_of_wine.htm)
- [92] PEZZELA F & MERELLI E, 2000, *A tabu search method guided by shifting bottleneck for the job-shop scheduling problem*, *European Journal of Operational Research*, **120**(3), pp. 297–310.
- [93] PINEDO M, 2002, *Scheduling: Theory, algorithms and systems*, 2<sup>nd</sup> Edition, Prentice Hall, Inc, Upper Saddle River (NJ).
- [94] PINEDO ML, 2005, *Planning and scheduling in manufacturing and services*, Springer Science Business Media, LLC, New York (NY).
- [95] PIRLOT M, 1996, *General local search methods*, *European Journal of Operational Research*, **92**, pp. 493–511.

- [96] PRACTICAL ACTION, TECHNOLOGY CHALLENGING POVERTY, 2006, *Grape wine*, [Online], [cited January 25th, 2007], Available from [http://practicalaction.org/docs/technical\\_information\\_service/grape\\_wine.pdf](http://practicalaction.org/docs/technical_information_service/grape_wine.pdf)
- [97] RAMUDHIN A & MARIER P, 1996, *The generalized shifting bottleneck procedure*, European Journal of Operational Research, **93**, pp. 34–48.
- [98] RANKINE B, 1989, *Making good wine: A manual of winemaking practice for Australia and New Zealand*, Sun Books, Melbourne.
- [99] REID RD & SANDERS NR, 2007, *Operations management*, [Online], [cited August 10th, 2007], Available from [www.csus.edu/mgmt/blakeh/RS-Ch10.ppt](http://www.csus.edu/mgmt/blakeh/RS-Ch10.ppt)
- [100] ROBINSON J (ED), 1994, *The Oxford companion to wine*, Oxford University Press, Oxford.
- [101] ROUX M, 2009, Viticulturist at *Wamakersvallei wine cellar*, [Personal Communication], Contactable at [marko@wamakersvallei.co.za](mailto:marko@wamakersvallei.co.za)
- [102] SA WINE COUNCIL, 2006, *SA Wine Council*, [Online], [cited March 19th, 2009], Available from <http://www.winecouncil.co.za/>
- [103] SA WINE INDUSTRY INFORMATION & SYSTEMS, 2006, *Annual report*, [Online], [cited May 2nd, 2007], Available from <http://www.sawis.co.za/SAWISPortal/DesktopDefault.aspx?tabindex=3&tabid=53>
- [104] SA WINE INDUSTRY INFORMATION & SYSTEMS, 2005, *Statistics of wine-grape vines as on 30 November 2005*, [Online], [cited May 2nd, 2007], Available from <http://www.sawis.co.za/SAWISPortal/DesktopDefault.aspx?tabindex=3&tabid=53>
- [105] SA WINE INDUSTRY INFORMATION & SYSTEMS, 2008, *Annual report*, [Online], [cited March 19th, 2009], Available from <http://www.sawis.co.za/info/annualpublication.php>
- [106] SABUNCUOGLU I & BAYIZ M, 1999, *Job shop scheduling with beam search*, European Journal of Operational Research, **118**, pp. 390–412.
- [107] SAIDI-MEHRABAD M & FATTAHI P, 2007, *Flexible job shop scheduling with tabu search algorithms*, International Journal of Advanced Manufacturing Technology, **32**, pp. 563–570.
- [108] SAWIS, 2007, *SA wine industry information & systems*, [Online], [cited May 22nd, 2007], Available from <http://www.sawis.co.za/>
- [109] SERAFINI P, 2003, *Asymptotic scheduling*, Mathematical Programming, **98**(1–3), pp. 1–15.
- [110] SEVASTIANOV S, 1998, *Nonstrict vector summation in multi-operation scheduling*, Annals of Operations Research, **83**, pp. 1–31.
- [111] SEVASTIANOV SV, 1994, *On some geometric methods in scheduling theory: A survey*, Discrete Applied Mathematics, **55**, pp. 59–82.

- [112] SHAFAEI R & BRUNN P, 1999, *Workshop scheduling using practical (inaccurate) data — Part 1: The performance of heuristic scheduling rules in dynamic job shop environment using a rolling time horizon approach*, International Journal of Production Research, **37**(17), pp. 3913–3925.
- [113] SINGER M, 2001, *Decomposition methods for large job shops*, Computers and Operations Research, **28**(3), pp. 193–207.
- [114] SLACK N, CHAMBERS S & JOHNSTON R, 1995, *Operations management*, 4<sup>th</sup> Edition, Study guide, Pearson Education, [Online], [cited August 6th, 2007], Available from [http://wps.pearsoned.co.uk/ema\\_uk\\_he\\_slack\\_opsman\\_4/0,8757,1144919-,00.html](http://wps.pearsoned.co.uk/ema_uk_he_slack_opsman_4/0,8757,1144919-,00.html)
- [115] SOUTH AFRICAN WINE, 2007, *Wamakersvallei Winery*, 2005, [Online], [cited November 27th, 2007], Available from <http://www.wamakersvallei.co.za>
- [116] SOUTH AFRICAN WINE INDUSTRY TRUST, 2007, [Brochure], Agricultural Research Centre, Manor House, R44 Arterial Road, Stellenbosch, 7600.
- [117] SOUTH AFRICAN WINE & BRANDY COMPANY, 2003, *The South African wine industry strategy plan (WIP)*, [Retrieved May 2nd, 2007], SAWB Online DataBase.
- [118] SOUTH AFRICAN WINE & BRANDY COMPANY, 2006, *The effects of deregulation on the South African wine industry*, [Retrieved May 4th, 2007], SAWB Online DataBase.
- [119] SOUTHERN HEMISPHERE WINE CENTER, 2007, *The history of South African wine*, [Online], [cited May 2nd, 2007], Available from <http://www.southernwines.com/history.cfm>
- [120] STEINHÖFFEL K, ALBERCHT A & WONG CK, 2002, *Fast parallel heuristics for the job shop scheduling problem*, Journal of the Operational Research Society, **39**, pp. 1163–1174.
- [121] STEVENSON WJ, 2007, *Operations management*, 9<sup>th</sup> Edition, McGraw-Hill Companies, Inc, New York (NY).
- [122] STORER RH, WU SD & VACCARI R, 1992, *New search spaces for sequencing problems with application to job shop scheduling*, Management Science, **38**, pp. 1495–1509.
- [123] SUN D, BATA R & LIN L, 1995, *Effective job shop scheduling through active chain manipulation*, Computers and Operations Research, **22**(2), pp. 159–172.
- [124] TAILLARD ED, 1994, *Parallel taboo search techniques for the job-shop scheduling problem*, Operations Research Society of America Journal on Computing, **6**, pp. 108–117.
- [125] TRUTER H, 2007, Cellar master at *Wamakersvallei wine cellar*, [Personal Communication], Contactable at [hugo@wamakersvallei.co.za](mailto:hugo@wamakersvallei.co.za)
- [126] TRUTER J, 2007, Cellar manager at *Wamakersvallei wine cellar*, [Personal Communication], Contactable at [johan@wamakersvallei.co.za](mailto:johan@wamakersvallei.co.za)
- [127] VAN DER MERWE K, 2007, Viticulturist at *Wamakersvallei wine cellar*, [Personal Communication], Contactable at [koos@wamakersvallei.co.za](mailto:koos@wamakersvallei.co.za)
- [128] VAN DYK FE, 2009, Principal supply chain analyst in the Built Environments department of the *Council for Scientific and Industrial Research (CSIR)* in South Africa, [Personal Communication], Contactable at [fevandyk@csir.co.za](mailto:fevandyk@csir.co.za)

- [129] VAN LAARHOVEN PJM & AARTS EHL, 1998, *Simulated annealing: Theory and applications*, D. Reidel Publishing Company, Dordrecht.
- [130] VAN LAARHOVEN PJM, AARTS EHL & LENSTRA JK, 1992, *Job-shop scheduling by simulated annealing*, *Operations Research*, **40**, pp. 113–125.
- [131] VAN SCHALKWYK H & ARCHER E, 2000, *Determining optimal ripeness in wine grapes*, Wynboer: A Technical Guide for Wine Producers, [Online], [cited June 5th, 2009], Available from <http://www.wynboer.co.za/recentarticles/0500optimum.php3>
- [132] VAESSENS RJM, AARTS EHL & LENSTRA JK, 1994, *Job shop scheduling by local search*, *INFORMS Journal on Computing*, **8**, pp. 302–317.
- [133] VARELA R, VELA CR, PUENTE J & GOMEZ A, 2002, *A knowledge-based evolutionary strategy for scheduling problems with bottlenecks*, *European Journal of Operational Research*, **145**, pp. 57–71.
- [134] VINE RP, HARKNESS EM, BROWNING T & WAGNER C, 1999, *Winemaking: From grape growing to marketplace*, Aspen Publishers, Inc, Gaithersburg (MD).
- [135] VISSER C, 2007, Winemaker at *Wamakersvallei wine cellar*, [Personal Communication], Contactable at [christiaan@wamakersvallei.co.za](mailto:christiaan@wamakersvallei.co.za)
- [136] WAGNER HM, 1959, *An integer linear-programming model for machine scheduling*, *Naval Research Logistics Quarterly*, **6**, pp. 131–140.
- [137] WANG L & ZHENG D, 2001, *An effective hybrid optimization strategy for job-shop scheduling problem*, *Computers and Operations Research*, **28**(6), pp. 585–596.
- [138] WATSON J, BECK JC, HOVE AE & WHITLEY, 2001, *Towards a descriptive model of local search cost in job shop scheduling*, *Proceedings of the Sixth European Conference on Planning*, ECP, Toledo.
- [139] WEISS KA & KELLY S, 2004, *Reading rehabilitation: A case on organizational layouts*, [Online], [Cited July 26th, 2007], Available from <http://courses.washington.edu/smartman/pdf/ReadingRehab.pdf>
- [140] WELLINGTON TOURISM BUREAU, 2007, *Wamakersvallei Winery*, [Online], [cited February 20th, 2007], Available from [http://www.wellington.co.za/taste\\_wellington.php](http://www.wellington.co.za/taste_wellington.php)
- [141] WELLINGTON TOURISM BUREAU, 2007, *Wellington Guide*, [Online], [cited February 20th, 2007], Available from <http://www.wellington.co.za/>
- [142] WIKIPEDIA, THE FREE ENCYCLOPEDIA, 2007, *History of South Africa in the apartheid era*, [Online], [cited August 13th, 2007], Available from [http://en.wikipedia.org/wiki/History\\_of\\_South\\_Africa\\_in\\_the\\_Apartheid\\_era#Sanctions](http://en.wikipedia.org/wiki/History_of_South_Africa_in_the_Apartheid_era#Sanctions)
- [143] WIKIPEDIA, THE FREE ENCYCLOPEDIA, 2007, *Tartaric acid*, [Online], [cited November 21st, 2007], Available from [http://en.wikipedia.org/wiki/Tartaric\\_acid](http://en.wikipedia.org/wiki/Tartaric_acid)
- [144] WIKIPEDIA, THE FREE ENCYCLOPEDIA, 2007, *Viticulture*, [Online], [cited January 31st, 2007], Available from <http://en.wikipedia.org/wiki/Viticulture>
- [145] WIKIPEDIA, THE FREE ENCYCLOPEDIA, 2007, *Wine*, [Online], [cited January 26th, 2007], Available from <http://en.wikipedia.org/wiki/Wine>

- [146] WIKIPEDIA, THE FREE ENCYCLOPEDIA, 2007, *Winemaking*, [Online], [cited January 26th, 2007], Available from <http://en.wikipedia.org/wiki/Winemaking>
- [147] WILLIAMSON DP, HALL LA, HOOGEVEEN JA, HURKENS CAJ, LENSTRA JK, SEVAST'JANOVE SV & SHMOYS DB, 1997, *Short shop schedules*, *Operations Research*, **45**, pp. 288–295.
- [148] WINE, 2007, *Wamakersvallei Winery*, [Online], [cited January 25th, 2007], Available from <http://www.wine.co.za/directory/winery.aspx?PRODUCERID=3205>
- [149] WINE CHARTER STEERING COMMITTEE, 2006, *The wine industry transformation charter, Consultative Draft I*, [Online], [cited May 21st, 2007], Available from [www.sawit.co.za/downloads/Wine\\_Industry\\_Transformation\\_Charter.pdf](http://www.sawit.co.za/downloads/Wine_Industry_Transformation_Charter.pdf)
- [150] WINETECH, 2006, *Annual report outline*, [Online], [cited May 2nd, 2007], Available from <http://www.winetech.co.za/docs2007/winetechjaarverslag2006engelsoutline.doc>
- [151] WINELAND, 2006, *Money makes the wine go round — but who foots the bill?*, Wineland, October 2006, [Online], [cited May 2nd, 2007], Available from <http://www.wineland.co.za/2006oct-money.php3>
- [152] WINSTON WL, 1994, *Operations research: Applications and algorithms*, 3<sup>rd</sup> Edition, Duxbury Press, Belmont (CA).
- [153] WITTEWER G & ROTHFIELD J, 2008, *Global wine — Australia in perspective*, The Global Wine Statistical Compendium, Australian Wine and Brandy Corporation, [Online], [cited March 19th, 2009], Available from <https://www.awbc.com.au/winefacts/data/free.asp?subcatid=97>
- [154] WOSA, 2007, *Wines of South Africa*, [Online], [cited May 2nd, 2007], Available from <http://www.wosa.co.za/>
- [155] YAMANDA T & NAKANO R, 1992, *A genetic algorithm applicable to large-scale job shop problems*, *Parallel Problem Solving from Nature*, **2**, pp. 281–290.
- [156] YAMANDA T & NAKANO R, 1996, *Job shop scheduling by simulated annealing combined with deterministic local search*, pp. 237–248 in OSMAN IH, KELLY JP, *Meta-heuristics: Theory & Applications*, pp. 237–248.
- [157] YANG S & WANG D, 2001, *A new adaptive neural network and heuristic hybrid approach for job-shop scheduling*, *Computers and Operations Research*, **28**, pp. 955–971.

---

---

## APPENDIX A

---

# Processor specifications

### Contents

A.1 Tank capacities and processor numbering . . . . .	181
A.1.1 Tank capacities . . . . .	181
A.1.2 Processor numbers . . . . .	188
A.2 Grape Intakes . . . . .	188

This appendix contains information regarding the equipment used in the cellar at Wamakersvallei Winery, such as the physical and actual capacities (where relevant) of processors. Furthermore, the necessary information regarding each of the processors forming part of the active cellar, and therefore part of the active cellar scheduling problem as described in Chapters 4–6, are considered. The appendix also contains a summary of information on the yearly grape intake at Wamakersvallei Winery from 2000 to 2006, presented in §A.2.

### A.1 Tank capacities and processor numbering

Each processor involved in the active cellar scheduling problem is assigned an unique processor number. In §A.1.1, these numbers are shown, where necessary deferring the introduction of some processor numbers to §A.1.2.

#### A.1.1 Tank capacities

Information with respect to tanks and their capacities, is listed in Table A.1. It is important to note the difference between the Vessel reference and the processor number. The Vessel reference is a label assigned to each of the processors by the winemaker in order to keep track of machine activity (using the cellar software), whereas the processor number,  $i$ , is used to label machine  $P_i$  for the active cellar scheduling problem in VINDSS. Table A.1 therefore contains the individual physical machine capacities, the machine locations in the cellar, their vessel numbers and their processor numbers. Store BS refers to the Blue Store, the remaining the store labels are self explanatory.

Vessel reference	Store	Description	Physical Capacity	Actual Capacity	Processor number
001	B	Settling tank	43 580		
002	B	Settling tank	43 513		
003	B	Settling tank	43 644		
004	B	Settling tank	43 612		
005	B	Settling tank	43 345		
006	B	Settling tank	44 021		
007	B	Settling tank	43 626		
008	B	Settling tank	43 624		
009	B	Settling tank	43 686		
010	B	Settling tank	43 776		
011	B	Settling tank	43 806		
012	B	Settling tank	43 580		
013	B	Settling tank	43 424		
014	B	Settling tank	43 662		
015	B	Settling tank	43 846		
016	B	Settling tank	43 487		
017	B	Fermentation tank	43 590		
018	B	Fermentation tank	43 470		
019	B	Fermentation tank	43 404		
020	B	Fermentation tank	43 610		
021	B	Fermentation tank	43 563		
022	B	Fermentation tank	43 599		
023	B	Fermentation tank	43 813		
024	B	Fermentation tank	43 851		
025	B	Fermentation tank	43 733		
026	B	Fermentation tank	43 746		
027	B	Fermentation tank	43 791		
028	B	Fermentation tank	43 820		
029	B	Fermentation tank	43 622		
030	B	Fermentation tank	43 512		
031	B	Fermentation tank	43 464		
032	B	Fermentation tank	43 573		
033	B	Fermentation tank	43 519		
034	B	Fermentation tank	43 509		
035	B	Fermentation tank	22 959		
036	B	Fermentation tank	22 867		
037	B	Fermentation tank	22 980		
038	B	Fermentation tank	22 934		
039	B	Fermentation tank	22 982		
040	B	Fermentation tank	22 926		
041	B	Fermentation tank	22 935		
042	B	Fermentation tank	22 950		

Table A.1: Physical capacities (in litres) of the different types of tanks found in Stores A to F of Wamakersvallei cellar, including the actual capacity (in tonnes) where it is relevant. A processor number is assigned to the tanks forming part of the active cellar [135].



Vessel reference	Store	Description	Physical Capacity	Actual Capacity	Processor number
043	B	Fermentation tank	22 949		
044	B	Fermentation tank	30 118		
045	B	Fermentation tank	30 119		
046	B	Fermentation tank	30 154		
047	B	Fermentation tank	30 300		
048	B	Fermentation tank	30 163		
049	B	Fermentation tank	30 261		
050	B	Fermentation tank	30 182		
051	B	Fermentation tank	30 158		
052	B	Fermentation tank	29 852		
053	B	Fermentation tank	29 882		
054	B	Fermentation tank	29 873		
055	B	Fermentation tank	29 887		
056	B	Fermentation tank	29 783		
057	B	Fermentation tank	29 865		
058	B	Fermentation tank	29 876		
059	B	Fermentation tank	29 832		
061	E	Fermentation tank	51 633		
062	E	Fermentation tank	50 908		
063	E	Fermentation tank	51 481		
064	E	Fermentation tank	51 571		
065	E	Fermentation tank	33 774		
066	E	Fermentation tank	33 655		
067	E	Fermentation tank	33 771		
068	E	Fermentation tank	33 556		
071	A	Fermentation tank	89 089		
072	A	Fermentation tank	89 014		
073	A	Fermentation tank	88 554		
081	BS	Storage tank	175 206		
082	BS	Storage tank	174 738		
083	BS	Storage tank	172 564		
084	BS	Storage tank	172 105		
090	A	Fermentation tank	5 483		
091	A	Fermentation tank	5 470		
092	A	Fermentation tank	5 382		
093	A	Fermentation tank	2 869		
094	A	Fermentation tank	2 868		
095	A	Storage tank	2 901		
096	A	Storage tank	2 882		
201	F	Fermentation Tank	53 343		
202	F	Fermentation Tank	53 319		
203	F	Fermentation Tank	53 451		

Table A.1 (continued): *Physical capacities (in litres) of the different types of tanks found in Stores A to F of Wamakersvallei cellar, including the actual capacity (in tonnes) where it is relevant. A processor number is assigned to the tanks forming part of the active cellar [135].*

Vessel reference	Store	Description	Physical Capacity	Actual Capacity	Processor number
204	F	Fermentation Tank	53 542		
205	F	Fermentation Tank	53 401		
206	F	Fermentation Tank	53 462		
207	F	Fermentation Tank	53 537		
208	F	Fermentation Tank	53 384		
209	F	Fermentation Tank	110 403		
210	F	Fermentation Tank	110 771		
211	F	Fermentation Tank	109 994		
212	F	Fermentation Tank	110 459		
213	F	Fermentation Tank	110 494		
214	F	Fermentation Tank	110 216		
215	F	Fermentation Tank	110 568		
216	F	Fermentation Tank	110 538		
217	F	Fermentation Tank	110 647		
218	F	Fermentation Tank	110 473		
219	F	Fermentation Tank	110 622		
220	F	Fermentation Tank	110 323		
221	F	Fermentation Tank	110 222		
222	F	Fermentation Tank	110 224		
223	F	Fermentation Tank	109 881		
224	F	Fermentation Tank	110 078		
225	F	Fermentation Tank	53 257		
226	F	Fermentation Tank	53 223		
227	F	Fermentation Tank	53 367		
228	F	Fermentation Tank	53 425		
229	F	Fermentation Tank	53 458		
230	F	Fermentation Tank	53 352		
231	F	Fermentation Tank	53 485		
232	F	Fermentation Tank	53 430		
233	F	Fermentation Tank	89 250		
234	F	Fermentation Tank	88 685		
235	F	Fermentation Tank	88 452		
236	F	Fermentation Tank	88 925		
237	F	Fermentation Tank	89 266		
238	F	Fermentation Tank	88 596		
239	F	Fermentation Tank	89 177		
240	F	Fermentation Tank	27 206		
241	F	Fermentation Tank	27 238		
242	F	Fermentation Tank	27 187		
243	F	Fermentation Tank	27 152		
244	F	Fermentation Tank	27 171		
245	F	Fermentation Tank	27 194		

Table A.1 (continued): *Physical capacities (in litres) of the different types of tanks found in Stores A to F of Wamakersvallei cellar, including the actual capacity (in tonnes) where it is relevant. A processor number is assigned to the tanks forming part of the active cellar [135].*

Vessel reference	Store	Description	Physical Capacity	Actual Capacity	Processor number
246	F	Fermentation Tank	27 324		
247	F	Fermentation Tank	27 195		
248	F	Fermentation Tank	27 342		
249	F	Fermentation Tank	27 358		
250	F	Fermentation Tank	27 188		
251	F	Fermentation Tank	27 185		
252	F	Fermentation Tank	27 191		
253	F	Fermentation Tank	27 258		
254	F	Fermentation Tank	27 226		
255	F	Fermentation Tank	27 171		
301	C back	Fermentation tanks	87 654		
302	C back	Fermentation tanks	87 569		
303	C back	Fermentation tanks	87 689		
304	C back	Fermentation tanks	87 369		
305	C back	Fermentation tanks	87 486		
306	C back	Fermentation tanks	87 592		
307	C back	Fermentation tanks	87 497		
308	C back	Fermentation tanks	110 927		
309	C back	Fermentation tanks	111 122		
310	C back	Fermentation tanks	111 210		
311	C back	Fermentation tanks	111 159		
312	C back	Fermentation tanks	110 849		
313	C back	Fermentation tanks	111 083		
314	C back	Fermentation tanks	87 419		
315	C back	Fermentation tanks	87 728		
316	C back	Fermentation tanks	87 481		
325	C front	Cold stabilisation	17 354		
326	C front	Cold stabilisation	17 431		
327	C front	Cold stabilisation	17 596		
328	C front	Buffer	21 501		
329	C front	Buffer	21 560		
330	C front	Buffer	21 503		
331	C front	Buffer	17 523		
332	C front	Buffer	17 433		
333	C front	Buffer	17 545		
334	C front	Buffer	11 310		
335	C front	Buffer	11 345		
336	C front	Buffer	11 204		
337	C front	Buffer	8 024		
338	C front	Buffer	8 030		
339	C front	Buffer	8 046		
340	C front	Cold stabilisation	5 358		

Table A.1 (continued): *Physical capacities (in litres) of the different types of tanks found in Stores A to F of Wamakersvallei cellar, including the actual capacity (in tonnes) where it is relevant. A processor number is assigned to the tanks forming part of the active cellar [135].*

Vessel reference	Store	Description	Physical Capacity	Actual Capacity	Processor number
341	C front	Cold stabilisation	5 364		
342	C front	Cold stabilisation	8 050		
343	C front	Cold stabilisation	11 276		
344	C front	Cold stabilisation	11 303		
345	C front	Buffer	26 865		
346	C front	Cold stabilisation	26 837		
347	C front	Cold stabilisation	26 906		
401	D	Fermentation tank	75 000		
402	D	Fermentation tank	75 000		
403	D	Fermentation tank	75 000		
404	D	Fermentation tank	75 000		
405	D	Fermentation tank	75 000		
406	D	Fermentation tank	75 000		
407	D	Fermentation tank	75 000		
408	D	Fermentation tank	75 000		
409	D	Fermentation tank	75 000		
410	D	Fermentation tank	75 000		
411	D	Fermentation tank	75 000		
412	D	Fermentation tank	75 000		
413	D	Fermentation tank	75 000		
414	D	Fermentation tank	75 000		
415	D	Fermentation tank	130 000		
416	D	Fermentation tank	130 000		
417	D	Fermentation tank	130 000		
418	D	Fermentation tank	130 000		
419	D	Fermentation tank	130 000		
420	D	Fermentation tank	130 000		
421	D	Fermentation tank	130 000		
422	D	Fermentation tank	130 000		
423	D	Fermentation tank	130 000		
424	D	Fermentation tank	130 000		
430	D	Fermentation tank	37 000		
431	D	Fermentation tank	37 000		
432	D	Fermentation tank	37 000		
433	D	Fermentation tank	37 000		
434	D	Fermentation tank	34 000		
435	D	Fermentation tank	34 000		
436	D	Fermentation tank	34 000		
437	D	Fermentation tank	34 000		
438	D	Fermentation tank	34 000		
439	D	Fermentation tank	34 000		
440	D	Fermentation tank	34 000		

Table A.1 (continued): *Physical capacities (in litres) of the different types of tanks found in Stores A to F of Wamakersvallei cellar, including the actual capacity (in tonnes) where it is relevant. A processor number is assigned to the tanks forming part of the active cellar [135].*

Vessel reference	Store	Description	Physical Capacity	Actual Capacity	Processor number
441	D	Fermentation tank	34 000		
442	D	Fermentation tank	34 000		
443	D	Fermentation tank	34 000		
449	D	Fermentation tank	20 000		
450	D	Fermentation tank	20 000		
451	D	Fermentation tank	20 000		
452	D	Fermentation tank	20 000		
453	D	Fermentation tank	20 000		
454	D	Fermentation tank	20 000		
455	D	Fermentation tank	20 000		
456	D	Fermentation tank	20 000		
457	D	Fermentation tank	20 000		
458	D	Fermentation tank	20 000		
A	E	Contech Buffer A	11 710		
B	E	Contech Buffer B	11 710		
DF1	E	Red fermentation tank	100 197	80	19
DF2	E	Red fermentation tank	100 194	80	20
DF3	E	Red fermentation tank	100 235	80	21
DF4	E	Red fermentation tank	100 564	80	22
DF5	E	Red fermentation tank	100 605	80	23
DF6	E	Red fermentation tank	100 475	80	24
DF7	E	Red fermentation tank	100 323	80	25
DF8	E	Red fermentation tank	100 785	80	26
DF9	E	Red fermentation tank	100 363	80	27
DF10	E	Red fermentation tank	100 085	80	28
RT317	C front	Red fermentation tank	55 742	50	29
RT318	C front	Red fermentation tank	55 793	50	30
RT319	C front	Red fermentation tank	55 804	50	31
RT320	C front	Red fermentation tank	54 169	50	32
RT321	C front	Red fermentation tank	56 081	50	33
RT322	C front	Red fermentation tank	55 615	50	34
RT323	C front	Red fermentation tank	55 868	50	35
RT324	C front	Red fermentation tank	55 783	50	36
RT325	C front	Red fermentation tank	88 000	80	37
RT326	C front	Red fermentation tank	88 000	80	38
RT327	C front	Red fermentation tank	88 000	80	39
RT328	C front	Red fermentation tank	88 000	80	40
RT329	C front	Red fermentation tank	95 000	80	41
RT330	C front	Red fermentation tank	95 000	80	42
RT331	C front	Red fermentation tank	95 000	80	43
RT332	C front	Red fermentation tank	95 000	80	44

Table A.1 (continued): *Physical capacities (in litres) of the different types of tanks found in Stores A to F of Wamakersvallei cellar, including the actual capacity (in tonnes) where it is relevant. A processor number is assigned to the tanks forming part of the active cellar [135].*

### A.1.2 Processor numbers

The remaining processor numbers are shown in Table A.2 together with their actual capacities, expressed in tonnes, where relevant.

Processor type	Processor number	Actual capacity
Tipping bins	1	—
	2	—
	3	—
Separators	4	20
	5	20
	6	20
	7	20
	8	20
	9	20
	10	20
Presses	11	15
	12	15
	13	25
	14	25
	15	25
	16	25
	17	25
	18	25

Table A.2: The processor numbers for the machinery in the active cellar, excluding the red wine fermentation tanks already listed in Table A.1.

## A.2 Grape Intakes

The daily total grape intake during the period 2000 to 2006 is shown in Table A.3 in order to provide the reader with a better understanding of the progression of the harvesting season at Wamakersvallei winery.

Month	Day	Year						
		2000	2001	2002	2003	2004	2005	2006
01	14						59	
01	15							
01	16							18
01	17						50	
01	18						204	
01	19	58					415	
01	20	89			69		355	
01	21				59		268	
01	22				141			
01	23		52		178	91		53
01	24	218	77	87	169		443	
01	25	435	102	84	7		400	9
01	26	433	100			36	590	
01	27	458			386	56	386	52
01	28	234		111	449		39	
01	29		148	109	554			
01	30		322	149	449	78		261
01	31	180	185	133	453		140	200
02	01	430	246	77			362	303
02	02	304	13			181	287	447
02	03	302			513	325	326	471
02	04	236		295	636	408	350	13
02	05		409	574	584	388		
02	06		530	259	609	461		480
02	07	427	508	515	499		438	549
02	08	535	434	414			536	539
02	09	531	379			513	700	610
02	10	510			477	714	515	406
02	11	337		512	504	444	480	
02	12		483	544	489	699		
02	13		531	470	495	544		422
02	14	356	584	461	352		553	452
02	15	471	494	133			5	454
02	16	336	340			414	541	262
02	17	381			479	522	517	316
02	18	174		598	490	527	279	
02	19		446	501	464	342		
02	20		457	426	298	443		83
02	21	192	419	214	71		251	152
02	22	243	309	332			503	307
02	23	250	256			353	368	506
02	24	216			311	321	491	399

Table A.3: A summary of the daily grape intakes from 2000 to 2006 at Wamakersvallei Winery, expressed in tons.

Month	Day	Year						
		2000	2001	2002	2003	2004	2005	2006
02	25	118		275	305	378	199	
02	26		309	135	316	350		
02	27		318	221	383	204		404
02	28	238	295	277	317		235	76
02	29	167						
03	01	177	286	192		342	279	
03	02	205	118			259	396	471
03	03	38			330	418	302	295
03	04			167	425	418	245	
03	05		208	176	165	79		
03	06	330	205	217	108			405
03	07	273	245	87	261		339	292
03	08	196	222	78		336	167	314
03	09	147	76			129	45	354
03	10	101			171	130	92	
03	11			71	49	378	98	
03	12		146	139		244		
03	13	142	106	127				96
03	14	113	3				29	294
03	15	119		11		138	39	360
03	16	15				97	5	268
03	17	2				237		109
03	18					36		
03	19					9		
03	20							
03	21							
03	22							87
03	23							27
Total		10 718	10 361	9 174	13 014	12 038	13 320	11 617

Table A.3 (continued): A summary of the daily grape intakes from 2000 to 2006 at Wamak-ersvallei Winery, expressed in tons.



---



---

## APPENDIX B

---

# Mathematical formulation of the scheduling problem

### Contents

B.1 IP formulation without pipe assignment . . . . .	191
B.2 Solving the problem instance in Example 5.1 . . . . .	193
B.3 The IP formulation including pipe assignment . . . . .	198

The mixed IP formulation of Chapter 5 is given in §B.1 of this appendix where pipe assignments are disregarded. The additional data required to solve the example problems of Chapter 5 were also given. The **Lingo** model for the IP approach is shown in §B.2. The possibility of applying the mixed IP to solve the cellar scheduling problem with pipe assignment is discussed in §B.3.

### B.1 IP formulation without pipe assignment

The objective in the scheduling problem inside the active cellar (without regarding the assignment of tasks to pipes) is to minimize  $C$  subject to the constraints

$$\begin{aligned}
 \sum_{j=1}^r t_{jk_j} + \sum_{j=r+1}^n f_{jk_j} &\leq C \\
 t_{j1} &\geq e_j, \quad j = 1, \dots, n \\
 t_{jk} + a_{ijk} p_{ijk} &\leq f_{jk}, \quad i = 1, \dots, m, \quad j = 1, \dots, n, \quad k = 1, \dots, k_j \\
 f_{jk} &= t_{j(k+1)}, \quad j = 1, \dots, n, \quad k = 1, \dots, k_j - 1 \\
 a_{ijk} &\leq \mu_{ijk}, \quad i = 1, \dots, m, \quad j = 0, \dots, n, \quad k = 1, \dots, k_j \\
 \sum_{i=1}^m a_{ij1} &= 1, \quad j = 1, \dots, n \\
 \sum_{i=m_1}^{m_2-1} a_{ij2} &= \left\lceil \frac{w_j}{20} \right\rceil y_{(j-r)}, \quad j = r+1, \dots, w \\
 f_{j2} - f_{\ell 2} &\geq 1 - (3 - z_{(j-r)(\ell-r)} - y_{(j-r)} - y_{(\ell-r)}) M, \quad j = r+1, \dots, w-1, \\
 &\quad \ell = j+1, \dots, w
 \end{aligned}$$

$$\begin{aligned}
f_{\ell 2} - f_{j 2} &\geq 1 - (z_{(j-r)(\ell-r)} + 2 - y_{(j-r)} - y_{(\ell-r)}) M, \quad j = r + 1, \dots, w - 1, \\
&\quad \ell = j + 1, \dots, w \\
\sum_{i=m_2}^{m_3-1} a_{ij2} c_i &\geq w_j (1 - y_{(j-r)}), \quad j = r + 1, \dots, w \\
a_{ij2} &\leq 1 - y_{(j-r)}, \quad i = m_2, \dots, m_3 - 1, \quad j = r + 1, \dots, w \\
a_{ij2} &\leq a_{ij3}, \quad i = m_2, \dots, m_3 - 1, \quad j = r + 1, \dots, w \\
\sum_{i=m_2}^{m_3-1} a_{ij3} c_i &\geq w_j, \quad j = r + 1, \dots, w \\
v_i + \sum_{j=1}^r a_{ij2} w_j &\leq c_i, \quad i = m_3, \dots, m \\
a_{ij2} + a_{i\ell 2} &\leq q_{j\ell} + 1, \quad i = m_3, \dots, m, \quad j = 1, \dots, r, \quad \ell = 1, \dots, r \text{ and } \ell \neq j \\
a_{ij2} &\leq tq_{ij}, \quad i = m_3, \dots, m, \quad j = 1, \dots, r \\
a_{ij1} &= tq_{ij}, \quad i = m_3, \dots, m, \quad j = w + 1, \dots, n \\
f_{j1} &\leq t_{\ell 1} + \left( 2 - \sum_{i=m_3}^{m_4-1} a_{ij1} - \sum_{i=m_3}^{m_4-1} a_{i\ell 1} \right) M, \quad j = w + 1, \dots, n, \\
&\quad \ell = j + 1, \dots, n \\
f_{j1} &\leq t_{\ell 1} + \left( 2 - \sum_{i=m_4}^m a_{ij1} - \sum_{i=m_4}^m a_{i\ell 1} \right) M, \quad j = w + 1, \dots, n, \\
&\quad \ell = j + 1, \dots, n \\
f_{jk} + s_{ij\ell} &\leq t_{\ell h} + (3 - x_{ij\ell} - a_{ijk} - a_{i\ell h}) M, \quad h = 1, \dots, k_\ell - 1, \quad i = 1, \dots, m, \\
&\quad j = 0, \dots, n, \quad k = 1, \dots, k_j - 1, \quad \ell = 1, \dots, n, \text{ and } \ell \neq j \\
\sum_{j=0}^n x_{ij\ell} &= \sum_{h=1}^{k_\ell} a_{i\ell h}, \quad i = 1, \dots, m_2 - 1, m_3, \dots, m, \quad \ell = 1, \dots, n \\
\sum_{\ell=0}^n x_{ij\ell} &= \sum_{k=1}^{k_j} a_{ijk}, \quad i = 1, \dots, m_2 - 1, m_3, \dots, m, \quad j = 0, \dots, n \\
&\quad k_\ell (\text{with } h \neq 3 \\
&\quad \text{if } r+1 \leq \ell \leq w) \\
\sum_{j=0}^n x_{ij\ell} &= \sum_{h=1}^{k_\ell} a_{i\ell h}, \quad i = m_2, \dots, m_3 - 1, \quad \ell = 1, \dots, n \\
&\quad k_j (\text{with } k \neq 3 \\
&\quad \text{if } r+1 \leq j \leq w) \\
\sum_{\ell=0}^n x_{ij\ell} &= \sum_{k=1}^{k_j} a_{ijk}, \quad i = m_2, \dots, m_3 - 1, \quad j = 0, \dots, n \\
t_{jk} &\geq 0, \quad j = 0, \dots, n, \quad k = 1, \dots, k_j \\
f_{jk} &\geq 0, \quad j = 0, \dots, n, \quad k = 1, \dots, k_j \\
a_{ijk} &\in \{0, 1\}, \quad i = 1, \dots, m, \quad j = 0, \dots, n, \quad k = 1, \dots, k_j \\
x_{ij\ell} &\in \{0, 1\}, \quad i = 1, \dots, m, \quad j = 0, \dots, n, \quad \ell = 1, \dots, n \\
y_{(j-r)} &\in \{0, 1\}, \quad j = r + 1, \dots, w
\end{aligned}$$

$$z_{(j-r)(\ell-r)} \in \{0, 1\}, j = r + 1, \dots, w, \ell = j + 1, \dots, w$$

## B.2 Solving the problem instance in Example 5.1

In order to solve Example 5.1, additional data are required. The values of all the parameters  $\mu_{ijk}$  are shown in Table B.1. This is an indication of the allowed processors of each task  $T_{jk}$ .

$J_j$	$T_{jk}$	$P_1$	$P_2$	$P_3$	$P_4$	$P_5$	$P_6$	$P_7$	$P_8$	$P_9$	$P_{10}$	$P_{11}$	$P_{12}$	$P_{13}$	$P_{14}$	$P_{15}$
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
5	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
6	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
	3	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
	2	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table B.1: The values of all  $\mu_{ijk}$  for the processors, jobs and their tasks in Example 5.1.

In Table B.2, the relevant non-zero setup times are listed for Example 5.1.

### Lingo 11.0 IP formulation

**Lingo 11.0** was used to solve the IP in Example 5.1. The **Lingo 11.0** problem formulation is presented below.

MODEL:

! A medium sized cellar with 15 machines excluding pipe assignment;

DATA:

```

r = 3; !The last red wine job;
w = 6; !The last white wine job;
n = 7; !The last job;
m1 = 4; !The first separator;
m2 = 7; !The first press;

```

Tipping bins $P_1, \dots, P_3$	Presses $P_7, \dots, P_9$
$s_{i14} = 1.5$	$s_{i45} = 1$
$s_{i15} = 1.5$	$s_{i46} = 1$
$s_{i16} = 1.5$	$s_{i47} = 1.5$
$s_{i24} = 1.5$	$s_{i54} = 1$
$s_{i25} = 1.5$	$s_{i56} = 1$
$s_{i26} = 1.5$	$s_{i57} = 1.5$
$s_{i34} = 1.5$	$s_{i64} = 1$
$s_{i35} = 1.5$	$s_{i65} = 1$
$s_{i36} = 1.5$	$s_{i67} = 1.5$
$s_{i41} = 0.5$	$s_{i74} = 2$
$s_{i42} = 0.5$	$s_{i75} = 2$
$s_{i43} = 0.5$	$s_{i76} = 2$
$s_{i51} = 0.5$	
$s_{i52} = 0.5$	
$s_{i53} = 0.5$	
$s_{i61} = 0.5$	
$s_{i62} = 0.5$	
$s_{i63} = 0.5$	

Table B.2: All non-zero setup times  $s_{ij\ell}$  for Example 5.1, expressed in hours.

```

m3 = 10; !The first DF tank;
m4 = 13; !The first RT tank;
m = 15;
NUM = 3;
LARGE = 1000;

ENDDATA

SETS:
    MACHINES: VOLUME, CAPACITY;
    JOBS: K_J, WEIGHT, SPLIT;
    TASKS;
    WHITE_JOBS /1..NUM/: y1;
    WHITE_JOBS_2 (WHITE_JOBS, WHITE_JOBS): y2;
    ALLOWED_TANKS( MACHINES, JOBS): TANK_MATCH;
    TASK( JOBS, TASKS): STARTT, ENDT;
    RED_JOB( JOBS, JOBS): MATCH;
    ASSIGN( MACHINES, JOBS, TASKS): ASSIGNMENT, DURATION, MU;
    FOLLOWING_JOBS( MACHINES, JOBS, JOBS): X, SETUP;
ENDSETS

! The objective;
    MIN = C;

! Defining the objective function;
    @SUM(JOBS(j)|j #NE# 1 #AND# (r+1) - j #GE# 0: STARTT(j,K_J(j))) +
    @SUM(JOBS(j)| (j - (r+2) #GE# 0): ENDT(j,K_J(j))) - C <= 0;

```

```

! Ensuring that the end and start times of a task leaves enough time for
production;
  @FOR(MACHINES(i):
    @FOR(JOBS(j)|(j #NE# 1):
      @FOR(TASKS(k)|(K_J(j) - k #GE# 0):
        STARTT(j,k) + ASSIGNMENT(i,j,k)*DURATION(i,j,k) -
        ENDT(j,k) <= 0));

! A task of a job should follow its previous task directly;
  @FOR(JOBS(j)|j #NE# 1:
    @FOR(TASKS(k)|K_J(j) - k #GE# 1:
      ENDT(j,k) - STARTT(j,(k+1)) = 0));

! Ensuring that a task is only assigned to an allowed machine;
  @FOR(MACHINES(i):
    @FOR(JOBS(j):
      @FOR(TASKS(k)|K_J(j) - k #GE# 0:
        ASSIGNMENT(i,j,k) - MU(i,j,k) <= 0));

! All first tasks may be assigned to exactly one processor;
  @FOR(JOBS(j)|(j #NE# 1):
    @SUM(MACHINES(i): ASSIGNMENT(i,j,1)) = 1);

  @FOR(JOBS(j)|((r+1) - j #GE# 0) #AND# (j #NE# 1):
    @SUM(MACHINES(i)|(i - m3 #GE# 0): ASSIGNMENT(i,j,2)) = 1);

! When a load of white grapes is assigned to different separators;
  @FOR(JOBS(j)|(j - (r+2) #GE# 0) #AND# ((w+1) - j #GE# 0):
    @SUM(MACHINES(i)|(i - m1 #GE# 0) #AND# ((m2-1) - i #GE# 0):
      CAPACITY(i)*ASSIGNMENT(i,j,2)) - WEIGHT(j)*y1(j-(r+1)) >= 0);

! Allowing only one separator to be emptied at any one time;
  @FOR(JOBS(j)|(j - (r+2) #GE# 0) #AND# ((w+1) - j #GE# 0):
    @FOR(JOBS(1)|(1 - (j+1) #GE# 0) #AND# ((w+1) - 1 #GE# 0):
      ENDT(j,2) - ENDT(1,2) + (3 - y2(j-(r+1),1-(r+1)) -
      y1(j-(r+1)) - y1(1-(r+1)))*LARGE >= 1;
      ENDT(1,2) - ENDT(j,2) + (y2(j-(r+1),1-(r+1)) + 2 -
      y1(j-(r+1)) - y1(1-(r+1)))*LARGE >= 1));

! Ensuring assignment to more than one press when needed;
  @FOR(JOBS(j)|(j - (r+2) #GE# 0) #AND# ((w+1) - j #GE# 0):
    @SUM(MACHINES(i)|(i - m2 #GE# 0) #AND# ((m3-1) - i #GE# 0):
      CAPACITY(i)*ASSIGNMENT(i,j,2)) - WEIGHT(j)*(1-y1(j-(r+1))) >= 0;
    @SUM(MACHINES(i)|(i - m2 #GE# 0) #AND# ((m3-1) - i #GE# 0):
      CAPACITY(i)*ASSIGNMENT(i,j,3)) - WEIGHT(j) >= 0);

  @FOR(JOBS(j)|(j - (w+2) #GE# 0):
    @SUM(MACHINES(i)|(i - m2 #GE# 0) #AND# ((m3-1) - i #GE# 0):

```

```
CAPACITY(i)*ASSIGNMENT(i,j,2)) - WEIGHT(j) >= 0);
```

! If a white grape job is separated and pressed at the presses it must be on the same press.;

```
@FOR(MACHINES(i)|(i - m2 #GE# 0) #AND# ((m3-1) - i #GE# 0):
  @FOR(JOBS(j)|(j - (r+2) #GE# 0) #AND# ((w+1) - j #GE# 0):
    ASSIGNMENT(i,j,2) - ASSIGNMENT(i,j,3) <= 0));
```

! Limiting the volume of a redwine tank;

```
@FOR(MACHINES(i)|(i - m3 #GE# 0) #AND# (m - i #GE# 0):
  VOLUME(i) + @SUM(JOBS(j)|((j - 2 #GE# 0) #AND# ((r+1) - j #GE# 0)):
    ASSIGNMENT(i,j,2)*WEIGHT(j)) - CAPACITY(i) <= 0);
```

! Allowing two tasks to be assigned to the same red wine tank only if the matching parameter allows it;

```
@FOR(MACHINES(i)|(i - m3 #GE# 0) #AND# (m - i #GE# 0):
  @FOR(JOBS(j)|((j #NE# 1) #AND# ((r+1) - j #GE# 0)):
    @FOR(JOBS(l)|((l #NE# 1) #AND# ((r+1) - j #GE# 0) #AND# (j -
      l #NE# 0)):
      ASSIGNMENT(i,j,2) + ASSIGNMENT(i,l,2) - MATCH(j,l) <=
        1));
```

! Limiting the assignment of task 2 red grape jobs to only allowed fermentation tanks;

```
@FOR(MACHINES(i)|(i - m3 #GE# 0) #AND# (m - i #GE# 0):
  @FOR(JOBS(j)|(j #NE# 1) #AND# ((r+1) - j #GE# 0):
    ASSIGNMENT(i,j,2) - TANK_MATCH(i,j) <= 0));
```

! The 'assignment' of a red wine to be emptied from the suitable fermentation tank;

```
@FOR(MACHINES(i)|(i - m3 #GE# 0) #AND# (m - i #GE# 0):
  @FOR(JOBS(j)|(j - (w+2) #GE# 0):
    ASSIGNMENT(i,j,1) = TANK_MATCH(i,j));
```

! If there are more than one fermentation tank to be emptied, only one task 1 at a time is allowed on each subset of the red wine fermentation tanks;

```
@FOR(JOBS(j)|j - (w+2) #GE# 0:
  @FOR(JOBS(l)|(l - (j+1) #GE# 0):
    ENDT(j,1) - STARTT(l,1) - (2 - @SUM(MACHINES(i)|(i - m3
      #GE# 0) #AND# ((m4-1) - i #GE# 0): ASSIGNMENT(i,j,1) +
      ASSIGNMENT(i,l,1)))*LARGE <= 0));
```

```
@FOR(JOBS(j)|j - (w+2) #GE# 0:
  @FOR(JOBS(l)|(l - (j+1) #GE# 0):
    ENDT(j,1) - STARTT(l,1) - (2 - @SUM(MACHINES(i)|(i - m4
      #GE# 0): ASSIGNMENT(i,j,1) + ASSIGNMENT(i,l,1)))*LARGE <=
      0));
```

! Using the setup times to set the starting times of tasks following on one

another on the same machine;

```

@FOR(MACHINES(i)|(m3-1) - i #GE# 0:
  @FOR(JOBS(j):
    @FOR(TASKS(k)|K_J(j) - k #GE# 0:
      @FOR(JOBS(l)|(l #NE# 1) #AND# (l - j #NE# 0):
        @FOR(TASKS(h)|K_J(l) - h #GE# 0:
          ENDT(j,k) + SETUP(i,j,l) - STARTT(l,h) -
          (3 - X(i,j,l) - ASSIGNMENT(i,l,h) -
          ASSIGNMENT(i,j,k))*LARGE <= 0)))));

```

! Defining X as a result of the assignments;

```

@FOR(MACHINES(i)|(i - m2 #GE# 0) #AND# ((m3-1) - i #GE# 0):
  @FOR(JOBS(l)|l #NE# 1:
    @SUM(JOBS(j): X(i,j,l)) - @SUM(TASKS(h)|((l - (r+2) #GE# 0)
    #AND# ((w+1) - l #GE# 0) #AND# (h #NE# 2) #AND# (K_J(l) - h
    #GE# 0)) #OR# (((l - (w+2) #GE# 0) #OR# ((r+1) - l #GE# 0))
    #AND# (K_J(l) - h #GE# 0)): ASSIGNMENT(i,l,h) = 0));

```

```

@FOR(MACHINES(i)|(i - m2 #GE# 0)#AND#((m3-1) - i #GE# 0):
  @FOR(JOBS(j):
    @SUM(JOBS(l): X(i,j,l)) - @SUM(TASKS(k)|((j - (r+2) #GE# 0)
    #AND# ((w+1) - j #GE# 0) #AND# (k #NE# 2) #AND# (K_J(j) - k
    #GE# 0)) #OR# (((j - (w+2) #GE# 0) #OR# ((r+1) - j #GE# 0))
    #AND# (K_J(j) - k #GE# 0)): ASSIGNMENT(i,j,k) = 0));

```

```

@FOR(MACHINES(i)|(i - m3 #GE# 0) #OR# ((m2-1) - i #GE# 0):
  @FOR(JOBS(l)|l #NE# 1:
    @SUM(JOBS(j): X(i,j,l)) - @SUM(TASKS(h)|K_J(l) - h #GE# 0:
    ASSIGNMENT(i,l,h) = 0));

```

```

@FOR(MACHINES(i)|(i - m3 #GE# 0) #OR# ((m2-1) - i #GE# 0):
  @FOR(JOBS(j):
    @SUM(JOBS(l): X(i,j,l)) - @SUM(TASKS(k)|K_J(j) - k #GE# 0:
    ASSIGNMENT(i,j,k) = 0));

```

! Define all X(i, j, j) to be zero for all machines;

```

@FOR(MACHINES(i):
  @FOR(JOBS(j): X(i,j,j) = 0));

```

! Set binary values for the required variables;

```

@FOR(MACHINES(i):
  @FOR(JOBS(j):
    @FOR(JOBS(l): @BIN(X(i,j,l))));

```

```

@FOR( ASSIGN: @BIN( ASSIGNMENT));

```

```

@FOR(JOBS(j)|(j - (r+2) #GE# 0) #AND# ((w+1) - j #GE# 0):
  @BIN(y1(j - (r+1))));

```

```

@FOR(JOBS(j)|(j - (r+2) #GE# 0) #AND# ((w+1) - j #GE# 0):
  @FOR(JOBS(1)|(1 - (r+2) #GE# 0) #AND# ((w+1) - 1 #GE# 0):
    @BIN(y2(j - (r+1),1 - (r+1)))));

@FOR(MACHINES(i):
  @FOR(JOBS(j):
    @FOR(TASKS(k)|k - K_J(j) #GE# 1:
      ASSIGNMENT(i,j,k) = 0)));

DATA:
! Import the data from Excel;
  JOBS, TASKS, MACHINES, DURATION, K_J, MU, SETUP, MATCH, WEIGHT, VOLUME,
  CAPACITY, TANK_MATCH = @OLE('C:\Documents and Settings\Adri\My Documents\
  2008\Masters\Lingo\No_pipes_4.xls');
ENDDATA

```

### B.3 The IP formulation including pipe assignment

The approach in §5.2 of a mathematical programming formulation for the scheduling problem in the active cellar where pipe assignment were disregarded in the main body of the thesis. In this section, the changes required to the mathematical programming model is considered in order to consider relevant pipe assignments as well. The tasks, their individual task types and required processors are listed in Tables B.3 and B.4 for the jobs consisting of red and white grapes respectively.

Red wine	First Job			Second job		
Task type	$T_1$	$T_6$	$T_4$	$T_5$	$T_6$	$T_3$
Tasks	$T_{j1}$	$T_{j2}$	$T_{j3}$	$T_{j'1}$	$T_{j'2}$	$T_{j'3}$
Processor	Tipping bins	Pipes	Fermentation tanks	Fermentation tanks	Pipes	Presses

Table B.3: The different tasks required in order to process a load of red grapes and their corresponding task types and processor requirements when pipe assignment is considered. The first job refers to the job,  $J_j$ , taking place on the day of receiving the grapes, whereas the second job,  $J_{j'}$ , refers to the job that requires processing after primary fermentation.

White wine					
Task type	$T_1$	$T_6$	$T_2$	$T_6$	$T_3$
Tasks	$T_{j1}$	$T_{j2}$	$T_{j3}$	$T_{j4}$	$T_{j5}$
Processor	Tipping bins	Pipes	Separators or Presses	Pipes	Presses

Table B.4: The different tasks required in order to process a load of white grapes, as well as the corresponding task types and processor requirements, when pipe assignment is considered.

Applying the parameters explained in §5.1, The constraint set is similar to that considered without pipe assignment since the pipes are now considered to be machines. The only additional requirement is that no job may be assigned to two machines directly after one another, but that



at least one pipe should be allowed in between. However, this problem constraint set is not discussed fully. However, the cellar graph with pipe numbering is included in order to provide the reader a better understanding of the approach. No solution to the suggested method is considered.

Using the same smaller active cellar as in Example 5.1, a solution to the mathematical programming model of §B.3 is illustrated. The numbered cellar graph with the dummy pipes indicated is shown in Figure B.1.

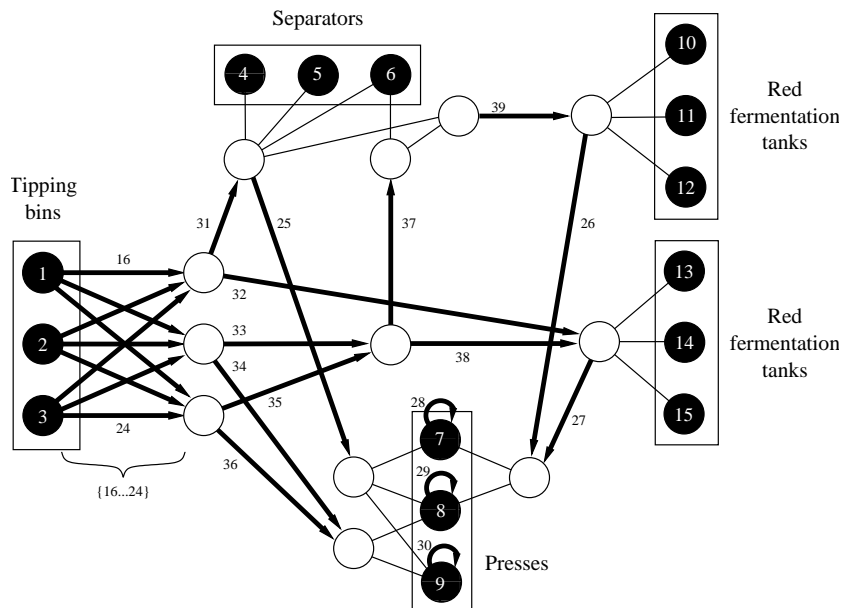


Figure B.1: The numbered cellar graph for the active cellar to illustrate the working of the mathematical programming model when pipe assignment is included.

The same grape loads as in Example 5.1 are expected and the current status of red wine tanks are also the same as those shown in Table 5.3. The expected jobs are also the same as those presented in Table 5.4.

The allowable successors are listed in Table B.7.

Job $j$	Task $k$	Allowed $P_i$	Corresponding $p_{ijk}$
0	1	$\mathcal{P}$	0
1	1	$P_1, \dots, P_3$	0.1
	2	$P_{10}, \dots, P_{15}$	144
2	1	$P_1, \dots, P_3$	0.1
	2	$P_{10}, \dots, P_{15}$	144
3	1	$P_1, \dots, P_3$	0.1
	2	$P_{10}, \dots, P_{15}$	120
4	1	$P_1, \dots, P_3$	0.1
	2	$P_4, \dots, P_6$	2.25
		$P_7, \dots, P_9$	0
	3	$P_7, \dots, P_9$	2.5
5	1	$P_1, \dots, P_3$	0.1
	2	$P_4, \dots, P_6$	1.25
		$P_7, \dots, P_9$	0
	3	$P_7, \dots, P_9$	2.5
6	1	$P_{10}, \dots, P_{15}$	5
	2	$P_7, \dots, P_9$	2.5

Table B.5: The duration  $p_{ijk}$  of processing task  $T_{jk}$  on an allowed set of processors (therefore all processors for which  $\mu_{ijk} = 1$ ).

Tipping bins $P_1, \dots, P_3$	Separators $P_4, \dots, P_6$	Presses $P_7, \dots, P_9$
$s_{i14} = 1.5$	$s_{i45} = 0.5$	$s_{i45} = 1$
$s_{i15} = 1.5$	$s_{i54} = 0.5$	$s_{i46} = 1.5$
$s_{i24} = 1.5$		$s_{i54} = 1$
$s_{i25} = 1.5$		$s_{i56} = 1.5$
$s_{i34} = 1.5$		$s_{i64} = 2$
$s_{i35} = 1.5$		$s_{i65} = 2$
$s_{i41} = 1$		
$s_{i42} = 1$		
$s_{i43} = 1$		
$s_{i51} = 1$		
$s_{i52} = 1$		
$s_{i53} = 1$		

Table B.6: All non-zero setup times  $s_{ijl}$  for Example 5.1, expressed in hours.

Processor	$P_1$	$P_2$	$P_3$	$P_4$	$P_5$	$P_6$	$P_7$	$P_8$	$P_9$	$P_{10}$	$P_{11}$	$P_{12}$	$P_{13}$	$P_{14}$
$P_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_2$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_3$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_5$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_6$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_7$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_8$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_9$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{10}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{11}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{12}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{13}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{14}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{15}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{16}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{17}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{18}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{19}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{20}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{21}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{22}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{23}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{24}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{25}$	0	0	0	0	0	0	1	1	1	0	0	0	0	0
$P_{26}$	0	0	0	0	0	0	1	1	0	0	0	0	0	0
$P_{27}$	0	0	0	0	0	0	1	1	0	0	0	0	0	0
$P_{28}$	0	0	0	0	0	0	1	0	0	0	0	0	0	0
$P_{29}$	0	0	0	0	0	0	0	1	0	0	0	0	0	0
$P_{30}$	0	0	0	0	0	0	0	0	1	0	0	0	0	0
$P_{31}$	0	0	0	1	1	1	0	0	0	0	0	0	0	0
$P_{32}$	0	0	0	0	0	0	0	0	0	0	0	0	1	1
$P_{33}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{34}$	0	0	0	0	0	0	0	1	1	0	0	0	0	0
$P_{35}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{36}$	0	0	0	0	0	0	0	1	1	0	0	0	0	0
$P_{37}$	0	0	0	0	0	1	0	0	0	0	0	0	0	0
$P_{38}$	0	0	0	0	0	0	0	0	0	0	0	0	1	1
$P_{39}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0

Table B.7: The values of the variable  $u_{i_1, i_2}$  where the entry  $(i_1, i_2)$  of the matrix corresponds to the value of variable  $u_{i_1, i_2}$ .

Processor	$P_{15}$	$P_{16}$	$P_{17}$	$P_{18}$	$P_{19}$	$P_{20}$	$P_{21}$	$P_{22}$	$P_{23}$	$P_{24}$	$P_{25}$	$P_{26}$	$P_{27}$	$P_{28}$
$P_1$	0	1	1	1	0	0	0	0	0	0	0	0	0	0
$P_2$	0	0	0	0	1	1	1	0	0	0	0	0	0	0
$P_3$	0	0	0	0	0	0	0	1	1	1	0	0	0	0
$P_4$	0	0	0	0	0	0	0	0	0	0	1	0	0	0
$P_5$	0	0	0	0	0	0	0	0	0	0	1	0	0	0
$P_6$	0	0	0	0	0	0	0	0	0	0	1	0	0	0
$P_7$	0	0	0	0	0	0	0	0	0	0	0	0	0	1
$P_8$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_9$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{10}$	0	0	0	0	0	0	0	0	0	0	0	1	0	0
$P_{11}$	0	0	0	0	0	0	0	0	0	0	0	1	0	0
$P_{12}$	0	0	0	0	0	0	0	0	0	0	0	1	0	0
$P_{13}$	0	0	0	0	0	0	0	0	0	0	0	0	1	0
$P_{14}$	0	0	0	0	0	0	0	0	0	0	0	0	1	0
$P_{15}$	0	0	0	0	0	0	0	0	0	0	0	0	1	0
$P_{16}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{17}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{18}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{19}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{20}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{21}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{22}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{23}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{24}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{25}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{26}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{27}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{28}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{29}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{30}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{31}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{32}$	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{33}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{34}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{35}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{36}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{37}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{38}$	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{39}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table B.7 (continued): The values of the variable  $u_{i_1, i_2}$  where the entry  $(i_1, i_2)$  of the matrix corresponds to the value of variable  $u_{i_1, i_2}$ .

Processor	$P_{29}$	$P_{30}$	$P_{31}$	$P_{32}$	$P_{33}$	$P_{34}$	$P_{35}$	$P_{36}$	$P_{37}$	$P_{38}$	$P_{39}$
$P_1$	0	0	0	0	0	0	0	0	0	0	0
$P_2$	0	0	0	0	0	0	0	0	0	0	0
$P_3$	0	0	0	0	0	0	0	0	0	0	0
$P_4$	0	0	0	0	0	0	0	0	0	0	0
$P_5$	0	0	0	0	0	0	0	0	0	0	0
$P_6$	0	0	0	0	0	0	0	0	0	0	0
$P_7$	0	0	0	0	0	0	0	0	0	0	0
$P_8$	1	0	0	0	0	0	0	0	0	0	0
$P_9$	0	1	0	0	0	0	0	0	0	0	0
$P_{10}$	0	0	0	0	0	0	0	0	0	0	0
$P_{11}$	0	0	0	0	0	0	0	0	0	0	0
$P_{12}$	0	0	0	0	0	0	0	0	0	0	0
$P_{13}$	0	0	0	0	0	0	0	0	0	0	0
$P_{14}$	0	0	0	0	0	0	0	0	0	0	0
$P_{15}$	0	0	0	0	0	0	0	0	0	0	0
$P_{16}$	0	0	1	1	0	0	0	0	0	0	0
$P_{17}$	0	0	0	0	1	1	0	0	0	0	0
$P_{18}$	0	0	0	0	0	0	1	1	0	0	0
$P_{19}$	0	0	1	1	0	0	0	0	0	0	0
$P_{20}$	0	0	0	0	1	1	0	0	0	0	0
$P_{21}$	0	0	0	0	0	0	1	1	0	0	0
$P_{22}$	0	0	1	1	0	0	0	0	0	0	0
$P_{23}$	0	0	0	0	1	1	0	0	0	0	0
$P_{24}$	0	0	0	0	0	0	1	1	0	0	0
$P_{25}$	0	0	0	0	0	0	0	0	0	0	0
$P_{26}$	0	0	0	0	0	0	0	0	0	0	0
$P_{27}$	0	0	0	0	0	0	0	0	0	0	0
$P_{28}$	0	0	0	0	0	0	0	0	0	0	0
$P_{29}$	0	0	0	0	0	0	0	0	0	0	0
$P_{30}$	0	0	0	0	0	0	0	0	0	0	0
$P_{31}$	0	0	0	0	0	0	0	0	0	0	1
$P_{32}$	0	0	0	0	0	0	0	0	0	0	0
$P_{33}$	0	0	0	0	0	0	0	0	1	1	0
$P_{34}$	0	0	0	0	0	0	0	0	0	0	0
$P_{35}$	0	0	0	0	0	0	0	0	1	1	0
$P_{36}$	0	0	0	0	0	0	0	0	0	0	0
$P_{37}$	0	0	0	0	0	0	0	0	0	0	1
$P_{38}$	0	0	0	0	0	0	0	0	0	0	0
$P_{39}$	0	0	0	0	0	0	0	0	0	0	0

Table B.7 (continued): The values of the variable  $u_{i_1, i_2}$  where the entry  $(i_1, i_2)$  of the matrix corresponds to the value of variable  $u_{i_1, i_2}$ .



---

---

## APPENDIX C

---

# Wamakervallei harvesting application data

### Contents

C.1 Sample sugar levels . . . . .	205
C.2 Example information . . . . .	217

### C.1 Sample sugar levels

The vineyard blocks for which sample sugar levels have been calculated at Wamakervallei are listed in Tables C.1 and C.2. These tables contain the sugar levels for the periods 26 January 2009 to 3 February 2009, and 4 February 2009 to 12 February 2009, respectively. The vineyard blocks are renamed as blocks AA to JH.

Vineyard block	Cultivar	26-Jan	27-Jan	28-Jan	29-Jan	30-Jan	02-Feb	03-Feb
AA	CABF							
AB	CABF							
AC	CABF							
AD	CABS							
AE	CABS	18.2						
AF	CABS	16.8						
AG	CABS							
AH	CABS							
AI	CABS							
AJ	CABS							
AK	CABS							
AL	CABS							
AM	CABS							
AN	CABS							

Table C.1: The sugar levels calculated from the samples received during the period of 26 January to 3 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.

Vineyard block	Cultivar	26-Jan	27-Jan	28-Jan	29-Jan	30-Jan	02-Feb	03-Feb
AO	CABS							
AP	CABS							
AQ	CABS							
AR	CABS							
AS	CABS							
AT	CABS		17.7				18.5	
AU	CABS							
AV	CABS							
AW	CABS							
AX	CABS							
AY	CABS							
*AZ	CHAR	23.4						
BA	CHAR							
BB	CHAR		20.5					
*BC	CHAR						24.8	
BD	CHAR						22.4	
BE	CHAR						23.8	
BF	CHAR		23.3				25	
BG	CHAR		19				22.5	
BH	CHAR		20.6		20.6		20.5	
BI	CHAR					21.1		
BJ	CHAR		21.3		20.7		20.5	
BK	CHAR							
BL	CHAR							
BM	CHAR		21.8					
BN	CHAR		21.2					
BO	CHAR		21.2					
BP	CHAR		20.5					
BQ	CHAR		18.8					
BR	CHAR		20.2					
BS	CHAR	19.2					21.3	
BT	CHAR							
*BU	CHAR		23.2					
*BV	CHAR						23.2	
BW	CHAR	20.4					19.5	
BX	CHAR							
BY	CHAR						21.3	
BZ	CHAR		21.2				23.9	
CA	CHEN				20.2			
CB	CHEN		19.0				20.3	
CC	CHEN						19.7	
CD	CHEN		17.0				20.5	

Table C.1 (continued): *The sugar levels calculated from the samples received during the period of 26 January to 3 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.*



Vineyard block	Cultivar	26-Jan	27-Jan	28-Jan	29-Jan	30-Jan	02-Feb	03-Feb
CE	CHEN		17.8				19.9	
CF	CHEN						24.5	
CG	CHEN	15.2						17.3
CH	CHEN	18.4						
CI	CHEN							20.5
CJ	CHEN							21.3
CK	CHEN	15.7						18.7
CL	CHEN	14.7						15.7
CM	CHEN	17.3						
CN	CHEN	15.0						18.6
CO	CHEN	18.8			18.9		19.9	
CP	CHEN	16.0						
CQ	CHEN							15.9
CR	CHEN							
CS	CHEN							
CT	CHEN	19.7					20.1	
CU	CHEN	19.5					22.1	
CV	CHEN	20.0					20.4	
CW	CHEN	19.6					20.4	
CX	CHEN	18.2						20.3
CY	CHEN	17.0						18.2
CZ	CHEN	15.8						19.8
DA	CHEN	16.8						19.2
DB	CHEN							21.0
DC	CHEN							
DD	CHEN							
DE	CHEN							
DF	CHEN	18.6					20.9	
DG	CHEN	18					19.3	
DH	CHEN		15.7					18.8
*DI	CHEN							21.9
DJ	CHEN						21.1	
DK	CHEN	17.8						
DL	CHEN	17.1						
DM	CHEN							
DN	CHEN							
DO	CHEN							
DP	CHEN	18.1					21.1	
DQ	CHEN	17.7						
DR	CHEN							
DS	CHEN					19.2		
DT	CHEN					16.8		

Table C.1 (continued): *The sugar levels calculated from the samples received during the period of 26 January to 3 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.*

Vineyard block	Cultivar	26-Jan	27-Jan	28-Jan	29-Jan	30-Jan	02-Feb	03-Feb
DU	CHEN					18.7		
DV	CHEN							
DW	CHEN							
DX	CHEN							
DY	CHEN						22.3	
DZ	CHEN						20.2	
EA	CHEN						22.4	
EB	CHEN						22.7	
EC	CHEN						21.5	
ED	CHEN							
EE	CHEN	18.2						21.7
EF	CHEN	16.3						18.4
EG	CHEN							
EH	CHEN	15.9						20.7
EI	CHEN	19.1						
EJ	CHEN	18.6						
EK	CHEN	20.7						
EL	CHEN					19.4		
EM	CHEN					21.1		
EN	CHEN	17.2					20.0	
EO	CHEN	18						
EP	CHEN	16.2						
EQ	CHEN	18.4						
ER	CHEN	18.7						
ES	CHEN	20.1					21.2	
ET	CHEN	17.9						
EU	CHEN	20.3					21.8	
EV	CHEN	19.1					19.9	
EW	CHEN	19.0					19.1	
EX	COLO							
EY	COLO							
EZ	MALB							
FA	MALB							
FB	MALB							
FC	MALB							
FD	MALB							
FE	MALB							
FF	MALB							
FG	MALB				20.3			
FH	MERL							22.2
FI	MERL							
FJ	MERL							

Table C.1 (continued): *The sugar levels calculated from the samples received during the period of 26 January to 3 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.*

Vineyard block	Cultivar	26-Jan	27-Jan	28-Jan	29-Jan	30-Jan	02-Feb	03-Feb
FK	MERL							
FL	MERL							
FM	MERL	21.6						
FN	MERL							
FO	MERL	20.1						
FP	MERL							
FQ	MERL					20.5		
FR	MERL		19.7					21.9
FS	MERL							
FT	MERL							
FU	MERL							
FV	MERL						19.7	
FW	MERL							21.9
FX	MERL					20.0		
FY	MERL							
FZ	MERL							
GA	MERL							
GB	MERL							
GC	MERL							
GD	MERL							
GE	MERL				21.3		24.8	
GF	MERL				22.5		23.2	
GG	MERL						21.6	
GH	PINO		21.6				23.8	
GI	PINO							
GJ	PINO		19.0					
GK	PINO		19.8					
GL	PINO		22.5				25.7	
*GM	PINO		20.5					
*GN	PINO		22.3					
GO	PINO							
GP	PINO		17.9					21.4
GQ	PINO		21.7					23.9
GR	PINO							
GS	PINO		21.4					
*GT	PINO						20.4	
*GU	PINO							
GV	PINO							
GW	PINO							
GX	PINO							
GY	PINO		24.5			25.7		
*GZ	PINO							

Table C.1 (continued): *The sugar levels calculated from the samples received during the period of 26 January to 3 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.*

Vineyard block	Cultivar	26-Jan	27-Jan	28-Jan	29-Jan	30-Jan	02-Feb	03-Feb
HA	PINO							
HB	PINO							
HC	PINO							
HD	PINO							
HE	PINO		25.2					
HF	PINO					29.0		
HG	PINO					24.2		
HH	PINO							
HI	PINO			23.0				21.0
HJ	PINO							23.5
HK	RIES					15.5		
HL	SAUV							
HM	SAUV							
HN	SAUV							
*HO	SAUV							
HP	SAUV							
*HQ	SAUV							
HR	SAUV							
*HS	SAUV							
*HT	SAUV							
*HU	SAUV							
HV	SAUV				20.0	20.6		
*HW	SAUV		22.3					
*HX	SAUV							
*HY	SAUV							
*HZ	SAUV							
*IH	SAUV							
*IB	SAUV							
*IC	SAUV							
*ID	SAUV							
IE	SAUV							
*IF	SAUV							
IG	SAUV							
IH	SAUV							
*II	SAUV							
*IJ	SAUV							
*IK	SAUV							
*IL	SAUV							
IM	SHIR							23.0
IN	SHIR							
IO	SHIR							
IP	SHIR							

Table C.1 (continued): *The sugar levels calculated from the samples received during the period of 26 January to 3 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.*

Vineyard block	Cultivar	26-Jan	27-Jan	28-Jan	29-Jan	30-Jan	02-Feb	03-Feb
IQ	SHIR	18.5						
IR	SHIR							
IS	SHIR							
IT	SHIR							
IU	SHIR							
IV	SHIR							
IW	VIOG						21.8	
IX	VIOG							
IY	VIOG							
IZ	VIOG							
JA	VIOG						19.9	
JB	VIOG					18.1		
JC	VIOG						22.0	
JD	VIOG							
JE	VIOG							
JF	VIOG							

Table C.1 (continued): The sugar levels calculated from the samples received during the period of 26 January to 3 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.

Vineyard block	Cultivar	04-Feb	05-Feb	06-Feb	09-Feb	10-Feb	11-Feb	12-Feb
AA	CABF			21.8				
AB	CABF		22.3		23.4			
AC	CABF				23			
AD	CABS		19.4		18.8			
AE	CABS		20.2		20.6			
AF	CABS		19.2		20.3			
AG	CABS						24	
AH	CABS						22.9	
AI	CABS						22.3	
AJ	CABS						24.5	
AK	CABS						24.6	
AL	CABS						23.6	
AM	CABS						24.2	
AN	CABS						23.7	
AO	CABS						23.0	
AP	CABS				24.0			
AQ	CABS				22.0			
AR	CABS				23.0			

Table C.2: The sugar levels calculated from the samples received during the period of 4 February to 12 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.

Vineyard block	Cultivar	04-Feb	05-Feb	06-Feb	09-Feb	10-Feb	11-Feb	12-Feb
AS	CABS				23.0			
AT	CABS					21.4		
AU	CABS				26.3			
AV	CABS				22.6			
AW	CABS					19.8		
AX	CABS					21.4		
AY	CABS				20.9			
*AZ	CHAR							
BA	CHAR	23.1						
BB	CHAR		22.4		22.8		23.0	
*BC	CHAR							
BD	CHAR							
BE	CHAR							
BF	CHAR							
BG	CHAR		22.8					
BH	CHAR			23.0				
BI	CHAR							
BJ	CHAR			23.5				
BK	CHAR							
BL	CHAR					21.3		22.6
BM	CHAR							
BN	CHAR							
BO	CHAR		23.5					
BP	CHAR							
BQ	CHAR		22.3					
BR	CHAR		22.0					
BS	CHAR			22.9	23.5			
BT	CHAR				19.4			19.8
*BU	CHAR							
*BV	CHAR			24.5				
BW	CHAR							
BX	CHAR						21.6	
BY	CHAR					24.2		
BZ	CHAR							
CA	CHEN		22.0		23.3			
CB	CHEN							
CC	CHEN		21.2					
CD	CHEN		22.5					
CE	CHEN		21.8					
CF	CHEN							
CG	CHEN				19.9			
CH	CHEN			23.0				

Table C.2 (continued): The sugar levels calculated from the samples received during the period of 4 February to 12 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.

Vineyard block	Cultivar	04-Feb	05-Feb	06-Feb	09-Feb	10-Feb	11-Feb	12-Feb
CI	CHEN				21.3			
CJ	CHEN							
CK	CHEN						19.2	
CL	CHEN						16.3	
CM	CHEN				19.7			
CN	CHEN				19.8			
CO	CHEN			20.9				
CP	CHEN							
CQ	CHEN							
CR	CHEN		21.4					
CS	CHEN			21.6		22.2		
CT	CHEN		20.8			23.6		
CU	CHEN		22.0			25.1		
CV	CHEN		22.1			24.2		
CW	CHEN							
CX	CHEN			21.8				
CY	CHEN				21.6			
CZ	CHEN						19.8	
DA	CHEN						21.2	
DB	CHEN							
DC	CHEN				21.3			
DD	CHEN						21.0	
DE	CHEN				18.3			
DF	CHEN	20.1		21.4	22.3			
DG	CHEN	19.8		21.4	22.3			
DH	CHEN							
*DI	CHEN							
DJ	CHEN							
DK	CHEN			21.2				
DL	CHEN							
DM	CHEN			21.4				
DN	CHEN			21.1				
DO	CHEN			19.8				
DP	CHEN			22.9				
DQ	CHEN			21.5				
DR	CHEN			22.8				
DS	CHEN			22.9				
DT	CHEN							
DU	CHEN							
DV	CHEN			22.1				
DW	CHEN			22.3				
DX	CHEN							

Table C.2 (continued): The sugar levels calculated from the samples received during the period of 4 February to 12 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.

Vineyard block	Cultivar	04-Feb	05-Feb	06-Feb	09-Feb	10-Feb	11-Feb	12-Feb
DY	CHEN			23.9				
DZ	CHEN							
EA	CHEN			24.2				
EB	CHEN							20.8
EC	CHEN			23.6				20.8
ED	CHEN				18.8			22
EE	CHEN			22.8				
EF	CHEN							
EG	CHEN			21.2				
EH	CHEN							
EI	CHEN			20.4				
EJ	CHEN			21.6				
EK	CHEN			22.7				
EL	CHEN							
EM	CHEN							
EN	CHEN						20.8	
EO	CHEN							
EP	CHEN							
EQ	CHEN							
ER	CHEN							
ES	CHEN			22.3				
ET	CHEN							
EU	CHEN			23.1				
EV	CHEN							
EW	CHEN							
EX	COLO				18.3			19.3
EY	COLO				20.9			19.2
EZ	MALB			22.8				
FA	MALB			22.1				
FB	MALB				21.6			
FC	MALB					21.6		
FD	MALB					18.8		
FE	MALB			23.2		24.6		
FF	MALB							
FG	MALB							
FH	MERL					24.8		
FI	MERL		20.5		23.7			
FJ	MERL			24	24.7			
FK	MERL			23.1				
FL	MERL			23.5	24.1			
FM	MERL		25.3					
FN	MERL		23.0					

Table C.2 (continued): *The sugar levels calculated from the samples received during the period of 4 February to 12 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.*



Vineyard block	Cultivar	04-Feb	05-Feb	06-Feb	09-Feb	10-Feb	11-Feb	12-Feb
FO	MERL		24.8					
FP	MERL				22.0			
FQ	MERL		23.7		26.6			
FR	MERL				24.3		25.3	
FS	MERL							27.8
FT	MERL							27.0
FU	MERL					23.5		
FV	MERL						24.6	
FW	MERL					23.5		
FX	MERL			23.3				
FY	MERL				24			
FZ	MERL			22.6		23.2		
GA	MERL			20.2		24.5		
GB	MERL							
GC	MERL							25.7
GD	MERL							27.2
GE	MERL							
GF	MERL			25.5				
GG	MERL			24.5				
GH	PINO							
GI	PINO		22.5		25.9			
GJ	PINO							
GK	PINO		21.9		24.5		24.7	
GL	PINO							
*GM	PINO					23.8		
*GN	PINO							
GO	PINO		22.6					
GP	PINO				23.2			24.4
GQ	PINO				26.8			27.3
GR	PINO							25.1
GS	PINO				22.6			
*GT	PINO							
*GU	PINO							
GV	PINO							
GW	PINO							
GX	PINO							
GY	PINO		24.5			25.7		
*GZ	PINO							
HA	PINO							
HB	PINO							
HC	PINO							
HD	PINO							

Table C.2 (continued): The sugar levels calculated from the samples received during the period of 4 February to 12 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.

Vineyard block	Cultivar	04-Feb	05-Feb	06-Feb	09-Feb	10-Feb	11-Feb	12-Feb
HE	PINO		25.2					
HF	PINO					29.0		
HG	PINO					24.2		
HH	PINO							
HI	PINO			23.0				21.0
HJ	PINO							23.5
HK	RIES					15.5		
HL	SAUV							
HM	SAUV							
HN	SAUV							
*HO	SAUV							
HP	SAUV							
*HQ	SAUV							
HR	SAUV							
*HS	SAUV							
*HT	SAUV							
*HU	SAUV							
HV	SAUV				20.0	20.6		
*HW	SAUV		22.3					
*HX	SAUV							
*HY	SAUV							
*HZ	SAUV							
*IH	SAUV							
*IB	SAUV							
*IC	SAUV							
*ID	SAUV							
IE	SAUV							
*IF	SAUV							
IG	SAUV							
IH	SAUV							
*II	SAUV							
*IJ	SAUV							
*IK	SAUV							
*IL	SAUV							
IM	SHIR					25.5		
IN	SHIR				23.5			
IO	SHIR			20.9				
IP	SHIR		21.7		23.2			
IQ	SHIR		23.5		24.4			
IR	SHIR		22.3		24.4			
IS	SHIR				23.0			
IT	SHIR				24.0			

Table C.2 (continued): The sugar levels calculated from the samples received during the period of 4 February to 12 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.

Vineyard block	Cultivar	04-Feb	05-Feb	06-Feb	09-Feb	10-Feb	11-Feb	12-Feb
IU	SHIR				22.0			
IV	SHIR			21.5		22.2		22.5
IW	VIOG							
IX	VIOG							21.8
IY	VIOG							25.7
IZ	VIOG							22.1
JA	VIOG							
JB	VIOG					23.4		
JC	VIOG							
JD	VIOG							24.0
JE	VIOG							20.9
JF	VIOG						22.7	

Table C.2 (continued): *The sugar levels calculated from the samples received during the period of 4 February to 12 February, 2009. The blocks that were harvested before 9 February are denoted by a darker row colour.*

## C.2 Example information

This section contains results obtained by applying the decision support system to the 2009 harvesting data of Wamakersvallei Winery. In Table C.3, the jobs generated as the first scenario when attempting to evaluate the initial harvest scheduling solution are listed along with the job types, weights, arrival times, allowed processors (where relevant), jobs that are allowed to be mixed (where relevant) and the additional processing time to be added to the normal processing time when two Type II truckloads are joined together.

$J_j$	Type	$w_j$	$e_j$	$P_i$ with $\mu_{ij} = 1$	all $J_\ell$ for which $q_{j\ell} = 1$	extra processing time
$J_1$	I	11.0	3.3	$P_{21}, \dots, P_{44}$	–	0.0
$J_2$	I	25.0	3.4	$P_{21}, \dots, P_{44}$	$J_3, J_4$	0.0
$J_3$	I	80.0	1.6	$P_{21}, \dots, P_{44}$	$J_2, J_4$	0.0
$J_4$	I	40.0	5.6	$P_{21}, \dots, P_{44}$	$J_2, J_3$	0.0
$J_5$	II	11.9	3.7	–	–	1.1
$J_6$	II	18.0	6.4	–	–	1.5
$J_7$	II	2.0	2.7	–	–	0.0
$J_8$	II	24.1	6.4	–	–	1.6
$J_9$	II	51.6	5.4	–	–	2.5
$J_{10}$	II	6.3	0.7	–	–	0.0
$J_{11}$	II	37.8	5.3	–	–	2.1
$J_{12}$	IV	0.0	3.2	–	–	0.0
$J_{13}$	IV	0.0	0.0	–	–	0.0
$J_{14}$	IV	0.0	4.5	–	–	0.0
$J_{15}$	IV	0.0	5.2	–	–	0.0
$J_{16}$	IV	0.0	5.6	–	–	0.0
$J_{17}$	IV	0.0	2.9	–	–	0.0
$J_{18}$	IV	0.0	3.4	–	–	0.0
$J_{19}$	IV	0.0	3.9	–	–	0.0
$J_{20}$	IV	0.0	3.7	–	–	0.0
$J_{21}$	IV	0.0	1.7	–	–	0.0
$J_{22}$	IV	0.0	5.0	–	–	0.0
$J_{23}$	IV	0.0	2.8	–	–	0.0
$J_{24}$	IV	0.0	5.0	–	–	0.0
$J_{25}$	IV	0.0	3.2	–	–	0.0
$J_{26}$	IV	0.0	6.2	–	–	0.0
$J_{27}$	IV	0.0	4.2	–	–	0.0
$J_{28}$	IV	0.0	6.0	–	–	0.0
$J_{29}$	IV	0.0	5.2	–	–	0.0
$J_{30}$	IV	0.0	0.0	–	–	0.0
$J_{31}$	IV	0.0	4.8	–	–	0.0
$J_{32}$	IV	0.0	5.6	–	–	0.0
$J_{33}$	IV	0.0	2.8	–	–	0.0
$J_{34}$	IV	0.0	6.0	–	–	0.0
$J_{35}$	IV	0.0	3.1	–	–	0.0
$J_{36}$	IV	0.0	2.8	–	–	0.0
$J_{37}$	IV	0.0	1.8	–	–	0.0
$J_{38}$	IV	0.0	0.8	–	–	0.0
$J_{39}$	IV	0.0	1.9	–	–	0.0
$J_{40}$	IV	0.0	6.0	–	–	0.0

Table C.3: Jobs  $J_1, \dots, J_{40}$  generated as part of the first scenario from the initial harvesting schedule in §8.2.

---

---

## APPENDIX D

---

# VinDSS user manual

### Contents

D.1 Importing data from Excel . . . . .	219
D.2 Generating a harvesting schedule from the imported data . . . . .	220

This appendix serves as a concise user manual for VINDSS with the main focus on the correct import of the data from Microsoft Excel [86] (discussed in §D.1). The process of generating and viewing the solution is briefly outlined §D.2.

### D.1 Importing data from Excel

The layout of the Microsoft Excel file used for the importing of data is shown in Figure D.1. Each column contains the name of the column as the first column entry. For example, Excel column A contains the referencing names for each of the vineyard blocks. Therefore, cell (A,1) contains the title ‘Block reference’. However, importing data also works perfectly fine without the column name. The importance lies in the order of the columns and also in the requirement that all entries should only start in row 2 since row 1 is considered a row containing only titles. The order of the columns is first, the block reference, then in column B the cultivar abbreviation. It is necessary for the abbreviation to exactly match the abbreviations listed in Table D.1. Column C contains the class of the grapes of the vineyard block and column D contains the expected vineyard block yield expressed in tonnes. The most recent sample data are included in Column E and the calculated sugar level is included in Column F, expressed in degrees Balling.

In the case where it is required that the vineyard block should be kept separate from other vineyard blocks, the name of the vineyard block is preceded by a ‘\$’. For example, if vineyard block AB may not be mixed with any other blocks, \$AB is entered as the vineyard block name. If, for any reason, a block in the Excel file should not be considered for harvesting during the current harvesting period, the name is preceded by a ‘\*’. For example, if block AB is referred to as \*AB it is not included in the harvesting schedule.

Column G is left open after which the necessary tank information is listed, starting with the tank name in Column H. It is not the tank name that is of the greatest importance, but rather the order — the fermentation tank information should always be listed in the indicated order, starting at tank DF1 up to the last DF tank, DF10 and then the RT tanks in numerical order

	A	B	C	D	E	F	G	H	I	J	K	L
	BLOCK REFERENCE	CULTIVAR	CLASS	TONNES	LATEST SAMPLE DATE	SUGAR LEVEL		TANK	TONNES	CULTIVAR	CLASS	EMPTYING DATE
2	AA	CABF	1	65	06-Feb	21.8		DF1	20	PINO	1	10-Feb
3	AB	CABF	3	8	05-Feb	22.3		DF2	12	PINO	2	10-Feb
4	BA	CHAR	3	16	04-Feb	23.1		DF3				
5	BB	CHAR	1	4	05-Feb	22.4		DF4				
6	BD	CHAR	1	70	02-Feb	22.4		DF5				
7	BE	CHAR	1	80	02-Feb	23.8		DF6				
8	BF	CHAR	1	23	02-Feb	25.0		DF7				
9	BG	CHAR	3	5	05-Feb	22.8		DF8				
10	BH	CHAR	1	75	06-Feb	23.0		DF9				
11	BI	CHAR	2	5	30-Jan	21.1		DF10				
12	BJ	CHAR	1	24	06-Feb	23.5		RT317				
13	BK	CHAR	1	50	05-Feb	21.5		RT318				
14	BP	CHAR	1	4	27-Jan	20.5		RT319				
15	BQ	CHAR	1	10	05-Feb	22.3		RT320				
16	BR	CHAR	1	7	05-Feb	22.0		RT321				
17	BS	CHAR	3	35	06-Feb	22.9		RT322				
18	BW	CHAR	1	10	02-Feb	19.5		RT323				
19	BZ	CHAR	1	20	02-Feb	23.9		RT324				
20	CB	CHBL	1	15	02-Feb	20.3		RT325				
21	CC	CHBL	1	55	05-Feb	21.2		RT326				
22	CD	CHBL	1	75	05-Feb	22.5		RT327				
23	CE	CHBL	1	65	05-Feb	21.8		RT328				
24	CF	CHBL	6	10	02-Feb	24.5		RT329				
25	CH	CHBL	1	16	06-Feb	23.0		RT330				
26	CI	CHBL	1	16	03-Feb	20.5		RT331				
27	CJ	CHBL	1	16	03-Feb	21.3		RT332				
28	CK	CHBL	1	15	03-Feb	18.7						
29	CR	CHBL	2	24	05-Feb	21.4						
30	CS	CHBL	2	15	06-Feb	21.6						
31	CV	CHBL	2	8	05-Feb	22.1						
32	CW	CHBL	2	3	02-Feb	20.4						
33	CX	CHBL	1	30	06-Feb	21.8						
34	CY	CHBL	1	75	03-Feb	18.2						
35	CZ	CHBL	1	30	03-Feb	19.8						
36	DA	CHBL	1	15	03-Feb	19.2						
37	DB	CHBL	2	15	03-Feb	21.0						
38	DF	CHBL	2	6	06-Feb	21.4						

Figure D.1: A screen shot the exact format of the Microsoft Excel file used to import data to VindSS.

starting at RT317 and ending at RT332. Column I contains the weight of the grapes that are currently contained in the tank and is left open if the tank is empty. It is only one third of this indicated weight actually being transported to the presses, since the free run juice is first drained and does not require pressing. The contents of the tank is included in columns J and K, indicating the cultivar and class, respectively. Finally, the expected emptying date of the tank is indicated in Column L.

Further important factors to keep in mind when importing the data from an Excel file, is that the data should be contained in the first Worksheet of the file and that the file should be saved as an xls-file. The Excel file should be saved as `C://HarvestScheduling/Data.xls`

## D.2 Generating a harvesting schedule from the imported data

Generating a harvesting schedule from the imported data is a very simple process. The user should select the number of days for which a schedule is sought, ranging between 3 and 5 days. Then the starting date should be selected from the drop-down menu, as shown in Figure D.2. VINDSS cannot tell the difference between weekdays and weekend days. Therefore, if

Cultivar number	Cultivar name	Cultivar reference
0	Cabernet Franc	CABF
1	Cabernet Sauvignon	CABS
2	Shiraz	SHIR
3	Petit Verdot	PETI
4	Pinotage	PINO
5	Merlot	MERL
6	Mourvedre	MOUR
7	Malbec	MALB
8	Roobernet	ROOB
9	Ruby Cabernet	RUBY
10	Cinsaut	CINS
11	Chenin Blanc	CHBL
12	Sauvignon Blanc	SAUV
13	Chardonnay	CHAR
14	Viognier	VIOG
15	Colombar	COLO
16	Hanepoot	HANE
17	SA Riesling	RIES
18	Weisser Riesling	WEIS

Table D.1: The abbreviations used to refer to the cultivars when the data importing function of VINDSS.

the schedule is generated on a Friday, the user should ensure that the selected date refers to the coming Monday. If the harvest scheduling process is started, it cannot be stopped until a feasible solution is found. If, for some reason, it was started unnecessarily, the application may be stopped by cancelling the process in the task manager.

The final harvesting schedule is displayed by selecting the tabbed pane labelled ‘Suggested Harvesting Schedule’.

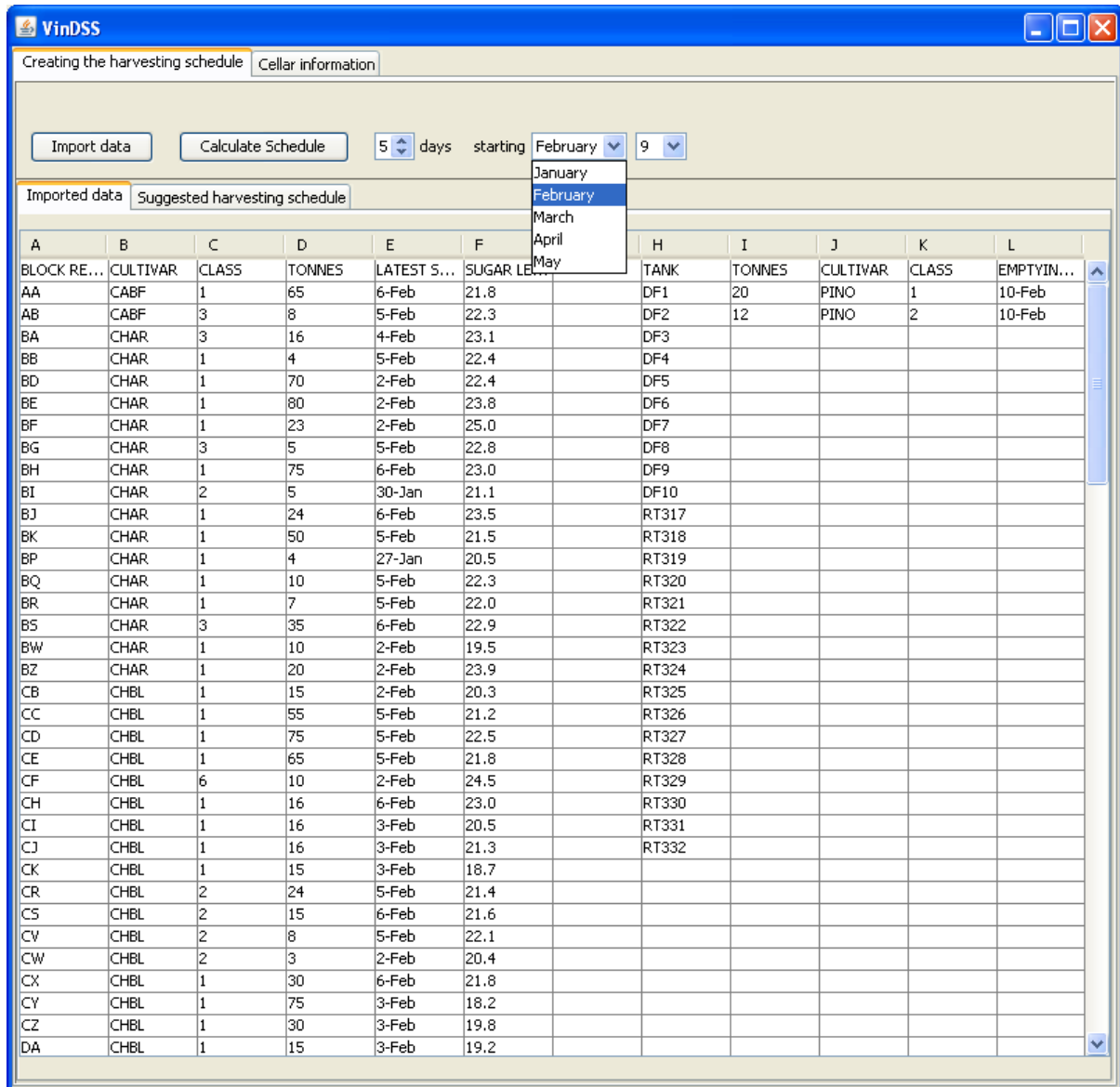


Figure D.2: A screen shot of the drop down month selection function of VINDSS, where only months considered as harvesting months is available for selection.