EVOLUTIONARY ALGORITHMS FOR SUPPLY CHAIN OPTIMISATION

Maksud Ibrahimov

Submitted in fulfillment of the requirements
of the degree of Doctor of Philosophy

Principal Supervisor: Professor Zbigniew Michalewicz
Co Supervisor: Dr. Arvind Mohais
Co Supervisor: Dr. Charles Lakos

October 2012
School of Computer Science
The University of Adelaide
Australia

Contents

List of Figures

List of Tables

Abstract

Many real-world problems can be modelled as a combination of several interacting components. Methods based on Evolutionary Algorithms seem to be appropriate for handling such problems, but they have not been extensively researched in such domains. In this thesis we study the applicability of Evolutionary Algorithms for today's high complexity real-world problems which consist of several interacting components. A natural source of such problems emerged from supply chain management problems which consist of several interacting components, and are also generally non-linear, heavily constrained, and involve many variables. We aim to study possible approaches for supply chain optimisation problems that seamlessly integrate algorithms addressing the local components, under the framework of global optimisation.

Thesis Declaration

Dedication

To my parents and my sister Nargiz.

Acknowledgements

I would like to thank my supervisory panel comprising of Prof. Zbigniew Michalewicz, Dr. Arvind Mohais and Dr. Charles Lakos for their support throughout the candidature. I especially want to acknowledge the support and guidance of my primary supervisor, Prof. Zbigniew Michalewicz. I was extremely fortunate to meet him, he has not only been the greatest mentor who inspired me and provided his invaluable insight into the field of evolutionary computation and its connection with business but also is a true role model for me. I would like to also mention help and support of Dr. Arvind Mohais. His guidance helped me to deeply understand many aspects of evolutionary computation.

Great credit is due to Matthew Michalewicz, Constantin Chiriac, Roland Spitty and Dr. Luigi Barone for their wise advices on different aspects of business, supply chain and software development. Additionally, I would like to thank support and help of Dr. Larisa Aksenova who supported me immeasurably.

I cannot thank enough my parents and my sister for their love and support.

I also thank William Winser for his assistance with proof-reading the text.

Chapter 1

Introduction

During the last few decades many algorithms were introduced that were inspired by various processes in nature, ranging from mimicking the behaviour of animals and cooling metals to immune systems and genetics. Many of them were based on the fundamental principles of natural selection and survival of the fittest, formulated by the nineteenth century naturalist Charles Darwin in his book "On the Origin of Species" [53]. These fundamental principles were used to create a family of new algorithms. Evolutionary Strategies invented by Rechenberg [150], Evolutionary Programming by Fogel [65], Holland's Genetic Algorithms [82] and Koza's Genetic Programming [93] were the first meta-heuristics inspired by Darwin's principles. These researchers were some of the pioneers of the field of Evolutionary Computation (EC).

1.1 Motivation

Over the last two decades, the Evolutionary Computation (EC) community experienced significant growth. In 1990 there were no journals specially dedicated to EC and there was just a single conference which ran every second year – the International Conference on Genetic Algorithms (ICGA). At the ICGA'91, research from all main EC branches was presented together for the first time. Since then the community has been growing rapidly [49]: thousands of papers have been written on evolutionary algorithms (with 40% annual growth during 1978 – 1998 [6]), journals dedicated to EC have emerged (IEEE Transactions on Evolutionary Computation and Evolutionary Computation Journal being the main two), and dedicated conferences (e.g. PPSN, IEEE CEC, GECCO) and special tracks at most Artificial Intelligence and Operations Research conferences take place. The number of practitioners of EC is also growing exponentially [85].

Many new variants of evolutionary algorithms were developed by incorporating new features into standard algorithms, such as different data structures for solution representation, several subpopulations and migration between them, various selection criteria, different recombination and mutation operators. Theoretical research focussed on the properties of evolutionary algorithms (e.g convergence rates, the balance between exploration and exploitation) as well as on the behaviour of evolutionary algorithms in problems with different characteristics (e.g. ruggedness of the landscape, deceptiveness). At the same time, experimental research concentrated on standard problems from operations research, such as many types of scheduling, travelling salesman, vehicle routing, inventory management, stock cutting, packing, graph colouring and knapsack problems. Many standard benchmarks were created to compare the performance of different algorithms in relation to these problems [1, 39, 163].

1.2 Theory and practice of evolutionary algorithms

Many claims have been made about the applicability of EC methods for solving real-world problems in many domains: defence, finance, management, economics [2, 37] and others. Despite the advances in EC research, growing interest in evolutionary algorithms, and a desire to apply research in evolutionary algorithms to real-world problems, some researchers, who are involved in real-world applications, have indicated that there is a significant and increasing gap between the theory and practice of evolutionary algorithms. Dave Davis talked about this gap 16 years ago at the Workshop on Evolutionary Algorithms, organised by the Institute for Mathematics and Its Applications, University of Minnesota, Minneapolis, Minnesota, October 21 - 25. Being one of the most recognised practitioners of Evolutionary Algorithms at that time, he stated that theoretical results often are of very little use when applied to real-world applications. Recent articles also mention the existence of this gap and its continuing growth [119, 120].

Research in evolutionary algorithms has made quite substantial progress, but the focus of most experimental research efforts in the EC community has remained almost the same for the last two decades – that is mostly on the classical operations research problems mentioned earlier. These problems were relevant to the real-world problems two decades ago, but since that time the complexity of business problems has skyrocketed. Researchers still use the standard experimental sets of functions for numerical optimisation (e.g. De Jong's test suite [57]) and standard benchmarks for combinatorial optimisation [39, 163]. Clearly these problems are NP-hard and very important for research, but they do not fully reflect the complexity of today's real-world problems [119, 120]. Due to globalisation, integration and advances in technology, many real-world businesses are becoming more complex in structure, involving several components with non-linear interactions between them. The larger the businesses grow, the greater the need for systems that assist in decision-making activities – *decision support systems.*

Consider the following example of a company that distributes water tanks. Water tanks are transported to geographically dispersed clients via trucks (possibly with trailers) of different capacities. Efficient packing of water tanks into trucks can be achieved by bundling smaller tanks inside the bigger ones as well as stacking them together (see Figure 1.1). Tanks can only be unbundled at certain locations prior to customer delivery and customers can be located at quite a distance from each other. Truck drivers may have different driving licences which allow them to drive only certain types of trucks (with or without trailers). The company faces a problem with daily decisions as to how to load trucks, whether to use additional trailers, which customer to serve, and what roads to take to deliver products to different customers on time. The goal is to minimise the overall cost of delivery. However, the calculation of cost is not trivial because it includes the cost of total distance travelled, salaries of drivers and the number of tanks delivered to customers on a trip.

To create a decision support system for this problem, the following problems need to be considered:

- the packing problem, involving determining the optimal bundling and stacking of tanks on the truck.

- the vehicle routing problem with pickup and delivery, to determine the optimal routes and loading schedules for each truck once the tanks are loaded.

- the problem of allocation of drivers to trucks.

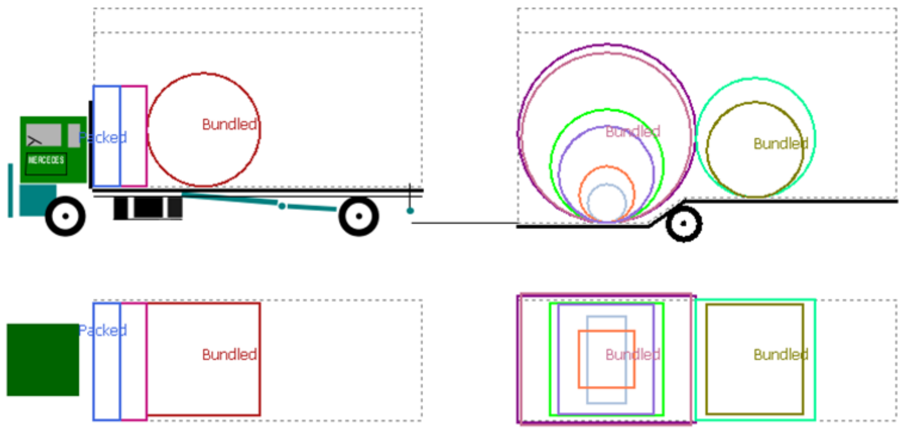


Figure 1.1: Packing of water tanks into the truck. The top picture presents the side view, the bottom presents the top view of the truck.

- the truck/trailer selection problem

Each of the above problems is hard to solve on its own. However, they are interrelated, so consideration of each of them in isolation will lead to a sub-optimal result of the overall problem. For example, the classical packing problem, which involves packing goods into a container to utilise container space as efficiently as possible, is a very well studied problem. However, considering optimal packing of water tanks without taking into account which driver is going to drive the truck and customer locations is useless as customers may reside in different parts of the country and the distances can be quite large. Additionally, a decision about using a particular truck with a trailer for a particular trip may prevent another delivery which requires a driver with appropriate qualifications. Thus, the optimal solution to the packing problem by itself has little practical overall significance as it constitutes just one component of a larger problem and any attempts to solve it in isolation are of little practical interest. From a business perspective solving a single component of the problem in isolation is called *local optimisation*, whereas consideration of all components of the problem will be referred to in this thesis as *global optimisation from the business perspective*. Note that the concept of global versus local optimisation from a business perspective is very different to the same concept from a classical mathematical perspective.

The EC community has been studying each of these problems in isolation; each of these problems represents only one aspect of the overall problem. All these components are interconnected and failure to consider any of these problems will result in a sub-optimal solution. In addition to all these problems there are specific business rules and constraints that need to be satisfied, e.g. tanks can be bundled and unbundled only at certain locations, the addition of the trailer to the truck allows bundling of some special types of tanks. Usually these types of constraints and business rules are not included in classical operations research problems. To address the overall problem of optimal tank distribution, all four components of the problem (packing, vehicle routing, driver allocation and truck/trailer selection) need to be considered together, as decisions about one component have immediate implications for the other components. This problem illustrates the case of global optimisation from the business perspective. Ackoff [3] correctly identified this issue over 30 years ago:

> Problems require holistic treatment. They cannot be treated effectively by decomposing them analytically into separate problems to which optimal solutions are sought.

While experimental research has focussed on the classical types of problems, the real-world problems have increased in complexity many times, particularly in consideration of the bigger scale of such problems. It should be noted that several factors contribute to the complexity of classic operations research problems:

- *size of the search space*, which is the number of total possible candidate solutions.
- *constraints*, which usually split the search space into feasible and infeasible regions.
- *multiple objectives*, that have to be taken into account.
- *variability*, which requires algorithms to operate on probability distributions rather than on exact values.
- *time-changing environments*, which introduce a time dimension to static problems.

The EC research community addressed all these factors [25, 58, 117, 91] well but did so in isolation. However, when we consider real-world problems with several components, each component includes several of these factors but on the top of that they expose additional issue of *non-linear interaction* between the components. This brings all the classic factors to another level. In problems with interconnected components the size of the search space becomes a multiplication of the cardinalities of the search spaces of individual components. In the case of the truck loading problem the search space becomes the multiplication of search spaces for packing, vehicle routing and assignment problems. Constraints may exist both on the component level as well as between the components; objectives in multi-component problems become multi-level, as objectives need to be satisfied on the components level, and in addition, the overall objectives need to be considered; variability becomes more complex as well.

Even when researching a single-component problem there is still some small mismatch between the research activities and practice. For example, the majority of published research in evolutionary algorithms on time-varying problems addresses the issue of time-varying objective functions. While these problems clearly deserve research efforts, in the real-world environment the objective function does not change very often. In most cases, businesses face a problem of time-varying constraints changing over time but the objective usually stays the same (e.g. maximisation of profit or minimisation of delays).

1.3 Objectives

The main purpose of this thesis is to research models of today's high complexity real-world problems which consist of several interacting components. We aim to study approaches that seamlessly integrate algorithms addressing the local components, under the framework of global optimisation. What approach should be taken to address problems of such complexity? Many papers by the operations research community offer to simplify the problem to linear or integer programming models. In this way a precise solution can be found using classical operations research algorithms. However, the linear model cannot reflect all the complexity of real-world problems and the solution to the linear model would not be palatable. The EC methods, on the other hand, allow the retention of all the non-linear interactions and constraints in the model and find a near-optimum solution. Thus, the more complex the model, the greater is the need for the EC methods.

We investigate algorithms based on *cooperative coevolution* for *supply chain* problems as they seem to be a natural fit: each subpopulation is responsible for the component of the problem and cooperative coevolution is responsible for the overall problem. In private conversation with Professor Kenneth De Jong, one of the pioneers of cooperative coevolution, at the Congress on Evolutionary Computation in 2010, he mentioned that to the best of his knowledge cooperative coevolution has not been applied to the problems of optimisation of supply chains. This research aims to bring the research in evolutionary algorithms one step closer to the real-world applications which are sought today by many industries. The choice of environments, where problems of such complexity arise in a natural way are *supply chains*.

Supply chains usually consist of multiple *silos* which are self-contained organisational entities. The *silo* is fundamental single component of any supply chain (Figure 1.2). Any supply chain can be assembled by combining silos. A special way of combining silos as a group of silos which share similar functionality and constitute a conceptual stage of a supply chain is called an *echelon*. One way of thinking of echelons is as a grouping tool to illustrate a high level structure of the supply chain.

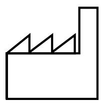


Figure 1.2: A single silo.

In this thesis the term *material flow* is defined as the way the silos of the supply chain are connected to each other. Several types of flows can be observed. One simple type is a *one-to-one* flow where one silo is directly connected to another. The first silo sends the output of its execution to the second silo. This type of sequential flow is shown in Figure 1.3. In this and in other supply chain diagrams of this thesis the direction of arrows representing flow shows the direction of material flow.

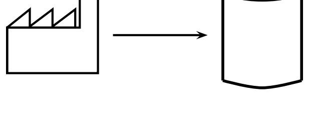


Figure 1.3: Two-silo model.

This two-silo model can also be thought of as a two-echelon model with one silo in each echelon (Figure 1.4). Considering multiple silos in each echelon leads us to more complex flows: *one-to-many* and *many-to-one*. In the former case, products of the execution of a single silo in the first echelon are distributed among multiple silos in the second echelon. The second case has several silos in the first echelon and they ship all products to a single silo in the second echelon. Other variations of flows between silos of two echelons may include a *many-to-many* flow, with a special case of a multi-silo supply chain with one-to-one flows between the silos. Diagrams of these flows are presented in Figure 1.5.

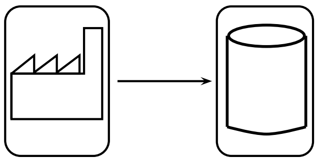


Figure 1.4: Two-echelon model with a single silo in each echelon.

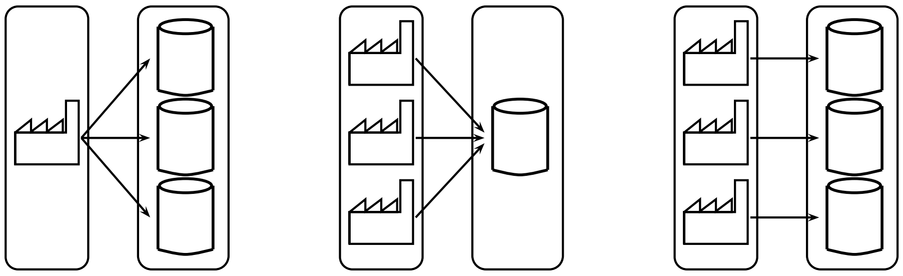


Figure 1.5: Two-echelon models.

Very often flows can also be within the echelons – *inter-echelon dependencies* (as distinct from *intra-echelon dependencies* which are dependencies between echelons). An example of intra-echelon flow studied in this thesis is material movement from one stockpile of a mine to another. The pictorial representation of such a supply chain is presented in Figure 1.6 which presents two echelons with inter- and intra- echelon dependencies.

Large enterprises often own many consecutive echelons of the supply chain. The example of such a supply chain with five echelons is presented in Figure 1.7.

Some large supply chains can be of a very complex structure that do not fit perfectly into the echelon model. An example of such a structure is shown in

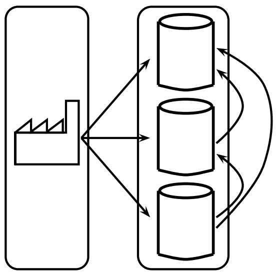


Figure 1.6: Two-echelon extended model

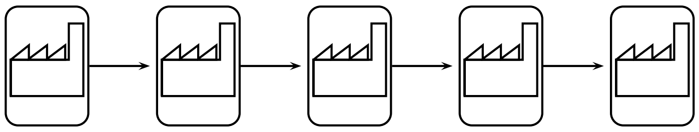


Figure 1.7: Five-echelon model

Figure 1.8. Silos with the same icons in this figure reveal similar functionality (e.g. transportation silo, production silo). In such a model still some silos can still be grouped into echelons, but we did not consider them in this model.

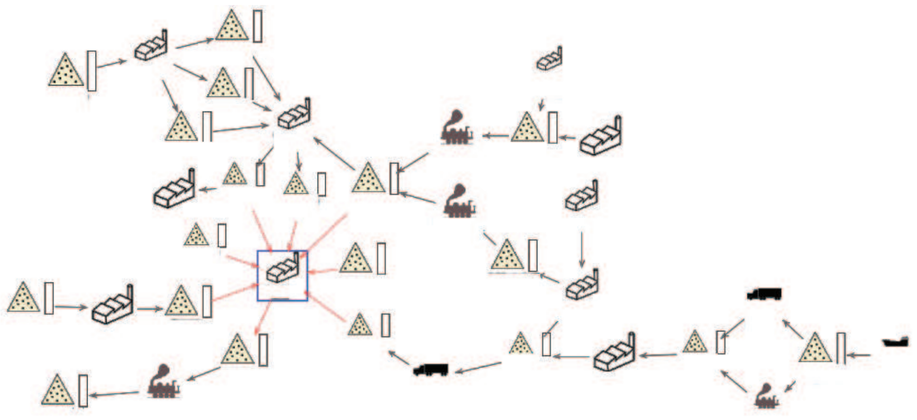


Figure 1.8: Multi-echelon supply chain with complex structure

The supply chain models mentioned earlier show the variety of different supply chains, from the very simple single silo to the multi-echelon, with different types of flows to supply chains that do not fit into the linear echelon model. In this thesis we will focus only on these models. However, there are an infinite number of supply chain models with different structures, material flows and constraints which we do not consider, and which will be considered in future research.

1.4 Organisation

The thesis is organised in the following way. Chapter 2 explains the basic concepts that are used throughout the thesis. We discuss basic terminology, management strategies and global optimisation in supply chains. Then we provide background information on evolutionary algorithms and their variations. Literature reviews of job shop scheduling and vehicle routing problems are provided in this chapter as some experiments in the further chapters are based on these problems.

We review the issue of dynamic environments within a single silo in Chapter 3. We articulate that, from the business perspective a dynamic environment in most cases implies time-varying constraints rather than a time-varying evaluation function. The review of the literature on evolutionary algorithms in dynamic environments presented in this chapter shows the need to investigate problems of time-varying constraints. The problem for the wine industry with time-varying constraints is presented and the evolutionary algorithm that addresses it is discussed. Moreover, this problem is used as a building block for the global optimisation of the whole wine supply chain in Chapter 7.

We investigate a one-to-one type of material flow based on an experimental supply chain in Chapter 4 (see Figure 1.3). Production and distribution silos of this supply chain are based on problems well-known in the operations research community. The functionality of the first silo is based on job shop scheduling and the functionality of the second silo is based on vehicle routing problems. We show that it is important to have cooperation between the silos during the optimisation in order to produce good results.

Chapter 5 presents a pair of examples of a supply chain with each echelon containing more than one silo. Several more complex material flow models are investigated within an experimental test bed. The second part of the chapter presents a relatively simple real-world example of a two-echelon supply chain with multiple silos within each echelon. The structural diagrams of these problems are shown in Figure 1.5.

Chapter 6 provides an extended real-world example of a two-echelon supply chain with a one-to-many material flow of silos in two echelons and additional inter-echelon dependencies. The problem is taken from the mining industry and involves scheduling of equipment in order to reach tonnage and quality targets of iron ore. The product movement in this problem is more complex than in the previous problems and involves many time-varying constraints.

Another real-world example, the wine supply chain consisting of five consecutive echelons, is presented in Chapter 7 (see Figure 1.7). Each echelon is described in detail in this chapter. The bottling problem presented in Chapter 3 represents one of the echelons of this supply chain. We investigate methods based on evolutionary algorithms to coordinate all the existing systems and reuse decision support modules within each one for global optimisation.

The decision support of complex product movement between the silos of a real-world supply chain is investigated in Chapter 8. An example of such a supply chain is shown in Figure 1.8. We propose methods based on fuzzy logic and evolutionary algorithms to address the complexity of the problem.

Finally, Chapter 9 summarises the results presented in this thesis and provides the directions for future research.

As the main contribution of this thesis, we have brought the research in evolutionary algorithms one step closer to practice by undertaking the following:

- investigating problems that consist of two or more interacting components
- investigating different types of interactions between supply chain echelons and silos
- showing that cooperation between silos during the optimisation results in better quality solutions
- developing a framework that addresses the issue of global optimisation from a business perspective by reusing the existing local silo algorithms
- successfully applying the global optimisation framework initially tested on experimental problems to the problems from the real world

1.5 Publications

The results of some parts of this thesis have been reported in the following publications:

- Maksud Ibrahimov, Arvind Mohais, Sven Schellenberg, Zbigniew Michalewicz *Scheduling in Iron Ore Open Pit Mining.* International Journal of Advanced Manufacturing Technology, 2012, submitted

- Maksud Ibrahimov, Arvind Mohais, Sven Schellenberg, Zbigniew Michalewicz *Evolutionary Approaches for Supply Chain Optimisation. Part I: Single and Two-component Supply Chains.* International Journal of Intelligent Computing and Cybernetics, volume 5, issue 4, 2012

- Maksud Ibrahimov, Arvind Mohais, Sven Schellenberg, Zbigniew Michalewicz *Evolutionary Approaches for Supply Chain Optimisation. Part II: Multi-silo Supply Chains.* International Journal of Intelligent Computing and Cybernetics, volume 5, issue 4, 2012

- Maksud Ibrahimov, Arvind Mohais, Maris Ozols, Sven Schellenberg, Zbigniew Michalewicz *Advanced Planning in Vertically Integrated Wine Supply Chains.* In Evolutionary Computation for Dynamic Optimization Problems, in Shengxiang Yang and Xin Yao (Editors), Springer, series on Studies in Computational Intelligence, 2012, pp 125-148

- Arvind Mohais, Maksud Ibrahimov, Sven Schellenberg, Neal Wagner, Zbigniew Michalewicz*An Evolutionary Approach to Practical Constraints in Scheduling: A Case-Study of the Wine Bottling Problem.* In Chiong, R., Weise, Th., and Michalewicz, Z. (Editors), Variants of Evolutionary Algorithms for Real-World Applications, Springer-Verlag, Berlin, 2012 pp 31-58

- Sven Schellenberg, Arvind Mohais, Maksud Ibrahimov, Neal Wagner, Zbigniew Michalewicz *A Fuzzy-Evolutionary Approach to the Problem of Optimisation and Decision- Support in Supply Chain Networks.* In Chiong, R., Weise, Th., and Michalewicz, Z. (Editors), Variants of Evolutionary Algorithms for Real-World Applications, Springer-Verlag, Berlin, 2012, pp 143-166

- Maksud Ibrahimov, Arvind Mohais, Sven Schellenberg, Zbigniew Michalewicz *Comparison of Cooperative, Multiobjective Cooperative and Classical Evolutionary Algorithms for Global Supply Chain Optimisation.* In Proceedings of Genetic and Evolutionary Computation Conference, Dublin, Ireland, July 12-16, 2011

- Maksud Ibrahimov, Arvind Mohais, Sven Schellenberg, Zbigniew Michalewicz *Comparison of Different Evolutionary Algorithms for Global Supply Chain Optimisation and Parameter Analysis.* In Proceedings of 2011 IEEE Congress on Evolutionary Computation, New Orleans, June 5-8, 2011, pp 2407-2414

- Maksud Ibrahimov, Arvind Mohais, Sven Schellenberg, Zbigniew Michalewicz *Advanced planning in vertically integrated supply chains.* In Intelligent Decision Systems in Large-Scale Distributed Environments, P. Bouvry, H. Gonzlez-Vlez, and J. Kolodziej (Editors), Springer, Volume 362, 2011, pp 125-148

- Maksud Ibrahimov, Neal Wagner, Arvind Mohais, Sven Schellenberg, Zbigniew Michalewicz *Comparison of multiobjective cooperative and classical evolutionary algorithms for global supply chain optimisation.* In Proceedings of the 24th European Conference on Operation Research, Lisbon, Portugal, 11-14 July 2010, pp 121-122

- Maksud Ibrahimov, Neal Wagner, Arvind Mohais, Sven Schellenberg, Zbigniew Michalewicz *Comparison of cooperative and classical evolutionary algorithms for global supply chain optimisation.* In Proceedings of the 2010 IEEE World Congress on Computational Intelligence, Barcelona, Spain, 18-23 July 2010, pp 1-8

- Arvind Mohais, Sven Schellenberg, Maksud Ibrahimov, Neal Wagner, Zbigniew Michalewicz *Time-Varying Constraints and Other Practical Problems in Real-World Scheduling Applications.* In Proceedings of the 2010 IEEE World Congress on Computational Intelligence, Barcelona, Spain, 18-23 July 2010, pp 1-8

- Sven Schellenberg, Arvind Mohais, Maksud Ibrahimov, Neal Wagner and Zbigniew Michalewicz *Optimising supply chain networks by means of a hybridised simulation based approach.* In Proceedings of the 2010 IEEE World Congress on Computational Intelligence, Barcelona, Spain, 18-23 July 2010, pp 1-6

- Zbigniew Michalewicz, Maksud Ibrahimov, Arvind Mohais *Applications of Evolutionary Methods for Complex Industrial Problems.* In Proceedings of EUROGEN'09, T. Burczynski and J. Periaux (Eds), 2009, pp 1-6

- Maksud Ibrahimov, Arvind Mohais, Zbigniew Michalewicz *Global optimization in supply chain operations chapter.* In Raymond Chiong and Sandeep Dhakal (Editors), Natural Intelligence for Scheduling, Planning and Packing problems, Springer, 2009, pp 1-28

Chapter 2

Basic Concepts and Literature Review

The first part of this chapter presents some background information on supply chains. It discusses basic terminology, push, pull and hybrid strategies for supply chains, and the complexity involved in decision making. Many businesses focus on making decisions for just a part of their supply chain without consideration of the whole supply chain (*local optimisation*), but often a much better value can be extracted from the supply chain when focussing on a broader subset of the supply chain using *global optimisation*. One of the reasons some companies still run their businesses based on the locally optimal decisions is that global optimisation comes with complex time-changing constraints and non-linear relationships between the components. To address this complexity, in further chapters of this thesis we use methods based on Computational Intelligence. We will mainly concentrate on the subset of Computational Intelligence – Evolutionary Algorithms. Background information on different classes of evolutionary algorithms and a literature review of their application to supply chain problems is provided in this chapter.

In the later we introduce experimental studies on two-echelon supply chains. The functionality of silos is based on two well known problems: job shop scheduling and vehicle routing problems under global optimisation based on cooperative coevolution. Section 2.4 describes the scheduling problem and its varieties, then provides a literature review focussing mainly on the Job Shop Scheduling Problem. Section 2.5 provides discussion on different types of the Vehicle Routing

Problem and surveys some of the algorithms that address it. Section 2.6 gives a review of basic concepts of cooperative coevolution and provides some literature review. The final section summarises the chapter.

2.1 Supply Chain Management

Supply chain management (SCM) is a field that combines management of various business functions with a broad range of theoretical domains such as systems theory, inventory control, optimisation, logistics, mathematical and computer modelling. A Global Supply Chain Forum defines Supply Chain Management as the integration of key business processes across the supply chain for the purpose of creating value for customers and stakeholders. The term *supply chain management* was first mentioned in 1982 by Oliver and Webber [130], but the concept of supply chain management was born long before in the early 20th century with the creation of the assembly line. In the 1960s the appearance of IBM computers helped raise interest in modern supply chain management.

Understanding and managing a company's supply chain is one of the hardest tasks procurement planners and supply chain managers face in today's business environment. Ideally, a supply chain is driven by demand generated by the placing of orders by customers. An order may include one or many items which are composed of processed and unprocessed raw materials and components. In addition to the customers' orders which, generally speaking, draw a final product out of the supply chain network (*pull factors*), supply chain networks are often subject to push factors which are caused by the feeding of raw material into the supply chain by suppliers (*push factors*) [80]. In most cases, the supply of unfinished goods is not able to be synchronised with the demand generated at the other end of the supply chain, or an adjustment is delayed. Suppliers constantly, periodically or spontaneously, deliver raw material or components into the supply chain network. According to these factors, several traditional supply chain strategies exist[158]:

- In **push-based** strategies, production and distribution decisions are based on long-term forecasts using orders received from the retailer's warehouses. It is much harder for these strategies to react to changes in the marketplace.

- In **pull-based** strategies production and distribution are driven by true customer demand rather than forecast. With pull strategies a company does not have an inventory and produces goods purely in response to customer demand. Therefore, pull strategies become impractical when times between order and production and then delivery of goods (*lead time*) are long.

- **push-pull** strategies evolved to take advantage of both types of strategies mentioned above. In this type of strategy, some stages of the supply chain (normally first stages) are operated based on push strategies while the remaining use pull strategies for operation.

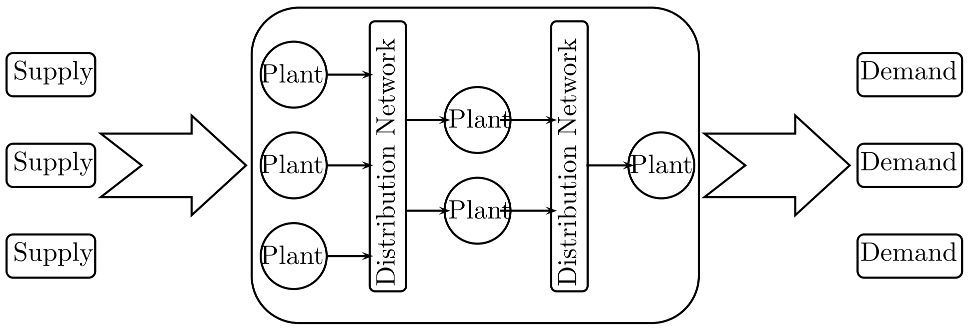


Figure 2.1: Schematic view of multi-echelon supply chain network

Reconciling supply and demand by determining the quantity of finished goods to produce becomes a labour-intense and time-consuming endeavour. It involves sourcing decisions when there are more than one internal or external suppliers of raw material, factory management decisions to define production plans and determine maintenance outages, and decisions on how to effectively distribute the finished goods within the supply chain and to the end customer. Generating an optimal plan which takes all the previous considerations into account becomes virtually impossible for human operators who are in most cases equipped only with spreadsheet tools.

In managing the supply chain thousands of individual decisions have to be made at regular intervals. These decisions are of different scope and significance (e.g., operational, tactical, strategic); there are also countless trade-offs between various decisions. Further, supply chain networks are extremely complex and they are set in constrained and dynamic environments, with many (usually conflicting) objectives. Moreover, most decisions should take *variability* into account, that is the degree to which a certain value can change, e.g. the truck may arrive ten minutes earlier/later or customer demand may often vary from the forecasted value. High demand variability may lead to the *bullwhip effect* – larger and larger swings in inventory as it progresses through the supply chain in response to changes in demand. Any successful implementation of an Advance Planning System for such a complex network should be capable of addressing several key issues; these include dealing with huge search spaces, many objectives, complex constraints and variability.

Due to the high level of complexity, it becomes virtually impossible for deterministic systems or human domain experts to find an optimal solution – not to mention that the term "optimal solution" loses its meaning in a multi-objective environment, as often we can talk only about trade-offs between different solutions. Moreover, the manual iteration and adjustment of scenarios (what-if scenarios and trade-off analysis), which is needed for strategic planning, becomes an expensive, possibly unaffordable, exercise.

Various *key performance indicators* (KPI) of the supply chain are usually measured in order to evaluate the quality of a particular activity in the supply chain e.g. revenue, number of orders *delivered in-full on-time* (DIFOT), and carbon emissions. Often in order to assess various KPIs of the decisions for a certain supply chain a *simulation* is run on a computer for the solution (decisions on how to run the supply chain), that is, an imitation of the behaviour of the supply chain.

Note that every time we simulate a supply chain or solve a problem we must realise that we are in reality only finding the solution or simulating only a *model* of the problem. All models are a simplification of the real world - otherwise they would be as complex and unwieldy as the natural setting itself. Thus, the process of problem solving consists of two separate general steps: (i) creating a model of the problem, and (ii) using that model to generate a solution [121]:

$$Problem \rightarrow Model \rightarrow Solution.$$

Note that the "solution" is only a solution in terms of the model. If our model has a high degree of fidelity, we can have more confidence that our solution and its simulation results will be meaningful. In contrast, if the model has too many unfulfilled assumptions and rough approximations, the solution may be meaningless, or worse.

Many texts on Advanced Planning and Supply Chain Management (e.g.,

[161]) describe several commercial software applications (e.g., AspenTech – aspenONE, i2 Technologies – i2 Six.Two, Oracle – JDEdwards EnterpriseOne Supply Chain Planning, SAP – SCM, and many others), which emerged mainly in the 1990s. However, it seems that the areas of supply chain management in general, and advanced planning in particular, are ready for a new genre of applications which are based on Computational Intelligence methods. Many supply chain related projects run at large corporations worldwide failed miserably (projects that span a few years and cost many millions). In [161] the authors wrote:

> In recent years since the peak of the e-business hype Supply Chain Management and especially Advanced Planning Systems were viewed more and more critically by industry firms, as many SCM projects failed or did not realise the promised business value.

The authors also identified three main reasons for such failures:

- the perception that the more you spend on IT (e.g., APS) the more value you will get from it,
- an inadequate alignment of the SCM concept with the supply chain strategy, and
- the organisational and managerial culture of industry firms.

While it is difficult to argue with the above points, it seems that a fourth (and unlisted) reason is the most important: maturity of technology. Small improvements and upgrades of systems created in 1990s do not any longer suffice for solving companies' problems in the 21st Century. A new approach is necessary, an approach which would combine seamlessly the forecasting, simulation and optimisation components in a new architecture. In this thesis we mainly focus on simulation and optimisation aspects. Further, many existing applications are not flexible enough in the sense that they cannot cope with any exceptions, i.e., it is very difficult, if not impossible, to include some problem-specific features; most businesses have some unique features which need to be included in the underlying model, and are not adequately captured by off-the-shelf standard applications. Thus the results are often not realistic and the team of operators return to their spreadsheets and whiteboards rather than relying on unrealistic recommendations of the software.

Several studies have investigated optimisation techniques for various supply chain components in isolation, including the job-shop scheduling problem [38, 96, 55], planning and cutting problems [105, 104], routing problems [168], allocation and distribution problems [44, 183], to name a few. Many algorithms were developed to solve various supply chain components in a single silo environment [114, 171, 182]. However, the optimisation of each silo without considering relationships with other silos usually does not lead to a globally optimal solution of the whole supply chain. Because of that, large businesses started to become more interested in optimisation of their whole system rather than the optimisation of single components of the system.

Large businesses typically breakdown their business into operational components such as purchasing, production and distribution. In the past, organisations have concentrated their efforts on these single operational components of the supply chain – *silos* (this term will be used interchangeably with the term *component* in this thesis). As optimisation of each individual silo in isolation may not lead to the global optimum, large businesses started to become more interested in optimisation of their whole system rather than optimisation of single components of the system. The first approach is commonly referred to in the literature as *coordinated supply chain management.*

It becomes an even more challenging task to address a *multi-echelon supply chain.* Recall that the term *echelon* refers to a group of silos which share similar functionality and constitute a conceptual stage of a supply chain. For example, all distribution units in the supply chain form one echelon, production plants form another one. Originally, the term *echelon* was introduced by Clark [41] instead of term "level" to avoid confusion with stock levels.

An example of a wine supply chain is presented in Figure 2.2. The dotted components represent external components to the company, for example dry goods suppliers, distribution companies, storages, etc. The components with applications screenshots represent the vertically integrated wine supply chain, all components of which belong to the given company. Material flow goes from left to right on this figure, whereas information flows from right to left. The vertically integrated supply chain on this figure is represented by five echelons: Maturity Models, Vintage Intake Planning, Crushing, Tank Farm and Bottling. These echelons will be described in detail later in Chapter 7.

The following section discusses the importance of optimal decisions from the global perspective in coordinated supply chain management.

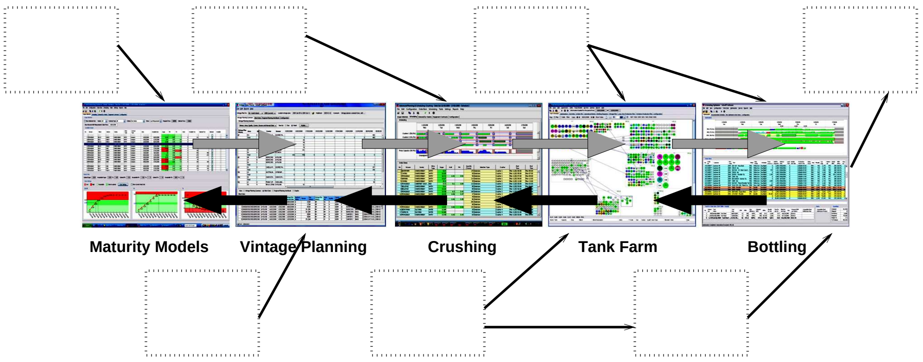


Figure 2.2: Wine Supply Chain. Grey arrows on this diagram represent a material flow. Black arrows represent information flow. The dotted components represent external components to the company

2.2 Global Optimisation

The general problem of global optimisation, in the *classical* sense, is to find $\vec{x}$ so as to

$$optimise f(\vec{x}), \vec{x} = (x_1, \dots, x_n) \in \mathcal{F} \subseteq \mathcal{S},$$

where $\mathcal{S}$ defines the search space and $\mathcal{F}$ defines *feasible region* which is bounded by the set of m constraints: $g_j(\vec{x}), \forall j = 1, \dots, m$.

That is, one is seeking to find an element in the search space $\mathcal{S}$ that optimises the objective function.

Usually, the search space $\mathcal{S}$ is quite simple: for numerical optimisation it is $\mathcal{S} \subseteq \mathbb{R}^n$; for combinatorial optimisation problems, $\mathcal{S}$ is discrete in nature, like a set of permutations, or a set of edges of a graph. An example of a combinatorial optimisation problem that might be encountered in a commercial setting is finding a permutation of n jobs to be executed on a machine that would minimise the amount of time spent re-configuring the machine between jobs (or from another perspective, maximises the amount of time the machine is in use).

In silo-focused optimisation, the search space $\mathcal{S}$ is typically built from a single type of variable. For example, as above, $\mathcal{S}$ may consist of all permutations of the set $\{1, 2, \dots, n\}$. Business global optimisation problems, on the other hand, typically have a more complex search space that consists of compositions of simpler search spaces. We will refer to these types as *single-silo* and *multi-silo* search spaces respectively.

Many aspects of commercial and industrial processes are amenable to optimisation, and this fact has been the source of many applications of modern heuristic methods. For example, the sequencing of jobs can be optimised in the classical job shop scheduling problem [38, 96, 55]. There are many algorithms that address different kinds of planning and cutting problems [105, 104], and many methods have been developed to solve different types of transportation [118], allocation [44, 183] and distribution problems. These are all examples of problems in which a globally optimal solution is sought for a specific, fairly narrow and focused problem. In real world business situations, however, it is often the case that the potential for optimising operations and profits can be increased by considering broader subsets of the business, and by looking at how major processes affect each other, in addition to the traditional approach of seeking independent opportunities for improvement. For example, in the case of job shop scheduling, the logistics of the provision of raw materials to the shop might be considered, as well as the details of post-production inventory shipping, marketing and retailing. From a *business* perspective, this is *global optimisation.*

Lawrence Davis, one of the experts in supply chain optimisation, describes this situation as follows [56]:

> . . . companies should be able to use solutions to optimise across their sourcing and procurement, production and distribution processes all at the same time. Instead, supply chain solutions tend to break those functions out into separate modules, each of which runs separately from the others. The answer you get when you first do production and then do distribution or any of these functions independently is not as good as the answer you get when you do them together.

Consider an example in which a company produces and distributes certain types of goods. Suppose that the company's factory is very efficient and, in a given month, produces an exceptionally large quantity of finished goods. From a local point of view this component (i.e. factory) of the supply-chain operation has performed excellently, but if we look at the broader picture of global production, if the excess goods are not readily distributed and consumed, the company will need to use storage space which will cost extra money, thus making this locally good production run sub-optimal, from a global point of view.

David Simchi-Levi characterised one of the objectives of supply chain management in the following way[158]:

> . . . the objective of supply chain management is to be efficient and cost effective *across the entire system*; total system wide costs , from transportation and distribution to inventories of raw materials , work in process and finished goods , are to be minimised . Thus, the emphasisis not on simply minimising transportation cost or reducing inventories , but rather on taking a systems approach to supply chain management.

It is possible to take the idea of global optimisation from a business perspective to increasingly greater heights by including into consideration more and more factors that affect a business' profitability. However, practical limits are often encountered when the scope of the optimisation problem begins to encroach on boundaries of another business entity's operations. For example, it may be that the provision of raw materials to a job shop is a matter handled by a different business entity (i.e. component) which may not be willing to divulge minute details of its operations.

Some typical characteristics of a problem that suggest that there might be an opportunity for global optimisation from a business perspective are as listed below.

- *Dependence or partial dependence among components (silos).* Problem specific constraints and business rules tie single silos together making dependency connections between them. It is noted that constraints here can be *intracomponent constraints*, which exist within a single component itself, and *intercomponent constraints*, that form the dependencies between the silos. Intracomponent constraints are the classic type of constraints encountered in many optimisation problems. Intercomponent constraints have a slightly different purpose in global optimisation compare to the classic intracomponent constraints.

- *Multiple objectives.* Having many single operational components in the problem with each component having its own or even several objectives yields the multiple objective optimisation problem. However, this multiobjective optimisation problem stands at a higher level compared to the classic multiobjective optimisation. Here an objective of each component of the problem may have its own subobjectives.

- *Need for understanding what-if scenarios.* Businesses and their competitors are rarely static; indeed, they are dynamic and business environment is constantly changing. Thus, companies want to respond to these changes by making more intelligent decisions and changes to their operations. There is a great need for businesses to foresee how a particular change or decision will impact their priorities, profit and other objectives. The complex multi-silo system with a large network of dependencies is a lot harder to understand and predict than a single component one.

When we make the distinction between local and global optimisation from a business perspective, the idea of global optimisation in the classical sense still holds in both cases. If we try to optimise over the entire supply chain, then in terms of the objective function being considered, we still seek a classical global optimum. On the other hand, if we restrict our attention to optimising the operations of a single factory, then this is local optimisation with respect to the broader picture of the supply chain; nevertheless we also seek a classical global optimum for the objective function associated with the optimisation of the factory. The distinction between local and global optimisation from a business perspective lies in the differences between their respective search spaces.

Although the field of supply chain management emerged recently, the idea of coordinated planning had already been proposed in 1960 by Clark and Scarf [42]. These authors investigated multi-echelon inventory problem based on sequential and tree-like models, and they approached the problem using a recursive decomposition method. Vidal and Goetschalckx [172] provide a review of strategic production-distribution systems with global integration. The authors stress that very little research exists that addresses optimisation of the whole supply chain rather than its single components. They mainly concentrate on mixed integer programming models. Capar and Ulengin in [34] classify all SCM literature by introducing special taxonomy.

The review by Thomas and Griffin [165] defines three categories of operational coordination: Buyer-Vendor coordination, Production-Distribution coordination and Inventory-Distribution coordination. The model described later in this chapter belongs to the second category. For the study of production-distribution coordination category, several researchers approached it with dynamic programming heuristics, mixed integer programming and Markov chains. The authors underline the complexity of problems of this category and discuss strategic planning supply chain models, the majority of which are mixed integer programming based models.

Aikens [4] presents a problem of the optimal location of warehouses starting with simple a uncapacitated single-commodity case with zero echelons and ending with more complex capacitated multicommodity and multi-echelon facility location models.

In one of the first papers on production-distribution coordination [77] the author considers several models with interacting silos and stochastic demands and discusses an analytical approach to find optimum inventory levels. Production orders are processed in weekly batches.

Pyke and Cohen [146] consider a single-product, three silo supply chain that includes a factory, a finished goods stockpile and a distribution centre. The whole model is demand driven and based on the Markov chain. The finished goods stockpile in this model is used more like a buffer between production and distribution silos. The authors present near-optimal algorithms to determine batch size, normal reorder point, and expedite reorder point. The companion paper [147] upgrades their system to support multi-product situations.

Chandra and Fisher [36] developed a two component demand driven model where products are first produced at the production plant and then distributed by the a fleet of trucks to a number of retail outlets (also known as a vehicle routing problem). Then the authors compare two approaches: in the first one they solve the production and vehicle routing problems separately, and then together combined under a single optimisation algorithm. The production planning problem is solved optimally and the vehicle routing problem is solved using heuristics. Their results show an the advantage of using the second approach. However, the authors use relatively small datasets for their problem.

Multi-echelon inventory optimisation systems have been studied by various researchers. Caggiano et al. [32] uses a greedy algorithm for his multi-item, multi-echelon service parts distribution system, [33] solves his multi-item, two-echelon problem with Lagrangian decomposition, [176] approaches the same problem with a greedy algorithm. Song et al. in [160] use metamodel to study the behaviour of the multi-echelon supply chain system.

Altiparmak et al. [8] propose algorithms using a mixed-integer, non-linear programming model for multi-objective optimisation of a supply chain network based on the real world problem of a company that produces plastic products. They compare three approaches to find the set of pareto-optimal solutions and discuss the pros and cons of each.

There are additional studies that address supply chain problems with the use of evolutionary algorithms; these are discussed in section 2.3.

2.3 Evolutionary Algorithms

This section provides background information on Computational Intelligence, ways to model the problem and evolutionary algorithms.

There are two main branches that tackle the optimisation of complex problems: operations research and computational intelligence. Operations research uses techniques such as linear programming, branch and bound, and dynamic programming. In this chapter our methods, however, will be based on the methods of Computational Intelligence and in particular Evolutionary Algorithms.

Computational Intelligence is considered to be an alternative to classical optimisation and it relies on heuristic algorithms (such as in fuzzy systems, neural networks and evolutionary computation). In addition, Computational Intelligence also embraces techniques such as swarm intelligence, neural networks, fractals and chaos theory, and artificial immune systems. Computational Intelligence techniques often combine elements of learning, adaptation, evolution and fuzzy logic to create programs that are, in some sense, intelligent.

An interesting question, which is raised from time to time, is concerned with guidance on the types of problems for which Computational Intelligence methods are more appropriate than standard operation research methods. From our perspective, the best answer to this question is given in a single word: complexity. Real-world problems are usually difficult to solve for several reasons:

- the number of possible solutions is so large as to forbid an exhaustive search for the best answer.

- the evaluation function that describes the quality of any proposed solution is noisy or varies with time, thereby requiring not just a single solution but an entire series of solutions.

- the possible solutions are so heavily constrained that constructing even one feasible answer is difficult, let alone searching for an optimum solution.

Naturally, this list could be extended to include many other possible obstacles. For example, we could include noise associated with our observations and measurements, uncertainly about given information, and the difficulties posed by problems that have multiple and possibly conflicting objectives (which may require a set of solutions rather than a single solution). All these reasons are just various aspects of the complexity of the problem.

When solving a problem, in reality we are finding a solution to a model of the problem as a simplification of a real-world. In solving such real-world problems there are at least two ways to proceed [121].

1. We can try to simplify the model so that traditional methods might return better answers, or
2. We can keep the model with all its complexities, and use non-traditional approaches, to find a near-optimum solution.

In other words, the more complex the problem (e.g., size of the search space, evaluation function, noise, constraints), the more appropriate it is to use a non-traditional method, e.g. the Computational Intelligence method. Further, it is difficult to obtain a precise solution to a problem because we either have to approximate a model or approximate the solution. A large volume of experimental evidence shows that the latter approach can often be used to practical advantage: many Computational Intelligence methods have already been incorporated into software applications that handle levels of supply chain complexity that are unapproachable by traditional methods (such as linear programming, dynamic programming, and branch and bound).

2.3.1 Structure of the Evolutionary Algorithm

Evolutionary Algorithms (EA) are a subclass of Computational Intelligence, a generic population-based metaheuristics, inspired by the principles of biological evolution. They are built on abstractions of genetic inheritance and the Darwinian principle of survival of the fittest, mimicking the smartest system – that of nature. In contrast with methods based on OR, these methods evolve the solution instead of constructing it step by step, i.e. at any point of time the algorithm has at least one complete solution (anytime response).

One of the best examples for illustrating how Evolutionary Algorithms work is based on the population of rabbits from [118]:

> ...at any given time there is a population of rabbits. Some of them are faster and smarter than other rabbits. These faster, smarter rabbits are less likely to be eaten by foxes, and therefore more of them survive to do what rabbits do best: make more rabbits. Of course, some of them slower, dumber rabbits will survive just because they are lucky. This surviving population of rabbits starts breeding.

> The breeding results in a good mixture of rabbit genetic material: some slow rabbits breed with fast rabbits, some fast with fast, some smart rabbits with dumb rabbits, and so on. And on the top of that, nature throws in a 'wild hare' every once in a while by mutating some of the rabbit genetic material. The resulting baby rabbits will (on average) be faster and smarter than these in the original population because more faster, smarter parents survived the foxes. (It is a good thing that the foxes are undergoing similar process – otherwise the rabbits might become too fast and smart for the foxes to catch any of them).

Algorithm 1: The Structure of Generic Evolutionary Algorithm

```
$generation \leftarrow 0$
initialise_population
evaluate_population
**while** not $termination\_condition$ **do**
    $generation \leftarrow generation + 1$
    select_individuals_for_reproduction
    apply_variation_operators
    evaluate_newborn_offspring
    select_individuals_for_survival
**end while**
```

Let E be a metric search space and F is an *objective function* that maps $E \rightarrow \mathbb{R}$; the objective is to maximise F on E $max_{x \in E} F(x)$. Evolutionary Algorithms maintain a set of solutions for each of the iterations. As these algorithms are inspired by nature, most of the terminology utilises biological metaphors: each solution is referred to as an *individual*, a set of solutions – a *population* and each iteration in which a population is updated is called a *generation*. The raw data structure behind the individual is called a *genotype* that consists of separate *genes*; the decoded data from the genotype used for evaluation is called a *phenotype*. Most Evolutionary Algorithms can be described by Algorithm 1.

The first step of the EA is *population initialisation*. The population is usually initialised randomly, but some smart initialisation procedure that biases individuals into some good regions may significantly improve the results of the optimiser. The diversity of the population is of paramount importance for the optimisation results as it lets the optimiser explore different regions of the search space. After the initialisation, each individual in the population is evaluated using the *fitness function* which returns the *fitness* of an individual i.e the quality of the solution.

After the initialisation part the main loop iterates through generations terminated by a *termination_condition*. The basic termination condition is based on the limit of number of generations to evolve. However, more sophisticated conditions are possible; for example, measuring the diversity of population or the progress of the best individual in the population.

The next step of the algorithm is to select parent individuals for breeding, taking fitness into account – stronger individuals have more chances to produce offspring for the next generation (survival of the fittest). Selection in the Evolutionary Algorithms also occurs during the survival selection where it is responsible for selecting individuals for the next generation. The two most commonly used selection methods are roulette selection and tournament selection.

Evolutionary operators are responsible for constructing new offspring based on the genetic material of parent individuals. These operators are always stochastic and executed according to the algorithm specific probabilities. Two common operators are:

- **recombination**: involves several parents (typically two) that exchange genetic material to produce offsprings.
- **mutation**: involves one parent that makes one or more changes to its genotype to produce an offspring.

The last part of the algorithm is the selection of individuals for survival. The *generational* strategy fully replaces all individuals in the current generation with new offspring. If new offspring replace only a few individuals in the current population then the survival strategy is called *steady-state*.

2.3.2 Variants of Evolutionary Algorithms

This subsection provides an overview of some common classes of evolutionary algorithms.

Evolution Strategies

One of the earliest Evolutionary Algorithms developed was Evolution Strategies introduced by Ingo Rechenberg in early 1960's and further developed by

Rechenberg [150] and Schwefel [156]. The original algorithm was involved with a single individual with real-valued vector encoding. Each of the real variables is changed by Gaussian mutation with a mean of zero and standard deviation σ. The mutated individual is accepted if it has better fitness than its parent and does not violate any constraints.

A single individual ES evolved into $(\mu + \lambda)$ and (μ, λ) forms. Here μ represents a number of solutions in the parent generation and λ a number of solutions generated from the parent generation. In the first form $(\mu + \lambda)$ the new population of μ individuals is selected from $\mu + \lambda$ individuals (new offspring plus parent population). In the second form μ individuals of initial population produce λ offspring $(\lambda > \mu)$ and then the new population of μ individuals is formed only from offspring. This produces better results on problems where the optimum is changing over time.

Genetic Algorithms

One of the most widely known types of Evolutionary Algorithms is the Genetic Algorithm (GA). The initial genetic algorithm introduced by John Holland, works on the binary genotype space. The phenotype of the algorithm can be an arbitrary space that can be coded into the bitstring. The probability of selection of a certain individual as a parent, is proportional to its fitness (for example, roulette selection). Bitstring crossover and mutation are used. During the survival phase all individuals in the population are replaced by newly created offspring.

Later the research community realised that the bitstring genotype is not universal and can be practically any space as long as genetic operators are defined to execute on this genotype [118]. Other additions to the "classical" genetic algorithm were tournament selection and elitism (the best individual is preserved to the next generation).

Evolutionary Programming

Evolutionary Programming was introduced by Lawrence J. Fogel [65]. They aimed to predict changes in an environment that consisted of sequences of symbols, with the algorithm producing the next symbol of the sequence. The evaluation was based on the payoff function that measured the accuracy of the prediction. Fogel used finite state machines as underlying chromosome representation. A population of finite state machines was maintained and evolved.

Evaluation of each state machine was based on running it against a sequence of symbols and averaging the accuracy of the number of predictions.

Genetic Programming

One of the earliest applications of genetic programming was created by Lawrence J. Fogel who applied it to the problem of discovering finite-state automata in 1964. Later it has been applied to the learning classifier systems and development of sets of sparse rules describing optimal policies for Markov decision processes. The first genetic programming approach that utilises procedural languages organised in tree-based structures was described by Nichael L. Cramer in 1985 [50]. Genetic Programming has been popularised by John Koza [93]. The idea is that the program should evolve itself during the evaluation process. The search space is a space of possible computer programs and the solution representation is a hierarchically structured computer program which is basically a tree. To evaluate a solution, a program-solution is run against a selected set of test cases and the fitness function is a sum of distances between the correct and produced results.

Evolutionary Algorithms with Multiple Populations

It is also possible to use more than one population in the run of algorithm, with each evolving by its own rules.

Island model evolutionary algorithms exchange individuals between the subpopulations throughout the optimisation to help maintain diversity, and to spread to new, potentially good areas of the search space. Subpopulation islands have a defined topology, so sending an individual is possible only between certain islands.

Another type of multi-population Evolutionary Algorithm is the *cooperative coevolution.* In coevolutionary algorithms, the subpopulations are called *species*; this approach borrowed from biology, where different species interact in a common ecosystem. Two main kinds of coevolution exist: *competitive* and *coevolutionary.* In the former, individuals from different species try to compete with each other (the arms race concept), whereas in the second type individuals from different species try to achieve a common goal of evolving together. Experiments in the current thesis mainly involve cooperative coevolution. Section 2.6 describes coevolution in more detail and gives a brief literature review.

Other classes of algorithms that belong to Evolutionary Algorithms include

Evolutionary Programming, Genetic Programming, Gene Expression Programming and Differential Evolution.

Although there are differences between these evolutionary algorithms, they are all share many commonalities. Many of them just use different terminology for the same thing. All of them are based on fundamental principles of Darwinian evolution; they maintain a population of solutions (in some cases population size is one); each of the individuals of the population have finite lifetime and stronger individuals produce offspring. In a broad sense, evolutionary algorithms provide a general framework for the way in which to approach complex problems.

In recent years, there has been an increased interest in solving supply chain management problems using evolutionary algorithms. David Naso et al. [125] look at the problem of coordination of just-in-time production and transportation in a network of partially independent facilities of ready-mixed concrete. They optimise the network of independent and distributed production centres serving a number of customers distributed across a certain geographical area. This problem, which has high complexity and strict time delivery constraints, is approached with a meta-heuristic based on a hybrid genetic algorithm with combined constructive heuristics.

Zhou et al. [182] present a novel genetic algorithm to solve bi-criteria, multiple warehouse allocation problem. The proposed method finds the pareto-front of a wide range of non-dominated solutions without the arbitrary determination of weighting coefficients.

Vergara et al. [171] developed an evolutionary algorithm for dealing with the coordination of supply chain flows between different members of the chain. The authors of that paper recognise the importance of an overarching algorithm that optimises the whole system. Their work looks at the flow of materials in terms of supply and demand, but does not consider the intricacies of production within each business silo.

Researchers had been working much earlier on supply chain problems as can be seen in the following references. Lee and Choi [101] apply genetic algorithms to solve a single machine scheduling problem with distinct dates, and they attempt to minimise all penalties. This method produces near optimal solutions, which they prove by comparison with an exact algorithm. Lee et al. [102] address the problem of inventory management of a refinery that imports several types of crude oil and proposes a mixed-integer linear programming model. Martin et al. [114] create a linear programming model to optimise flat glass production.

2.4 Scheduling

Scheduling plays a fundamental role in many industries especially in manufacturing and service. It becomes even more crucial in the current competitive environment, where it is almost impossible for human domain experts to take into account all the information about deadlines, millions of customers and business processes. Failure to meet customer expectations may result in significant financial and credibility loss.

Many different types of scheduling problem exist. Some of the most popular variations are listed below.

- **Job Shop** (JSSP). In this version of the scheduling problem there are m machines and each job has a set of operations that need to be executed on these machines in a certain order. JSSP is used in several experiments in the current thesis.
- **Flow Shop**. Each of the jobs has to be processed on each machine following the same route, i.e M_1, $M_2, \ldots$ and so on.
- **Flexible Flow Shop**. This is a generalisation of the previous problem where there are c stages in series with a number of identical machines in parallel at each stage . Each job should be executed on the specific order of stages, i.e stage 1, stage 2, etc.
- **Open Shop**. As with previous problems each job has to be executed on each of the machines, but some execution time may be zero. There are no restrictions enforced on the routing of each job through machines.
- Many other variations of scheduling problems include the addition of various constraints, such as deadlines, release dates, precedence constraints, batch processing and machine breakdowns.

The job shop scheduling problem is a canonical example of a scheduling problem. It is one of the most difficult combinatorial optimisation problems, and as is the case with most scheduling problems, it is NP-complete [68]. A very good survey of job-shop scheduling problems and different solution representations is given in [38]. Davis [55] discusses the application of genetic algorithms to the job-shop scheduling problem. Wang and Zheng solve it using a modified genetic algorithm [174]. In their work, to avoid premature convergence and improve neighbourhood search, the classical mutation operator is replaced by the metropolis sample process of simulated annealing with a probabilistic jumping property. A special crossover operator is used in the simulate annealing-based main loop. The simulated annealing approach to tackle job-shop scheduling problem is discussed by Laarohoven [96].

The job shop scheduling problem mathematically is formulated as follows. There is a set of jobs $J = \{J_1, \ldots, J_n\}$ and a set of machines $M = \{M_1, \ldots, M_n\}$ which these jobs can be executed on. Each job has a set of operations O and a specific order of execution on each of the existing machines. Each operation i of the job J_j on machine M_k takes a certain processing time and is denoted by $P_{i,j,k}$. Two operations cannot be executed on a machine at overlapping intervals of time. The goal is to find an ordering of these jobs on the machines that minimises the total time needed to execute all the jobs.

As it is one of the most popular NP-hard problems, many evolutionary algorithms have been proposed for its solution. Yamada and Reeves [178] propose an evolutionary algorithm to solve the permutation flow shop problem. They combine stochastic sampling and best descent methods into one unified method to reach effective results. They use a multi-step crossover and mutation as well as a representative neighborhood method, which is constructed by clustering the original neighbourhood and choosing the best representative from each cluster. This method is described by Nowicki and Smutnicki in their taboo search algorithm for the flow shop problem [129].

There are also many published reports of the application of evolutionary algorithms to solve scheduling problems in various domains of application.

Marchiori and Steenbeek [113] developed an evolutionary algorithm for the real-world airline crew scheduling problem. Results of the real-world benchmark instances are compared with the results produced by the commercial systems and produce effective, competitive results.

Ponnambalam and Reddy [136] developed a multiobjective hybrid search algorithm for lot sizing and sequencing in flow-line scheduling. The main idea is to use the memetic type of algorithm that combines the genetic algorithm with a local search procedure.

A practical problem of optimal oil production planning was discussed by Ray and Sarker [149]. Production is based on several oil wells, from which a crude oil is extracted by means of the injection of a high pressure gas. The goal of the problem was to optimise the amount of gas needed to be used in each of wells to maximise output of oil, taking into account the limited amount of gas each day. Single objective and multiple objective versions of the problems are considered.

Burke and Smith [26] investigated a real-world problem that addressed the maintenance problem of a British regional electricity grid. They compare the performance of a proposed memetic algorithm with other methods.

Martinelli [115] showed the optimisation of time-varying objective functions using a stochastic comparison algorithm. Values of the time-varying functions are known through estimates. Noise filtering is introduced to decrease the probability of wrong moves.

Tinós in [166] introduced a self-organising random immigrants scheme for algorithms in dynamic environments. Newly immigrated individuals are held in the subpopulation for some time until they develop good fitness values.

Yang in his chapter [179] discussed a memory scheme approach as a method of improving performance of evolutionary algorithms for dynamic environments. Two kinds of memory schemes are described: direct and associative. These schemes are applied to genetic algorithms and to univariate marginal distribution algorithms in dynamic environments. These algorithms are run on a set of generated tests: cyclic, random and with addition of noise.

Lutz [155] considered the application of evolutionary strategies for numerical optimisation problems in dynamic environment. The main parameters of evolutionary strategies for problems in dynamic environments are presented. Performance measures are discussed, with their advantages and disadvantages.

Several researchers considered scheduling problems in the mining industry. Atashkin [9] looked at the problem of distribution of the loads on the faces and of choosing the most effective technique and conditions for each face. The problem is based on the mining of coal and the objective is to optimise the cost. He proposes an algorithm based on passive stochastic programming.

In the last few years scheduling problems have been successfully solved by the genetic programming approaches [90, 89, 12, 127, 128].

2.5 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is the combinatorial problem of finding a set of delivery routes from the depot (or several depots for the multi-depot version) to customers, using a fleet of trucks and satisfying a number of constraints. Since it was first proposed by Dantzig and Ramser [52] five decades ago, it has become one of the most famous combinatorial problems in operation research. It was initially known as the Truck Dispatching Problem. The authors described the problem of distributing gasoline between the terminal and the service stations with a fleet of identical trucks. Many distribution management and logistics problems can be modelled using the Vehicle Routing Problem.

The Vehicle Routing Problem is defined as follows. Let $G = (V, E)$ is a graph where $V = \{v_0, \ldots, v_n\}$ is a set of customer locations, and $E = \{e_1, \ldots, e_k\}$ is the set of routes between the locations. For every route (i, j), where $i \neq j$, there is a associated cost c_{ij}, which can be interpreted as travel cost or travel time. There is as well a fleet of m trucks, where $m_L \leq m \leq m_H$, which are based at the location v_0 called *depot* (in some variations of the problem there maybe more than one depot). Here, m_L and m_H define lower and upper truck number limits respectively. In the case $m_L = m_H$ the fleet is said to be *fixed*. When the fleet is not fixed each truck has a fixed cost. The problem is to minimise the cost of routes and trucks in such a way that:

- each city except for depot v_0 is visited only once and by only one vehicle;
- each route starts and ends at the depot;
- additional capacity limitation constraints are satisfied. Depending on variations of the problem these constraints may include vehicle capacity constraints, route length constraints, time windows that customer needs to be visited, and precedence constraints between customers.

Some of the most popular variations of the Vehicle Routing Problem listed below.

- **Capacitated Vehicle Routing Problem (CVRP)**. This is a basic version of the problem where all trucks are identical, there is a single depot and all the demands are deterministic and known in advance. The only constraint imposed is a vehicle capacity limitation. If the problem is constrained by the total distance of the route it is called a distance-constrained *Vehicle Routing Problem.*
- **Vehicle Routing Problem with time windows**. This variation is an extension of the previous problem with the additional constraint that each customer v_i should be visited only during an interval $[a_i, b_i]$ called a

time window. In case of the early arrival of the truck, it may wait at the customer location.

- **Vehicle Routing Problem with backhauls**. This problem is also an extension of the CVRP where two types of customers exist: *linehaul customers* that require a certain quantity of product to be delivered, and *backhaul customers* that require a given quantity of product to be picked up. All linehaul customers need to be visited before any backhaul customer. Additionally, this problem can be further complicated with the inclusion of time windows.
- **Vehicle Routing Problem with pickup and delivery**. In this version of the problem each customer has a quantity of product to pickup and to deliver. At each location it is assumed that the delivery takes place before pickup and at no point can the time total load of each truck be negative. As with the previous variant of the problem this problem can be enhanced with time windows.
- **Multi-depot Vehicle Routing Problem**. In this case the problem is extended to determine the routes for several trucks from more than one depot to transport the product to customers and then return to the same depot.

The Vehicle Routing Problem is a combination of the *Travelling Salesman Problem* and the *Bin Packing Problem* which are both known to be NP-hard. Thus, the exact deterministic algorithms for the Vehicle Routing Problem are only practical for small instances of the problem. Exact algorithms for the VRP are mostly dominated by branch-and-bound [97, 63, 98] branch-and-cut methods [10, 16] and integer programming [14].

An exact algorithm for the Capacitated Vehicle Routing problem using a set partitioning formulation was proposed by Baldacci et al. [13]. Azi et al. [11] described an algorithm for the VRP with time windows where one truck may perform more than route. One of the best exact methods was created by Fukasawa et al. [66] where authors hybridised branch-and-cut with the q-routes approach.

For the big instances of the VRP, heuristics and metaheuristics based approaches are more practical. There are a few heuristics that have been popular mainly due to their simplicity and speed. The Clarke and Wright [43] savings based constructive heuristic uses simple merging rules to construct the routes.

Sequential and parallel versions exist for this algorithm. It is used in several algorithms presented in this thesis. The sweep mechanism of Gillett and Miller [69], cluster-first route-second methods of Fisher and Jaikumar [64], and route-first cluster-second methods by Beasley [18] are some other popular heuristics which are often used for the population initialisation in population based meta-heuristics.

Simulated Annealing and Tabu Search are one of the most popular and most successful metaheuristics to solve the VRP. The unified tabu search algorithm by Cordeau [45, 46] can be used for multi-depot and time window constrained VRPs. It searches neighbourhood of solution by moving a vertex from its route between two of its closest neighbours in another route.

Ergun et al. [62] uses a very large neighbourhood search to address the VRP. At every iteration it searches a very large neighbourhood of solutions operating on several routes simultaneously.

Many metaheuristics based on genetic algorithms were created as well. We will be using Prins' [145] algorithm in several experiments throughout the thesis. It uses a local search technique to produce solutions, therefore it is classified as a *memetic* algorithm. Another memetic algorithm was described by Berger and Barkaoui [19] which uses two population with migration component in between them.

Bouthillier and Crainic [100] in their paper describe a parallel asynchronously cooperative multi-search algorithm for the VRP. The threads exchange information through a special module which manages a pool of solutions. Each process can be represented by a different meta-heuristic, such as evolutionary algorithms and tabu search.

Alba and Dorronsoro [7] provide a solution to the VRP using a cellular genetic algorithm. By adding local search and small overlapping neighbourhoods the authors preserve population diversity and produce a good quality solutions. In this algorithm the population is presented as a 2D toroidal grid and it uses edge recombination and simple mutation operators to evolve new solutions.1

Reimann proposes savings algorithm based on ant colony optimisation [151]. A good survey on the latest VRP algorithms can be found in [47].

2.6 Coevolution

Coevolution is a simultaneous evolution of several genetically isolated subpopulations of individuals that exist in a common ecosystem. Each subpopulation is called species and mate only within its species. In evolutionary computation coevolution can be of two types: competitive and cooperative.

In *competitive* coevolution, multiple species coevolve separately in such a way that fitness of individual from one species is assigned based on how well it competes against individuals from the other species. One of the early examples of competitive coevolution is the work by Hillis [79], where he applied a competitive predator-prey model to the evolution of sorting networks. Rosin and Belew [153] used a competitive model of coevolution to solve a number of game-learning problems including Tic-Tac-Toe, Nim and a small version of Go.

Cooperative coevolution uses a divide and conquer strategy: all parts of the problem evolve separately, and the fitness of an individual of a particular species is assigned based on the degree of collaboration with individuals of other species. Individuals of each species evolve to contribute to their part within the single ecosystem. Two different models of cooperative coevolution have been proposed: the model by Potter [141] and the model by Moriarty and Miikkulainen [123]. This thesis mainly focuses on the Potter model, which is a more general model compared to the model proposed by Husbands and Mill [86]. Also, a somewhat similar model of evolution was proposed previously by Holland [83]. Potter applied cooperative coevolution artificial neural networks [143] to concept learning for the construction of immune systems [144], and the optimisation of some standard functions [142].

Wiegand analysed cooperative coevolution on several pseudo-boolean functions, showing the relationship between cooperative coevolution and symmetric games. He proposed the hierarchical categorisation of design choices for coevolutionary algorithms, performance effects on update timing, and analysis methods, using evolutionary game theory and dynamic systems [175].

Popovici in her work used dynamical systems theory to analyse cooperative coevolutionary evolutionary algorithms (CCEAs). She looked at the dynamics of individuals using fitness landscapes, explained the problem of convergence to the Nash Equilibrium with best response curves [139], analysed sequential and parallel CCEAs [137], and investigated relationships between internal and external metrics [140]. She used two dimensional fitness landscapes with the parameter α to vary landscape surface in most of her work.

Parameter setting issues of cooperative coevolution were explored by Luke et al. [107], investigating how to choose collaboration methods, the number of populations and number of trials per fitness evaluation. Some of their recommendations were to use parallel update timing over sequential, more subpopulations for strongly linked problems, and preferred use of shuffling methods.

A recent article by Popovici et al. [138] presented a wide survey of theoretical and empirical issues in cooperative coevolution.

2.7 Summary

There exists a large body of research on the application of optimisation techniques to highly-focused aspects of real-world business problems. This research may be suitable for small businesses seeking to optimise their production efficiency and profit. However for bigger businesses, especially ones that are vertically integrated, optimisation needs to be examined from a much broader point of view.

Optimisation of any of that business' silos would definitely lead to an improvement in its operations in terms of time and money, and separate efforts to optimise multiple silos would also yield benefits. But businesses should be more interested in optimisation of their whole system rather than optimisation of single components. A larger opportunity exists for *global optimisation from a business perspective.* By modelling the situation in such a way as to have limited communications between those parts of the model representing the various silos, we can achieve a level of optimisation that is better than that attainable by isolating and optimising individual components.

From an abstract point of view, business-perspective global optimisation fits into the framework of traditional global optimisation. What distinguishes it as a subclass from traditional business optimisation scenarios is that its problems can be formulated as objective functions with rather more complex-structured search spaces. For example, whereas a traditional problem might have the set of permutation of a finite number of objects as its search space, a business-perspective global optimisation problem might instead have as its search space the vector space of n elements, each of which is taken from the space of permutations of a finite number of objects.

As several experimental studies in this thesis focus on job shop scheduling and vehicle routing problems, their basic concepts and a relevant literature review was discussed in this chapter.

Chapter 3

Single Silo Model

The dynamic nature of real-world problems is a key differentiator with respect to more artificially formulated problems. It creates challenging variations on what might otherwise be considered straightforward optimisation problems from an evolutionary algorithms perspective. In this chapter we divide dynamic problems into three categories based on the characteristic that varies with time: the objective function, the input variables, or the constraints. A review of the literature with regard to the way in which real-world problems in each of these categories have been addressed shows that there is a need to investigate time-varying problems of the third kind, that is, problems in which the constraints are dynamic. While time-varying objective functions are an important direction of research, in most real-world problems the objective function stays the same (i.e. minimisation of cost or maximisation of profit). However, the dynamic environment from the business perspective usually implies constraints that change over time. This appears to be another gap between theory and practice and in this chapter we endeavour to shorten this gap by looking into this issue using a problem from the wine industry.

In this chapter we present methods for dealing with time-varying constraints within a single silo. We show how each challenge can be dealt with by making adjustments to genotypic representation, phenotypic decoding, or to the evaluation function itself. The ideas presented are exemplified by means of a single silo case study of a real-world commercial problem, namely that of bottling wine in a mass-production environment. Addressing the issues raised by this problem is critical for a successful solution, especially when the evolutionary algorithm being designed is for the purpose of driving a software application that must be robust enough to meet the demands and expectations of everyday use by application domain specialists who are not necessarily optimisation experts. The methods described in this chapter have the benefit of having been proven by fully-fledged implementation into a software application. The application undergoes continual and vigorous use in a complex production environment in which time-varying constraints arising in multiple different combinations are routinely present.

3.1 Introduction

The application of evolutionary algorithms to real-world problems brings us face to face with some challenging issues related to the time-varying nature of such problems. These issues are over and above the fundamental problems being solved. In fact, they can be so critical from the standpoint of creating a robust and truly usable software application that taking them into consideration significantly alters the approach taken to solve the problem.

The issues in question arise because of the dynamic nature of real-world problems. One hardly ever encounters a situation that is static for any significant period of time. In everyday business life, things change almost constantly. The number of orders to be scheduled in a factory changes, the run rate of a machine varies due to environmental factors, the quantity of a particular item ordered is modified after it has already been allocated, machines break down, and delivery trucks arrive late.

Take scheduling problems as an example. Many factories deal with these kinds of problems constantly. From a pure-problem perspective, evolutionary algorithms have been used for quite a long time to solve scheduling problems [55, 96, 174]. A general scheduling problem requires that we make decisions on how to assign a number of tasks to a fixed set of machines able to perform those tasks. The assignment must be done in such a way as to minimise the amount of time required to complete all of the tasks (referred to as the makespan). There are many variations to this problem; for example in some cases, the processing of an object requires that it be processed in pre-defined phases, each possibly taking place on a different machine. All of the variants however are equivalent to the NP-Complete problem of *job shop scheduling* [68]. This of course makes it virtually unsolvable in a reasonable amount of time using currently known techniques, and so the usual approach is to look for an approximate solution using evolutionary algorithms. The essence of a straightforward evolutionary approach is to represent a candidate solution as an ordered list of (machine, job) pairs, with the natural order of the list being representative of the sequence of job execution. Using a well-constructed fitness function that embodies the problem constraints and parameters, and perturbing the representation using simple operators will usually guide the population to a good solution.

This kind of approach is quickly seen to be insufficient when it comes to creating a fully-fledged software application that is to be used in a real-world business environment to manage the day-to-day scheduling needs of a large enterprise. There are many issues that intrude on the purity of a simple approach such as the one described above. However these issues are usually critical and require careful consideration so that the business needs can be satisfied in a manner compatible with the architecture of the evolutionary algorithm.

This chapter continues with a classification of time-varying issues into three categories, alongside a brief literature review of work in each category. An industrial wine bottling problem is then elaborated in sufficient detail to enable the reader to more easily visualise the kinds of problems being addressed by this work. We go on to describe the real-world business issues that had to be considered and resolved in order to build a software solution around an evolutionary algorithm core. The design and construction of the EA itself is fully specified and we illustrate how the various time-varying issues were resolved in relation to the EA components.

3.2 Dynamic Optimisation Problems

The dynamic nature of real-world optimisation problems is well known, but a closer examination reveals a few different aspects of the problem that can be described as *dynamic*. In this section we introduce a classification of dynamic optimisation problems into three categories, each of which is characterised by:

1. time-varying objective functions.
2. time-varying input variables.
3. time-varying constraints.

There is a large body of EA research literature that addresses the dynamic property of such optimisation problems, and we present a brief review of work undertaken in each of the areas. It will be noticed that while there is an abundance of work on problems fitting into the categories of time-varying objective functions and time-varying input variables, there are relatively few published reports dealing with dynamic optimisation problems of the third kind, that is problems that deal with time-varying constraints.

3.2.1 Time-varying objective functions

These are problems in which the shape of the fitness landscape varies with time. The location of the optimum value is continually shifting. The literature shows that there has been a considerable amount of research done on this kind of problem from the early days of evolutionary methods, to the present [24, 23, 124, 35, 180]. Some are based on artificially constructed objective functions and others on real-world applications. In many published reports, situations were set up wherein during a single run of an evolutionary algorithm, the location of the optimum would move; researchers were therefore particularly interested in developing EAs that could detect that there had been a change in the optimum and could continue to alter their search to find the new optimum.

Some examples of the earliest papers related to dealing with time-varying objective functions using evolutionary algorithms are by Pettit and Swigger [133] and Krishnakumar [94]. These papers reported on attempts to track optima in fluctuating, non-stationary environments (objective functions). Pettit and Swigger studied environments that fluctuated stochastically at different rates and they envisioned applications to areas such as voice recognition and synthesis.

Grefenstette extended the work of Krisnakumar [73], by pursuing ideas related to maintaining diversity. His work considered an abstract problem that used an optimisation surface that would change randomly every 20 generations of the genetic algorithm, with new peaks appearing and previous optima being replaced by new ones. Although not addressed as an application in that paper, reference was made to real-world dynamic problems such as optimising the fuel mixture for a jet engine based on the measurement of engine output. This problem is dynamic because of both slow and rapid changes; the components of the engine slowly degrade over time and hence slowly alter the function being optimised, and it is also possible to experience a rapid change due to the sudden failure of a component. It was clear that the intent of the work was to serve as a foundation for dealing with these types of problems.

Different authors have worked to create standard test suites that can be used to study time-varying objective functions. For instance, Branke [22] has published a test function called the moving peaks benchmark (MPB) problem. This is an abstract function that represents a landscape with several peaks. Each peak is defined by a height, a width and a position in the xy-plane. The function is dynamic because these three characteristics of each peak are altered slightly as time progresses. It is easy to imagine this function surface slowly changing with time, with the global optimum slowly moving its xy-coordinates, so that eventually the global maximum may shift from one of the peaks to another.

The case study that will be considered next in this chapter, does not fall into the category of dynamic problems discussed in this section. When considered from a broad perspective, the wine bottling problem can be thought of as having a time-varying objective function, but it is not one that changes during the course of an EA run, rather the objective function changes with each run of the application as business conditions change.

3.2.2 Time-varying input variables

These are problems in which the input data being processed in the optimisation scenario change from day to day, or in principle from run to run of the EA. For example, if we are interested in optimising the production schedule of a factory, then each day, or possibly each minute, the set of customer orders to be allocated and sequenced on machines will change. This is because we are dealing with a live business environment; as time passes, old orders are completed and new ones arrive.

Johnston and Adorf [116] described how artificial neural networks can be used to solve complex scheduling problems with many constraints; they reported that the real-world problem of scheduling the assigned usage of the NASA/ESA Hubble Space Telescope was solved using an application based on these techniques.

Chryssolouris and Subramaniam [40] explored the dynamic job shop problem where multiple criteria are considered and jobs may be processed by alternate machines. Although they recognised the lack of work on dynamic constraints such as random job arrival, in terms of input to the evolutionary algorithm, their work primarily focused on multiple job routes.

Madureira, Ramos and Carmo Silva [111] used a genetic algorithm to study how to produce schedules in a highly dynamic environment where new input variables require continual re-scheduling. Their work was based on an extended job-shop in a dynamic environment with simple products requiring several stages of assembly.

Another example of recent work done in this area include González et al. [72]. They study the multimodal transport problem that searches for the most economical route in the distribution of cast iron ductile piping.

3.2.3 Time-varying constraints

Problems involving time-varying constraints arise when the constraints of the environment within which a solution must be found change over time. These varying constraints add an additional level of complexity to the problem because a good EA approach must be able to operate equally well regardless of how many of these constraints are in place, and in what particular combination they occur. In general, we do not know what the current schedule might be at any given moment in time, yet we must create an EA that creates a new solution that matches up with the existing one, and still produces an optimal result for the long term.

Jain and Elmaraghy [88] discussed the types of problems that can be categorised as time-varying constraints;they described circumstances that require the generation of regular new production schedules due to uncertainties (both expected and unexpected) in the production environment. They touched on typical examples such as machine breakdowns, increased order priority, rush orders arrival and cancellations. All of these issues are also considered in this chapter, from the perspective of an integrated evolutionary algorithms-based software solution. There are many other example of works recognising this type of problem (see [132, 95, 84]).

Based on our experience in solving real-world optimisation problems for commercial organisations, we have found that the type of problems commonly experienced in industry are those that belong to the second and third categories. Interestingly we have also found that the vast majority of published research in evolutionary algorithms addresses the first category and to a lesser extent the second category. However, dynamic optimisation of the third kind, i.e. where the problem involves time-varying constraints, although well-recognised in other domains, has been the subject of relatively little investigation in evolutionary algorithms literature. This observation is especially true when we extend our search to the fully-fledged application of an EA to a dynamic real-world problem.

Figure 3.1 illustrates some examples of issues that typically arise in real-world problems.

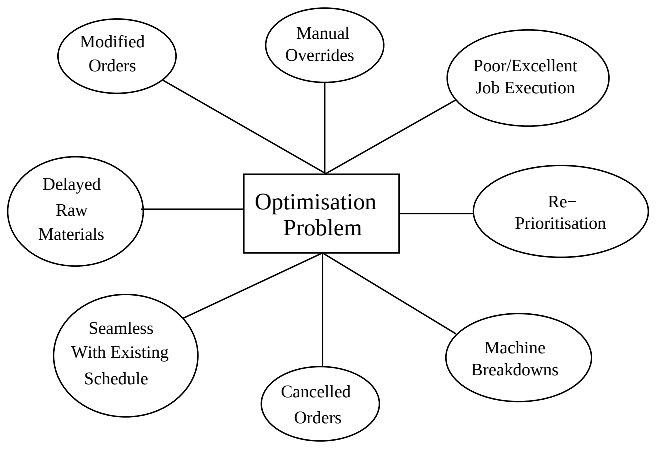


Figure 3.1: Typical Issues in Real-World Problems

3.3 Problem

Wine Manufacturing and Evolutionary Algorithms go particularly well together. From the starting point of planting grape vines and reaping the mature fruit, all the way through, crushing of the grapes, management of bulk tank movements during the fermentation process, to bottling of the finished product and sales, the wine manufacturing industry is a rich source of real-world application areas. Many of these problems are based on classic optimisation problems, and on that basis alone are quite difficult to solve. Evolutionary Algorithms of course by their very nature are natural takers for these kinds of challenges. In this section, we will examine what is involved in one of the wine manufacturing steps just described, namely the bottling of the finished product.

Before getting to the point where wine is bottled into a finished product, the liquid would have gone through a series of fermentation and other processing steps. We will assume that we are at the point where the liquid is in a finished, consumable state, and is residing in a bulk storage tank, which can be anywhere from several tens of thousands of litres, up to more than one million litres in volume. This bulk liquid remains in storage awaiting the bottling process during which it is pumped into a bottling factory and put into consumer size bottles, with a typical volume of less than one litre.

A bottling factory houses several bottling *lines*. These are machines that are connected to intermediate feeding tanks that contain finished wine, and are used to transfer the wine into glass bottles and hence produce the items that everyday consumers are accustomed to purchasing. The bottling lines also take care of related tasks such as capping the bottle with a screw cap or a cork, applying a label to the bottle, and packaging the bottles into cartons. Each bottling line is capable of bottling a subset of the types of finished wine products manufactured by the wine company.

These two elements, the bulk wine liquid, and the bottling lines, constitute the basic working elements of the wine bottling problem. The bottling process is illustrated in Figure 3.2. Which bulk wines are put into which bottles, and the time at which this is done, are determined by customer orders. The wine company receives orders for particular finished goods from their clients and it is those orders that must be carefully considered in order to determine the best way of running the bottling plant. In an ideal situation, customers place their orders with sufficient notice to ensure timely bottling of their goods.

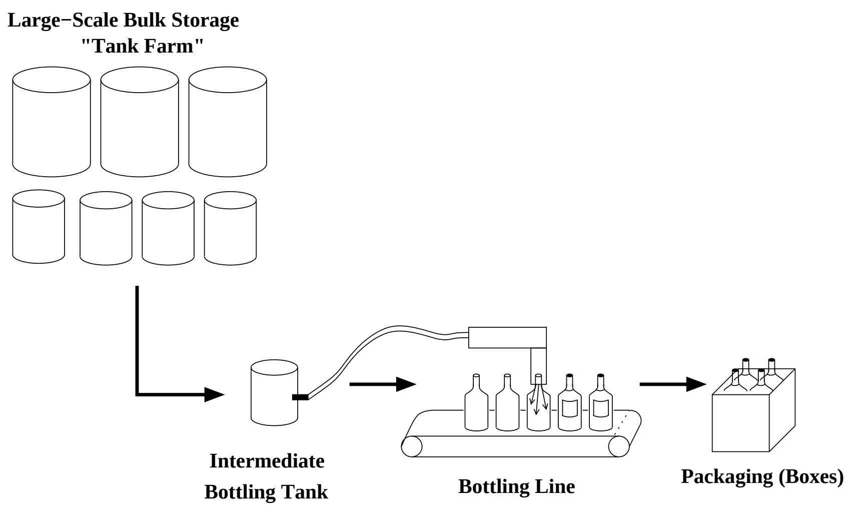


Figure 3.2: The Wine Bottling Process

The problem is to determine a sequence of orders to be carried out on each bottling machine hence classifying it as a scheduling problem, so that optimal use is make of the company's resources, from the point of view of making maximum profit, and also maximising customer satisfaction. Hence a good schedule will minimise production costs, and at the same time ensure that orders are produced in full, before their due dates. Deciding how to schedule production so as to make the best use of limited resources is the fundamental part of the wine bottling problem.

Wine making, as is the case in a multitude of other real-world applications, is a highly variable and complex business; what, from the point of view of using an evolutionary algorithm, would otherwise be considered a straightforward scheduling application, presents a series of significant challenges. The practical day-to-day business considerations are what make this problem highly dynamic. We classify the problem as a dynamic optimisation problem of the third kind, due to the fact that many of the issues are based on time-varying constraints.

Before discussing the business intricacies that give rise to the time-varying constraints, we will first consider some more basic issues that affect the scheduling problem.

- *Due dates*: When a customer places an order, the sales department will assign it a due date that is acceptable to the customer, and which should also be realistic taking into account the size of the order and available resources at the bottling plant. Sometimes for various reasons due dates are unrealistic, but nevertheless the scheduling application needs to employ techniques that strive to maximise the number of orders that are *delivered in full on time* (this expression is sometimes abbreviated as DIFOT). In cases where it is impossible to have orders delivered on time, the algorithm must strive to minimise the delays incurred in relation to each order. This is a very important objective in the application, since in business, it is of utmost important that customers not be displeased due to the late delivery of orders.

- *Bulk wine availability*: Some orders may need to be inevitably delayed due to the fact that the bulk wine needed to fill the bottles may not yet be ready. This can easily happen since the process of creating wine is quite variable, and batches may not have responded to the fermentation process as expected, and may require additional processes to get the wine to the required specification and taste. There are a number of other processes involved in the preparation of the bulk wine, such as filtration and temperature stabilisation, that can lead to delays.

- *Dry Materials*: In addition to the liquid wine, a few dry goods are required to produce a finished bottle of wine. First there is the glass bottle, and its covering which may be a screw cap or a cork, and other items such as labels, of which there may be several (for the front, back and possibly neck of the bottle), and foils and wire hoods for sparkling wines. If an order is scheduled at a particular time, then to proceed, the appropriate amounts of each dry good must be ready for installation into the bottling machine. Hence it is a requirement that the optimisation software look at the availability of these materials in order to produce feasible schedules.

- *Job run lengths*: It is inefficient to have machines frequently changing from one type of bottling job to another because this incurs set-up and take-down time and reduces the overall utilisation of the machine. Hence, the scheduling algorithm must attempt to group similar orders for sequential execution so this type of inefficiency is avoided.

- *Wine changeovers*: Wines are categorised broadly in terms of their colour – red, white, and rose (pink). Below this level of classification, there are more detailed distinctions such as sweet red, dry red, aromatic white, full-bodied white, sparkling red, sparkling white, and fortified wine. Even for a very large wine company, it may not be the case that there is sufficient production volume to justify dedicating individual machines exclusively to the production of one type of wine. Hence the reality is that the same machine must be used at different times to bottle different types of wine. When a bottling line finishes working with one type of wine and switches over to another type, this is referred to as a *wine changeover*. The bottling line must be cleaned during changeovers, to varying degrees, depending on the nature of the change. Certain types of wine changeovers are undesirable and need to be avoided where possible by the optimisation algorithm. For example, if we are changing over from a run of white wine to a run of red wine, then a relatively brief cleaning is required, since residual amounts of white wine entering into red is not much of a problem. However, the reverse situation where we go from a run of red to a run of white requires a very extensive cleaning process, including sterilisation. Minute amounts of red wine entering into a run of white is likely to cause a great deal of damage.

- *Other changeovers*: Although wine changeovers incur the most often, there are a variety of other changeovers that can happen, even within a run of the same colour wine. Each different finished good that is being produced requires a particular size and type of glass bottle, as well as a particular type of covering, which may be a screw cap or a cork, of which there are several varieties, and as well as unique labels for each brand of wine. The physical re-configuration of the bottling line hardware to accommodate these items requires a strip-down time to remove the items used by the previous run, and a set-up time to insert the new items required for the next run. The optimisation algorithm needs to try to minimise these changeovers as far as possible by appropriately grouping orders.

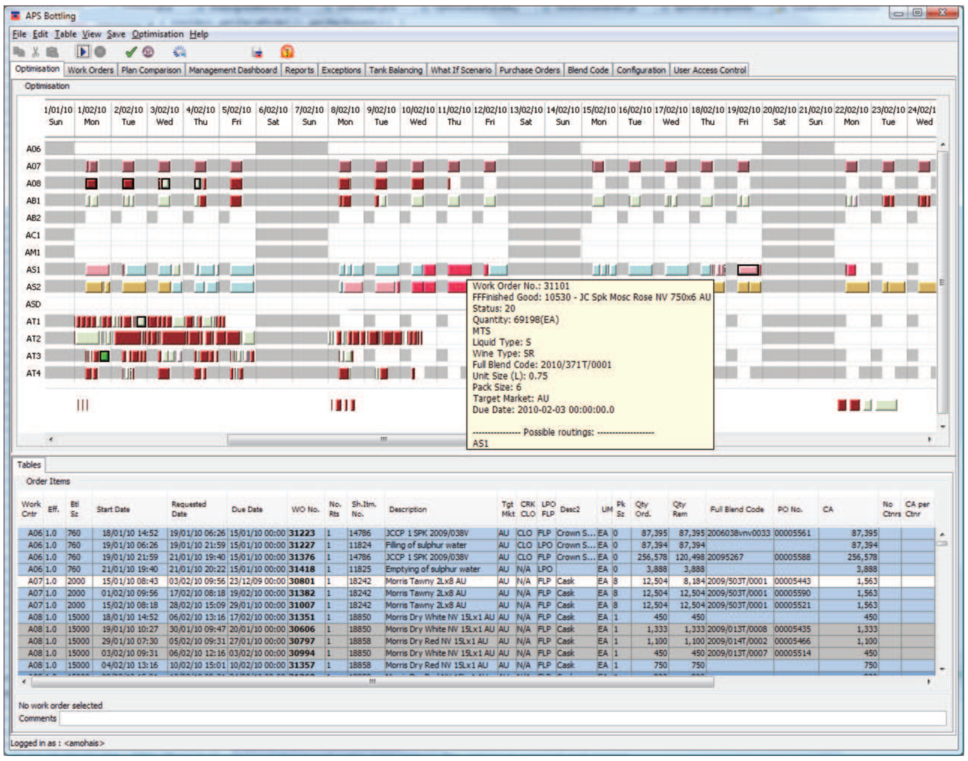
- *Bottling line availability*: Some industries use machines that are kept in operation continuously. This is sometimes the case in large-scale wine companies, but for only limited periods of time. In most scenarios, bottling lines have typical hours of operation that correspond to an average workday, which may be for example 8:00am to 6:00pm. The optimisation algorithm needs to consider the availability of each machine in order to make appropriate scheduling decisions since one of the critical factors affecting the evaluation of a proposed schedule is the number of orders that are satisfied on time. In computing machine availability must be factored in relation to the amount of time that a job will take to perform and to decision-making about where to assign it.

- *Routings*: Each product can be bottled on a number of different lines. The choices however are usually a proper subset of all of the machines. Although the same product could be bottled on a few different machines, the performance characteristics of each machine are likely to be different, sometimes significantly so. Set-up and strip-down time could vary, and also the speed at which the bottles are processed can vary significantly (this is called the *run rate*). Each possible assignment of a finished good to a bottling line, together with its associated performance data is referred to as a routing. The optimisation algorithm must strive to choose the best routing to use under the circumstances. If all else is equal, then naturally the fastest routing would be chosen, but this is often not the case as some lines may be heavily loaded thus preventing such a choice from being made. In order to achieve better load balancing between the machines, alternate routings may have to be used.

Our work on the wine bottling problem resulted in a full-featured piece of software integrated around a core evolutionary algorithm that deals with all of the above listed issues, and creates feasible, optimal schedules for satisfying customer orders for wine. The application was launched for a major global wine company, and it experiences heavy daily usage at international sites. It is a cornerstone of their scheduling department, and we think it is a good example of an integrated evolutionary algorithm serving robustly in prime time. The main screen of the application is shown in Figure 3.3 with all confidential client information removed. The area of horizontally adjacent rectangles shown in the upper portion of the picture is a graphical representation of the schedule produced by the software. Each rectangle corresponds to a customer order that has been assigned to a bottling line at a particular time. The length of the rectangle is indicative of the amount of time required to execute the job and its colour coding lets the user know at a glance the type of wine being produced. The graphical interface also allows the user to drag and drop the rectangles to make manual adjustments to the schedule produced by the optimisation algorithm.

Figure 3.4 shows the configuration interface of the software application that allows the user to enter parameters that define the availability of each bottling line. This interface needed to be completely flexible in terms of allowing the user to specify any possible set of available and unavailable times for each line. In this application the user is allowed to specify a typical weekly timetable, for example a machine may be available from Monday to Friday from 8:00am to 6:00pm, and unavailable otherwise. It also allows the user to specify periods of time in which there is a deviation from this typical schedule. For example, ahead of a festive season, it might be desirable to make the machines work longer hours, and possibly on weekends. This interface also allows the user to allocate periods of time for regularly scheduled downtime for maintenance of the machines.

Figure 3.5 shows a material requirements dialog that is part of the software application. For any given order, the user may view the materials required to produce the required number of bottles of wine, such as caps, corks, labels, and so on. The application provides the user with information on the availability of each material, based on knowledge of the opening inventory stock, that is used for a running simulation over the time frame of the production schedule. Additional knowledge of still-open purchase orders for additional materials, and also knowledge of the lead times of various suppliers of these materials are catered for; this allows the software's underlying algorithm to make appropriate scheduling decisions so as to best ensure that orders are sequenced in a feasible


APS Bottling
File Edit Table View Save Optimisation Help
Optimisation | Work Orders | Plan Comparison | Management Dashboard | Reports | Exceptions | Tank Balancing | What If Scenario | Purchase Orders | Blend Code | Configuration | User Access Control
Optimisation
1/01/10 Sun | 1/02/10 Mon | 2/02/10 Tue | 3/02/10 Wed | 4/02/10 Thu | 5/02/10 Fri | 6/02/10 Sat | 7/02/10 Sun | 8/02/10 Mon | 9/02/10 Tue | 10/02/10 Wed | 11/02/10 Thu | 12/02/10 Fri | 13/02/10 Sat | 14/02/10 Sun | 15/02/10 Mon | 16/02/10 Tue | 17/02/10 Wed | 18/02/10 Thu | 19/02/10 Fri | 20/02/10 Sat | 21/02/10 Sun | 22/02/10 Mon | 23/02/10 Tue | 24/02/1 Wed
A06 A07 A08 AB1 AB2 AC1 AM1 AS1 AS2 ASD AT1 AT2 AT3 AT4
Tables
Order Items

Work Order No.: 31101
FFFinished Good: 10530 - JC Spk Mosc Rose NV 750x6 AU
Status: 20
Quantity: 69198(EA)
MTS
Liquid Type: S
Wine Type: SR
Full Blend Code: 2010/371T/0001
Unit Size (L): 0.75
Pack Size: 6
Target Market: AU
Due Date: 2010-02-03 00:00:00.0

---------------- Possible routings: ------------------
AS1

Work Cntr	Eff.	Btl Sz	Start Date	Requested Date	Due Date	WO No.	No. Rts	Sh.Itm. No.	Description	Tgt Mkt	CRK CLO	LPO FLP	Desc2	UM	Pk Sz	Qty Ord.	Qty Rem	Full Blend Code	PO No.	CA	No Ctnrs	CA per Ctnr
A06	1.0	760	18/01/10 14:52	19/01/10 06:26	15/01/10 00:00	**31223**	1	14786	JCCP 1 SPK 2009/038V	AU	CLO	FLP	Crown S...	EA	0	87,395	87,395	2006038vnv0033	00005561	87,395		
A06	1.0	760	19/01/10 06:26	19/01/10 21:59	15/01/10 00:00	**31227**	1	11824	Filling of sulphur water	AU	CLO	LPO	Crown S...	EA	0	87,394	87,394			87,394		
A06	1.0	760	19/01/10 21:59	21/01/10 19:40	15/01/10 00:00	**31376**	1	14786	JCCP 1 SPK 2009/038V	AU	CLO	FLP	Crown S...	EA	0	256,578	120,498	20095267	00005588	256,578		
A06	1.0	760	21/01/10 19:40	21/01/10 20:22	15/01/10 00:00	**31418**	1	11825	Emptying of sulphur water	AU	N/A	LPO		EA	0	3,888	3,888			3,888		
A07	1.0	2000	15/01/10 08:43	03/02/10 09:56	23/12/09 00:00	**30801**	1	18242	Morris Tawny 2Lx8 AU	AU	N/A	FLP	Cask	EA	8	12,504	8,184	2009/503T/0001	00005443	1,563		
A07	1.0	2000	01/02/10 09:56	17/02/10 08:18	19/02/10 00:00	**31382**	1	18242	Morris Tawny 2Lx8 AU	AU	N/A	FLP	Cask	EA	8	12,504	12,504	2009/503T/0001	00005590	1,563		
A07	1.0	2000	15/02/10 08:18	28/02/10 15:09	29/01/10 00:00	**31007**	1	18242	Morris Tawny 2Lx8 AU	AU	N/A	FLP	Cask	EA	8	12,504	12,504	2009/503T/0001	00005521	1,563		
A08	1.0	15000	18/01/10 14:52	06/02/10 13:16	17/02/10 00:00	**31351**	1	18850	Morris Dry White NV 15Lx1 AU	AU	N/A	FLP	Cask	EA	1	450	450			450		
A08	1.0	15000	19/01/10 10:27	30/01/10 09:47	20/01/10 00:00	**30606**	1	18850	Morris Dry White NV 15Lx1 AU	AU	N/A	FLP	Cask	EA	1	1,333	1,333	2009/013T/0008	00005435	1,333		
A08	1.0	15000	29/01/10 07:30	05/02/10 09:31	27/01/10 00:00	**30797**	1	18858	Morris Dry Red NV 15Lx1 AU	AU	N/A	FLP	Cask	EA	1	1,100	1,100	2009/014T/0002	00005466	1,100		
A08	1.0	15000	03/02/10 09:31	06/02/10 12:16	03/02/10 00:00	**30994**	1	18850	Morris Dry White NV 15Lx1 AU	AU	N/A	FLP	Cask	EA	1	450	450	2009/013T/0007	00005514	450		
A08	1.0	15000	04/02/10 13:16	10/02/10 15:01	10/02/10 00:00	**31357**	1	18858	Morris Dry Red NV 15Lx1 AU	AU	N/A	FLP	Cask	EA	1	750	750			750		

No work order selected
Comments
Logged in as : <amohais>

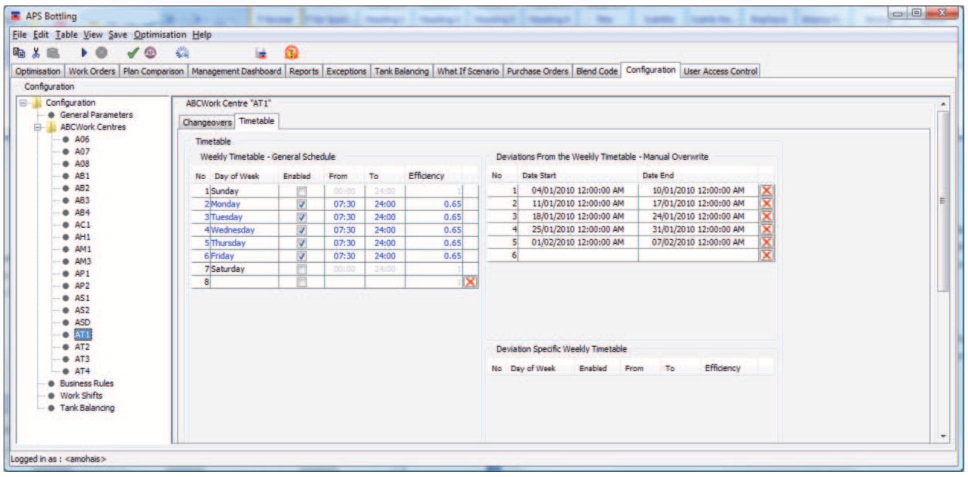
Figure 3.3: Visualisation of a schedule on multiple production lines in the software application

manner. If scheduled too early, at a time when the materials would not be available, an order would have to be passed over, possibly leading to other disruptions in the schedule and hence in the factory itself.

3.4 Time-Varying Challenges in Wine Bottling

In this section, we will look at some business requirements that led to time-varying constraints that had to be addressed in the software. Here we will only consider the issues; their actual solution, including algorithmic details, will be covered in a later section.

- *Manual assignments*: There are various scenarios in which the human scheduler would need the ability to override the schedule that was produced by the evolutionary algorithm. For instance, it might be that some of the information used as input by the algorithm may for some reason be


APS Bottling
File Edit Table View Save Optimisation Help
Optimisation | Work Orders | Plan Comparison | Management Dashboard | Reports | Exceptions | Tank Balancing | What If Scenario | Purchase Orders | Blend Code | Configuration | User Access Control
Configuration
Configuration
General Parameters
ABCWork Centres
A06
A07
A08
AB1
AB2
AB3
AB4
AC1
AH1
AM1
AM3
AP1
AP2
AS1
AS2
ASD
AT1
AT2
AT3
AT4
Business Rules
Work Shifts
Tank Balancing
Changeovers
Timetable
Logged in as : <amohais>

ABCWork Centre "AT1"

Timetable

Weekly Timetable - General Schedule

No	Day of Week	Enabled	From	To	Efficiency
1	Sunday	☐	00:00	24:00	1
2	Monday	✓	07:30	24:00	0.65
3	Tuesday	✓	07:30	24:00	0.65
4	Wednesday	✓	07:30	24:00	0.65
5	Thursday	✓	07:30	24:00	0.65
6	Friday	✓	07:30	24:00	0.65
7	Saturday	☐	00:00	24:00	1
8		☐			1

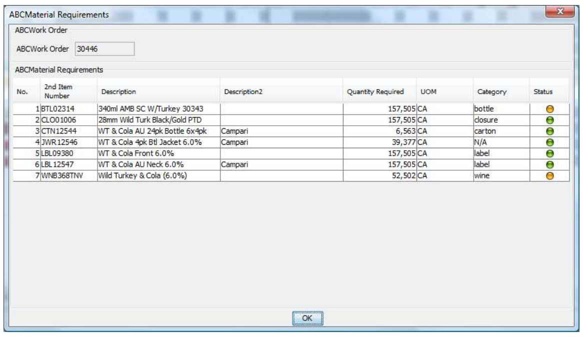
Deviations From the Weekly Timetable - Manual Overwrite

No	Date Start	Date End
1	04/01/2010 12:00:00 AM	10/01/2010 12:00:00 AM
2	11/01/2010 12:00:00 AM	17/01/2010 12:00:00 AM
3	18/01/2010 12:00:00 AM	24/01/2010 12:00:00 AM
4	25/01/2010 12:00:00 AM	31/01/2010 12:00:00 AM
5	01/02/2010 12:00:00 AM	07/02/2010 12:00:00 AM
6		

Deviation Specific Weekly Timetable

No	Day of Week	Enabled	From	To	Efficiency

Figure 3.4: Machine Availability Interface



ABCWork Order

ABCWork Order 30446

ABCMaterial Requirements

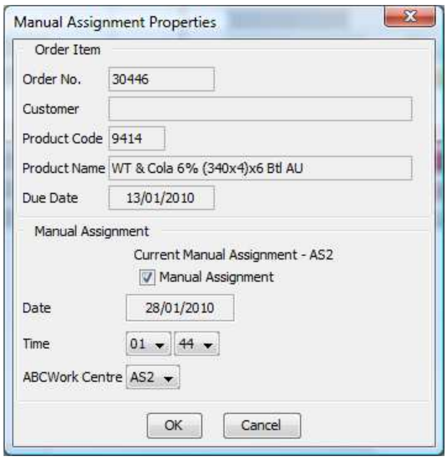
No.	2nd Item Number	Description	Description2	Quantity Required	UOM	Category	Status
1	BTL02314	340ml AMB SC W/Turkey 30343		157,505	CA	bottle	●
2	CLO01006	28mm Wild Turk Black/Gold PTD		157,505	CA	closure	●
3	CTN12544	WT & Cola AU 24pk Bottle 6x4pk	Campari	6,563	CA	carton	●
4	JWR12546	WT & Cola 4pk Btl Jacket 6.0%	Campari	39,377	CA	N/A	●
5	LBL09380	WT & Cola Front 6.0%		157,505	CA	label	●
6	LBL12547	WT & Cola AU Neck 6.0%	Campari	157,505	CA	label	●
7	WNB368TNV	Wild Turkey & Cola (6.0%)		52,502	CA	wine	●

Figure 3.5: Material Requirements Interface

inaccurate; examples are the availability of dry goods and expected delivery times of raw materials, which are all imported automatically from the company's Enterprise Resource Planning (ERP) system. Hence the schedule that is created may not be feasible and would need to be corrected by the human scheduler. Another typical scenario is one where a very important customer makes a late, but urgent, request for a large quantity of wine. Even if this causes severe disruptions to the smooth running of the bottling plant, this type of request is usually accommodated due to the high value placed on some customers, and the importance of maintaining a good business relationship with them. In this kind of situation the unusual placement of that urgent order in the production sequence is not something that the evolutionary algorithm could perform because it goes against most of the optimisation objectives that were trying to be achieved in the first place. The software application that we developed had to be flexible enough to allow its built-in evolutionary algorithm to work in such a way that it could actively seek out an optimal solution that satisfied the constraints described before, but at the same time allowed inefficient manual overrides dictated by a human operator to co-exist with the otherwise optimal solution. Another way of looking at this is that the structure of the search algorithm had to be flexible enough to find locally optimal solutions in restricted neighbourhoods of the overall search space, with those neighbourhoods being defined by user-specified manual assignments.

As mentioned earlier, the application presents the schedule with both graphical and tabular representations, each of which may be intuitively manipulated by the user, simply by dragging and dropping graphical rectangles, or rows of a table. This provides the capability of altering the sequencing of orders to suit a human users preference. Additionally, as shown in Figure 3.6, there is a manual assignment interface that is accessed on a per-order basis and that allows the user to manually specify all details about when and where a particular job should be scheduled.

- *Machine breakdowns*: From time to time, a bottling line will break down and become unavailable for use. It may be that a solution found by the optimiser previously would have been planned around that machine being available during a period of time that has now become unavailable due to the breakdown. The software must be flexible enough to repair the


Manual Assignment Properties

Order Item

Order No.	30446
Customer	
Product Code	9414
Product Name	WT & Cola 6% (340x4)x6 Btl AU
Due Date	13/01/2010

Manual Assignment

Current Manual Assignment - AS2

[✓] Manual Assignment

Date	28/01/2010
Time	01 44
ABCWork Centre	AS2

Figure 3.6: Manual Assignment Interface

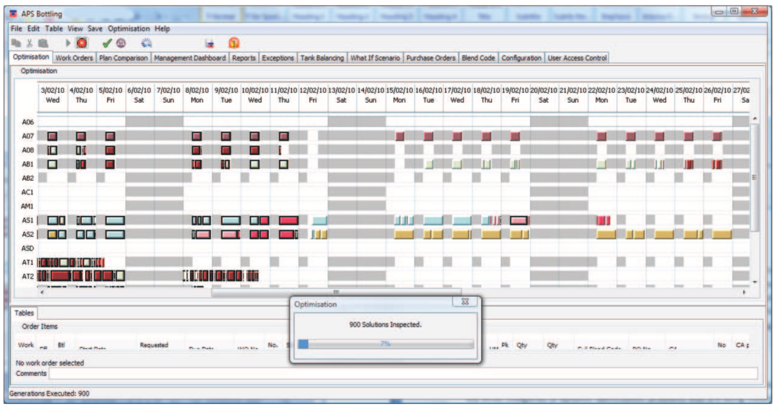
previous solution to take into account the breakdown. A very simple initial solution to this could be to merely shift the sequence of orders previously assigned to a machine, starting at the point where the machine breakdown started. All subsequent orders are scheduled later in time, by an amount that roughly works out to be the duration of the breakdown period. In some cases a more sophisticated approach is required where a complete re-optimisation is performed to repair damage that was done to the percentage of orders that will be delivered on time. This kind of optimisation must attempt to keep as much of the existing schedule intact while repairing the placement of the affected orders.

- *Freeze periods*: Every time the evolutionary algorithm is run, the possibility exists that a very different schedule could be produced when compared to the one previously generated. This happens simply as a result of the EA doing its job of finding an optimal solution as dictated by its objective function (which changes each time new orders and constraints appear in the system, which is virtually always). In a real-world application, it is highly unlikely that this kind of approach could be tolerated. The preferred mode of operation is the following. The software is used to initially create a schedule for a fairly long period of time, for example 2-4 months. However, once that schedule is accepted and saved to an internal database, subsequent daily use of the optimiser to schedule newly arriving customer orders must be done in a controlled manner so that there is a buffer period at the beginning of the schedule that remains the same as it was the previous day. This unchanging portion of the schedule is referred to as a *freeze period*. It is usually defined in terms of a number of days, for example 7 days. In order to achieve this effect, the evolutionary algorithm has to perform what we refer to as seamless *meshing* with an existing schedule. The first 7 days of the existing schedule are frozen, and some of the previously existing orders, along with new ones are re-shuffled by the optimisation algorithm and attached in a neat continuous way with the orders in the freeze period. The contents of the freeze period vary with time, as each day some orders are executed and so are removed completely from the schedule, and others that were once outside the freeze period gradually move into it. This is referred to as a *rolling time window*.

On a more granular level, freeze periods are affected by live update feedback data coming from the factory floor. The *current order* on a given bottling line is the first order showing up on the schedule. At any given moment, it usually corresponds to an order that physically is in the process of being executed in the factory. Our system is designed to receive updates in near-real-time (every 5 minutes), thus letting it know how much of that order has been carried out. That in turn affects the orders in the freeze period because the size of the current order needs to be gradually reduced to reflect what remains to be done, and consequently at the right-hand end of the freeze period, more orders will come in. This sometimes leads to a situation in which there is an order that straddles the freeze period and the open-optimisation period.

Figure 3.7 shows a screen capture from our software application that illustrates how the frozen period is managed. In the graphical area of the window, where orders are represented by rectangles, on the left-hand side of the screen there are a number of orders that are surrounded by a thick bold border, indicating that they have been frozen. On the right-hand side, are the orders that the optimiser is free to move around to find an optimal schedule.

- *Modified orders*: Once a schedule has been created and saved, there might be a situation in which the next time the software package is opened, it realises that a modification has been made to one of the scheduled orders in the database. This might be, for example, a change in quantity. More or fewer bottles of wine may be required and, by adjusting the scheduled



900 Solutions Inspected.

No work order selected

Generations Executed: 900

Figure 3.7: Freeze Period Illustration

order, the start and end times of all subsequent orders on the same bottling line become affected.

- *Poor/Excellent Job Execution*: Due to a number of variables, the efficiency of a bottling machine may be better or worse on any given day, and the factory manager would expect the scheduling optimiser to take this into account when creating a new schedule, or when adjusting an existing one. A machine that is performing poorly will delay jobs scheduled further down in time, and likewise a machine that is performing unusually well, will bring jobs up earlier, which could in some cases have a detrimental effect when we take into consideration the availability of dry goods. The optimisation software must be able to deal with this time-varying issue by re-adjusting the solution, or by re-optimising as necessary.

3.5 The Solution Using an Evolutionary Algorithm.

In this section we will look at the structural, algorithmic and programmatic details required to solve the problem using an evolutionary algorithm. We begin by examining the representation of a candidate solution and then look at how that abstract representation is converted (decoded) into an actual schedule with dates and times. Next we look at the key operators employed to alter candidate individuals. Finally we re-visit the time-varying problems described above and describe how various elements of the evolutionary algorithm had to be modified to accommodate those issues.

3.5.1 Representation

One of the most important aspects to the solution of a problem using an evolutionary algorithm is the representation used to encode a candidate solution. Forcing the use of a particular representation may significantly impact on the quality of the solutions found since many useful operators may be overlooked, or may be cumbersome to program, thereby slowing down the execution of the algorithm. For this application, we chose a representation that closely matched the actual assignment of orders to machines.

For the wine bottling scheduling problem, the core of the problem was conceptualised as having a number of orders that must be placed on a fixed number of bottling machines in an efficient sequence. Hence the natural representation to use is one of a mapping of lists of orders to machines. This can be visualised as in Figure 3.8. The representation illustrated in this diagram is quite similar to the final schedule presented visually in Figure 3.3 above. The difference is that the actual decoding on the individual into a real-world schedule also takes into consideration several other time-varying factors such as machine availability, splitting of single orders into several sub-jobs and meshing with an existing schedule.

The representation shown in Figure 3.8 is stored programmatically as a map of machines to variable-length lists of orders. The list on any given machine is sorted chronologically in terms of which orders will be carried out first. The individual is constructed in such a way that the assignment of orders to machines is always valid, in other words the assigned orders always respect product routings.

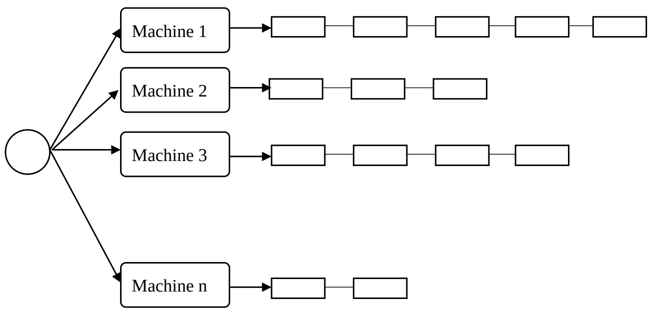


Figure 3.8: Scheduling Individual Representation

3.5.2 Decoding

In order to manage the decoding process, we employed a concept we referred to as *time blocks*. A time block, illustrated in Figure 3.9, is a data structure that keeps track of a contiguous period of time. Each such block keeps track of a start time, an end time and an activity that is performed during that time, allow for the possibility that nothing is done actually done, in which case the time block is referred to as an *available* time block. The link to the activity performed during a time block may point to an external data structure that contains any information on any level of detail that is required to accurately model a scenario.

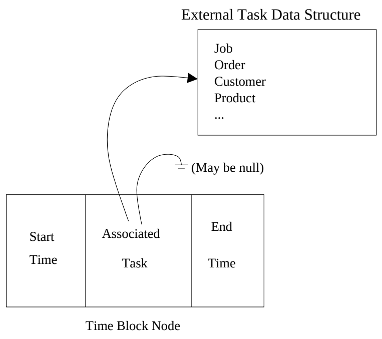


Figure 3.9: A Time Block Node

Decoding begins with a series of multiple linked lists of time block nodes associated with each machine. Each machine has a list of available time blocks, and occupied time blocks. At the outset, before anything is placed on a machine, it would only contain a list of available time blocks, each representing a chunk of time during which the machine is available for use. This is what would happen in the nominal scenario. If there are manual decisions that were already made by a human operator, then these would be reflected in the existence of some occupied time blocks. More detail on this issue is given in section 3.5.4 below.

Decoding proceeds by going through all machine-job pairs found in the individual representation and proceeding to the corresponding machine, finding an available time block node, and marking it as occupied for the corresponding job. Some jobs may not be able to fit in the first available time block, and may need to be split into multiple parts. How this is done, indeed, if it is permissible at all, depends on the policies of the business for which the scheduling application is being created. In the case of the wine bottling application that we are considering, orders were split across adjacent available time blocks. This process is illustrated in Figure 3.10 below.

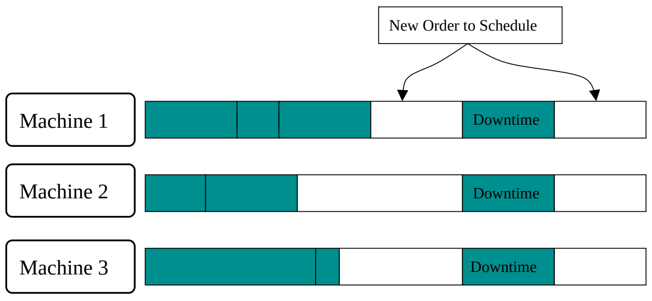


Figure 3.10: Illustration of the Decoding Process

An important issue that arises when decoding is performed in this way for a scheduling problem is the question: in what order should the machine-job pairs be considered? Should we select Machine-A first and process all of the orders in its list, and then proceed to Machine-B and so on? Alternatively, should we process one job from Machine-A's list, then one job from Machine-B's list eventually cycling back to Machine-A, and keep on in that manner until all jobs have been decoded? The answer to this question depends on the industry being considered. For the wine bottling problem there are no dependency or constraints between jobs running on different machines, and hence the former approach was used. However, there are cases in which the second approach (or possibly others) may be needed. For example in some industries there might be a limit to the number of changeovers that are permitted to occur across an entire plant in one day. In those cases, it is important not to bias the decoding (or alternatively to deliberately and carefully bias it) to favor one machine, or a group of machines.

3.5.3 Operators

A number of operators were used to manipulate the above representation. To avoid the problem of having to perform extensive repairs based on invalid representation states, crossover-type operators were avoided. Those listed below, which are typical examples from the set used, may all be considered as mutation operators.

- *Routing Mutation*: This operator modifies the machine that was selected to execute a job. An alternative is randomly chosen from the set of possible options.

- *Load Balancing Mutation*: This is a variation to the routing mutation operator which, instead of merely randomly choosing an alternative machine for an order, could choose from a subset of machines that are under-loaded. Such an approach would help with load-balancing of the machines.

- *Grouping*: This operator groups orders based on some common characteristic, such as wine colour, bottle type, or destination export country. There are several variants of the grouping operators. Some operate quite randomly, looking for a group of orders based on some characteristic, and then looking left or right for a similar group and then merging the two. Other examples are given below.

- *Recursive Grouping*: A more directed variation on grouping is recursive grouping. This operator seeks out an existing group of jobs based on wine colour, then within that group performs random grouping based on some other characteristic, such as bottle size. This process may then be repeated inside one of the subgroups.

- *Outward Grouping*: Outward grouping is a term we use to describe the process of identifying groups in a list of jobs based on a primary characteristic, then randomly selecting one of them and from that location looking left and right for another group with a common secondary characteristic and finally bringing the two together. As a concrete example, we may first identify groups based on a primary attribute, say closure type that is, whether a screw cap or a cork is used to close the bottle – and then randomly select a group that uses screw caps as a starting point. Next a secondary attribute such as wine colour is considered. Suppose the group of screw caps involves white wines, then we look randomly to the left or to the right for a group based on the primary attribute that also involves white wines. In the end we may end up merging with a group of jobs that involves corks, but which happens to involve only white wines. The end result is we have a larger contiguous group of white wine, with two subgroups, one with screw caps, the other with corks. The process of finding a starting point and then looking outwards for subgroups to merge with gave rise to the term *outward grouping*.

- *Order Prioritisation*: Orders that may be showing up as being produced late after decoding an individual are stochastically prioritised by moving them left in the decoding queue. Again, this type of operator could be made more intelligent than random by moving groups of jobs along with the one that is identified as being late. As a result the job gets prioritised, but at the same time disruption to the grouping is minimised. This operator works on one of the most important objectives of the solution, namely to maximise DIFOT.

3.5.4 Solving the Dynamic Issues

We will now look at how we dealt with the time-varying issues that were identified in section 3.4. As will be observed, the problems were addressed by a combination of modifications made to the initial time block node linked-lists, to the decoding process, by altering the input variables to the optimiser, and by introducing a step between the optimiser and the human user called *solution re-alignment*.

- *Solving Manual Overrides*: This problem was solved by applying a constraint to the decoding process, and indirectly affecting the fitness function. The software application allows the user to select a particular order, and to specify which machine it should be done on, as well as the date and time of assignment. This becomes a timetable constraint for the genotype decoder. When a candidate individual is being decoded, the initial state of the machine time block usages includes the manually assigned orders as part of the list of occupied time blocks. Subsequent decoding of non-manually-assigned jobs would take place as usual using the remaining available time blocks.

  The fitness function would then evaluate how well the individual was decoded into the partially occupied initial machine state. As in the case where there are no manual assignments, it is up to the evolutionary operators to modify the individual in such a way that its decoded form meshes well with the fixed orders to reduce changeovers and so on. The initial state of available time blocks used for decoding is illustrated in figure 3.11.

  In this approach, it is primarily the fitness function that would guide the search to a relatively good solution built around the manual assignment. However it is also important to de-emphasise some of the more structured and aggressive operators such as recursive grouping and outward grouping, which were designed with a clean slate in mind. They would still contribute toward the search for a good solution in parts of the time period that do not contain manual assignments, but the main evolutionary algorithm loop should keep track of the performance of these operators and adjust their probability of application accordingly.

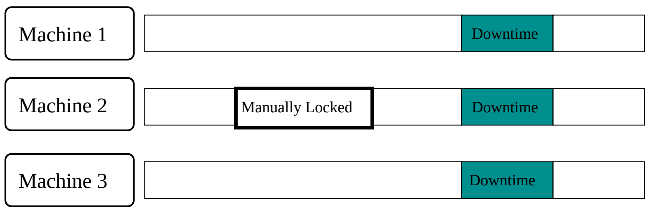


Figure 3.11: Modified Initial Time Blocks for Manual Locks

- *Solving Freeze Periods*: The application allows the user to select a freeze date and time on each machine used in the bottling plant. During an optimisation run, all assignments in the existing schedule that are before the freeze period cutoff are internally marked as manual assignments, and therefore behave in exactly the same way as described above for user-defined manual assignments.

  It is also important to pay attention to jobs that might straddle the freeze period cutoff date. Such orders need to be fully frozen to ensure correct results. Another situation that needs attention is where a job may be split into several pieces, due, for example, to relatively low machine availability on consecutive days, with some of those split pieces falling into the frozen period. Care must be taken to also freeze the parts that are outside of the freeze period, but which belong to the same order.

- *Solving Machine Breakdowns, Modified Orders, and Poor/Excellent Job Execution*: These three problems were solved using a similar approach. In the case of Modified Orders, when the application re-loads and realises that an existing order has been modified, for example its quantity has been increased or decreased, then the necessary action to be taken is at the level of modifying the existing solution, before it is fed back into the next run of the evolutionary algorithm. This was accomplished using a process call *solution re-alignment.*

Essentially, on each machine, all assigned orders, sorted in order by start date, are examined for any changes to the underlying orders. A new instance of an available time block nodes linked-list is created and the assigned orders are then re-inserted into it. By reviewing each order as it last existed in the schedule, with the order record coming from the company's ERP system, it is possible to allow assignments to be shortened or lengthened as needed. Following the re-insertion of a modified assignment, all subsequent assignments are modified to fit in the resulting changed available space. This process has a direct effect on the number of orders ending up in the freeze period, and therefore on the amount of available time that the optimiser has at its disposal for the next run. The process of solution re-alignment is illustrated in figure 3.12

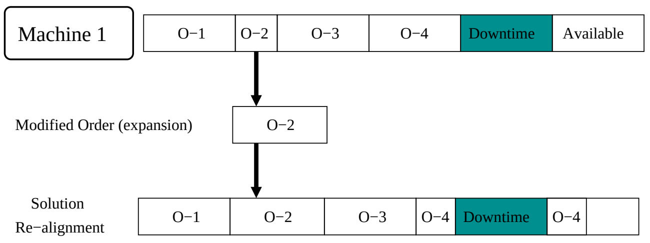


Figure 3.12: Realigning a Solution

Solution re-alignment is a step that links the internal decoded representation of an individual with the dynamic world of the human operator in which multiple constraints can vary from one use of the application to another. It allows an existing solution to be adapted to match constraints that may have changed, and possibly subsequently allow the evolutionary algorithm to run with a freeze period that now accurately reflects the effect of the changed constraints. Re-alignment can be thought of as a second-level decoding process that works with an already decoded representation as opposed to working with a genotypic representation.

As we already mentioned earlier in chapter, the system has been deployed into a live production environment at a few locations, and is in daily use.

3.6 Summary

In this chapter, we looked at issues that arise when dealing with dynamic optimisation problems in real-world applications within a single silo. A literature review was presented in line with our classification of dynamic problems according to time-varying objective functions, input variables and constraints. An important consideration, that was emphasised throughout the chapter, was that complexities arising due to time-variability had be dealt with adequately in order to ensure that the solution produced is usable in an actual business environment. Given that a large real-world software application was presented as the main outcome of the research presented here, the need to ensure that those issues were resolved in a practical way that still allowed the power of evolutionary algorithms to be applied was a main theme.

From the three types of dynamic optimisation problems, we identified the need for algorithms addressing the issue of time-varying constraints when dealing with real-world applications. The majority of research on dynamic optimisation problems is focussed mainly on time-varying objective functions, which clearly deserve research efforts. However, from the business perspective the main issue of a dynamic environment are the constraints that change over time. The issues raised were exemplified by a case study based on an evolutionary algorithm scheduling optimiser that was implemented for use in the wine bottling industry. The software, having been deployed and put into daily use in a live production environment, serves as empirical evidence that the approaches put forward in this chapter have passed the litmus test of suitability for real-world applications.

One of the key issues was the need to be able to allow a human operator to manually decide certain parts of the eventual solution. In effect, what this does is modifies the search space explored by the evolutionary algorithm. Although there are undoubtedly better solutions that exist in the unconstrained search space, when we take into account these so-called manual assignments, the optimal solution based on those restrictions must be found; this would satisfy the immediate business needs that gave rise to the need for human intervention into the search process.

Another key issue that was explored was the need to ensure that future schedules mesh seamlessly with existing ones. This was handled by extending the concept of manual-assignments to the concept of a freeze period in which existing schedule assignments are treated as unchanging, and therefore tantamount to numerous manual assignments. This has the benefit of allowing factory operators to continue working with the assurance that preparations made for the next few days will not be disrupted, and planning can proceed in a smooth and normal manner.

Overall, what has been illustrated is that in implementing a scheduling evolutionary algorithm for use in a practical commercial application, it is necessary to design almost everything, from the representation, the decoding process, the operators and the solution structure. This must be done in such a way that maximum flexibility is maintained with respect, first, to allowing time-varying constraints to be easily considered by the core algorithm that finds an optimal schedule, and second, to allowing an easy and natural flow between data structures used directly by the optimisation algorithm and the user interface that is manipulated by the human operator.

From the supply chain perspective, this chapter provided an example of a single silo problem. However, as have been mentioned before, the real business value of optimisation is evident when the optimisation is done in the context of the whole supply chain. In Chapter 7 this bottling system is used as a part of the whole wine supply chain to find a global optimum from the business perspective.

Chapter 4

Two-silo Supply Chain

This chapter considers an artificial two-silo supply chain and compares several techniques for global supply chain optimisation. One technique employs sequential evolutionary optimisation of silos, that is, one silo is optimised and its output is then used as an initial input parameter for the optimisation of the second silo. This technique is compared with several techniques based on cooperative coevolution, which employ optimisation of both silos in parallel with periodic cooperation between the silos. Each technique is tested on an experimental supply chain. The results show that parallel optimisation with cooperation yields higher *quality* solutions.

4.1 The Problem

A simulated supply chain with two silos has been built to investigate the issues connected with its global optimisation (see Figure 4.1). The first silo (S1) is a production silo that assembles goods and then ships these finished products for consideration into the second silo. The second silo (S2) serves as a distribution component. It transports goods to customers using a fleet of trucks. This simulates processes in real-world supply chains. The goal of optimisation for this supply chain is to minimise overall expenses that occur with production and distribution. Below is a more detailed description of functionality of each silo.

The *first* component in the supply chain is a production scheduling silo, and it is represented in the literature as a *job-shop scheduling problem* (JSSP).

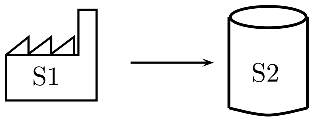


Figure 4.1: Simulated two-silo supply chain

This problem consists of a set of jobs $J = \{J_1, \ldots, J_n\}$ and a set of machines $M = \{M_1, \ldots, M_n\}$ on which these jobs can be executed. Each job has a set of operations O and a specific order of execution on each of the existing machines. Each operation i of the job J_j on machine M_k takes certain processing time and is denoted by $P_{i,j,k}$. Two operations cannot be executed on a machine at the overlapping intervals of time. The goal is to find such ordering of these jobs to minimise the total time needed to produce all of them.

The *second* component of the system is a distribution silo and it represents the *vehicle routing problem* (VRP). The basic VRP is defined as follows. Given the undirected graph $G = (V, E)$ where $V = \{0, \ldots, n\}$ is the set of vertices, each vertex $v \in V \backslash \{0\}$ represents a customer destination with non-negative demand q_i and vertex 0 represents a depot. $E = \{(i, j) : i, j \in V, i < j\}$ is a set of edges which serve as routes between destinations with travel cost c_{ij}. The depot has a fleet of m identical vehicles each with a maximum vehicle capacity of Q and a maximum route length of L. The goal of the problem is to find a set of m or at most m routes with the minimal total travel cost such that each customer is visited exactly once and each route starts and ends at the depot. The solution needs to satisfy maximal capacity and maximal route length constraints.

The combination of these two components yields our simulation of the supply chain system: one silo is producing goods, and the other one is responsible for distributing them to customers. The overall supply chain works as follows. After an order is produced at the production silo it is moved to the distribution centre's second silo depot. The depot then assigns a truck and selects a route for delivery to the customer. To make this model closer to the real world, trucks deliver goods to customers by the end of each production period of length τ. For example if $\tau = 7$ then delivery of goods starts at the end of the week for products produced during the week. If during the production week, products a and b have been produced, then both are shipped to customers at the end of the week. Later in the chapter we will refer to orders produced within one τ-period as a *bucket*.

The performance of the first silo is measured in time units, while the performance of the vehicle routing problem is measured in kilometres. In order to view the global performance of the supply chain, all this is converted to a money equivalent:

$$Cost = \alpha X + \beta \sum Y_i \tag{4.1}$$

where X is a makespan of the produced schedule in time units, and Y_i is a total trip length of all trucks during the production period i. Thus αX is a production cost and $\beta \sum Y_i$ – a transportation cost. The goal of the problem on this supply chain is to minimise $Cost$. Although this problem can be considered as a multi-objective optimisation problem in this thesis we will be aggregating the objectives into a single value.

For the optimisation of the VRP silo, we use Prins' algorithm [145] and a customised algorithm based on common techniques cited in the literature [38] for optimisation of JSSP silo. Using these algorithms as *building blocks*, two different approaches to solving the problem are discussed in the next section: a sequential evolutionary approach and an approach based on cooperative coevolution [141].

4.2 Algorithms

Two local algorithms to address JSSP and VRP are described in this section. Later in the section, we explain how to use these algorithms as building blocks to construct algorithms that solve the problem of global optimisation from the business perspective of this supply chain.

Sequential and cooperative coevolutionary (parallel) approaches are shown in this section. The sequential approach uses the optimal solution of the first silo to feed into the second silo as input, and based on that, produce a solution for the second. Several variations based on cooperative coevolution are proposed: basic coevolutionary, coevolutionary based on pareto front selection, and an approach with immediate evaluation based on Clarke and Wright's algorithm [43].

4.2.1 Algorithm for the JSSP

The following version of the evolutionary algorithm was used for this problem.

Each solution in a chromosome is encoded as a permutation in the following way:

$$X = (x_1, \dots, x_{nm}) \tag{4.2}$$

where n is the number of jobs and m is the number of machines with a total of $n \times m$ genes. This representation is and operation-based representation, which means that each gene in the chromosome is one operation and each job appears in the chromosome exactly m times. An advantage of this representation is that all permutations make a valid schedule. This representation was proposed by Gen et al. [108]. An example of such representation is chromosome

$$(1, 2, 1, 2, 3, 3, 2, 1, 3).$$

which means that operations will be executed in the following order: O_{11}, O_{21}, O_{12}, O_{22}, O_{31}, O_{32}, O_{23}, O_{13}, O_{33}. Here O_{ij} is j-th operation of the job J_i. To build a schedule from this sequence we start from m empty machines with no orders on them and then add operations in the order specified by the chromosome. Each operation is assigned greedily to its machine in such a way that it is executed as soon as possible without violating these constraints: operation $O_{i,j+1}$ cannot be executed before operation O_{ij} is finished and no two jobs can be executed on the same machine at the same time.

The operators that follow were used for the algorithm:

- *Partially mapped crossover (PMX).* This operator can be thought of as a crossover for permutations as it preserves a count of each element in the permutation. This operator has been proposed by Goldberg and Lingle [71] to address the blind travelling salesman problem. The basic algorithm under PMX is the following. Firstly, two chromosomes are aligned and two cut points are selected at random. Sections between these cut points are swapped. In the matching section, numbers that stand across each other define the numbers to be swapped in each chromosome. Figure 4.2 shows an example of insertion mutation.

- *Insertion Mutation.* This operator selects a gene at random and then inserts it to another random position. Figure 4.3 shows an example of insertion mutation.

- *Inversion Mutation.* This operator [70] randomly selects two points in the

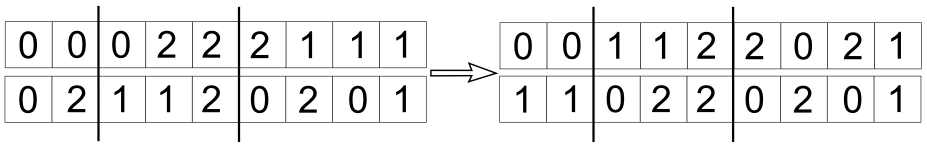


Figure 4.2: Partially mapped crossover (PMX)

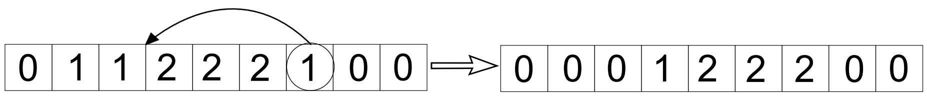


Figure 4.3: Insertion mutation

chromosome and inverts all values in between these points. Figure 4.4 shows an example of this operator.

- *Swap Mutation.* This operator, also known as a reciprocal exchange mutation operator, selects two genes in the chromosome at random and then swaps them. Figure 4.5 shows an example of this mutation.

The first step of the algorithm is population initialisation which creates a population with random individuals. Each individual is assigned a fitness value by an evaluation function which is designed so that its value is indicative of how well the individual solves the problem. For this problem the evaluation function is a makespan of the schedule created from the chromosome taken with the negative sign: $FitnessSched = -Makespan(schedule)$

During the main algorithm loop, individuals with higher fitness values are given more chances to reproduce. The reproduction process is carried out with the help of operators which take one or two individuals and produce offspring. In every generation each of the operators are executed with specified probability. This algorithm uses a steady state population and the tournament selection of size two.

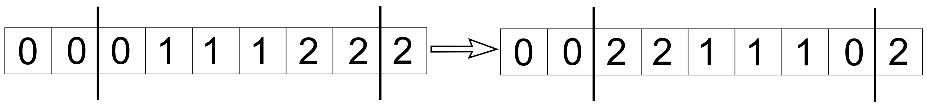


Figure 4.4: Inversion mutation

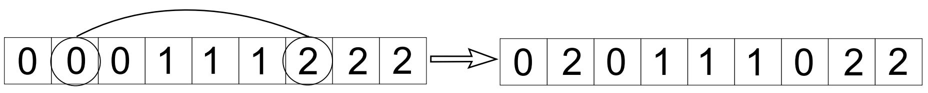


Figure 4.5: Swap mutation

4.2.2 Algorithm for the VRP

An evolutionary algorithm was used to solve the VRP problem. This algorithm is based on a memetic algorithm created by Prins [145].

Each chromosome of the population is the permutation of n clients. In contrast of some algorithms [109] for the VRP this permutation does not contain trip delimiters. The order of genes in the chromosome represents order in which these customers would be visited by a single truck. The optimal splitting procedure *Split* is then used to produce the best partitioning of the trips. It is based on finding the shortest path on the auxiliary graph, as in the work of Beasley [17] about the route-first cluster-second heuristic.

This weighted auxiliary graph $H = (X, A, Z)$ contains a set of $n + 1$ nodes $X = \{0, \ldots, n\}$, a set of edges between nodes A and the trip cost of the edge z_{ij}. Here, edge (i, j), where $i < j$, exists if the trip from customer $i + 1$ to j is feasible, that is, it does not violate the capacity and route length constraints.

An optimal route partition corresponds to the shortest path from the depot, which is node 0, to the customer n in the auxiliary graph. The shortest path can be computed sufficiently fast using Dijkstra's algorithm. A good description of this algorithm can be found in Cormen et al. [48].

The current implementation does not actually build H in memory but rather builds it online getting the list of all neighbouring edges only when needed.

The operators discussed below were used for the algorithm.

- *Order Crossover (OX).* Since each chromosome of the population is a simple permutation, a classic permutation operator such as OX can be applied. It was first proposed by Davis [54]. Basically, it builds offspring by choosing a subsequence from one parent and preserving the relative order of genes from the other. Firstly, two cut points are selected at random, and segments between these points are copied to the offspring. Next, starting from the second cut point of one parent, genes from the second parent are copied in the same order, omitting the symbols already present in the first parent. After reaching the end of the sequence, we continue from the start of the string. Figure 4.6 shows an example of this

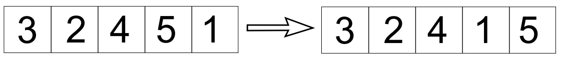


Figure 4.6: Order crossover (OX)

operator.

- *Local Search Operator (LS).* This operator, using the *Split* procedure, first converts the genotype (permutation) to the phenotype which is a set of truck routes. Then, it looks at certain $O(n^2)$ neighbourhoods generated with 9 different trip modifications. For details of these modifications see [145]. Each iteration of LS stops at the first improving modification. After this modification is applied the process is repeated until no modification can be found.

The fitness value of an individual in the population would be a sum of all route lengths taken with the negative sign. In order to avoid premature convergence a special *dispersal property* is maintained in the population. The basic idea of the dispersal property is to have the population sorted in increasing order of fitness values, and the fitness difference between two consecutive individuals set to at least $\Delta > 0$.

$$\forall P_1, P_2 \in \Pi : P_1 \neq P_2 \Rightarrow |F(P_1) - F(P_2)| \geq \Delta \tag{4.3}$$

where Π is a population and F is the evaluation function for the chromosome.

The initial population consists of σ chromosomes generated at random in such a way that they comply with the dispersal property 4.3. They are generated one at a time, and if the new chromosome has a difference with one of the chromosomes already in the population of less than Δ it gets dropped and a new one is generated, otherwise it is inserted into the specific position in the population to keep the population sorted.

During the main algorithm loop, individuals with higher fitness values are given more chances to reproduce. The selection of parents is made with a tournament selection of size two. Every generation two parents are selected, run through the OX operator and then one of the offspring produced is selected. This offspring is improved by the LS operator with a certain probability p_m. After this, if the created offspring does not violate the dispersal property 4.3, it replaces an individual from the second half of the population, i.e the individual with an index greater or equal than $\lfloor \sigma/2 \rfloor$. A steady state population is used for this evolutionary algorithm.

4.2.3 Sequential Approach

The two algorithms discussed above (for JSSP and VRP) produce reasonably good solutions to the problems. They were checked against the standard data sets. These algorithms are used to optimise the proposed model of the supply chain.

For the actual supply chains, the production silo has a manager who is in charge of optimal scheduling of operations and the distribution silo has another manager who tries to minimise transportation costs of the goods received from the first silo. Both managers try to optimise their work thinking only about their own silo without much exchange of information.

In this experiment a simulated supply chain is created that mimics this process. The JSSP algorithm will be run on the dataset of the first silo producing an optimal production schedule. The produced schedule will then be fed into the algorithm of the second silo. In the production silo, the whole makespan is divided into intervals of length τ. These intervals correspond to the production periods when trucks start to deliver goods produced during this period. For orders of each period (bucket), the VRP algorithm is run and results of each run are summed. The total performance of the supply chain is computed using equation 4.1. This method will produce a global solution of the supply chain and will be referred to further in this chapter as the *sequential approach* and abbreviated as *SEQ*.

4.2.4 Cooperative Coevolutionary Approach

Another approach to tackle the aforementioned supply chain problem is based on cooperative coevolution. In this method two algorithms, each corresponding to a single silo, are run in parallel. Cooperation between silos occurs during evaluation of individual solutions. Solutions from one silo are evaluated based on their performance when combined with a representative solution from the other silo and vice versa. The JSSP and VRP algorithms described above are for silos 1 and 2 respectively.

The main difference of the cooperative coevolutionary algorithm from others is the way it evaluates individuals. In order to evaluate the fitness of the individual, several *partner representatives* from another species are selected and combined with the individual, thus forming the global solution for the supply chain. The fitness of the best combination is then assigned to the individual and this partner is saved in the chromosome.

The question here is how to combine the two individuals? A scheduling individual produces a product order in which goods are made. From this product order, job order buckets are extracted. Each job order in a bucket has a customer and each gene of the VRP individual also corresponds to a customer. Therefore job order buckets can be combined with a VRP individual to produce truck routes. To get routing information from a VRP individual corresponding to a particular bucket, only genes that refer to customers from job orders of that bucket need to be considered. All other genes are ignored. Suppose, we have a bucket with orders that need to go to customers 2, 5 and 7 and our VRP chromosome is (5, 2, 3, 4, 6, 1, 7, 8). The modified chromosome for this bucket is (5, 2, 7), so to these orders will be distributed in this exact order. The *Split* is then applied to form the routes. The same combining procedure is applied to all the buckets, then distances of all routes are added and multiplied by the coefficient β to get the total transportation cost. The global cost of the solution is transportation cost plus production cost computed with the equation 4.1.

In the beginning, each of the species is initialised with the random individuals by procedure $initialise()$. As with the VRP algorithm, random initialisation in the VRP species should be done in such a way that the population of VRP species complies with the dispersal property 4.3.

During the main algorithm loop, the JSSP algorithm evolves the scheduling species and VRP algorithm evolves the VRP species in exactly the same way as in the single silo optimisation, with a few exceptions. One difference is that fitness evaluation is done in a cooperative way, as was described above rather than in the sequential way. Each species uses the same operators as were used in the original algorithms, except additional *InsertionMutation*, *InversionMutation* and *SwapMutation* are applied and the *LS* operator improves routes only after solutions are combined and routes created. The function $applyOperators(species)$ applies operators relevant to each species.

The selection of partner representatives is done by the procedure $selectPartnerIndividuals()$ in the following way. For representatives from scheduling species, 3 individuals are selected: the best, and two random ones. For partner representatives from VRP species, te population is divided into three equal intervals and a random individual from each interval is chosen. Since individuals in the population are sorted by fitness values, this approach makes sure that a diverse range of individuals is selected. In addition to 3 random individuals the best individual (the first individual in population) is also selected. Due to the high dependency between the two silos the strictly greedy algorithm of choosing just the best individuals will not produce good results. This dependency effect was studied by Potter [141] and may be explained that, with a best partner strategy, the algorithm gets frozen in the Nash equilibrium.

To give each species some time to adapt to representatives of the other species, partner representatives are not selected every generation; instead they are selected every certain number of generations, called *freezing time* and stored in memory. Each species has its own representative freezing time.

During the evolutionary loop both species use steady state populations and tournament selection of size two for parent selection, as in the original JSSP and VRP algorithms.

Algorithm 2: Coevolutionary global optimisation

```
$gen \leftarrow 0$
for $species \in allSpecies$ do
  $initialise(species)$
end for
for $species \in allSpecies$ do
  $evaluate(species)$
end for
while $gen < maxGen$ do
  $gen \leftarrow gen + 1$
  $selectPartnerIndividuals()$
  for $species \in allSpecies$ do
    $applyOperators(species)$
    $evaluateNewIndividuals()$
  end for
end while
```

In the remaining part of this chapter this algorithm will be referred to with the abbreviation *B+2R* (best plus two random).

4.2.5 Cooperative Coevolution with Non-dominated Parent Selection

One of the main decisions when designing an evolutionary algorithm based on cooperative coevolution is the choice of the method of selecting partner individuals. The approach discussed in this subsection employs an algorithm very similar to the one discussed in the previous subsection, with the difference being the way the partner individuals are selected. This method is based on a non-dominated sorting procedure, that is, sorting all solutions by the level of their dominance relative to the other solutions. With this algorithm, all non-dominated solutions are of level one, then these solutions are disregarded, and with this consideration all non-dominated solutions from the rest of the population are assigned a level two. This procedure is repeated until all solutions are assigned a dominance level.

Each silo in the supply chain can be treated as a separate problem without considering the other silos and, hence has its own objective. The production makespan is the individual objective for the first silo without considering the second one. For the VRP problem the total length of the route is calculated as its objective. However, the objective of the second silo considers orders grouped in time buckets from the random individual of the first production silo. In this way each global solution (i.e. the combination of solutions from the first and second silo) apart from the combined fitness have two additional independent fitnesses.

Partner selection with the current approach works as follows. At each generation all current global solutions are put on the plane according to their two objectives. Then the first non-dominated set of solutions is found. A partner individual is randomly chosen from this set.

In the remaining part of this chapter this algorithm will be referred to with the abbreviation *NDPS* (non-dominated partner selection).

4.2.6 Cooperative Approach with On-the-fly Partner Generation

This algorithm addresses the two-silo problem with a variation of the cooperative coevolutionary algorithm described above. The scheduling individual is produced in the same way and a population of scheduling individuals is stored. In contrast to the previous algorithm, VRP individuals are not stored, and are generated on the fly using Clarke and Wright algorithm [43].

Clarke and Wright's algorithm for n customers starts with n routes, one route per customer. Then the algorithm tries to merge routes to create routes that serve several customers per route, thus reducing the total cost of the solution. This procedure is continued until no further route merge can be made.

In the current approach, the solution for the scheduling problem using an evolutionary algorithm is described in subsection 4.2.1. As with the cooperative coevolutionary approach, evaluation of a scheduling individual is done by combining it with the partner vehicle routing individual. However, in this algorithm a partner vehicle routing individual is produced deterministically during each evaluation, based on the scheduling individual, so the population of VRP individuals is not stored.

When a particular scheduling individual needs to be evaluated, its schedule is divided into time buckets. Then a vehicle routing problem for each bucket is solved and the result is added to the total fitness of the vehicle routing individual and then total fitness is calculated according to the formula 4.1.

In the remaining part of this chapter this algorithm will be referred to by the abbreviation *FLY*.

4.3 Experiments and Results

This section presents the results of experimental runs of the sequential and coevolutinary approaches to the two-silo supply chain problem.

The evolutionary algorithm for this problem was implemented in Java. A set of tests was developed based on the standard data sets for vehicle routing and job-shop scheduling problems. JSSP data sets were taken from instances proposed by Taillard in [163]; in particular ta11, ta35, ta51, ta70 and ta71 were used. For the VRP test cases instances proposed by Christofides et al. [39] were taken, in particular CMT-1, CMT-2 and CMT-3. In table 4.1 the column *Size* corresponds to the number of customers and hence the number of orders in the experiment.

In order to make a data set for the supply chain model one instance of the JSSP data set for the first silo and one instance from the VRP dataset for the second silo, were taken. The following combinations were used: ta11 with CMT-1, ta35 with CMT-1, ta51 with CMT-2, ta70 with CMT-2 and ta71 with CMT-3. This defines job orders, depot location, customers and customer coordinates. The missing link between the two silos is the mapping between job orders and customers that was developed for each data set.

Due to the stochastic nature of evolutionary algorithms, each experiment was executed 50 times, so as to allow for statistically accurate results. The average, standard deviation, minimal and maximum of all runs were recorded. For the total price evaluation from equation 4.1 $\alpha = 1.0$ and $\beta = 2.0$ were taken.

Parameters for the sequential approach are the following:

- JSSP algorithm:
  - Number of generations = 1000000
  - Population size = 50
  - PMX operator probability = 0.7
  - InsertionMutation operator probability = 0.3
  - InversionMutation operator probability = 0.3
  - SwapMutation operator probability = 0.3
- VRP algorithm:
  - Number of generations = 10000
  - Population size = 30
  - Crossover operator was executed every generation
  - Local search probability = 0.2
  - $\Delta = 0.5$

Below are parameters for the coevolutionary approach:

- Number of generations = 200000
- Parameters used for scheduling species:
  - Species size = 20
  - Freeze time = 50
  - PMX operator probability = 0.7
  - InsertionMutation operator probability = 0.5
  - InversionMutation operator probability = 0.5
  - SwapMutation operator probability = 0.5
- The following parameters were used for the VRP species:
  - Species size = 20
  - Freeze time = 50
  - Order crossover operator probability = 1.0

- InsertionMutation operator probability = 0.5
- InversionMutation operator probability = 0.5
- SwapMutation operator probability = 0.5
- $\Delta = 0.5$

These parameters were chosen by the series of manual exploratory experiments. Table 4.1 shows the comparison of the results. In all the data sets we can see that all coevolutionary approaches (B+2R, NDPF, FLY) produce better results on average with smaller standard deviation. In the first dataset B+2R and FLY produce very similar averages (2446.19 and 2442.85) with different standard deviations. The two-tailed p-score of this case is 0.50. This is due to the small data set size. The number of fitness evaluations in the coevolutionary approach is about 1.4 times larger than in the sequential one. However, an increasing number of fitness evaluations in the latter method would not produce significantly better results, as after a number of generations used in the current algorithm it converges to the local optimum. Another issue with the cooperative approach is that fitness evaluation of each pair of individuals is a much more computationally expensive simulation. With the sequential approach, the simulation needs to be run only for the evaluation of VRP individuals.

After comparing the coevolutionary approaches, despite our hopes, the NDPS did not produce solutions of as good quality as B+2R and the FLY. It was also the most computationally expensive approach as a pareto front needs to be recreated (at least partially) for each generation. The advantage of the FLY algorithm is that it is much quicker than the B+2R, due to the deterministic Clarke and Wright heuristic, but it was not superior to the simple B+2R approach, except being slightly superior in the first test case (size 20). But if we take into account its standard deviation, B+2R is a clear winner; in most of the experiments it has a significantly lower standard deviation than the other approaches. This experiment shows also that very often a simple methodology (in this case the selection mechanism) creates better solutions than more sophisticated algorithms.

This experiment confirms our initial hypothesis that cooperation between components while optimisation produces higher quality solutions. Figure 4.7 shows the visual representation of the solution produced by the coevolutionary approach. Figure 4.8 illustrates the solution produced by the sequential approach on the same dataset. It is clear that the second picture has a better quality solution for the JSSP, whereas routings for the VRP are very chaotic;

Experiment	**Size**	**Average**	**Min**	**Max**	**StdDev**
SEQ	20	2603.28	2407.01	2800.69	93.57
B+2R	20	2446.19	2395.45	**2484.06**	**32.07**
NDPS	20	2527.40	2375.39	2702.21	78.69
FLY	20	**2442.85**	**2285.76**	2653.84	83.62
SEQ	30	3761.86	3451.21	4127.81	105.98
B+2R	30	**3514.18**	3389.71	**3597.83**	**68.16**
NDPS	30	3569.85	**3336.29**	3798.74	115.41
FLY	30	3530.78	3358.27	3732.18	89.19
SEQ	50	5941.21	5453.55	6390.89	205.68
B+2R	50	**5658.66**	5372.54	5969.93	157.08
NDPS	50	5840.32	5463.54	6215.18	**142.15**
FLY	50	5760.12	5447.79	6164.37	170.89
SEQ	50	6387.10	6046.69	6729.65	156.76
B+2R	50	**6103.58**	**5732.16**	**6452.84**	144.94
NDPS	50	6287.59	5997.84	6575.29	**141.08**
FLY	50	6169.14	5785.13	6582.39	170.41
SEQ	100	11859.45	11205.47	12532.39	295.91
B+2R	100	**10844.74**	10788.89	10900.58	**55.84**
NDPS	100	11265.19	**10733.88**	**11791.87**	245.41
FLY	100	10863.28	10782.85	11795.49	267.23

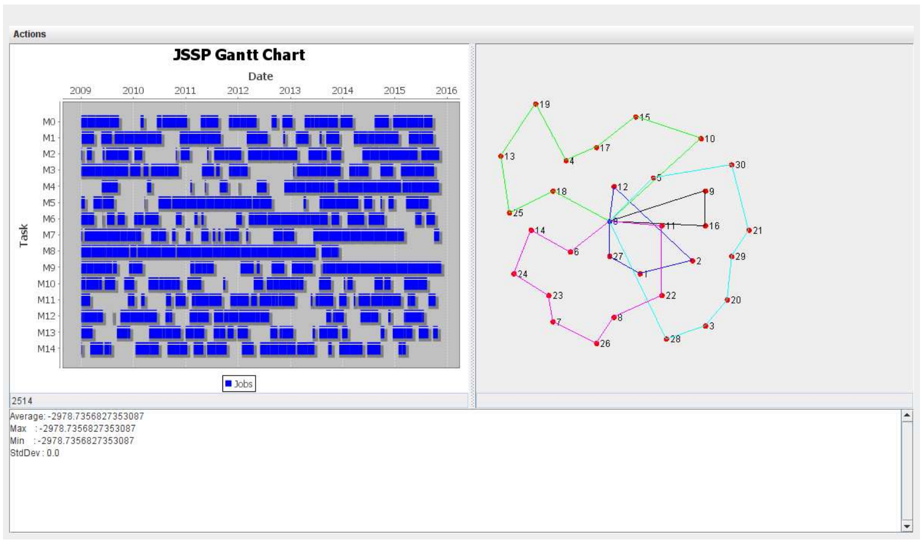
Table 4.1: Run results of the two-silo experiment

customers within the same road are located at quite a distance from each other. Regarding the solution produced by the cooperative coevolutionary approach, the routings are more clustered for a specific area, and the total cost of execution of the second scenario is lower than the first. Here the second solution cooperated with the first to reach a better overall solution.

4.3.1 Optimal Parameter Setting in the Two-silo Experiment

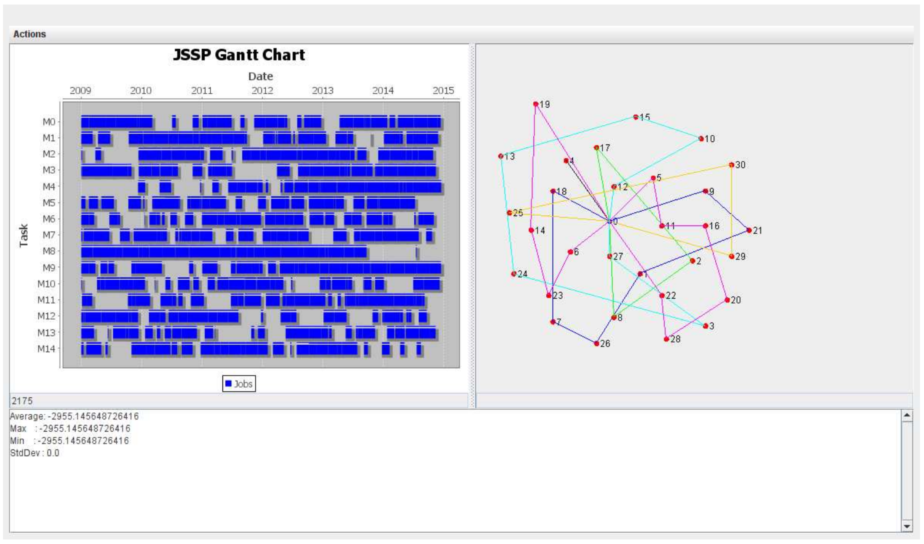
One of the characteristics of algorithms based on cooperative coevolution is that they make it very hard to understand population dynamics and parameter choice. From our experimentation species size was found to be the parameter that most affected the result of the algorithm.

The following experiment has been conducted in order to find out the best parameter setting for the cooperative coevolutionary algorithm that has been used. The experiment was run for all parameter combinations in the range



```
2514
Average: -2978.7356827353087
Max  : -2978.7356827353087
Min  : -2978.7356827353087
StdDev : 0.0
```

Figure 4.7: Solutions produced by cooperative coevolutionary approach



```
2175
Average: -2955.145648726416
Max  : -2955.145648726416
Min  : -2955.145648726416
StdDev : 0.0
```

Figure 4.8: Solutions produced by sequential approach

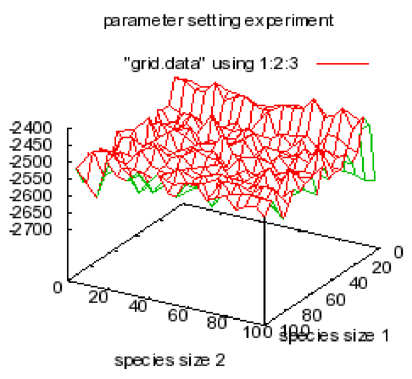


Figure 4.9: 3D grid for parameter setting

from 5 to 100 with a step of 5 for both species sizes. For each parameter pair, the experiment was run 10 times and the results of each parameter set were accumulated and analysed. Ten runs of experiment were conducted for species size pairs (5, 5), (5, 10), (5, 15), . . . , (5, 95), (5, 100), (10, 5), (10, 10), . . . , (10, 100), . . . (100, 100). Here a pair (x, y) means that for the given experiment population size for the first species was x, and for the second was y.

The aggregated result of the experiment has been plotted on the 3D grid and shown in figure 4.9. X and Y coordinates on the plot represent species sizes of scheduling and vehicle routing problems respectively, and the Z axis shows the fitness.

The results show that using a small population size for the scheduling silo produces significantly better results than for other population sizes. Another observation from this grid is that a change in population size of the first species has a more significant effect on optimisation than the second parameter.

4.4 Summary

In this chapter we considered a two-silo simulated supply chain problem with the production and distribution components. Several approaches were employed: a sequential evolutionary and a few variations of cooperative coevolutionary approach. The latter produced higher quality solutions due to the use of cooop-eration between supply chain silos. Differences between cooperative approaches were discussed.

The next chapter upgrades the existing two-silo model to support multiple silos and uses similar techniques based on cooperative coevolution to produce good quality results.

Chapter 5

Two-echelon Multi-silo Supply Chain

The previous chapter described an experimental supply chain with a relatively simple structure – two connected silos. This two-silo model can also be thought of as a two-echelon model with one silo in each echelon. This chapter enhances the previous supply chain by having several silos in each of two echelons with more complex flows between the echelons. The first section presents an artificial multi-silo two-echelon supply chain, and compares several techniques for global supply chain optimisation similar to the ones presented in the previous chapter. One technique employs a sequential evolutionary optimisation of silos, that is, the output of one silo is used as initial input parameter for the connected silos of the next echelon. It is compared with the cooperative coevolutionary approach which optimises all silos in parallel with periodic interaction between the participating components. Each technique is tested on an experimental supply chain. Two principal flow structures of the supply chain are explored: one-to-many and many-to-one flows. Results show that parallel optimisation with cooperation yields higher *quality* solutions.

Since supply chain models have different types of flows and many silos, it becomes complicated to run, test and analyse these models. The right software design decisions need to be made in order to easily modify different experimental runs with the same codebase. Hence, the simulation framework has been developed to address these challenges, and is discussed in subsection 5.1.3.

The second part of the chapter describes a real-world example of multi-silo two-echelon optimisation with slightly different flows between the silos of the two echelons: each silo of the first echelon has one-to-one flow with another silo of the second echelon. This problem is based on an existing Australian company that specialises in producing sheet steel. A proposed solution based on the variation of the cooperative approach with on-the-fly partner generation (FLY), and the experiments conducted, are presented. It also shows the importance of global optimisation from the business perspective: in this problem optimal batching by itself, without consideration of customer locations does not bring much value to the final solution, so the overall supply chain is considered during the optimisation.

5.1 Two-echelon Experimental Model

The following subsection describes an experimental two-echelon supply chain, flows between the components, proposed algorithms and presents a comparison of the results of running these algorithms.

5.1.1 Introduction

The previous chapter discussed the two-silo supply chain which represents one type of material flow between silos in supply chains: a *one-to-one* flow. This is the simplest type of sequential supply chain. However, real-world supply chains may contain more sophisticated kinds of flows where several silos of one echelon process materials concurrently. One of these sorts of flows is the *many-to-one* (MTO) flow. An example of such a supply chain is presented in Figure 5.1. In this figure three silos of the first echelon (S1, S2, S3) process material and then send the product produced to the single silo (S4) of the second echelon. Another type of multi-silo supply chain is the *one-to-many* (OTM) flow. An example of such a supply chain is presented in Figure 5.2. Here, one silo of the first echelon (S1) is connected with three silos from the second echelon (S2, S3, S4). The one-to-many is somewhat more complicated than the many-to-one supply chain, since, apart from decisions made within silos, additional decisions should be made on where in the second echelon to route products of the silo from the first echelon.

Understanding these two types of material flows lets us produce more complex structures, such as the *many-to-many* type of flows in *multi-echelon* supply chains. This chapter investigates methods of optimal decision support in OTM

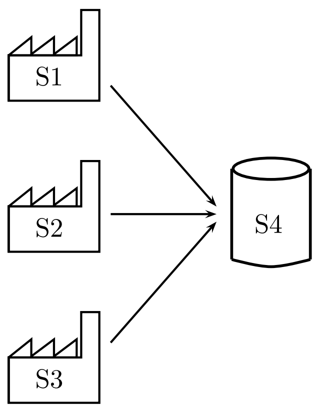


Figure 5.1: Simulated two-echelon supply chain with many-to-one flow

and MTO supply chains so they can be used as building blocks to tackle the complexity of the real-word problems. The ability to apply the right techniques to address problems in these systems is of paramount importance when the organisation wants to take into account larger scale aspects of the supply chain rather than focussing on its particular component. It is a way to move from local to global optimisation, thus taking one step further towards closing the gap between theoretical research and real-world problems.

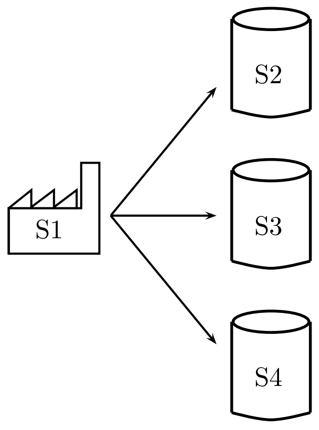


Figure 5.2: Simulated two-echelon supply chain with one-to-many flow

5.1.2 Problem

The two experimental supply chains that will be described in this chapter are based on the two-silo model described in Chapter 4. Each silo of the first production echelon is represented by the job shop scheduling problem, while the function of the second echelon is distribution, so each silo represented by the vehicle routing problem.

The first experiment involves an experimental supply chain model with two echelons and MTO material flow between silos, i.e several production silos, based on job shop scheduling problem, of the first echelon send produced products to the distribution silo based on the vehicle routing problem (second echelon). The description and algorithms for JSSP and VRP were presented in the previous chapters.

The overall MTO supply chain works as follows. There is a set of customers, with their locations and orders they wish to receive. These orders are scheduled at one of the production silos. When the order is ready it is moved to the distribution depot. For this problem it is assumed that it is already known in advance which production plant should be used to produce a particular order. The distribution silo ships goods to customers using a fleet of trucks with the predefined frequency τ (measured in days). The goal of optimisation is to minimise overall expenses that occur during production and distribution, i.e. to minimise the total makespan of all production silos and the total distance travelled by trucks. The total performance of the production silos is measured in time units, and the performance of the distribution silo in kilometres. The overall performance of the solution is calculated converting both measures to the money equivalent:

$$Cost = \alpha \sum X_i + \beta \sum Y_i \tag{5.1}$$

where X_i is a makespan of the produced schedule of silo i in time units, Y_i is a total trip length of all trucks during the production period i, and α and β are conversion coefficients. The goal of the problem is to minimise $Cost$.

The second experimental supply chain model has a very similar functionality of echelon silos but has an OTM topological setup. The production echelon has only one production silo that sends produced items to one of several silos of the distribution echelon.

The overall OTM supply chain functionality is similar to the previous model (MTO). Given a set of customers and their orders, these are scheduled to be produced in the production plant, then the decision has to be made as to which of several existing distribution depots a produced order to be sent for further delivery to the customer. All the distribution silos ship orders to customers with time intervals of τ days. As with the MTO model, the production silo's performance is based on the production makespan and distribution centres measure their performance in the distance travelled by trucks. These two measures are converted to a monetary value by a formula similar to 5.1:

$$Cost = \alpha X + \beta \sum_i \sum_j Y_i^j \tag{5.2}$$

where X is the makespan of the produced schedule in time units, Y_i^j is a total trip length of all trucks of echelon j during the production period i, and α and β are conversion coefficients. The goal of the problem is to minimise $Cost$.

5.1.3 Simulation Framework

An important part of both algorithms presented in this section is *simulation*, which helps in evaluation of *global solutions*. In this chapter we will refer to the *global solution* as a combination of decisions from the local solutions, i.e decisions across the whole supply chain.

A special simulation framework has been implemented to model the processes in the supply chain. The supply chain in this framework is represented as a directed graph with silos as nodes and dependencies between silos as edges. All silos have a set of incoming and outgoing edges and know what silos are on the other end of the edge. Each silo implements a silo specific functionality based on the local solution and sends its output to the virtual object called the *switch*. The purpose of the switch is to implement the logic behind the distribution of the output to the connected set of silos. In the case where there is a single connected silo then the switch just forwards the output to this connected silo (forwarding switch); in the case where there are several connected silos, it uses a heuristic (or even a metaheuristic) to control the distribution of outputs (distributing switch).

The processing logic of each silo accepts an input from three sources: from the predecessor silos, the silo specific data model and the silo solution. The input from the predecessor silos is stored in special input buffers and retrieved only when the silo gets its order to be executed. The silo specific data model stores information about the silo itself, for example the number of production lines (for the JSSP), or the coordinates of depot and customers (for the VRP). The last part, the silo solution, represents the decisions made by the local optimiser.

Before the start of the simulation, the supply chain graph is topologically sorted and silo execution is done in this order. Next, the local solutions are attached to the corresponding silos and execution starts at each silo, one by one, in the topological order. The results of the execution are collected by the global module and can be accessed when needed.

One of the main advantages of creating the simulation framework for these supply chains is its reusability. The framework helps us quickly model and evaluate different types of supply chain – from simple two-silo to more complicated OTM and MTO supply chains with varying numbers of silos in each echelon. In Chapter 7 we successfully reuse the same framework to model five sequential echelons of the wine supply chain.

An additional advantage of the framework is that it provides a single point of evaluation of the quality of solutions. Therefore a solution evaluation of the two approaches discussed in this chapter will be done *using the same evaluation model* despite the difference in nature of these two algorithms. Before running the experiments, the simulation framework was thoroughly tested for correctness. If a separate evaluation procedure would have to be written for each experiment (of which there are 12 in this chapter) each one of them would need to be tested separately, but we would still use potentially different evaluation models when comparing the solution quality of the two approaches.

5.1.4 Algorithm

The two supply chain models that have been described are based on two classical problems: JSSP and VRP. There are many algorithms that address these problems. These two problems and the algorithms tackling them will be used as basic building blocks for global optimisation algorithms that address the supply chain as the whole that most large enterprises require. For both JSSP and VRP, the same algorithms as in the preceding chapter, will be used (see subsections 4.2.1 and 4.2.2). Both algorithms produce reasonably good results on the standard data sets. As in the preceding chapter the algorithmic focus is not to be able to solve the local problems with the best result, but rather to be able to use the existing components to build an optimisation mechanism on top of these components.

Two approaches are taken for both supply chain models: *sequential* and *cooperative coevolutionary.* The next subsections describe them in detail.

The first approach for tackling optimisation in these two supply chain models is based on the sequential approach. In many real-world supply chains each silo is operated by a manager who sends the produced product to the manager of the next silo in the chain. In this way they execute sequential optimisation: from one silo to the next one with the optimal solution within each silo.

The sequential approach follows the same sequential pattern. The JSSP optimisation starts running for all silos of the production echelon (just one instance in the case of OTM), one at a time. Each time period the production silos send the produced items to the distribution silo. In case of the OTM, where there are several distribution silos, a decision has to be made as to which distribution depot items are to be sent. For this problem a simple closest distance measure has been implemented, i.e the item is sent to the distribution point which is the closest to the client. Prince's VRP algorithm [145] is executed at each distribution centre for each time period. The results of all silo optimisations are combined using formula 5.1 for the MTO and formula 5.2 for the OTM supply chain model.

The second approach to address the problem of multi-silo optimisation within the two supply chain models is based on cooperative coevolution. For this global optimisation coevolutionary algorithm, each silo has a corresponding algorithm that runs in parallel with the other algorithms and is managed by the *global module*. The global module coordinates the parallel run of the algorithms, exchanges partner representatives from the species (populations of algorithm at each silo) and controls the termination of the execution of all silo algorithms.

Each silo of the first echelon (production) has a corresponding evolutionary algorithm solving the job shop scheduling problem (algorithm described in subsection 4.2.1). Similarly, each silo of the distribution echelon has an evolutionary algorithm for VRP (Prince's algorithm discussed in subsection 4.2.2). The only change made in these local algorithms is the way the individuals are evaluated. The fitness evaluation is based on the performance of the individual when combined with representative individuals from the other silos.

Representative individuals from other silos are distributed through the global module in the following way. When the population is initialised at each silo, a random individual representative is sent to the global module. When the global module gathers representative individuals from all local optimisers, it then distributes all individuals between the local optimisers. In this way every local optimiser has a copy of the representative individual and starts the solution evaluation. The procedure of exchanging representative individuals will be referred further to in this chapter as *representative synchronisation*. Evaluation is done combining representative individuals, putting them into simulation and executing the simulation. When completed the simulation returns the fitness. After the initial population evaluation the algorithm starts running. At every specified number of generations, the global module kicks off representative synchronisation, but this time local algorithms send one best and one random representative. When running the simulation, the optimiser randomly chooses one of them to put into the simulator. Each such synchronisation represents a start of the new *epoch* of the algorithm.

The reason local algorithms send two representatives instead of one is to provide the diversity. If just the best representative is sent, the global algorithm converges too quickly to the Nash equilibrium. On the other hand, the reason only one random representative is put into the simulation is that simulation is a very computationally expensive process, and the number of evaluations of all combinations would be 2^S, where S is the total number of silos in the supply chain.

Convergence to the Nash equilibrium is one of the biggest issues when dealing with cooperative coevolution. Preliminary test runs of the algorithm have shown that the current implementation of the algorithm still converges prematurely. Some sort of shuffling of populations needs to be done in order to break out from the Nash equilibrium. To achieve this, a special *PopulationEvolver* operator has been created. It kicks off when there is no improvement can be found for a certain number of generations. It takes half of the current population and tries to evolve it for a specified number of generations using a local fitness function. The local fitness function only evaluates the quality of the local solution without considering cooperation with solutions from the other silos. After the local population is evolved it is then put back into the original population and the changed individuals are re-evaluated by the simulation framework (now considering the cooperation with the other solutions).

This cooperative coevolutionary algorithm is run for a certain number of eras. It is general enough to apply to both MTO and OTM models. The following section reports on the results of both the cooperative and sequential algorithms.

5.1.5 Experiments and Results

The described algorithms were implemented in Java. A set of test cases representing both MTO and OTM models was developed based on the standard data sets for the VRP (instances proposed by Christofides et al. [39]) and JSSP (instances proposed by Taillard [163]). Three tests were created for the MTO and another three for the OTM. The following data sets were developed:

- OTM: one JSSP silo + two VRP (1+2). The JSSP silo is represented by *ta40* data set and data sets for VRP are *vrp1*, *vrp2*. Total of 30 orders.
- OTM: one JSSP silo + three VRP (1+3). The JSSP silo is represented by *ta70* data set and data sets for VRP are *vrp1*, *vrp2* and *vrp3*. Total of 50 orders.
- OTM: one JSSP silo + three VRP (1+3). The JSSP silo is represented by *ta100* data set and data sets for VRP are *vrp1*, *vrp2* and *vrp3*. Total of 100 orders.
- MTO: one JSSP silo + two VRP (2+1). The JSSP silos are represented by *ta01* and *ta02* data sets and *vrp1* for the VRP. Total of 50 orders.
- MTO: one JSSP silo + three VRP (3+1). The JSSP silos are represented by *ta01*, *ta02* and *ta03* data sets and *vrp1* for the VRP. Total of 50 orders.
- MTO: one JSSP silo + three VRP (4+1). The JSSP silos are represented by *ta01*, *ta02*, *ta03* and *ta04* data sets and *vrp2* for the VRP. Total of 60 orders.

The parameters used for the sequential approach are the same as in the previous chapter. For the coevolutionary approach the following parameters were used:

- Number of generations = 100,000 with new epoch coming every 30 generations
- Population size of 20 was used for all species.
- These parameters were used for the JSSP species:
  - PMX operator probability = 0.7
  - InsertionMutation operator probability = 0.3

Experiment	**Silos**	**Orders**	**Average**	**Min**	**Max**	**StdDev**
OTM1 Seq	1 + 2	30	3097.68	2874.09	3458.18	195.54
OTM1 CC	1 + 2	30	**2893.16**	**2829.20**	**3412.79**	**163.93**
OTM2 Seq	1 + 3	50	5893.44	5512.92	**6329.40**	**139.18**
OTM2 CC	1 + 3	50	**5801.19**	**5480.85**	6333.76	156.53
OTM3 Seq	1 + 3	100	10342.17	10029.78	11087.40	314.44
OTM3 CC	1 + 3	100	**10240.56**	**9755.64**	**10987.63**	**265.92**
MTO1 Seq	2 + 1	50	3933.92	3694.37	4221.76	132.50
MTO1 CC	2 + 1	50	**3782.48**	**3657.42**	**4193.67**	**121.17**
MTO2 Seq	3 + 1	50	6043.88	5822.95	6390.01	189.03
MTO2 CC	3 + 1	50	**5853.14**	**5635.36**	**6200.22**	**139.66**
MTO3 Seq	4 + 1	60	7801.37	7318.77	8077.48	247.64
MTO3 CC	4 + 1	60	**7514.68**	**7178.61**	**7892.87**	**152.71**

Table 5.1: Run results of the multi-silo experimental supply chain

  - InversionMutation operator probability = 0.3
  - SwapMutation operator probability = 0.3

- The following parameters were used for the VRP species:
  - Order crossover operator probability = 0.7
  - InsertionMutation operator probability = 0.3
  - InversionMutation operator probability = 0.3
  - SwapMutation operator probability = 0.5
  - $\Delta = 0.5$

These parameters were chosen by the series of manual exploratory experiments. For each of the datasets two algorithms were executed: coevolutionary and sequential (total of 12 experiments). Each experiment was executed for 50 times for stochastically correct results. Result average, standard deviation, minimum and maximum values are reported in Table 5.1. Here suffix *CC* corresponds to the cooperative coevolutionary approach and *Seq* to the sequential approach. The bold font shows better result. There is a tie in the second data set. Two tailed p-value for this case is 0.002. This may have happened because of relatively small data set size, all larger datasets have clearly better results for the coevolutionary algorithm.

The results of these experiments confirm that the communication between the components during the cooperative coevolution produces higher quality solutions. Maximum and standard deviation of the second data set are slightly better for the sequential case, but minimum and average are better for the coevolutionary approach.

5.2 Two-echelon Case Study

This section provides an example of real-world supply chain with two echelons. To protect the privacy of the company it will be called Global Steel in this chapter.

5.2.1 Problem

Global Steel has several plants across Australia and specialises in producing sheet steel. Sheet steel products are basically thin sheets of steel that have been coiled into rolls. The company can manufacture many different types of products, that differ in chemical composition, density and width. Furthermore, the rolls can be configured with different diameters and cores (see Figure 5.3). When a plant manufactures a roll of steel, the width of the roll cannot be changed. However, customers typically place orders for a smaller width, and so, large rolls must be cut into pieces along their width to satisfy these orders. This requires orders of similar configuration to be grouped together before the cutting of the sheets (see Figure 5.4).

The constraints and business rules of this problem are listed below; many of them are similar to the previous problem.

- Each plant's daily work hours are limited.
- Each plant can produce a subset of all possible goods. Some products can be produced at multiple plants, but the cost and rate of production is different at different plants.
- Each plant is equipped with a particular number of 'knives', which are used to cut large rolls into smaller ones. Each plant can only produce sheets of fixed width and has special knives to cut to the width needed.
- Each plant has a fixed daily operating cost which is the cost of running the plant, and does not depend on the actual products produced.
- After production, products are delivered to customers, and this incurs a transportation cost. There are defined transportation costs between each

customer and plants. Some plants cannot ship to some customers due to specific business rules.

- Due to shipping constraints, export orders must be produced and shipped from a single plant, whereas for others orders, part of the order can be produced at one plant and other parts at other plants.
- Orders are produced in batches by the plants; this will be explained in detail later in this section.

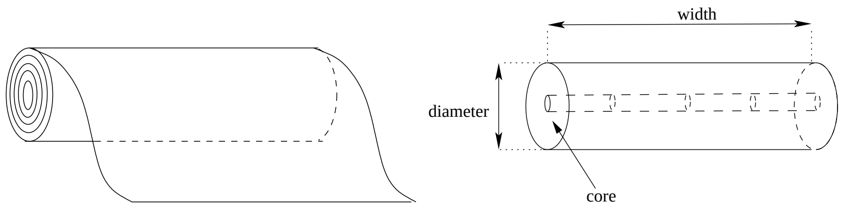


Figure 5.3: A roll of sheet steel.

In order to understand the problem better, let us look at a simplified example. Suppose we have 5 plants: P_1, P_2, P_3, P_4, P_5 and 7 customers C_1, C_2, ..., C_7. Let us say that customer C_2 has an order for 5 tonnes of product I_1, and 8 tonnes of I_3. Customer C_6 wants 15 tonnes of I_1, 3 tonnes of I_2, and 10 tonnes of I_3. I_1 can be produced only at P_2, I_2 can be produced at P_1 and P_4 and I_3 can be produced at P_2, P_3 and P_5 although with different costs of production. (See Table 5.2(a)). Table 5.2(b) shows the products that can be produced at each plant, as well as the tonnes per hour rate of production for each.

Suppose that we have to complete these orders in a time frame of 100 hours. Let us look at one possible way of distributing this workload amongst the plants. As I_1 can only be produced at plant P_2 we are forced to assign all 100 tonnes of I_1 there. This then implies that the required 30 tonnes of I_2 must be produced at P_4, using up 20 hours of the available time. 280 tonnes of I_3 still need to be produced. P_2 could produce 200 tonnes of it, P_3 could produce 150 tonnes and P_4, because it only has 80 hours of available time left, could produce 120 tonnes. Thus the 280 required tonnes of I_4 can be split amongst these three plants in various ways.

Plants can produce rolls of sheet steel of a certain width only, which is generally much larger than the widths typically required by customers, and therefore, ordered items that have the same type, core, and diameter must be

	Product		
Customer	I_1	I_2	I_3
C_2	50		180
C_6	50	30	100
Total	**20**	**3**	**18**

(a)

	Plant			
Product	P_1	P_2	P_3	P_4
I_1	1.0			
I_2	1.0			1.5
I_3		2.0	1.5	1.5

(b)

Table 5.2: (a) Demand table: The quantity of each item ordered by each customer. (b) Production capacity table: Tonnes per hour production rates of each plant for different products.

batched into a single roll. Then a large roll is cut into pieces to satisfy these orders. This is illustrated in Figure 5.4(a).

This process of batching sometimes results in wastage. To illustrate, suppose that customers C_1 and C_2 need rolls of product I_1, of the same diameter and with the same type of core. C_1 needs 2 rolls of width 500 mm, and C_2 needs 4 rolls of width 600 mm. Let us assume that the only plant that can produce I_1, must produce rolls of steel of width 2000 mm. In this situation, batching can be done as shown in Figure 5.4(b). The plant can produce two rolls of I_1 of width 2000mm. One can be cut into 4 pieces: 3 pieces of 600mm and 1 piece of 200mm. The 3 pieces of 600mm can be used for orders, but the remaining 200mm is wasted. It may be recycled, but a loss is incurred nevertheless. The second roll of 2000mm can be cut into widths of: 600mm, 500mm, 500mm, 400mm, the first 3 of which would go towards satisfying orders, and the last piece ends up as wastage. Some wastage is unavoidable, but clearly it is desirable to have as little wastage as possible in order to maximise production efficiency and hence profit. Intelligent grouping of orders can minimise wastage.

Plants use cutting tools called knives, physically arranged in parallel, to cut large rolls into smaller ones. Different plants are equipped with different numbers of knives, and this presents another constraint to consider when batching orders. A plant equipped with n knives can cut a roll into at most $n+1$ pieces. In the above example (Figure 5.4(b)) cutting a large roll into four small ones, would have required three cuts and hence three knives.

The goal of the problem is to find a set of assignments of orders to plants that optimises the overall profit of the company. Profit is determined by the following formula:

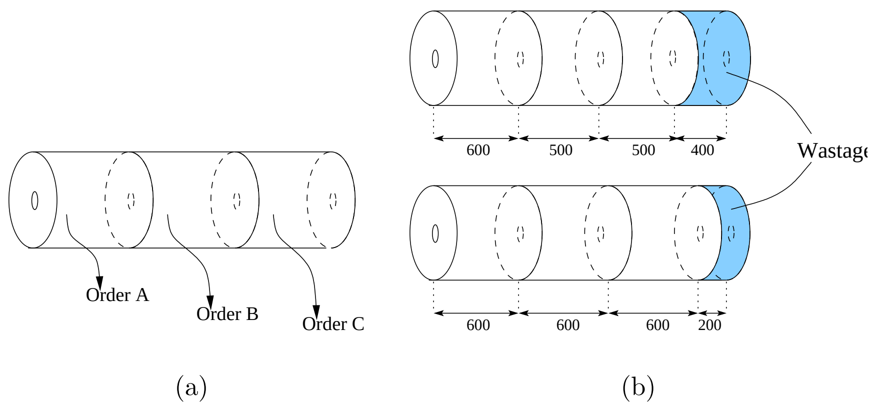


Figure 5.4: (a) Batching of orders into a single roll. (b) Wastage resulting from the batching process. 4 rolls of 600mm, and 2 of 500mm needed.

$$Profit = S - C_m - C_t \tag{5.3}$$

where S is *selling price* of the goods produced, C_m is the *cost of manufacturing* those goods, and C_t is the *cost of transporting* the goods to customers. Selling prices may vary for the same product depending on the market in which it is being sold. The cost of manufacturing a particular order item is the sum of the cost per hour of producing that grade of sheet steel, and the fixed hourly cost of running the relevant plant for the time it takes to produce the required tonnage. Transportation costs vary with the destination address of the customer. Export orders naturally incur higher transportation costs. These last two components are the negative components of the profit; the bigger they are, the less is the profit. The selling price is the positive component of profit; it is known before hand, and does not depend on which plant an order is allocated to. Therefore, the goal of the problem is to minimise the transportation and manufacturing costs.

This problem is a typical example of the multi-silo two-echelon supply chain. All *production* activities within plants form the first echelon and each plant can be treated as a single production silo. The next echelon is comprised of *batching and distribution* silos where, firstly, rolls are cut, and then the cut pieces distributed to the customers. Each pair of the production and the corresponding batching and distribution silos is very similar to the two-silo concept described in chapter 4, except that the current model puts more emphasis on the batching process rather than the distribution (which is straightforward and does not present optimisational complexity). Since the current model consists of several production silos, the original two-silo model is upgraded to a two-echelon model with one-to-one dependencies between components. Figure 5.5 shows the diagram of this supply chain.

The next section describes the algorithm to address the problem of steel production. The approach is based on the on-the-fly (FLY) cooperative approach described in Chapter 4.

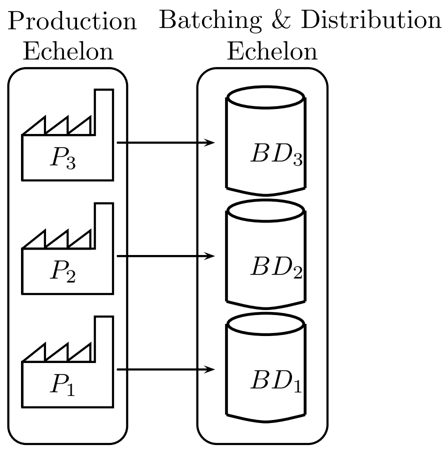
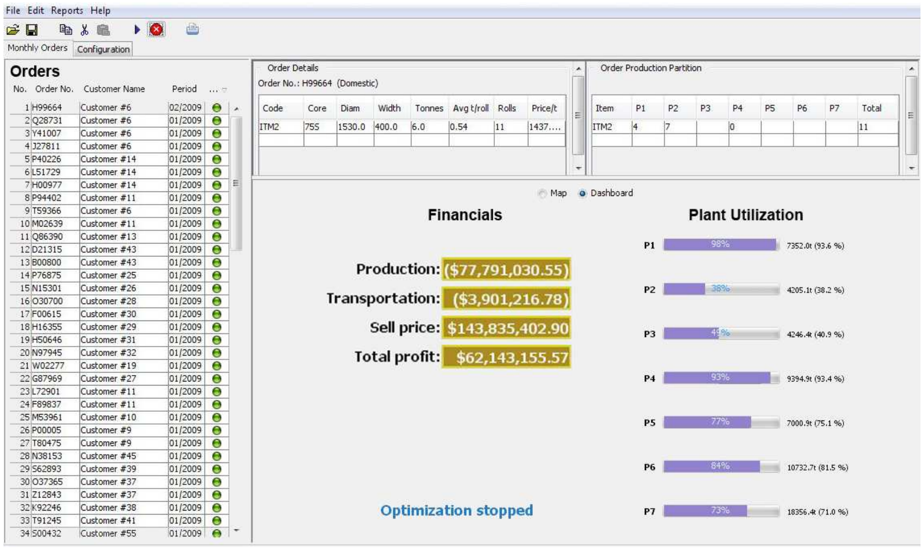


Figure 5.5: Two-echelon multi-silo supply chain with one-to-one dependencies between components

5.2.2 Solution

The software created for this problem used an evolutionary algorithm to find a solution. It is shown in Figure 5.6. The software is fully configurable to cater for various selling prices, transportation and manufacturing costs. Given the projected orders for an upcoming month, it seeks an optimal set of assignments that maximises the profit of the company. Being a decision support tool, the software also provides financial comparisons with human-proposed allocations. In practice, the software suggests decisions that have better financial bottom lines.

The algorithm can be divided into two parts, one for each echelon. The first part seeks the optimal allocation of all orders to plants, whereas the second provides support for batching of orders and their further distribution.



Orders

No.	Order No.	Customer Name	Period	...
1	H99664	Customer #6	02/2009	●
2	Q28731	Customer #6	01/2009	●
3	Y41007	Customer #6	01/2009	●
4	J27811	Customer #6	01/2009	●
5	P40226	Customer #14	01/2009	●
6	L51729	Customer #14	01/2009	●
7	H00977	Customer #14	01/2009	●
8	P94402	Customer #11	01/2009	●
9	T59366	Customer #6	01/2009	●
10	M02639	Customer #11	01/2009	●
11	Q86390	Customer #13	01/2009	●
12	D21315	Customer #43	01/2009	●
13	B00800	Customer #43	01/2009	●
14	P76875	Customer #25	01/2009	●
15	N15301	Customer #26	01/2009	●
16	O30700	Customer #28	01/2009	●
17	F00615	Customer #30	01/2009	●
18	H16355	Customer #29	01/2009	●
19	H50646	Customer #31	01/2009	●
20	N97945	Customer #32	01/2009	●
21	W02277	Customer #19	01/2009	●
22	G87969	Customer #27	01/2009	●
23	L72901	Customer #11	01/2009	●
24	F89837	Customer #11	01/2009	●
25	M53961	Customer #10	01/2009	●
26	P00005	Customer #9	01/2009	●
27	T80475	Customer #9	01/2009	●
28	N38153	Customer #45	01/2009	●
29	S62893	Customer #39	01/2009	●
30	O37365	Customer #37	01/2009	●
31	Z12843	Customer #37	01/2009	●
32	K92246	Customer #38	01/2009	●
33	T91245	Customer #41	01/2009	●
34	S00432	Customer #55	01/2009	●

Order Details

Order No.: H99664 (Domestic)

Code	Core	Diam	Width	Tonnes	Avg t/roll	Rolls	Price/t
ITM2	75S	1530.0	400.0	6.0	0.54	11	1437....

Order Production Partition

Item	P1	P2	P3	P4	P5	P6	P7	Total
ITM2	4	7		0				11

Financials

Production:	($77,791,030.55)
Transportation:	($3,901,216.78)
Sell price:	$143,835,402.90
Total profit:	$62,143,155.57

Plant Utilization

P1	98%	7352.0t (93.6 %)
P2	38%	4205.1t (38.2 %)
P3	4[illegible]%	4246.4t (40.9 %)
P4	93%	9394.9t (93.4 %)
P5	77%	7000.9t (75.1 %)
P6	84%	10732.7t (81.5 %)
P7	73%	18356.4t (71.0 %)

Optimization stopped

Figure 5.6: Steel optimiser's main screen.

The representation used for the evolutionary algorithm consists of two parts. The first part is a string of quantities assigned to each capable plant for various order items. Figure 5.7 illustrates the structure of this representation hierarchically. It can be visualised as a string of s records, one for each order being considered. In turn, each order record can be thought of as consisting of a number of smaller records, one for each item in that order. And finally, each order item record can be thought of as consisting of a number of allocation records, one for each plant that is capable of producing the item under consideration. Such a lowest-level record is called a *plant quantity allocation (PQA)*. More formally, assuming that there are s orders to be satisfied, a candidate solution is represented as $X = (x_1, \ldots, x_s)$, where for $1 \leq i \leq s$, $x_i = (y_1, \ldots, y_{m_i})$, where m_i is the number of items in order i. And further, for $1 \leq j \leq m_i$, $y_j = (z_1, \ldots, z_{k_{ij}})$, where k_{ij} is the number of plants capable of producing item j of order i. Each z_{ij} is a tuple (OI_i^j, p, q); it is a plant quantity allocation in which plant p is allocated q rolls of item j in order i.

The second part of the representation is a string of permutations of the plants in the system. Each permutation represents the preferred order of the plants for use in export orders. This can be formalised as $X_{pref} = \{x_{pref_1}, \ldots, x_{pref_s}\}$, where each element x_{pref_i} is a permutation of the set of plants associated with order i. Figure 5.8 illustrates the second part of the representation.

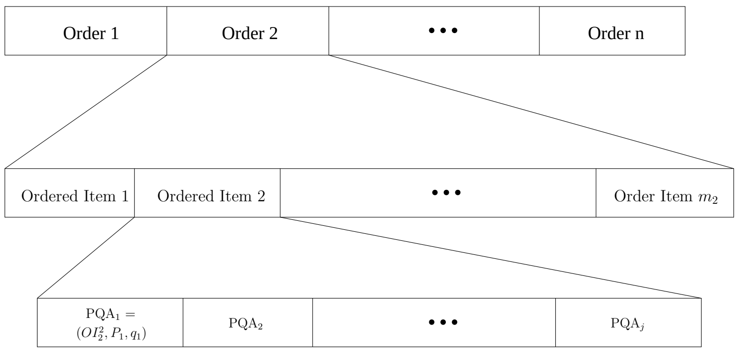


Figure 5.7: First part of the representation of an individual.

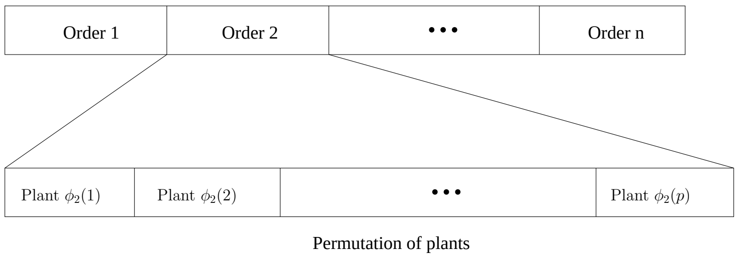


Figure 5.8: Second part of the representation of an individual.

At the start of the evolutionary algorithm, a population of randomly initialised individuals is created. First, each order item cell in Figure 5.7 is initialised by uniformly randomly dividing the number of rolls of steel required for that order item into the PQA cells lower down in the hierarchy. Second, each order cell in Figure 5.8 is assigned a randomly generated permutation of the steel plants.

After the population of individuals is initialised, the evolutionary algorithm enters the main loop in which the population is *evolved* until a specific criterion is satisfied. Each individual is assigned a fitness value by an evaluation function which is designed so that its value is indicative of how well the individual solves the problem. The evaluation functions in our work make use of penalty functions which essentially assign a value to an individual in proportion to how badly the individual violates some constraint.

During the main loop, individuals of higher fitness are probabilistically given more chances to *reproduce*, i.e. to make modified copies of themselves to replace them if the modified copies have better fitness values. The changes to be made to an individual are accomplished by means of *operators*, which are functions that take one or more individuals and combine, alter and manipulate their structures in a partially random way. After operators are applied, some of the resulting new individuals are passed through *repair* functions, that attempt to randomly, but intelligently alter their structure so as to reduce the extent of their constraint violations.

Not every candidate solution, that can be constructed as described above, would represent a feasible solution. For example, if in a candidate solution, a plant is assigned more work than it could possibly produce in the time frame under consideration, then the solution is infeasible. Many techniques are known to handle different types of constraints [122] and most of them are based on penalty functions. The approach used for this algorithm is also based on penalty functions. Penalties were used to deal with both hard and soft constraints. Penalty functions for hard constraints were given larger weights than those for soft constraints. The penalty functions developed were:

- *the quantity production penalty:* This penalty is used to avoid underproduction or overproduction of an ordered item. If an order item requires N_1 rolls of a given product, and the sum of the PQA quantities for this order item is N_2, then this penalty function returns a value that is proportional to $|N_1 - N_2|$. This is a soft constraint because it is not always possible to produce the exact amount demanded, as plant capabilities are limited and fixed.

- *the plant capacity violation penalty:* If the amount of work allocated to a plant exceeds its capacity for the time period, then this function penalises the solution in proportion to the over-allocation. This is a hard constraint because a plant cannot produce more than its capacity.

- *the wastage penalty:* Wastage is possible as a result of the batching procedure that occurs during production. This function penalises in proportion to the number of tonnes of wastage. This is also a soft constraint.

- *the export order single source penalty:* Orders for export customers should be produced at the same plant, and this function penalises export orders that use multiple plants. This is a hard constraint.

Each candidate solution is assigned a fitness value by an *evaluation function* that returns a value that is indicative of how well that candidate solves the problem. For this problem, the fitness function is defined as:

$$f(X) = Profit(X) - \sum_{i} (R_i \times Penalty_i(X)) \tag{5.4}$$

where X is a candidate solution, and R_i is the coefficient of the penalty function $Penalty_i(X)$. $Profit(X)$ is calculated using three components: manufacturing cost, transportation cost and selling price, as described previously. Our evolutionary algorithm tries to maximise this function $f(X)$, i.e. maximise profit, and minimise constraint violations.

An evolutionary operator takes one or more candidate solutions and transforms it (or them), into a new candidate solution. This is typically done by using partially random manipulations of the representation. Operators are employed probabilistically. We used several operators to solve this problem. Some of them look like the classic mutation and crossover, and some of them are modified versions of these two. The operators used are the following:

- *Quantity crossover:* This is a classic crossover-like operator. It takes two individuals, randomly chooses a crossover point at the boundary of an order item cell, and performs a crossover at this point.

- *Quantity mutation:* This operator is similar to the classical mutation. It takes an individual, and probabilistically chooses several PQA cells and changes their quantity values by random amounts within a certain range. The changed amount is chosen randomly in the range $[-\alpha, \alpha]$, where

$$\alpha = \frac{quantityOrdered - quantityAllocated}{|\lambda(OI)|} \tag{5.5}$$

  where $\lambda(OI)$ is the quantity required for a particular order item.

- *PQA Shuffling:* This operator randomly chooses one or several ordered items and shuffles quantities assigned to plants within each ordered item. It preserves the sum of quantities within one ordered item before and after the shuffling. So, if we randomly choose ordered item OI_k then $G(OI_k) = G'(OI_k)$, where G is the sum of all quantities of ordered item OI_k assigned to all capable plants before applying the shuffling operator, and G' is the sum after applying it.

- *Preferred plant operator:* This operator looks at the second part of the representation of an individual, namely the permutation of plants, and performs a random transposition on it, i.e. it randomly swaps two elements of the permutation.

A *repairer* is an operator that tries to modify a candidate solution in such a way as to eliminate or reduce the number of constraints that it violates. In our evolutionary algorithm we use a set of problem-specific *probabilistic repairers*. The main difference between these and the classical repairers is that they are not executed each time a solution gets evaluated, but instead are applied probabilistically, which allows for tuning for better performance. The repairers used are:

- *the quantity allocation repairer:* This repairer tries to fix errors in over- and under- allocation of a particular order item amongst capable plants. It works in the following way. We choose an order which has underproduction or overproduction and increase or decrease the quantity produced at each plant. The increment or decrement are not random. The algorithm decides on the changes to be made. This will increase the fitness of individual, but being a random change, some diversity will be maintained.

	P_1	P_2	P_3	P_4	Total
Quantity before repair	6	13	16	9	44
Quantity after repair	5	11	15	8	38

  For instance, suppose that for a particular order item, 36 rolls are needed, but according to an individual about to be repaired, the capable plants have been allocated 44 rolls. The second row shows what might be the result of applying the quantity allocation repairer. Some values increased, some decreased. The new solution is still not perfect, but overproduction has been reduced.

- *the overallocation repairer:* This repairer is different from the quantity allocation repairer. It checks the *total* workload allocated to a plant, to see if it exceeds that plant's capacity for the time period. If it does, the repairer randomly chooses some order item that uses the plant in question, and reduces its quantity allocation by one. It repeats this procedure until the plant can produce its assigned amount in the given time period.

- *the export order source repairer:* This operator checks for export orders in the individual that use multiple plants, and performs repairs according to the preferred plant permutation. For any such order, it tries to assign the whole work to the first capable plant in the permutation list.

- *the final repairer:* Unlike previous repairers, this repairer is run only at the end of evolutionary algorithm to "polish" the solution. It uses non-probabilistic versions of previous algorithms, since at the end of the algorithm, we want to make solutions as good as possible without worrying about diversity of individuals.

One of the challenges of this problem is grouping ordered items into batches; which is a second echelon of the supply chain. The problem is known in the literature as the *stock cutting problem*. Recall that the problem is the following. We have a set of ordered items all with the same product, core, and diameter, but with different widths. These must be allocated to a plant that can produce rolls of a fixed width which can later be cut into smaller rolls to satisfy the orders. For convenience, let us refer to these large fixed-width rolls as *max-rolls*. We need to group the ordered items into batches in such a way that the number of max-rolls produced by the plant is minimised, thus minimising the wastage.

For this problem, a greedy deterministic approach was taken. The algorithm looks at all order items that are assigned to a particular plant, groups those that are compatible (i.e. that share the same product, diameter and core) and forms batches in the following way. The order item with the largest width is assigned to the first max-roll. Some of the width of this max-roll may be unassigned. The next largest order item is considered next. If there is a max-roll in which it can fit, it will be placed there, otherwise a new max-roll will be created. The same procedure is repeated for all order items. The pseudo-code of this algorithm is shown below:

The procedure *canFindAvailableSpaceInMaxRolls(orderItem)* iterates through all available max-rolls and checks if the given *orderItem* can be placed in a max-roll. If it can find such a max-roll, it will return true, if no such max-roll is available, the procedure returns false. The procedure *placeItemInAvailableSpace(item)* places a given *item* in the available roll that has been found by *canFindAvailableSpaceInOrders*.

The batching algorithm is executed for every individual fitness evaluation and contributes to several penalty functions. This is the same concept as was applied in the two-silo model in Chapter 4 where Clarke and Wright was executed at every fitness evaluation instead of the batching algorithm. The next

Algorithm 3: Batching algorithm

```
$maxRolls \leftarrow \emptyset$
for $group \in orderItemGroups$ do
  $sortByWidth(group)$
  for $orderItem \in group$ do
    if $canFindAvailableSpaceInMaxRolls(item)$ then
      $placeItemInAvailableMaxRoll(item)$
    else
      $newMaxRoll \leftarrow createNewMaxRoll()$
      $newMaxRoll.add(orderItem)$
      $maxRolls.add(newMaxRoll)$
    end if
  end for
end for
```

section presents results of the current evolutionary algorithm and compares the produced results with the company's production solution.

5.2.3 Experiments and Results

The evolutionary algorithm in this study was implemented in Java, and was tested on real world industrial data. The purpose of the software is to provide decision support to business managers who approve production allocation.

In this subsection we present the results of experimental runs of this algorithm with a view to gaining an understanding of how the algorithm solves the problem.

An experiment was performed on a set of real-world data for a particular month. Due to the stochastic nature of evolutionary algorithms, the experiment was executed 90 [1] times so as to allow for statistically accurate results. The specific parameters used for the experiment are shown below. The penalty coefficients were tuned for the business entity under consideration, and would perhaps not be ideal if this algorithm were to be replicated for another industry or business environment. In such a case, it is likely that re-tuning of the coefficients would be necessary. As such, it would not be prudent to try to interpret the significance of these penalty coefficient values.

- Population size = 250

[1]It was meant to be 100 runs but my mistake we put 90 and we realised this only when the experiment finished and the results were automatically computed.

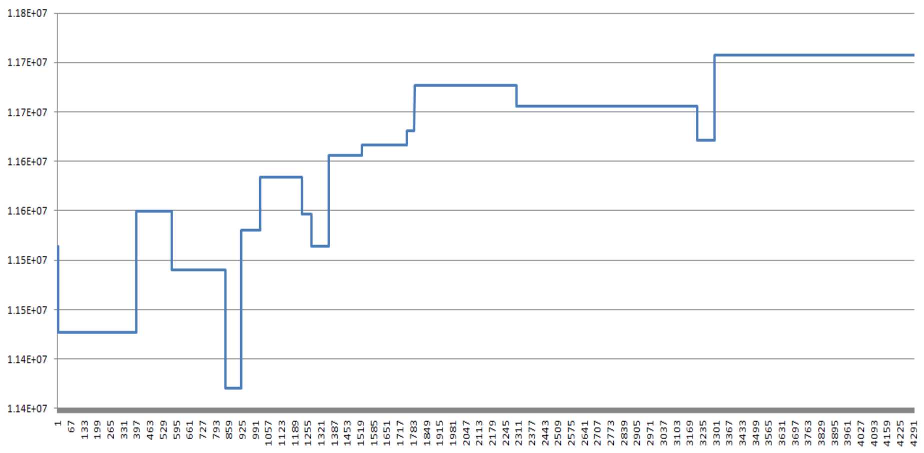


Figure 5.9: Sample run. Profit over generations.

- Number of trials = 90
- Mutation probability = 0.3
- Overallocation repair probability = 0.1
- Underproduction penalty coefficient = 10 000 000
- Overproduction penalty coefficient = 100 000
- Wastage penalty coefficient = 1
- Plant capacity violation penalty coefficient = 1 000 000 000
- Export order multi source penalty coefficient = 10 000 000 000
- Transportation penalty coefficient = 1
- Tournament selection of size 2
- A steady state population is used

Figure 5.9 illustrates the evolution of the *profit* variable during the lifetime of a single run of the experiment. From time to time, there is a decrease in the profit value; this corresponds to instances where the algorithm is able to find a solution that dramatically reduces constraint violations, but at the expense of less profit being generated. However, based on the penalty values being generated, the solution is still considered to be superior to what was previously

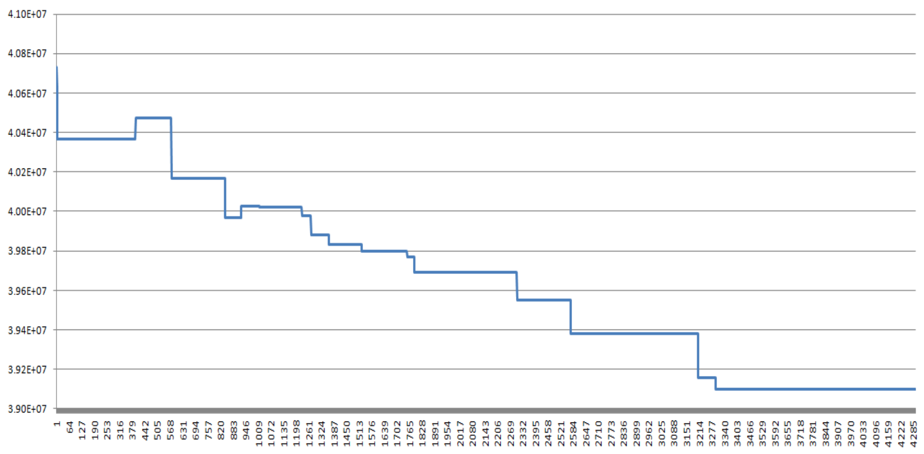


Figure 5.10: Sample run. Penalty function over generations.

in hand. However, the algorithm is eventually able to produce highly profitable solutions.

Figure 5.10 shows the decrease in the summed penalty function values over the lifetime of a single run of the experiment.

A 90 run execution shows median 1.18E+07 and 52110.52404 standard deviation of profit.

The assignments generated by our software system produces financial results that are 4% better than those produced manually by the company's experienced team of planners.

5.3 Summary

In this chapter we have considered a more sophisticated supply chain with multiple silos and one-to-many and many-to-one flows. As in the previous chapter the functionality of the supply chain was based on scheduling and distribution.

Special attention is devoted to the architecture of the optimisation system. A simulation framework is built for evaluation of global solutions. Two main advantages of the simulation framework are the reusability of the framework for different supply chains, and a single point of evaluation; this makes a fair comparison of models. The software architecture of the optimisation systems is often neglected in the research publications but it is of paramount importance when dealing with optimisation in real-world applications. Applying the right software architecture for the optimisation systems speeds up the development, makes it easier to experiment with different variations of the problem, reduces chances of error and in this way lets the architected software solution be more naturally applied for the real-world systems.

Two algorithms addressing the presented supply chain models are discussed: sequential and cooperative coevolutionary. Six different supply chains are considered for experimentation, both MTO and OTM. Each of the two algorithms were executed on these supply chains and the results show that cooperative approach beats the sequential.

The second part of the chapter presents a two-echelon supply chain based on real-world problem from the industry that produces sheet steel. It has multiple silos within each echelon with one-to-one flows between the silos of the first and second echelons. A variation of the FLY algorithm is proposed to tackle this problem. The solution produced satisfactory results, beating the current production schedule by 4%.

Chapter 6

Two-echelon Multi-silo Real-world Case Study

The previous chapter presented a multi-silo two-echelon supply chain with different material flows between the silos. This chapter describes a two-echelon supply chain problem from the mining industry. It is a mine planning problem which involves the existence of several diggers in the mine and a few plants that crush, process or store ore. The problem is how to schedule diggers, trucks, and plants, and how to route the ore in order to get enough tonnage of a product of a required quality. The supply chain is two-echelon with a one-to-many flow, but the difference between the previous supply chains is that there are dependencies between the silos of the second echelon.

The first section of the chapter presents some basic concepts about mining and provides a literature review. Section 6.2 explains a mine planning problem in detail, and the following section 6.3 explains the mathematical model of the problem as well as a model from the supply chain perspective. The algorithm used to tackle the mine planning problem is described in section 6.4, and the following section shows the results. Section 6.6 gives an overview of the functionality of the built software. The last section concludes the chapter.

6.1 Basic Concepts and Literature Review

Mining is the process of the extraction for profit of valuable minerals or other geological materials from the earth. The following major types of minerals are commonly mined.

- **Metallic ores**. These include the *ferrous* metals (iron, manganese), the *base* metals (copper, lead, zinc), the *precious* metals (gold, silver, platinum) and the *radioactive* metals (uranium, radium).
- **Nonmetaillic minerals**. These include phosphate, limestone and sulphur.
- **Fossil fuels**. These include coal, petroleum, and natural gas. Extraction of petroleum and gas which have different physical characteristics requires a different mining technology.

Mining is amongst the most profitable of industries, and therefore proper planning and scheduling is vital for optimal mine exploitation. There are two main ways to recover materials from the ground: *surface mining* and *underground mining*. In this chapter we will be focusing on the former. Many problems arise in open pit mining: orebody modelling, creation of the life-of-mine (LOM) schedule, determination of mine equipment requirements and optimal operating layout, and the optimal transportation of ore from pit to port. Each one of these problems is very complex so the operation of a large open pit mine is an enormously difficult task.

In terms of the time horizon, there can be *operational*, *short term*, *medium term*, *long term planning* (LTP), and *life of mine* (LOM) planning. The operational and short term planning involve a very detailed scheduling of processes and equipment, where it is important to know what each piece of equipment should be doing during each day and even each hour. It also operates based on fairly precise data. In contrast, the LTP involves looking at a global picture, working with estimated data, various uncertainties, risk analysis, and evolving economic criteria. This research is mostly focused on the LTP, but future developments may include short term planning as well.

Hustrulid and Kuchta [87] identified five important stages of mining.

1. **Prospecting**. At this stage geologists use rough methods to discover mineral deposits such as a visual examination of earth and gravitational, seismic and magnetic measurements.
2. **Exploration**. This stage uses more refined techniques to determine the size and the value of a mineral deposit.

3. **Development**. This stage involves obtaining rights for the land and solving additional legal issues regarding the mining. The mine is prepared for excavation by stripping the overburden and other kinds of preparation for the main digging phase.

4. **Exploitation**. During this phase the actual recovery of minerals from the earth takes place; this is normally either *surface mining* or *underground mining*.

5. **Reclamation**. At the last stage the area mined is restored to its natural state.

In their book Hustrulid and Kuchta [87] provide detailed description and describe the challenges of each of these stages. Optimisation techniques have been applied mostly during the development and exploitation stages. Researchers started to get interested in mining from the early 60s.

Caccetta and Hill [30] discuss some techniques for LOM optimal production schedule. They describe mixed integer linear programming (MILP) model and give a survey of several methods to address the problem, including parametrisation method and a method based on Lagrangian relaxation. This model has some basic elements of the model proposed in the current research. The authors also review some commercial systems including Earthworks NPV Scheduler, MineMax and XPAC. Several heuristic based approaches are reviewed (for example [167]). However, they acknowledge that real world problems are too large for the model based on MILP. The authors explain their method based on the *branch and cut* algorithm and compare results with the ones produced by MineMax. In their later work Caccetta and Hill [31] use the *branch and bound* method to solve the problem based on MILP. They use a combination of best first search and depth first search to achieve a “good spread” of possible pit schedules whilst benefiting from using depth first search. Good lower bounds are obtained using the LP-Heuristic. The authors use CPLEX to solve LP subproblems. Bley et. al [20] use the same model formulation as Caccetta and Hill while strengthening the problem by adding inequalities derived by combining the precedence (knapsack substructure) and production constraints. The goal is to maximise the net present value (NPV).

Lizotte and Elbrond [106], and Yun and Yegulalp [181] tackle the mine scheduling problem with techniques based on the dynamic programming. Smith and Tao [159] address the multi-objective phosphate mine scheduling problem via goal programming techniques.

Handling uncertainty and risk analysis is another important topic in mining. Dimitrakopoulos and Ramazan [59] address a problem of complex multi-element scheduling in nickel-cobalt open pit mining with uncertainty in block grades, equipment access, mobility, blending and other aspects. The formulation is based on expected block grades and the probabilities of grades being above required cutoffs. Their approach mines the more certain areas of the deposit in earlier time periods, leaving uncertain areas for later when more information on them is available. The objective function does not directly maximise the NPV, but rather does it indirectly while creating a feasible schedule.

In the current chapter, we will mostly focus on the extended version of the *block sequencing problem* i.e what blocks to dig at what time. Dagdelen and Johnson [51] propose an algorithm based on Lagrangian relaxation for maximising NPV with constraints. The extension of this work has been done by Akaike and Dagdelen [5] by iteratively changing the values of Lagrangian multipliers, until the solution satisfies the constraints. Another extension has been done by Kawahata [92] in his doctoral thesis. He added variable cutoff grades, stockpiling and wastedump restrictions.

Busnach et al. [27] propose a heuristic algorithm that addresses a block sequencing problem for a phosphate mine in Israel. Samanta et. al [154] apply a genetic algorithm to the problem of *grade control planning* in a bauxite mine. The problem that is discussed in this paper has grade control planning problem as its part. The goal of the problem is to minimise quality deviations of two elements: silicon and aluminium.

Another important problem in mining is the problem of *ultimate pit limits* that is finding a shape of the mine at the the end of its life that provides the maximum profit. This problem is also efficiently solved by Lerchs-Grossman graph theoretic algorithm [103] and Picard's network flow method [134]. Caccetta and Giannini [28, 29] describe some approaches to other problems. Hochbaum and Chen [81] review most of the commonly used methods for this problem.

Osanloo et al. [131] has a good review of long-term optimisation models for open-pit mining. A very good review on operations research methods in mine planning (both surface and underground) is provided by Newman et al. [126].

Many problems in the mining industry can be classified as scheduling problems. The problem that is discussed in this chapter also belongs to this category. Several different types of scheduling problems exist, such as job-shop scheduling, flow-shop, flexible flow-shop and open-shop. These are described in detail in [135]. Many other algorithms for different types of scheduling problems were developed, for example [38, 68, 178, 129, 174, 136, 26, 149, 113, 96, 9, 152].

6.2 Problem Description

Since prehistoric times people have been mining different ores and minerals to produce various goods for their lives. Mining goes hand in hand with human history, and so even major cultural eras are called by various metals mined: the Bronze Age (4000 to 5000 BC), the Iron Age (1500 BC to 1780 AD), the Steel Age (1780 to 1945), and the Nuclear Age (1945 to the present). In the modern world mining is still one of the most important and advanced fields of industry. There are several main types of mining, and in this case study the open pit type of mining problem will be described. In particular, a metal ore mining system will be discussed.

A typical mine has a hierarchical structure. At the top level the mine is divided into several sub-mines called *models*. Typically each model is an island separated by non-ore material. Each model consists of one or several *pits*. Each pit in turn is sliced into horizontal layers called *benches*. Each bench contains multiple *blocks* (which is the last level of hierarchy). Each block has specific information about its coordinates, tonnage, characteristics, percentage of different metals and non-metals (e.g. iron, aluminum, phosphorus) and waste. A block can fully consist of waste, or it can contain certain metals of high or low grade.

There are two main types of plants on the mining site: *crushers* and *washplants* (sometimes referred to as a *beneficiation plant*). Depending on the nature of the mine there can be other types of plants, however, in this chapter only these two are considered.

A *crusher* is a plant designed to reduce large block chunks into smaller rocks for further processing. Different crushers have different crushing speeds measured in tonnes crushed per hour. Two main types of crushers considered in this chapter are high grade crushers and low grade crushers that are designed to operate with high and low grade materials respectively.

In order to improve low grade ore it needs to be processed by the *washplant*. The process of washing removes contaminations, impurities, and other things that lower the quality of the ore. Several methods are known to achieve this task: magnetic separation, advanced gravity separation, jigging, washing and

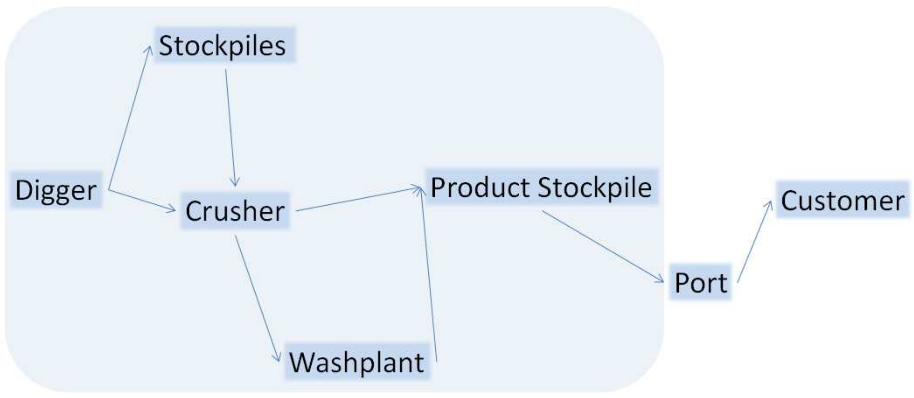


Figure 6.1: Product movement for mine planning.

others.

There are two types of mobile equipment available for mining: *diggers* and *trucks*. *Trucks* are used to transport material from one place to another. One or several fleets of trucks can be available. Each fleet has a number of trucks of the same type with same speed and load capacity. *Diggers* are used to excavate material and load it to trucks. Each digger has its own digging rate measured in tonnes of material per hour. They normally have a very low speed (2 km/hr).

Mine models, pits, benches, crushers, washplants, stockpiles and waste-dumps are connected by the road network. Materials are transported by a means of trucks through this network. Each pit has one or several pit exits through which trucks enter the pit. Each bench has one or several starting blocks, called *toe blocks*, with the road connected to them.

The mining process as follows. First, a block is blasted by explosives, then a *digger* enters the blast area and excavates the blasted material. It loads the material onto trucks which drive it to the next destination. The destination can be different depending on the type and quality of material. All high grade materials from the excavated block are transported to high grade crushers and then the crushed product is shipped to customers. The low grade material is transported first to the washplant to purify it, then to the low grade crusher and only after that shipped to customers. All excavated waste is taken to *wastedumps* which are basically piles of waste material. Along with the destinations mentioned above, excavated material can also go to *stockpiles* which can be thought of as temporary places to put material that can be used in the future. Figure 6.1 shows the movement of ore from the block to the customer. The current mine planning problem covers (shaded region) movement from the block to the product stockpile. After this process, ore from the product stockpile is transported by trains to the port and put into ships.

The current problem is a long term planning and scheduling problem. Decisions have to be made the order in which blocks of the mine should be excavated, and how to utilise the equipment, crushers, washplants, stockpiles and truck fleets. In order to understand the objectives of the problem, a mining time period concept needs to be discussed. As a long term scheduling problem, operation decisions should be made in a time horizon which is a few times smaller than the life of the mine term (which is the long term planning horizon). The whole decision time horizon is divided into smaller *time periods* which. The number of diggers and digging rates, and the crusher and washplant operating rates, capacities of wastedumps and stockpiles can be different across time periods. In addition, each time period has its *targets* which can be understood as a milestone to be reached at the end of the period.

- **Tonnage target.** As has been described earlier each block contains a certain tonnage of desired metal. A block can also consist of total waste, which will not be accounted for in reaching the targets.
- **Quality target.** All mining blocks have metal ore of a different quality. However, clients want ore of a predefined quality. This means that during a time period a set of blocks should be excavated such that, after the blending of all the excavated blocks during the time period, the blended quality should be close to reaching the desired quality. It is very hard to match the quality target exactly; this is a reason a quality tolerance is introduced, measured in percentage from the target. All qualities within this tolerance level are considered to be acceptable.

The objective is to generate a plan that meets these targets by utilising the provided equipment and machines. However, this problem is even more complicated than is described so far, due to additional constraints.

The addition of time periods to the problem adds an issue of time-varying constraints and time-varying objectives to the problem.

6.2.1 Constraints and Business Rules

This subsection describes different types of problem specific dependencies and business rules. Most of them fall into one of two categories: dependency constraints and capacity constraints. The current formulation of the problem recognises two types of dependency constraints: *block dependencies* and *area dependencies*. Block dependency is the basic constraint of the problem. There are three types of block dependencies:

- **clear Above Dependencies.** Due to the physical nature of the mine, a block cannot be excavated before the block on top of it is cleared. It can be thought of as a vertical dependency.
- **clear Ahead Dependencies.** This is a horizontal type of dependency. When digging the bench all side blocks should be excavated first before getting to the inner ones.
- **user Defined Dependencies.** Sometimes due to certain business rules or specific circumstances there is a need to define custom dependencies between blocks.

Another type of dependency constraint is area dependency. An area is basically an arbitrary collection of user-defined blocks. It can be an arbitrary collection of blocks but usually it involves a few adjacent pits that share a certain characteristic. The concept of an area gives a flexibility to the define custom dependencies. To indicate that one part of the mine should be excavated before another one, these two parts should be marked as different areas and connected by the dependency.

Capacity constraints, on the other hand, are the type of constraints that deal with the physical workload of machines and equipment. All of them are hard constraints, meaning violation of any of them will render a solution infeasible. A number of these hard constraints are:

- **digger capacity constraint**. Each digger can excavate only a certain tonnage per time period.
- **truck constraints**. There are speed and capacity constraints for each type of truck. Furthermore, the speed of the truck depends on whether it is loaded with ore, waste or unloaded, and also on the slope of the road.
- **crusher constraints**. As mentioned before crushers can operate with a limited speed and have processing capacity limits.
- **stockpile constraints**. Each stockpile has a limited capacity.

- **wastedump constraints**. Each wastedump has a limited capacity.

There are also some other constraints defined in this application, for example:

- **pit tonnage limits.** There is a limit on how much maximum tonnage can be excavated from each pit. Some pits may not have any limits.
- **digger proximity constraint.** Due to safety reasons, two diggers cannot operate too close to each other.

6.2.2 Overall Objectives

Taking into account all the dependency, capacity and other constraints described above, the problem becomes a highly complex combinatorial optimisation problem. The system that addresses the described mining problem has been implemented with an optimisation component based on a population based metaheuristic. This system was built for a large mining company, and it is currently in the process of being integrated into the existing IT structure. The key features of the system are as follows:

- **meeting targets**. The system is taking into account all constraints, configuration and targets and on that basis produces a 5 year plan for the way to utilise diggers, plants and other equipment, and for the block at which each digger should operate at a given time.
- **tradeoff analysis**. There are some configurations where it is not possible to satisfy both tonnage and quality targets. Then the system produces a set of solutions which show different tradeoffs in tonnage and quality
- **manual changes and what-if scenarios**. Numerous business rules are a predominant feature of many real-world systems. It is not always possible to incorporate all business rules into the software. That is why for any given solution produced by the system the operator can manually override decisions made by the system and see the impact of the changes instantly. Apart from the flexibility of manual changes this also reveals another interesting feature of the application: the ability to analyse various what-if scenarios. In a matter of minutes the user can see the result of adding or removing a digger or building another crusher on the mining site.

6.3 The Model

This section firstly describes the model of the current problem from the supply chain point of view, explaining the two echelon inter- and intra- echelon dependencies. The second part of the section presents the mathematical model of the current problem.

6.3.1 Supply Chain Model

Looking at this problem from the supply chain perspective, a two-echelon supply chain can be identified. The first is an *ore digging* echelon, and the second is an *ore processing* echelon.

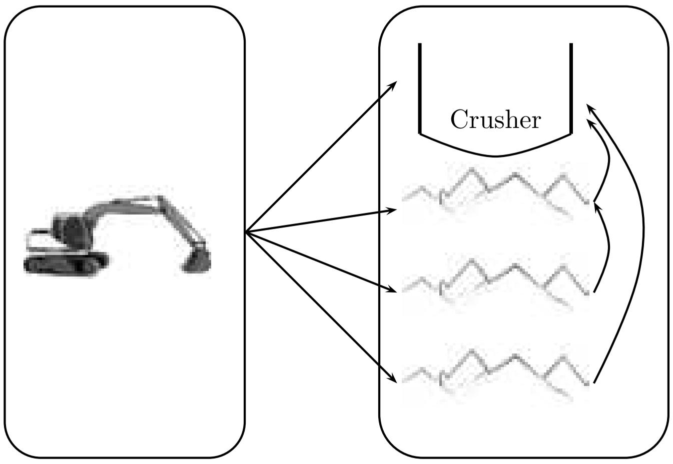


Figure 6.2: Mining supply chain with inter- and intra-echelon interactions

The ore digging echelon is responsible for finding the correct bench in the mine that would contribute to the total tonnage and quality targets and for selecting the correct digger. Depending on the resolution of the supply chain, each digger can be treated as a separate silo or, alternatively, all diggers can be aggregated into one digging silo. In this chapter we use the second model, treating all diggers as a single silo. The mining supply chain diagram on Figure 6.2 describes the second option.

The ore processing echelon consists of different crushers, stockpiles and wash-plants. The ore dug in the first echelon can take different routes depending on how far the actual quality and tonnage from the desired tonnage. For example, if tonnage and quality targets are satisfied during the current time period, it makes sense to send the ore to one of the stockpiles, so it can be reclaimed at later time periods; if quality needs to be improved, ore can go through the washplant or the ore can go straight to the crusher avoiding all the stockpiles. Since all crushers, stockpiles and washplants share the common property of processing ore (hence the name ore processing echelon) but, on the other hand, implement different behaviour in terms of functionality, they are considered to be different silos within one echelon with *inter-echelon interactions.*

Taking into account the two-echelon model of the supply chain, the approach presented in section 6.4 addresses each echelon separately: module 1 (initialisation) and module 2 (evolutionary algorithm) assign diggers to work on a specific set of benches while module 3 (decoder) makes deterministic decisions based on input from the first echelon.

The rest of the section presents the mathematical model of the problem proposed above.

6.3.2 General Notations

The following basic sets are defined for the mine $Mine$:

- Blocks $B = \{B_1, \ldots, B_{|B|}\}$ in the ore body. Each block is determined by the following characteristics: geometrical coordinates of the block are given; $tonnage(B_i, el, gr)$ is a tonnage of material el of grade gr.

- Benches $Ben = \{Ben_1, \ldots, Ben_{|Ben|}\}$. A bench is represented as a set of blocks $Ben_i = \{x : x \in B\}$. Each bench has a *toe block* defined as $b_0(Ben_i)$.

- Pits $P = \{P_1, \ldots, P_{|P|}\}$. A pit consists of set of benches $P_i = \{x : x \in Ben\}$.

- Model $M = \{M_1, \ldots, M_{|M|}\}$. A model contains a set of pits $M_i = \{x : x \in P\}$. The top level hierarchy is the mine which contains a set of models $Mine = \{x : x \in M\}$.

- Areas $A = \{A_1, \ldots, A_{|A|}\}$. An area is a set of blocks $A_i = \{x : x \in B\}$. Additionally, an area may have the earliest start date $esd(A_i)$ and latest end date $led(A_i)$. Each area has a set of dependent areas defined as $Dep(Ai) = \{x : x \in A\}$. If area A_j is dependent on area A_i then

$$\begin{aligned} \forall B_k, B_l : (\alpha(B_k) = A_i, \alpha(B_l) = A_j) \\ \Rightarrow t_l(B_k) < t_s(B_l) \end{aligned} \tag{6.1}$$

where $t_s(B_i)$ is a start time of digging block B_i, $t_e(B_i)$ is an end time of digging block B_i and $\alpha(B_i)$ is an area to which the block belongs. This dependency relation is expressed as $A_i \prec A_j$.

- Time periods $T = \{T_1, \ldots, T_{|T|}\}$. Each time period is determined by its start date and end date. Each period has information about various aspects such as the capacities of diggers, crushers, targets, and limits.

- Diggers $D = \{D_1, \ldots, D_{|D|}\}$. Each digger has own specific characteristics. $cost(D_i)$ is a cost of operating a digger. $speed(D_i)$ is a digger tramming speed. $location(D_i) \in B$ is an initial block of the digger. $rate(D_i, T_j)$ – how many tonnes of ore a digger can excavate per hour. $utilisation(D_i, T_j)$ – effective utilisation, which determines the percentage of the maximum rate the digger can operate in the real world due to day/night shifts, maintenance and other factors. Speed and cost remain fixed during the whole process, but utilisation and rate change across time periods.

- Crushers $C = \{C_1, \ldots, C_{|C|}\}$. Each crusher has its own characteristics. Its location is given by geometrical coordinates. $capacity(C_i, T_j)$ is a feed capacity, i.e. number of tonnes it can process per hour. $utilisation(C_i, T_j)$ – effective utilisation, which determines what percent of the maximum rate the crusher can operate in the real world.

$Q^k(T_i)$ is a k-th quality target for time period T_i with the tolerance q_i. For each time period T_i there are total of $K(T_i)$ targets. $\Omega(T_i)$ is a tonnage target for time period T_i

6.3.3 Objectives of the Optimiser

The goal is to find a vector $X = \{X_1, \ldots, X_{|B|}\}$ that minimises

$$min\Delta_T = \sum_{T_i \in T} |\Omega_T(T_i) - \Omega(T_i)| \tag{6.2}$$

and

$$min\Delta_Q = \sum_{T_i \in T} \sum_{1 \leq k \leq K(T_i)} \rho(k)\chi(|Q_T(T_i) - Q^k(T_i)| - q_i) \tag{6.3}$$

Here each element of X is a tuple $X_i = (st, digger, block)$ corresponding to digging start time, digger and block to be dug respectively. The end time can be calculated in the following way

$$endT(X_i) = startT + weight(block)/rate(digger, T_k) \tag{6.4}$$

where $weight(block)$ corresponds to the physical weight of the block and $rate(digger, T_k)$ defines rate (in tonnes per hour) in time period T_k during which the digging of the block occurs.

$\Omega_T(T_k)$ is a total tonnage of ore excavated during time period T_k and defined as follows: for each time period T_k,

$$\Omega_T(T_k) = \sum_{X_i \in T_k} tonnage(X_i.block, el, grade) \tag{6.5}$$

Thus, the first objective is to minimise total deviation from the desired tonnage

The second objective defined by 6.3 is to minimise total deviation from the desired quality across all time periods. In the equation 6.3 $\chi(x)$ is a positive indicator function which returns 0 if x is negative and the actual argument x otherwise. Since there are several quality targets, each target has its own priority which is determined by the coefficient $\rho(k)$.

6.3.4 Constraints

$\lambda_{AH}(B_i, B_j)$ – clear ahead dependency, 1 if dependency violated, 0 otherwise

$\lambda_{AB}(B_i, B_j)$ – clear above dependency, 1 if dependency violated, 0 otherwise

The solution made with the vector X should comply with the clear ahead and clear above constraints, so

$$\sum_{i,j} \lambda_{AH}(B_i, B_j) + \sum_{i,j} \lambda_{AB}(B_i, B_j) = 0$$

Also, no two blocks could be executed simultaneously on the same digger:

$$\forall i, j | X_i.digger = X_j.digger X_i \cap X_j = \emptyset$$

here $X_i \cap X_j$ operation returns the intersection between time intervals X_i and X_j.

All the constraints described above are hard dependencies.

$$\forall X_k.block \in A_i, X_m.block \in A_j | A_i \prec A_j$$

The following holds $X_m.st \leq endTime(X_k)$. Here relation $A_i \prec A_j$ defines the area dependency that area A_i should be mined before area A_j.

6.4 Approach

This section describes a proposed algorithm for the described problem. Firstly, the representation of the solution is described, then the evaluation quality score is explained, and lastly, the three stages of the proposed optimiser are shown.

6.4.1 Structure of a Solution

The structure of a solution for the optimiser is as follows. For each time period a table stores a set of benches that are scheduled to be dug in this time period. If only a part of the bench is scheduled during the selected time period, then the fraction of the bench is recorded. Table 6.1 illustrates this concept. The first column in this table lists all available benches. The other columns represent time periods when a particular bench is scheduled. For example, bench B3 is scheduled for digging in the 3rd time period and bench B4 is scheduled as follows: 60% during the 4th and 40% during the 5th time periods. We refer to this data structure as a Tonnage Schedule (TS). During the execution of the optimiser, the Tonnage Schedule is initialised and later modified at every iteration of the process; special operators make sure that clear above and area dependencies are not violated (i.e. that the solution is feasible from the perspective of these dependencies).

The size of the search space is quite significant; for example, for 20 time periods (this would correspond to 5 years, assuming quarterly time periods), with 500 benches available in total and 150 benches available per time period (due to various dependencies), and at least 10 benches fully allocated for a time period, the number of possible solutions is in the order of 10^{240}. Note that if the number of allocated benches is higher (say, 25), the size of the search space grows dramatically much further. Note also that, to compare the number of

Bench	TP1	TP2	TP3	TP4	TP5	TP6	TP7
B1	1						
B2		1					
B3			1				
B4				0.6	0.4		
B5						1	
B6						0.5	0.5
B7							1

Table 6.1: Solution representation

solutions in the search space, the current calculations of the number of atoms in the observable universe is close to 10^{80}.

6.4.2 The Quality Score of the Solution

The quality score of a solution is based on a combination of various penalty functions:

- Quality violation penalty. This penalty is calculated as a sum of deviations from the desired qualities of each quality target.
- Tonnage violation penalty. This penalty is calculated as a deviation of actual tonnage from the desired tonnage.
- Digger workload violation penalty. This penalty represents how much total digger workload exceeds the maximum digger workload. If total digger workload does not exceed the maximum workload the penalty is 0.
- Haulage workload violation penalty. This penalty represents how much total haulage workload exceeds the maximum haulage workload. If total haulage workload does not exceed the maximum workload the penalty is 0.

6.4.3 The Optimiser

The optimiser used in the system consists of 3 modules:

- Module 1: Initialisation
- Module 2: Problem-specific Evolutionary Algorithm
- Module 3: Decoder

which are executed sequentially. These three modules are discussed in turn.

Module 1: Initialisation

As the size of the search space for the planning problem is prohibitive, module 1 is responsible for creating the initial set of candidate solutions. The main parameters that control the execution of this algorithm are: n - population size, m - the number of offspring produced by a parent, k - the elitism factor (the number of the best individuals to maintain), and g - the diversity factor (to generate a pool of diverse solutions).

This module initialises the pool of candidate solutions in a few stages. It starts with a pre-distribution that considers the frozen time period, where all parts of the solution that fall into the frozen time period are copied from a previously generated solution (if it exists) into the new one. Then a *pre-distribution of dependency chains* is performed by analysing the pits' dependency chains for possible bottlenecks. A certain number of pits are selected and pre-distributed over several time periods to release bottleneck dependencies early. Many different arrangements (e.g. a different number of selected pits, different distributions over time periods) are possible. At this stage we have a population of size n, which consists of n (incomplete) solutions created during the first two stages. During the third stage of initialisation the module allocates benches to all considered time periods taking into account (a) all available benches from all pits that are not held by any dependencies (area dependencies and clear above dependencies), (b) pit limits (if specified), (c) the target tonnage, and (d) the target quality. Such allocation is run in a loop while at least one of the following conditions holds:

1. actual tonnage is greater than that desired
2. haulage workload capacity is exceeded for the benches scheduled for the current time period
3. digger workload capacity is exceeded for the benches scheduled for the current time period

By eliminating these violations the algorithm makes sure that, after exiting the loop, none of the hard constraints are violated i.e. the solutions are feasible. Further, spare digger capacity is assigned to dig waste benches by rescheduling some waste benches - during such rescheduling, the algorithm checks that no haulage workload and digger workload limits are violated.

The above steps are performed m times for each individual in the current population to produce an array of solutions. Elitist selection is then performed on this array with the elitism factor of k, i.e. k individuals with the best fitness are then selected from this array as a population for the next time period. This means that for all time periods except for the first, k solutions are selected from $m \times k$ candidate solutions for further processing. The quality measure of an individual solution is based on penalties (as discussed earlier). The selection algorithm maintains the diversity of the population by eliminating solutions that are too similar to other solutions - this is controlled by the diversity factor g.

At the final stages of the initialisation, reclaiming from stockpiles is performed in order to bring the quality closer to the desired levels. If the actual quality is within the desired quality tolerance range, this stage is skipped. The tonnage to be reclaimed is calculated in such a way that digger workload and haulage workload are not violated.

Module 2: Problem specific Evolutionary Algorithm

The main parameters that control the execution of this algorithm are: n_{ea} - population size, m_{ea} - the number of offspring per parent, k_{ea} - the elitism factor (the number of the best individuals to maintain), g_{ea} - diversity factor. A population of solutions generated during the initialisation phase is fed into the evolutionary algorithm that improves the population during the iterative process by applying variation operators to existing candidate solutions thus generating new offsprings. Sample variation operators include:

- **move right operator.** This operator reschedules a selected bench to the next time period. The operator is performed in such a way that no clear above rule is violated.
- **move left operator.** This operator does the same job as the previous one except that it reschedules benches to the previous time period instead.
- **repair operators.** These operators restore the feasibility of the solution after modification.

The evolutionary algorithm uses steady state tournament selection with operators that have the adaptive probabilities of being applied. The diversity of population is preserved by ignoring solutions that lie within a certain distance of each other.

Module 3: Decoder

The decoding phase consists of 2 parts:

- Converting from Tonnage Schedule into Bench Schedule.
- Converting from Bench Schedule into Block Schedule.

The module uses solutions generated by the previous module for making decisions on digger assignments and further material movements. During this stage only deterministic decisions are made. For example, the module decides where, to which crusher, stockpile, or waste dump, the excavated material (low grade, high grade, and waste) should be sent.

6.5 Results

The implemented optimiser can be run in different modes, with certain optimiser options being chosen or not chosen. The available options are to:

1. continue equipment utilisation after targets achieved, i.e use all spare digger capacity to dig more blocks and send to stockpiles.
2. allow or deny stockpile reclamation.
3. use total truck capacity as a hard constraint.
4. use top ranked quality target only.
5. enforce area dates.
6. enforce or ignore pit tonnage limits.

Each one of these points can be selected or deselected independently of the others. This gives a flexibility in varying different types of limitations during the run (total number of choices $2^6 = 64$).

Each of the optimiser executions has been run on live mine data provided by the mining company. The production schedule developed by the company's expert schedulers team has been analysed as well. However, a much more narrow quality tolerance boundary of 0.4% has been enforced in our system in contrast with the expert quality tolerance of 1.5%. The software produced a valid schedule within approximately 5 minutes and has been evaluated as fully operational by the expert team. The expert team used their own software suite based on

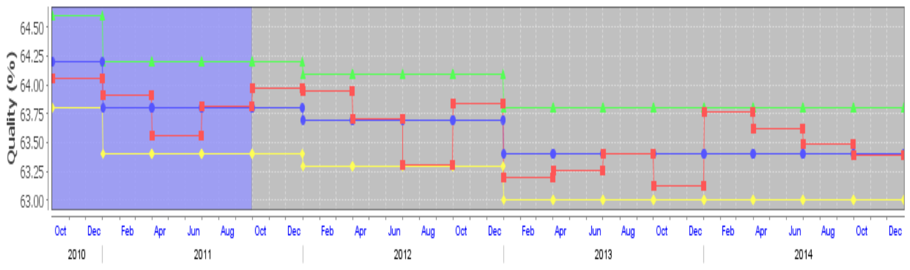


Figure 6.3: Result of the optimiser.

the XPAC mining system that has various scripts which help to find a solution. Building a solution with their system requires the operation of several systems from the suite to produce even one solution. The approximate time to build one solution can be around a day which is significantly slower than software based on our metaheuristics. Any variation in the schedule, such as the changing of business rules or the testing of what-if scenarios would incur a full schedule creation process.

Figure 6.3 shows the quality results of the first quality target with optimiser options 1, 2 and 4 selected. Tonnage targets were matched with zero total deviation from the desired tonnage. This configuration produced best results on both tonnage and quality objectives. The blue line on the graph represents the desired quality, red line shows the actual quality of the schedule, green and yellow lines show higher and lower tolerance levels (of 0.4%).

The addition of optimiser option 3 (enforce truck capacity) has an impact on the solution quality. Tonnage and quality graphs are shown on figures 6.4 and 6.5. This was achieved with the setting of 31 standard trucks working fully per time period. This result shows that it is very hard (if not impossible) to produce a sufficiently good schedule with this quantity of trucks and the current configuration. Both the previous results of a good schedule without constraining truck capacity, together with the current result, should tell schedulers to change certain values, the most likely being to increase the number of trucks in the problematic time periods and then to rerun the optimiser to see the result. This experiment shows the strengths of the software in quickly analysing various what-if scenarios.

The next experimental run of the optimiser allowing the utilisation of the stockpiles, i.e. options 1 and 2 were enabled. This optimiser configuration uses all quality targets prioritising them by rank. Here we set the priority to high and

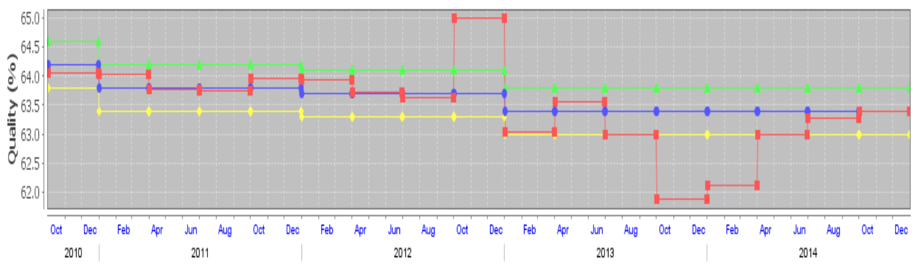


Figure 6.4: Result of the optimiser. Quality of the first objective with the limited truck capacity.

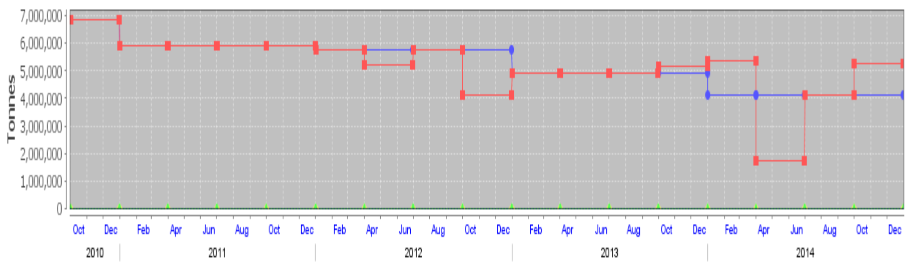


Figure 6.5: Result of the optimiser. Tonnage graph with the limited truck capacity.

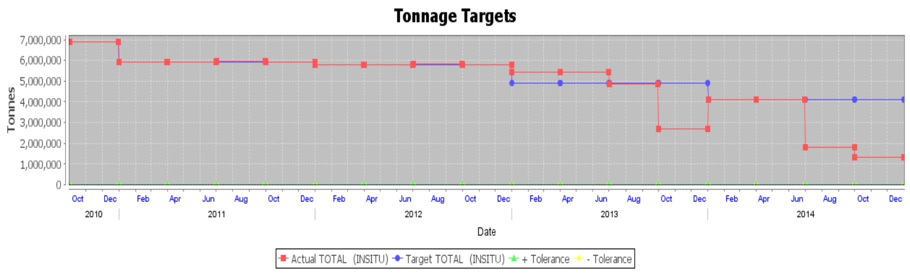


Figure 6.6: Result of the optimiser. Tonnage graph.

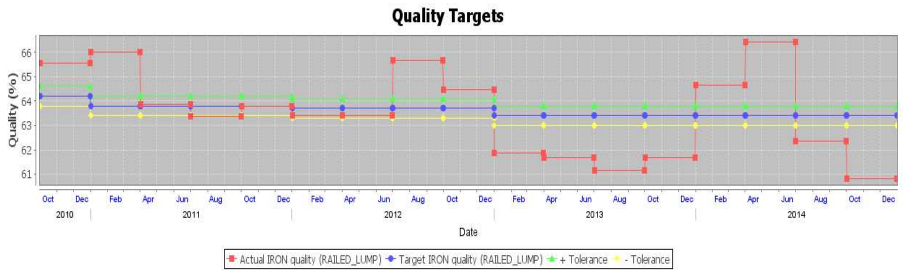


Figure 6.7: Result of the optimiser. Graph of quality of Iron.

low quality iron ore over other materials. This will assign the highest coefficient to iron ore quality deviation while evaluating the solution. In the real world schedulers are interested in both optimisations: the optimisation that considers only the first quality target, and the optimisation that considers all quality targets with priorities.

Results presented in figures 6.6, 6.8 and 6.8 show that considering all quality objectives makes the first objective worse while improving on other objectives. This converts the current problem from a 2 objective problem into a 9-objective problem, as the default configuration consists of 8 quality targets per quarter.

As with the previous optimiser run, the results give schedulers knowledge of what happens with the current configuration. By changing certain values and rerunning the optimiser, they were able to see the impact of the change within 5-10 minutes. Previously, it would take at least one day to see the results of the change. Additionally, the client utilised several systems to produce the result and had to feed the output of one system manually into another as well as manually configure each system for current settings. This, certainly, would increase the chance of making an error.

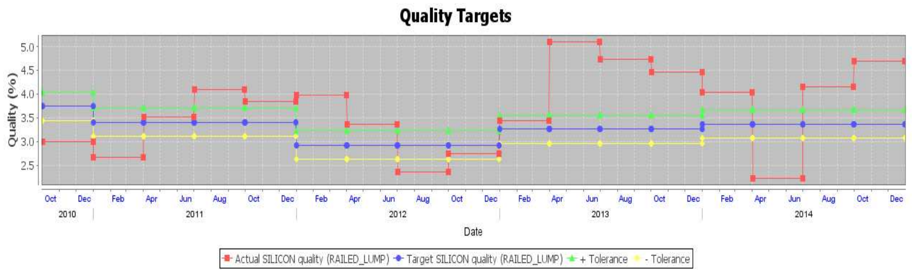


Figure 6.8: Result of the optimiser. Graph of quality of Silicon.

A very powerful tool for the guiding optimisation in the desired direction is the ability to set minimum and maximum pit limits. However, the outcome of some unconsidered settings can be very dangerous, as slowing down certain parts of the mine can cause other parts of the mine to be blocked due to dependencies. The use of area dates also allows an experienced scheduler to force the system to obtain a required result. However, each of these configurations may cause complications from failing to meet targets.

The described results show 2 main strengths of the system not found in most of the other systems:

- quick optimisation.
- powerful what-if analysis.

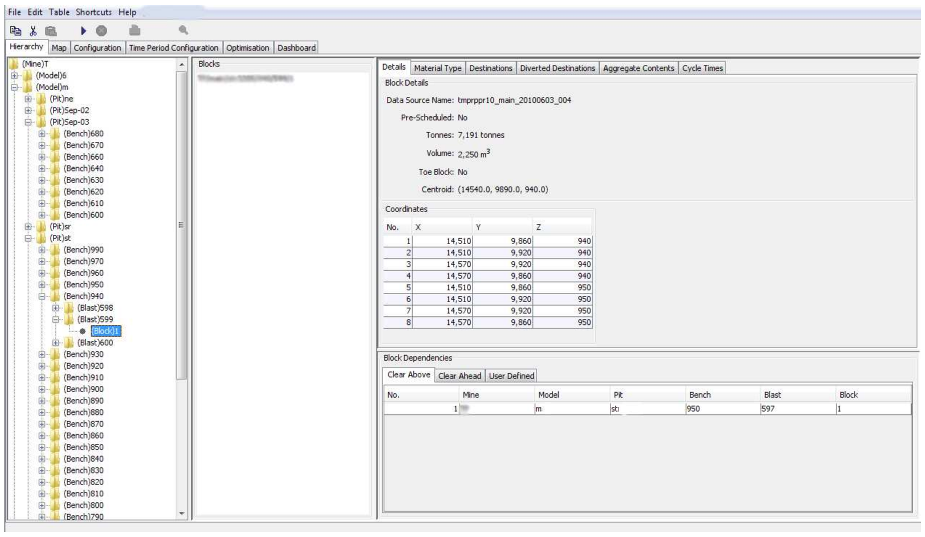
The next section describes the main functionality and configuration settings of the system.

6.6 Functionality

This section describes the main software functionality of the system. It has been developed in Java. The developed software has several main tabs: Hierarchy, Map, Configuration, Optimisation and Dashboard. These will be described in the following subsections.

6.6.1 Hierarchy Tab

The hierarchy tab presents the mine in the hierarchical view: Mine - Model - Pit - Bench - Block. As has been described in the previous sections the basic unit of the mine is a block. Each block is specified by its geometrical coordinates and



(Mine)T
- (Model)6
- (Model)m
  - (Pit)ne
  - (Pit)Sep-02
  - (Pit)Sep-03
    - (Bench)680
    - (Bench)670
    - (Bench)660
    - (Bench)640
    - (Bench)630
    - (Bench)620
    - (Bench)610
    - (Bench)600
  - (Pit)sr
  - (Pit)st
    - (Bench)990
    - (Bench)970
    - (Bench)960
    - (Bench)950
    - (Bench)940
      - (Blast)598
      - (Blast)599
        - (Block)1
      - (Blast)600
    - (Bench)930
    - (Bench)920
    - (Bench)910
    - (Bench)900
    - (Bench)890
    - (Bench)880
    - (Bench)870
    - (Bench)860
    - (Bench)850
    - (Bench)840
    - (Bench)830
    - (Bench)820
    - (Bench)810
    - (Bench)800
    - (Bench)790

Blocks
[redacted]

Block Details

Data Source Name: tmprppr10_main_20100603_004
Pre-Scheduled: No
Tonnes: 7,191 tonnes
Volume: 2,250 m^3
Toe Block: No
Centroid: (14540.0, 9890.0, 940.0)

Coordinates

No.	X	Y	Z
1	14,510	9,860	940
2	14,510	9,920	940
3	14,570	9,920	940
4	14,570	9,860	940
5	14,510	9,860	950
6	14,510	9,920	950
7	14,570	9,920	950
8	14,570	9,860	950

Block Dependencies

No.	Mine	Model	Pit	Bench	Blast	Block
1	[redacted]	m	st	950	597	1

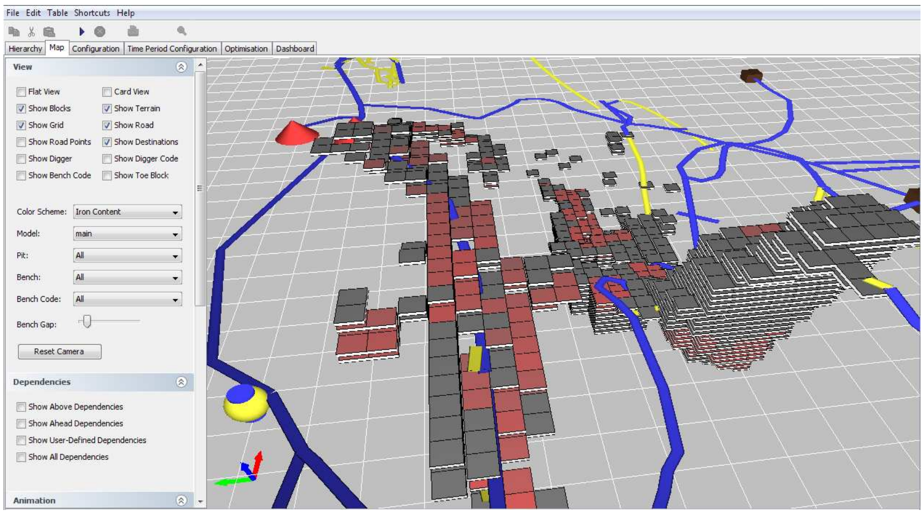
Figure 6.9: Screenshot presents the hierarchy tab with coordinates and tonnages of a selected block and its dependencies.

tonnages of certain material type and grade (for example high grade iron ore), which are exported from file. From the geometrical coordinates clear above and clear ahead dependencies are calculated.

The road network of the whole mine can be exported from the file as well. Each road consists of road segments which are represented by the geometrical coordinates. All parts of the mine, such as plants, crushers, stockpiles and wastedumps, are connected by the road network, and therefore it is possible to calculate the shortest distances between each block and destination. Users of the software can manually override the shortest path destination. The situations where some types of business rules override decisions of the system are very common for real world problems, and they are normally not considered in the classical research problems. Figure 6.9 shows the screenshot of the hierarchy tab with one block selected. On the right side of the screen the user is presented with all characteristics of the block including its dependencies.

6.6.2 Map Tab

The map tab presents the user with the an interactive 3D map of the mine, the road network and ore destinations fully constructed from the raw geometrical coordinates. The software provides the ability to "fly" over the mine, zoom



[ ] Flat View
[✓] Show Blocks
[✓] Show Grid
[ ] Show Road Points
[ ] Show Digger
[ ] Show Bench Code
[ ] Card View
[✓] Show Terrain
[✓] Show Road
[✓] Show Destinations
[ ] Show Digger Code
[ ] Show Toe Block

Color Scheme: Iron Content
Model: main
Pit: All
Bench: All
Bench Code: All
Bench Gap:

[ ] Show Above Dependencies
[ ] Show Ahead Dependencies
[ ] Show User-Defined Dependencies
[ ] Show All Dependencies

Figure 6.10: Screenshot presents the 3D view of the mine and map controls.

in and out and explore the structure. It also provides a user friendly way of assigning toe blocks, the setting of destinations and the visual exploration of the contents of the block. Different colour schemes visualise the mine by the concentration of iron in the blocks (the higher concentration the more intense is the colour), by pit, by model, by the time period planned for mining, and whether the block is going to the crusher or the stockpile. Figure 6.10 shows the screenshot of the map tab. The main part of the screen is a described 3D interactive map, where the right panel allows users to perform operations such as to hide or show various elements of the map, to change colour schemes, or to filter specific parts of the mine.

6.6.3 Configuration Tab

This is one of the most important tabs in the software as the configuration of all different parts of the mine and optimiser is presented here. Configuration is split into several categories: dependencies, plants, mobile equipment, wastedumps, stockpiles, scenario and time period configuration. This part of the system contains most of the constraints and business rules. The time period configuration category defines dynamic constraints i.e. they change over the time periods. These two types of constraints in the configuration are described below.

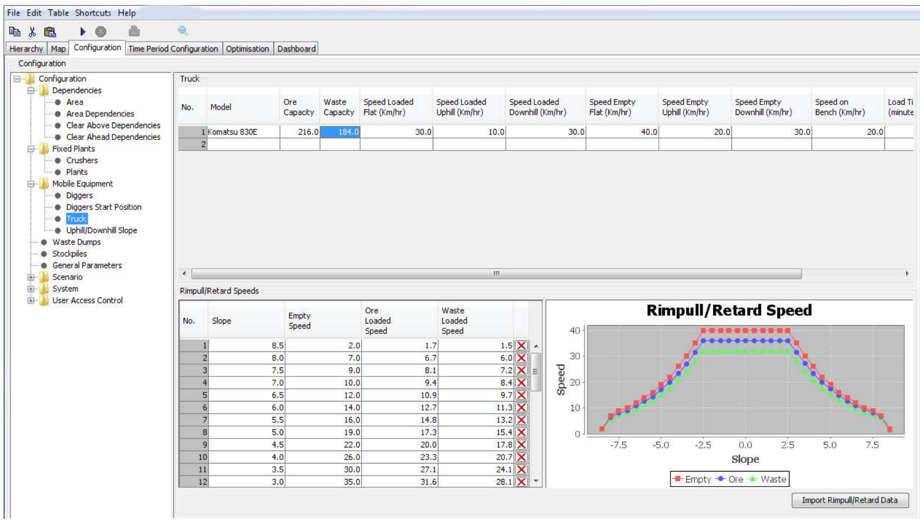
Static Configuration

The *dependencies* in the system are *area* dependences, *clear above* and *clear ahead* dependencies (see section 6.2). The first step in configuring an area is to create a master record in the area configuration screen. Once the area has been defined, the planner can then configure the area within the map screen by adding a set of blocks, a bench, a pit or even a model. The next step is to configure the area dependencies based on the defined areas. The dependencies category also allows a change of settings of values of clear above and clear ahead dependencies (for example, how many blocks should be cleared before a certain block can be mined in the clear ahead dependencies screen). Additionally, each area can be set to be mined after a certain time period, before the certain time period, or in between two time periods according to a certain business rule. The dependencies concept in mining is very essential and allows schedulers to restrict excavation of one part of the mine before the other one is excavated or to excavate it before or during certain time periods. From the optimisation point of view it significantly constrains the search space and in many cases makes finding even one feasible solution a very challenging problem.

The *plant* category lets users add or remove plants and crushers available in the system. At the static stage of the configuration plants and crushers are defined by their coordinates on the plane.

The *mobile equipment* configuration category lets the scheduler set the number of diggers, their type, speed and operating cost. Each digger may have different settings, so the configuration is not homogeneous. An additional constraint for diggers is their starting position on a certain block. As the travelling speed of each digger is very limited (typically around 2 kph), each digger is limited to excavating only a specified part of the mine because it is normally not optimal to spend most of the time in a time period transporting the digger.

Transportation of the ore from the mine to crushers, plants and stockpiles is done via trucks over the road network. This is represented in the system as fleets of homogeneous trucks. Each fleet defines parameters for each of the trucks in the fleet. Some of the parameters include the capacity of truck loaded with ore, and the capacity loaded with waste (as ore and waste have different densities). The speed of each truck depends on the slope of the road and type of material loaded on the truck. The calculation of speed is based on rimpull curves - a mathematical curve used to lookup the speed based on slope and load of the truck. As road segments have geometrical coordinates the slope can be



Truck

No.	Model	Ore Capacity	Waste Capacity	Speed Loaded Flat (Km/hr)	Speed Loaded Uphill (Km/hr)	Speed Loaded Downhill (Km/hr)	Speed Empty Flat (Km/hr)	Speed Empty Uphill (Km/hr)	Speed Empty Downhill (Km/hr)	Speed on Bench (Km/hr)	Load Ti (minute
1	Komatsu 830E	216.0	184.0	30.0	10.0	30.0	40.0	20.0	30.0	20.0	
2											

Rimpull/Retard Speeds

No.	Slope	Empty Speed	Ore Loaded Speed	Waste Loaded Speed
1	8.5	2.0	1.7	1.5
2	8.0	7.0	6.7	6.0
3	7.5	9.0	8.1	7.2
4	7.0	10.0	9.4	8.4
5	6.5	12.0	10.9	9.7
6	6.0	14.0	12.7	11.3
7	5.5	16.0	14.8	13.2
8	5.0	19.0	17.3	15.4
9	4.5	22.0	20.0	17.8
10	4.0	26.0	23.3	20.7
11	3.5	30.0	27.1	24.1
12	3.0	35.0	31.6	28.1

Figure 6.11: Screenshot presents the configuration tab.

easily calculated. The time it takes to travel a certain route of the network can be calculated summing travel times for road segments of that route. Note that the speed of the truck over a road segment is different if the truck is loaded with ore, waste or if it is unloaded.

The *wastedumps and stockpiles* category lets users define wastedumps and stockpiles that exist in the mine. Apart from their geographical coordinates the total capacity and opening balance of the structure are given.

The *scenario* category lets users define various optimiser configurations. If the *equipment utilisation after targets achieved* flag is blank then the optimiser will not attempt to plan any unused capacity that is available once the specified targets have been achieved.If this flag is selected then it will attempt to utilise diggers and trucks on waste blocks which will not affect the tonnage targets. If the *allow stockpile reclamation* flag is not selected then the system will not reclaim any tonnage from the stockpile to the crusher in generating a plan. If the *constrain optimisation by haulage capacity* is blank then the optimiser will not attempt to constrain the optimiser by exceeding specified truck capacity. If this flag is selected then the optimiser will not exceed truck capacity for each time period. If the *use top ranked quality target* is selected the optimiser will only focus on achieving the top weighted quality target for each time period. If this flag is not selected it will attempt to reach all quality targets according to the priority set by the weighting settings in the time period configuration. If the *enforce area dates* is selected the optimiser will respect the area dates set in the configuration screen. If this flag is not selected it will ignore the area dates in the optimisation process. If the *enforce pit tonnage limits* is selected the optimiser will respect the specified limits set in the time period configuration screen. If this flag is not selected it will ignore the limits in the optimisation process.

There is the capability of setting a time where the optimiser will not optimise prior to this date. Therefore the mine sequence that exists in the plan will remain unchanged up until the date specified. Note that there is an option of No Freeze where the optimiser will reschedule from the current date and not apply any freeze during the optimisation process.

Dynamic Constraints Configuration

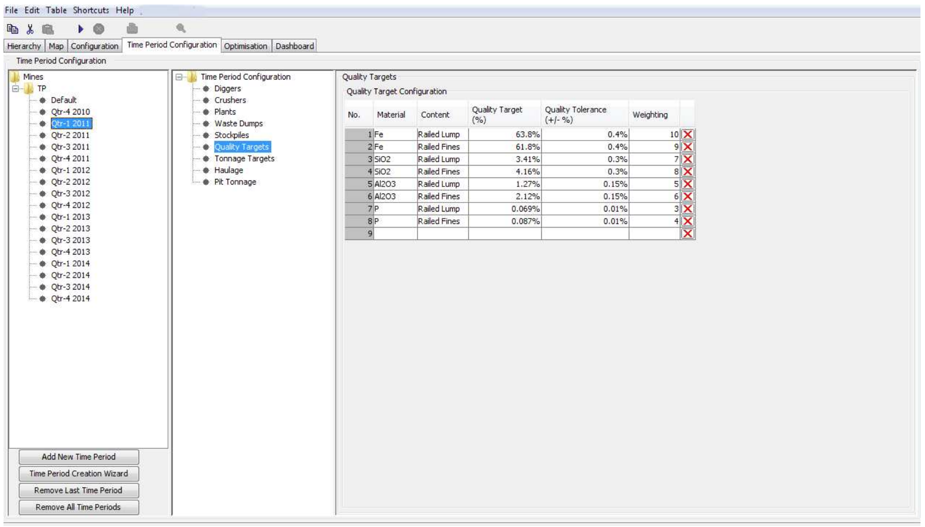
The dynamic constraints configuration can be achieved configuring each time period with its own parameters.

The following parameters can be changed over the time periods: rate and utilisation of diggers, feed capacity (in tonnes per hour) and utilisation of crushers, input capacity (in tonnes per hour) and effective utilisation of plants, capacity of wastedump (it is limited per time period and different from the total capacity of the wastedump), maximum and minimum capacity of stockpiles, number of trucks in the fleet and their effective utilisation. Additionally, each pit has a minimum and maximum tonnage that should be excavated.

The main part of the time period configuration is the configuration of targets. As has been described in section 6.2, two types of targets exist: tonnage targets and quality targets. Each time period has its own setting for both.

6.6.4 Optimisation Tab

The optimisation tab presents optimisation results. It has 2 main graphs that show actual and desired tonnage and quality targets. The quality graph also has two tolerance lines so it is very obvious if the solution is within the tolerance range or not. Figure 6.13 shows the screenshot of this tab. The top part shows a detailed breakdown of the parts of the mine to be dug during each quarter, the bottom part displays desired and actual graphs on tonnage and quality,

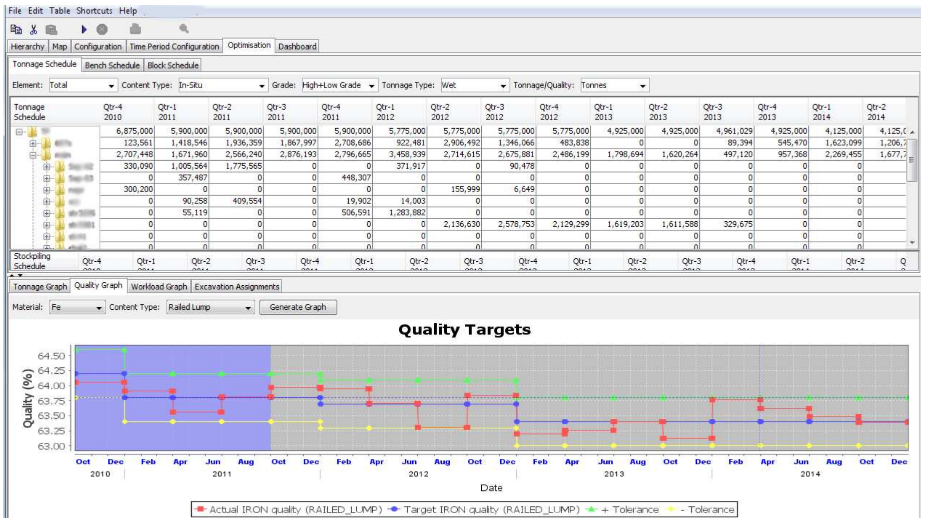


Quality Targets

Quality Target Configuration

No.	Material	Content	Quality Target (%)	Quality Tolerance (+/- %)	Weighting	
1	Fe	Railed Lump	63.8%	0.4%	10	❌
2	Fe	Railed Fines	61.8%	0.4%	9	❌
3	SiO2	Railed Lump	3.41%	0.3%	7	❌
4	SiO2	Railed Fines	4.16%	0.3%	8	❌
5	Al2O3	Railed Lump	1.27%	0.15%	5	❌
6	Al2O3	Railed Fines	2.12%	0.15%	6	❌
7	P	Railed Lump	0.069%	0.01%	3	❌
8	P	Railed Fines	0.087%	0.01%	4	❌
9						❌

Figure 6.12: Screenshot presenting the dynamic configuration by time period tab.



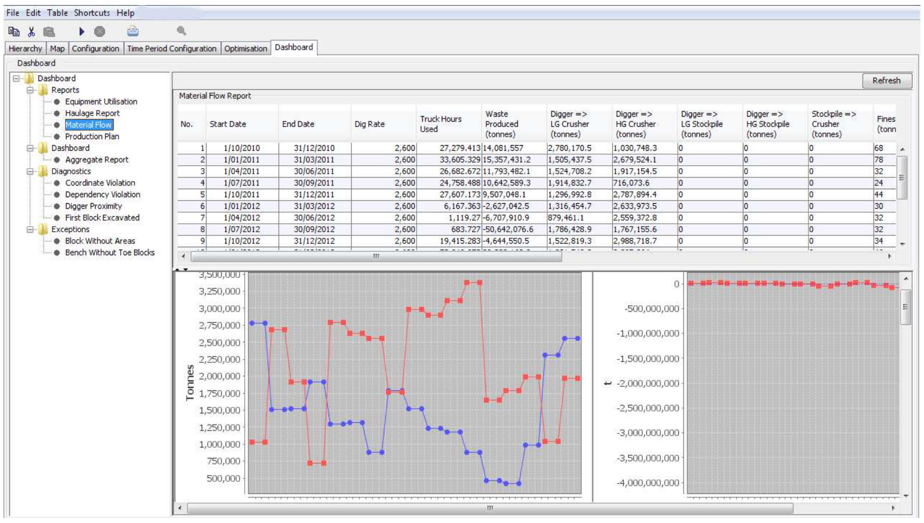
Element: Total Content Type: In-Situ Grade: High+Low Grade Tonnage Type: Wet Tonnage/Quality: Tonnes

Tonnage Schedule	Qtr-4 2010	Qtr-1 2011	Qtr-2 2011	Qtr-3 2011	Qtr-4 2011	Qtr-1 2012	Qtr-2 2012	Qtr-3 2012	Qtr-4 2012	Qtr-1 2013	Qtr-2 2013	Qtr-3 2013	Qtr-4 2013	Qtr-1 2014	Qtr-2 2014
[redacted]	6,875,000	5,900,000	5,900,000	5,900,000	5,900,000	5,775,000	5,775,000	5,775,000	5,775,000	4,925,000	4,925,000	4,961,029	4,925,000	4,125,000	4,125,(
[redacted]	123,561	1,418,546	1,936,359	1,867,997	2,708,686	922,481	2,906,492	1,346,066	483,838	0	0	89,394	545,470	1,623,099	1,206,7
[redacted]	2,707,448	1,671,960	2,566,240	2,876,193	2,796,665	3,458,939	2,714,615	2,675,881	2,486,199	1,798,694	1,620,264	497,120	957,368	2,269,455	1,677,7
[redacted]	330,090	1,005,564	1,775,565	0	0	371,917	0	90,478	0	0	0	0	0	0	
[redacted]	0	357,487	0	0	448,307	0	0	0	0	0	0	0	0	0	
[redacted]	300,200	0	0	0	0	0	155,999	6,649	0	0	0	0	0	0	
[redacted]	0	90,258	409,554	0	19,902	14,003	0	0	0	0	0	0	0	0	
[redacted]	0	55,119	0	0	506,591	1,283,882	0	0	0	0	0	0	0	0	
[redacted]	0	0	0	0	0	0	2,136,630	2,578,753	2,129,299	1,619,203	1,611,588	329,675	0	0	
[redacted]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
[redacted]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Stockpiling Schedule	Qtr-4	Qtr-1	Qtr-2	Qtr-3	Qtr-4	Qtr-1	Qtr-2	Qtr-3	Qtr-4	Qtr-1	Qtr-2	Qtr-3	Qtr-4	Qtr-1	Qtr-2	Q

Material: Fe Content Type: Railed Lump

Figure 6.13: Screenshot of the optimisation tab with the performance graphs and detailed schedule.



No.	Start Date	End Date	Dig Rate	Truck Hours Used	Waste Produced (tonnes)	Digger => LG Crusher (tonnes)	Digger => HG Crusher (tonnes)	Digger => LG Stockpile (tonnes)	Digger => HG Stockpile (tonnes)	Stockpile => Crusher (tonnes)	Fines (tonn
1	1/10/2010	31/12/2010	2,600	27,279.413	14,081,557	2,780,170.5	1,030,748.3	0	0	0	68
2	1/01/2011	31/03/2011	2,600	33,605.329	15,357,431.2	1,505,437.5	2,679,524.1	0	0	0	78
3	1/04/2011	30/06/2011	2,600	26,682.672	11,793,482.1	1,524,708.2	1,917,154.5	0	0	0	32
4	1/07/2011	30/09/2011	2,600	24,758.488	10,642,589.3	1,914,832.7	716,073.6	0	0	0	24
5	1/10/2011	31/12/2011	2,600	27,607.173	9,507,048.1	1,296,992.8	2,787,894.4	0	0	0	44
6	1/01/2012	31/03/2012	2,600	6,167.363	-2,627,042.5	1,316,454.7	2,633,973.5	0	0	0	30
7	1/04/2012	30/06/2012	2,600	1,119.27	-6,707,910.9	879,461.1	2,559,372.8	0	0	0	32
8	1/07/2012	30/09/2012	2,600	683.727	-50,642,076.6	1,786,428.9	1,767,155.6	0	0	0	32
9	1/10/2012	31/12/2012	2,600	19,415.283	-4,644,550.5	1,522,819.3	2,988,718.7	0	0	0	34

Figure 6.14: Screenshot of the dashboard tab.

6.6.5 Dashboard Tab

The last tab is the dashboard tab which presents the user with various kinds of reports, showing different KPIs, error and diagnostic information. For example, equipment utilisation, haulage reports, various aggregated and material flow reports, coordinate, dependency and other violations and data exceptions.

6.6.6 What-if Functionality

The software is allowsg the user to configure hypothetical mine and equipment configurations and then run sequencing optimisations and view KPI reports, thereby enabling an evaluation of what would happen if those scenarios were actually implemented. The factors with which the user would be able to experiment under what-if scenarios are:

- changing the location of a stock pile or dump
- changing equipment capacity and / or availability
- changing quality & tonnes targets
- reacting to events such as slope failure or flooding

6.7 Summary

In this chapter we considered a highly constrained mining problem. Firstly, the description of the problem was presented, followed by the approach based on metaheuristics, and then there is a description of the functionality of the software along with its configuration and constraints.

Each of the configuration categories presented in section 6.6 present additional constraints to the problem, which makes it extremely hard to find even one feasible solution. The complexity of the problem can be thought of from several perspectives. If we take, for example, such NP-hard problems as the travelling salesman problem, vehicle routing problem, various scheduling problems and other classical optimisation problems, they present very hard combinatorial complexity. Further, normally they are not very constrained, which makes them hard to apply in practical applications. The real-world problems usually have an additional layer of complexity – complexity by constraints. In addition to an enormous number of constraints, they are also non-linear and very often change over time. This chapter shows that the methods of computational intelligence are appropriate to solve highly constrained problems such as mining.

This work concentrated on optimal scheduling within a single mine. However, the concepts described here can be extended to multiple mines. This problem is known as integrated planning in the mining industry.

The following chapter describes a five echelon vertically integrated supply chain from the wine industry.

Chapter 7

Multi-echelon Supply Chain

In this chapter a wine supply chain of five echelons is described. Section 7.1 briefly discusses different components of the wine supply chain, three of which are described in consecutive sections. Section 7.2 explains vintage intake planning which is a process of harvesting grape blocks from vineyards, and carrying the grapes into wineries where they will be further processed. Section 7.3 introduces the tank farm process of the supply chain. The last section concludes the chapter. The bottling component of the supply chain has already been described in detail in Chapter 3.

7.1 Wine Supply Chain

A wine supply chain is a complex multi-component system with various kinds of inter- and intra- component dependencies. A typical supply chain may consist of components such as maturity models, vintage intake planning, crushing, tank farm, bottling, supply of dry goods (for example bottles, corks, and bottle labels), various distribution components (such as distribution depots and hubs), storage components (warehouses), and external sourcing of raw materials. In [161] Stadler and Kilger explain that a

> "...supply chain consists of two or more legally separated organizations, being linked by material, information and financial flows."

In order to reduce the complexity of this large supply chain, a company may concentrate on the subset of the supply chain that belongs to this company.

This type of management control where several consecutive stages of a supply chain are operated by a single company is called *vertical integration*, and a supply chain with this style of management is called a *vertically integrated supply chain*. In Figure 7.1 the dotted components represent components external to the company, for example dry goods suppliers, distribution companies, and storages. The components, with applications screenshots, represent the vertically integrated wine supply chain, all components of which belong to the given company. Material flow goes from left to right in this figure, whereas information flows from right to left. The vertically integrated supply chain in this figure is represented by five components: Maturity Models, Vintage Intake Planning, Crushing, Tank Farm and Bottling. These components will be described in detail later in this section.

From the broader perspective, these entities may not be legally separated, especially in vertically integrated supply chains, as they are within the jurisdiction of a single wine company. In the more generic sense, a supply chain consists of generally separate entities (within one structure) involved in the production and distribution of a certain commodity, with possibly different key performance indicators (KPIs), being linked by material, information and financial flows. This vertically integrated system can have two types of KPIs: *global*, which reflect a measure of the supply chain as a whole, and *local*, which show the performance of the individual component.

Focussing on the vertically integrated supply chain, which is a core element in the whole system, rather than on the individual components offers many benefits from the financial and business points of view for the company. A software application has been developed for each of the components. All these software applications were based on the paradigms of Computational Intelligence. When deployed together, these applications can optimise all planning and scheduling activities across a winery's entire supply chain, and can help wineries to optimise various business processes, predict and deal with unexpected events, and address key operational issues, such as:

- creating optimal harvest schedules that can accommodate last-minute changes.
- maximising the utilisation of crushers, pressers, and fermenters.
- optimising tank farm transfers and activities.
- dealing with sudden changes (e.g., delayed transport, demand spikes,

equipment failure, extreme weather conditions)

- improving resource allocation under risk and uncertainty.
- minimising transportation and other logistics costs.
- improving the accuracy of demand forecasts.

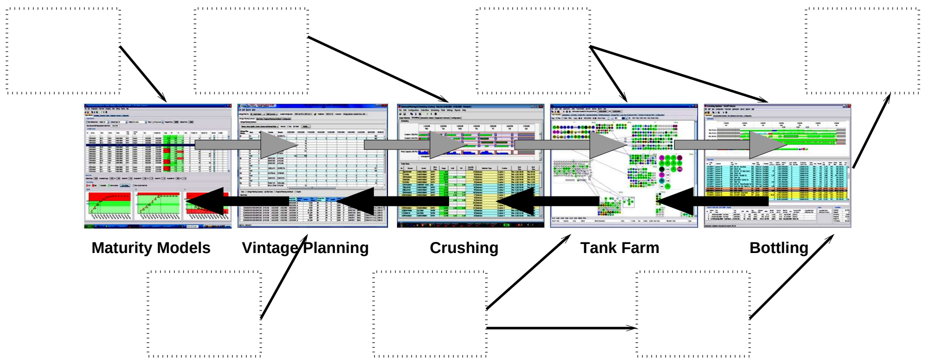


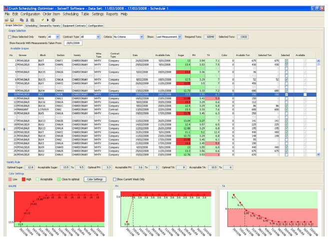
Figure 7.1: Wine Supply Chain. Grey arrows on this diagram represent a material flow. Black arrows represent information flow.

Having a system that controls the whole supply chain becomes even more essential because of the dynamic nature of the market and constantly arriving and changing customer orders with many, often conflicting, objectives. Apart from finding the globally optimal solution of how to run the supply chain in a relatively short period of time, this vertically integrated system has additional benefits such as the *ability to quickly react to dynamic changes to the schedule* giving the company a highly competitive advantage, as well as the ability to *quickly identify bottlenecks and problems*, thereby significantly decreasing the chance of human error. One of the most significant benefits is the possibility of a *global what-if scenario analysis*, enabling the quick analysis of the impact of certain changes in one or several parts of the supply chain on other parts. Such analysis is essential for schedulers and analysts. Looking at KPIs (both global and local) schedulers can evaluate whether a certain change had a positive or negative effect on the whole system.

This section briefly describes five components of a vertically integrated wine supply chain. These software applications include predictive modelling for grape maturity (using weather forecasts and readings on Baum, PH, and TA), vintage planning, crush scheduling, tank farm optimisation, bottling-line sequencing, and demand forecasting. In the later sections, due to space limitations, only three of these components are described in detail.

All the systems are coordinated by the *global module* which makes possible the cooperation between the components for the whole vertically integrated supply chain. The global module was implemented with algorithms based on cooperative coevolution [141] as described in previous chapters, and is further discussed in section 7.4.

7.1.1 Maturity Models


Crush Scheduling Optimizer - SolveIT Software - Data Set: 11/02/2008 - 17/02/2008 - Schedule 1
File Edit Configuration Order Item Scheduling Table Settings Reports Help
Grape Selection
Scheduling
Demand by Variety
Equipment Contracts
Configuration
Color Settings
Color Settings
BAUME
PH
TA

Grape Selection

☐ Show Selected Only · Variety: All · Contract Type: All · Criteria: No Criteria · Show: Last Measurement · Required Tons: 60048 · Selected Tons: 13632

Show Records With Measurements Taken From: 28/01/2008

Available Grapes

No	Grower	Block	Section	Variety	Wine Type	Contract Type	Date	Available Date	Sugar	PH	TA	Color	Available Ton	Selected Ton	Selected	Available
1	MSWLDELR	BLKT	CHAT1	CHARDONNAY	WHTV	Company	26/02/2008	5/01/2008	13	3.54	7.1	0	675	675	✓	✓
2	MSWLDELR	BLKM	CHAM1	CHARDONNAY	WHTV	Company	29/02/2008	9/01/2008	13.4	3.53	7.5	0	430	430	✓	✓
3	MSWLDELR	BLKA	SAUA1	SAUVIGNON BLANC	WHTV	Unknown	11/02/2008		[illegible]	[illegible]	[illegible]	0	80	0	☐	
4	MSWLBALR	BLK15	CHA2A	CHARDONNAY	WHTV	Company	5/02/2008	18/01/2008	13.3	3.36	7	0	36	0	☐	✓
5	MSWLDELR	BLKN	SAUN1	SAUVIGNON BLANC	WHTV	Unknown	9/02/2008		[illegible]	[illegible]	[illegible]	0	1,400	0	☐	
6	MSWLBALR	BLK15	CHA1A	CHARDONNAY	WHTV	Company	8/02/2008	18/01/2008	12.5	3.29	7	0	52	52	✓	✓
7	MSWLDELR	BLKY	CHAY2	CHARDONNAY	WHTV	Company	26/02/2008	11/01/2008	13.55	3.5	7	0	450	0	☐	✓
8	MSWLDELR	BLKK	SAUK1	SAUVIGNON BLANC	WHTV	Unknown	5/02/2008		11.63	[illegible]	6.4	0	585	0	☐	
9	MSWLDELR	BLKW	CHAW1	CHARDONNAY	WHTV	Company	13/02/2008	11/01/2008	12.73	3.33	7.2	0	680	680	✓	✓
10	MSWLDELR	BLKJ	CHAJ6	CHARDONNAY	WHTV	Company	29/02/2008	18/01/2008	13.5	3.55	6.7	0	150	150	✓	✓
11	MSWLBALR	BLK6	CHA11	CHARDONNAY	WHTV	Company	18/02/2008	23/01/2008	13.88	3.43	5.9	0	150	0	☐	✓
12	MSWLBALR	BLK16	CHA1A	CHARDONNAY	WHTV	Company	9/02/2008	20/01/2008	13.4	3.29	6.6	0	112	0	☐	✓
13	MSWLBALR	BLK16	CHA1C	CHARDONNAY	WHTV	Company	9/02/2008	20/01/2008	12.9	3.29	6.8	0	86	86	✓	✓
14	MANGANT	LOT6919	CHA1	CHARDONNAY	WHTV	Contract	13/02/2008	22/01/2008	13.3	3.33	7.6	0	600	600	✓	✓
15	MSWLBALR	BLK5	CHA1A	CHARDONNAY	WHTV	Company	19/02/2008	17/01/2008	13.78	3.45	6.3	0	250	0	☐	✓
16	MSWLDELR	BLKR	SAUR1	SAUVIGNON BLANC	WHTV	Unknown	11/02/2008		[illegible]	[illegible]	[illegible]	0	778	0	☐	
17	MSWLBALR	BLK13	CHA1B	CHARDONNAY	WHTV	Company	12/02/2008	26/01/2008	13.04	3.27	7	0	141	141	✓	✓
18	MSWLDELR	BLKU	CHAU4	CHARDONNAY	WHTV	Company	14/02/2008	11/01/2008	12.4	3.57	6.5	0	225	225	✓	✓
19	MSWLBALR	BLK13	CHA1A	CHARDONNAY	WHTV	Company	12/02/2008	26/01/2008	12.88	3.29	6.8	0	195	195	✓	✓
20	MSWLDELR	BLKT	CHAT2	CHARDONNAY	WHTV	Company	31/01/2008	5/01/2008	11.1	3.1	12.4	0	440	440	✓	✓
21	MSWLBALR	BLK14	CHA1B	CHARDONNAY	WHTV	Company	11/02/2008	23/01/2008	12.73	3.29	6.8	0	148	148	✓	✓
22	MSWLBALR	BLK4	CHA2	CHARDONNAY	WHTV	Company	19/02/2008	17/01/2008	13.92	3.45	6.1	0	55	0	☐	✓
23	MSWLBALR	BLK3	CHA1A	CHARDONNAY	WHTV	Company	21/02/2008	17/01/2008	14.4	3.45	5.9	0	190	0	☐	✓
24	MSWLDELR	BLKH	CHAH3	CHARDONNAY	WHTV	Company	4/03/2008	5/01/2008	13	3.55	6.9	0	440	440	✓	✓
25	MSWLDELR	BLKU	CHAU5	CHARDONNAY	WHTV	Company	25/02/2008	11/01/2008	13.3	3.56	6.6	0	675	675	✓	✓
26	MSWLDELR	BLKU	CHAU1	CHARDONNAY	WHTV	Company	15/02/2008	23/01/2008	12.76	3.53	1	0	670	0	☐	✓

Variety Rule

Optimal Sugar: 12.8 · Acceptable Sugar: 13.5 To 9.5 · Optimal PH: 3.3 · Acceptable PH: 3.6 To 3 · Optimal TA: 8 · Acceptable TA: 10.5 To 6

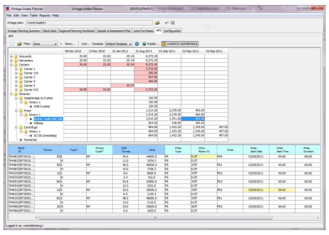
Color Settings: Low · High · Acceptable · Close to optimal · ☐ Show Current Week Only

Vintage intake plans are heavily dependent on the prediction of expected grape maturity dates. It is possible to export the prediction dates from some external system that functions as a black box providing only one date when it believes the harvesting should occur. However, limited visibility into the prediction process often prompts requests to revisit the prediction functionality of this process. So the maturity models deploy a new prediction module that provides improved prediction dates and the visibility of the prediction-calculation provided.

Grape maturity can be defined as the physiological age of the berry on the vine. It is important to define the optimal grape maturity for wine production and to develop clear chemical or biochemical traits that can be used to define the peak of ripeness. The definition of optimal maturity will vary depending upon the style of wine being made; the working definition of quality; varietal; rootstock; site; interaction of varietal, rootstock and site; seasonal specific factors; viticultural practices; and downstream processing events and goals. If a clear descriptive analysis of the quality target exists, then the time of harvest can be optimised to meet those goals. Several grape and cluster characteristics have been used to assess ripeness (e.g. sugar, pH, acidity, berry metabolites, berry proteins, taste). There are, of course, other non-compositional factors that influence the decision to harvest, including labour availability; seasonal changes such as rainfall; heat waves; tank space limitations; and other factors beyond the winemaker's control.

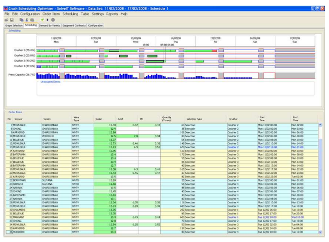
A "black box" prediction approach provides no audit capability for the user, making it difficult to detect and promptly address issues related to accuracy of prediction. These factors can easily cause errors in forecasting maturity dates to go unnoticed and unrectified for prolonged periods of time. Decisions on when to book certain grapes for harvesting and crushing rely heavily on the experience of the personnel involved in the process, and may result in a non-optimal allocation of harvesting and crushing resources. Each of these situations could result in higher costs for harvesting, transportation, and crushing, and reduction in grape quality.

7.1.2 Vintage Intake Planning



Vintage intake planning manages the grape supply intake from the "vineyard to the weighbridge". The functionality of this module supports the creation and maintenance of vintage intake plans that satisfy capacity constraints and facilitate the harvesting of grapes during periods of time when the quality is the highest. This stage is described in detail in section 7.2.

7.1.3 Crushing

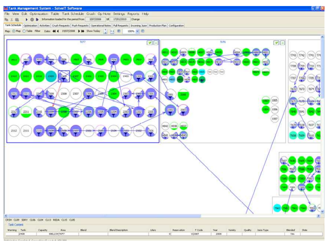


Crushers are used to process wine grapes and are often connected to different types of pressing machines, The optimal processing capacity of the crushing machines is about 40-45 tons per hour. However, if necessary, it may be increased to 60-80 tons per hour. The most important limiting factor is the capacity of the pressing machines and fermentation containers. The processing capacity for the pressing machines ranges from 4 to 12 tonnes per hour depending on the type of grapes. It is important to generate optimal schedules for all crushers over some time horizon. However, the generated weekly schedule may incur frequent changes due to contractual influences, weather conditions, seasonal influences, and daily production variances. When the changes occur, the Crush Scheduler re-optimises and generates alternative schedules to fill available capacity. Also, a variety of constraints are present in this part of the wine supply chain, including:

- constraints in time (e.g., not processing fruit during the hottest part of the day).

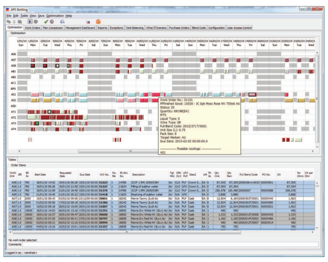
- constraints in the throughput of presses.
- constraints in the throughput of fermentation vessels.
- the throughput of trucks via the crusher to be a continuous flow.
- scheduled repairs and maintenance of equipment.
- scheduled changeover and clean up (white to red, or lower grade grape-to-higher grade grape).
- special demand-fulfilling variety shortages to address meeting capacity needs.

7.1.4 Tank Farm



Wineries make daily decisions on the processing and storage of wines and juices in their tank farms, with major wineries having several hundred tanks that have differing capacities and attributes. These tanks may be insulated or refrigerated, for instance, and could have an agitator. Some of these tanks might be used to store juice after the grapes are crushed, fermented, and pressed, while others might be primarily designated for blending. Different types of juices may also require specific tank attributes for processing, such as refrigeration jackets or agitators. This stage is described in detail in section 7.3.

7.1.5 Bottling



The primary task of the Bottling-Line Scheduler is to generate optimal production schedules for the wineries' bottling operations. The software uses advanced optimisation techniques for generating optimal production schedules. Opportunities for optimisation include manipulating the sequencing order, selecting which bottling lines to use, consolidating similar orders within the planning horizon, and suggesting changes to the requested dates that improve the overall schedule. Some of the key objectives are to maximise export and domestic service levels (i.e. DIFOT), maximising production efficiency, and minimising cost. This stage is described in detail in Chapter 3.

7.2 Vintage Intake Planning

This section describes the Vintage Intake Planning component of the wine supply chain in detail.

7.2.1 Description of the Problem

The aim of the vintage intake is to harvest grape blocks from vineyards, and to carry the grapes into wineries where they will be further processed. Vintage intake planning, within the scope of this chapter, describes the planning of all necessary actions for bringing wine grapes from vineyards into wineries and the immediate subsequent grape processing steps within the winery. The vintage intake planning finishes once the juice's fermentation is over, after which the wine will be refined in the tank farm (described in section 7.3). Parties involved in the intake process are harvesting and carrier companies and the winery personnel.

While wineries and the winery equipment such as crushers, presses, centrifuges and fermenters are owned by the wine producing company, the harvesting and transportation of grapes into the winery is carried out by contractors. Those contractors usually provide their services to many wine producers. Due to the nature of the ripening process of grapes and the inherent variations caused by different grape varieties, locations of the blocks and constantly changing weather conditions during the maturity process, the availability of contractors such as harvesters or truck providers fluctuates significantly, and is often unknown until very close to the actual date when their services become necessary.

Both the uncertainty about logistics resources and the inability to control them by vintage planners demand a constant update of the plans. The further uncertainty imposed by sudden unpredicted weather changes, which impacts on the grape maturity and thus the preferred wine intake da,y limits the planning horizon to only one or two weeks. These short-term plans are created in the week before the planning week commences, but do not get locked in until two days prior. At this point in time, the logistics manager for a particular region will have finished the process of requesting the required capacities to harvest and carry the grapes, and the service providers will have responded with their offer of available resources that meet or lower the necessary requirements. This process will continue until all resource requirements are met. Or until none of the available providers within the region has any capacities left. The resources will be booked and locked in to the plan. After locking in the available resources, only unexpected events such as technical problems or human error can compromise the plan.

Harvesters and trucks are not the only resources to take into consideration when drafting a vintage intake plan. The intake does not finish at the arrival at the winery, but after the wine grapes have been processed. At the winery the processing of the grapes begins:

- the grapes get weighted on weighbridges.
- the grapes will be pressed depending on which process for the harvested grapes is followed (usually only red grapes get pressed).
- some crushed grapes may require to stay together with the stalks and skins for a few days.
- the grapes get pressed (usually whites).
- the resulting press cut juices are centrifuged.
- finally, the juice is fermented in fermenters for about 1-2 weeks.

The above steps are by no means a complete list of possible processes, but they represent steps that pose major constraints on the grape intake. They represent the constraints that human planners would take into account when assessing the winery intake capacity for drafting an intake plan.

7.2.2 Constraints

The logistic constraints during peak season of the vintage are probably the most pressing capacity constraints faced by a vintage intake planner. Not only do they take into account the growers' preferred harvesters, but also they compete with other wine producing companies in allocating and booking carrier capacities.

The first resource at the winery to pose a capacity bottleneck is the weighbridge as in most cases it can only be used sequentially. In some instances, it may be connected with the crushers in which case grapes will be loaded into special bins which incorporate a weighing machine. The filled bin will be subsequently tipped into the crushers which grind the grapes. Prior to the crushing, the winemaker undertakes a final assessment of the quality of the grapes and

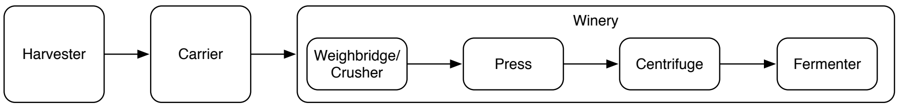
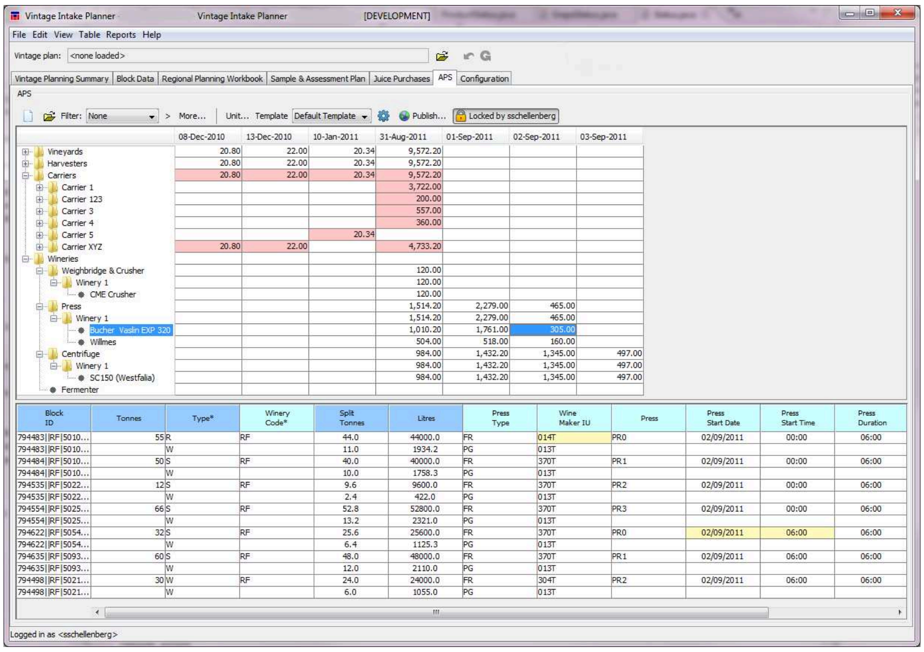


Figure 7.2: Grape Flow at intake. Note that not all grape processing steps within a winery are mandatory.

classifies the grapes into their final usage. This final usage determines the processes the grapes go through in the manufacturing network all the way to the final bottling of the product.


Vintage Intake Planner — Vintage Intake Planner [DEVELOPMENT]

File Edit View Table Reports Help

Vintage plan: <none loaded>

Vintage Planning Summary | Block Data | Regional Planning Workbook | Sample & Assessment Plan | Juice Purchases | APS | Configuration

APS

Filter: None > More... Unit... Template Default Template Publish... Locked by sschellenberg

	08-Dec-2010	13-Dec-2010	10-Jan-2011	31-Aug-2011	01-Sep-2011	02-Sep-2011	03-Sep-2011
Vineyards	20.80	22.00	20.34	9,572.20			
Harvesters	20.80	22.00	20.34	9,572.20			
Carriers	20.80	22.00	20.34	9,572.20			
Carrier 1				3,722.00			
Carrier 123				200.00			
Carrier 3				557.00			
Carrier 4				360.00			
Carrier 5			20.34				
Carrier XYZ	20.80	22.00		4,733.20			
Wineries							
Weighbridge & Crusher				120.00			
Winery 1				120.00			
CME Crusher				120.00			
Press				1,514.20	2,279.00	465.00	
Winery 1				1,514.20	2,279.00	465.00	
Bucher Vaslin EXP 320				1,010.20	1,761.00	305.00	
Willmes				504.00	518.00	160.00	
Centrifuge				984.00	1,432.20	1,345.00	497.00
Winery 1				984.00	1,432.20	1,345.00	497.00
SC150 (Westfalia)				984.00	1,432.20	1,345.00	497.00
Fermenter							

Block ID	Tonnes	Type*	Winery Code*	Split Tonnes	Litres	Press Type	Wine Maker IU	Press	Press Start Date	Press Start Time	Press Duration
794483||RF|5010...	55	R	RF	44.0	44000.0	FR	014T	PR0	02/09/2011	00:00	06:00
794483||RF|5010...		W		11.0	1934.2	PG	013T				
794484||RF|5010...	50	S	RF	40.0	40000.0	FR	370T	PR1	02/09/2011	00:00	06:00
794484||RF|5010...		W		10.0	1758.3	PG	013T				
794535||RF|5022...	12	S	RF	9.6	9600.0	FR	370T	PR2	02/09/2011	00:00	06:00
794535||RF|5022...		W		2.4	422.0	PG	013T				
794554||RF|5025...	66	S	RF	52.8	52800.0	FR	370T	PR3	02/09/2011	00:00	06:00
794554||RF|5025...		W		13.2	2321.0	PG	013T				
794622||RF|5054...	32	S	RF	25.6	25600.0	FR	370T	PR0	02/09/2011	06:00	06:00
794622||RF|5054...		W		6.4	1125.3	PG	013T				
794635||RF|5093...	60	S	RF	48.0	48000.0	FR	370T	PR1	02/09/2011	06:00	06:00
794635||RF|5093...		W		12.0	2110.0	PG	013T				
794498||RF|5021...	30	W	RF	24.0	24000.0	FR	304T	PR2	02/09/2011	06:00	06:00
794498||RF|5021...		W		6.0	1055.0	PG	013T				

Logged in as <sschellenberg>

Figure 7.3: Screenshot of the Vintage Intake Planning System

Depending on the quality of the grapes and the intended final product to which they contribute, the grapes will be routed through different processing states.

Presses pose the next potential bottleneck in the grape processing pipeline. Presses are also used to extract juice from grapes, but they use a much finer extraction technique than crushers. By pressing, the winemaker is able to extract different quality grades from the grapes, the so-called *press cuts* or *pressings*. They are constrained by the capacity of grapes they can be filled (typically around 40 tonnes) and the actual time it takes to carry out the pressing (approximately six hours, depending on the load and type of press).

Further clarification prior to fermentation is carried out in the centrifuges. Unlike presses, centrifuges are able to process a constant stream of juice. Depending on the type of centrifuge and the state of the infeed, centrifuges can reach a rate of $60,000$L of juice per hour [21].

The last step of the intake is the fermentation of juice. Fermenters can be filled over a period of days. It takes approximately 10-14 days for the fermentation to be finished; the fermentation vessel is then available for new juice. In some instances, however, the fermentation tank is also used for storage, so it will not return to the pool of empty tanks. This may be the case when capacity shortages in the tank farm occur, and which may be for reasons the planner cannot anticipate.

The above mentioned winery equipment has a physical processing capacity limit. Those limits vary only rarely so that planning of their usage and anticipated completion time can be easily determined. Issues arise if winery equipment is not functional (unplanned outages), or personnel who operate the equipment are not available. In these cases subsequent assignments become infeasible as they rely on the operation of the now unavailable equipment. The next weekly plan has to take into account and adapt to the change in operational resources.

Figure 7.3 shows the screenshot of the Vintage Intake Planning system illustrating the flow of material and the conversion from grapes to juice to wine by various processing steps. The figures in the table represent the quantity of grapes being harvested, carried and crushed. From the press stage on downwards, the values are shown in litres. Note that constraint violations are highlighted in the upper overview table (red background cell colour). A selection of the violating cell will bring up the below detailed table with all necessary information on the root cause of the violation.

7.3 Tank Farm

This section describes the Tank Farm component of the wine supply chain in detail.

7.3.1 Description of the Problem

The tank farm of a large winery is a major and integral part of the successful and efficient wine manufacturing enterprise. It interfaces with all the other major processes, by: accepting juice and wine from the intake process, providing wine for bottling, and being the source and destination of the company-level activities of inter-winery transfers, bulk wine exports, and wine purchased for blending. It is also the scene for wine production through the carrying out of additive, maturation, fermentation and blending tasks. The complexity and range of the varied products and activities in the tank make it a significant planning problem, and one operating at a wide range of time scales, thus needing to adapt to a dynamically changing environment.

The tank farm itself is generally a large area containing many (in the order of several hundred to a thousand) wine tanks in a range of sizes. At any given time a proportion of these tanks will be empty in order to provide spare capacity for receiving wine and to increase the flexibility for blending and other tasks that may require transfers between tanks. Within the tank farm there may be groupings of tanks by function (for example, an area with tanks receiving intake; or an area for storing wines ready for bottling) but overall the tank farm is usually in a relatively compact area (e.g. in an approximate square) in order to allow relatively flexible transfers between any pair of tanks, without incurring the penalty of needing to transfer over extremely large distances.

Apart from size variations, some tanks are distinguished by having specific features, such as agitators, refrigeration, insulation, or provision for oaking wine. Apart from the tanks themselves, the tank farm may include a number of additional elements. These include various types of equipment (such as filters, centrifuges or heat exchangers); some of this equipment is portable, but some is at fixed locations within the tank farm. In addition the tank farm may have a a fixed piping and manifold system for transferring wine between tanks, supplemented with portable ground pipes for the same purpose, providing more flexibility but with higher labour and setup costs.

Tank Selection

Overall there are two main optimisation tasks for organising activities within the tank farm, and these can be simply described as relating to *which tank(s)* should be chosen, and *when* a specific operation on the tanks (such as a transfer) should occur. The first of these activities can be divided between the two quite distinct tasks of *Source Tank Selection* and *Destination Tank Selection*.

Obviously the primary constraint in source tank selection concerns the actual content of a tank from which we plan to draw. If we need to bottle an order for 5,000 litres of 2009 Shiraz, then we must draw from source tanks with the correct variety and vintage year. This constraint is common to any system of warehousing or stocking of multiple types of product. What is specific to the wine industry, and the tank farm situation in particular, is the need to do this in a way that avoids oxidation of wine through contact with the air. If wine is left in a partially filled tank, then, over time, the wine will degrade and lose significant value. It is such a serious problem in the wine industry that it has specific terms: *ullage* is the proportion of air within a wine tank, and *being on ullage* refers to a tank and its content having non-zero ullage.

For destination tank selection, we are frequently dealing with empty tanks. An important rule for optimising the global behaviour is to ensure that specialist tanks are not selected if not needed - for example, fermentation tanks should not be selected as a destination for a generic transfer (particularly during the vintage season) unless other tanks are not available.

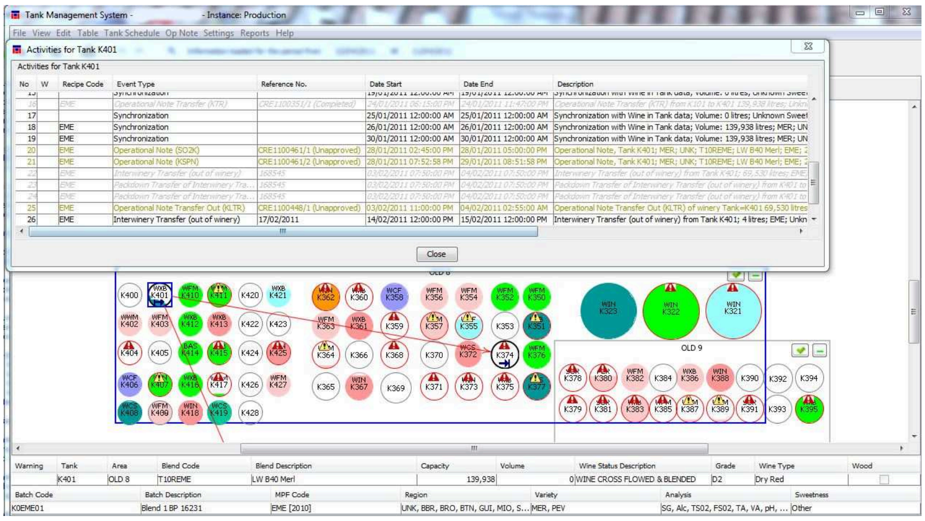
Other rules that relate equally to source and destination tank selection, revolve around reducing the costs associated with the execution of the transfers. A transfer over a longer distance is more expensive, due to increased setup times (for example to lay out a ground pipe) and the additional wastage and water use that is required for cleaning out the longer pipe. In a similar way, having many transfers to or from many smaller tanks will be more expensive than a single transfer involving a single pair of tanks.

Labour and Resource Scheduling

Once the operations of the tank farm, and the selection of tanks have been decided, there remains the scheduling issue of when the operation will occur, by whom, and with what specific equipment resources.

Each operation can be divided into a number of subtasks, such as setup, start-up, execution, and finish-up. For each such task fixed amounts of time, labour and, potentially, equipment, are required. With an execution task, such as a transfer between tanks, there may also be a variable cost in time, depending on the number of litres being transferred.

For routine planning, a standard schedule of times of various shifts of tank farm workers is defined, and the various types and number of equipment can


Tank Management System - - Instance: Production
File View Edit Table Tank Schedule Op Note Settings Reports Help
Activities for Tank K401
Close
OLD 8
OLD 9

Activities for Tank K401

No	W	Recipe Code	Event Type	Reference No.	Date Start	Date End	Description
15			Synchronization		19/01/2011 12:00:00 AM	19/01/2011 12:00:00 AM	Synchronization with Wine in Tank data; Volume: 0 litres; Unknown Sweet
16		EME	Operational Note Transfer (KTR)	CRE1100351/1 (Completed)	24/01/2011 06:15:00 PM	24/01/2011 11:47:00 PM	Operational Note Transfer (KTR) from K101 to K401 139,938 litres; Unkn
17			Synchronization		25/01/2011 12:00:00 AM	25/01/2011 12:00:00 AM	Synchronization with Wine in Tank data; Volume: 0 litres; Unknown Sweet
18		EME	Synchronization		26/01/2011 12:00:00 AM	26/01/2011 12:00:00 AM	Synchronization with Wine in Tank data; Volume: 139,938 litres; MER; UN
19		EME	Synchronization		30/01/2011 12:00:00 AM	30/01/2011 12:00:00 AM	Synchronization with Wine in Tank data; Volume: 139,938 litres; MER; UN
20		EME	Operational Note (SO2K)	CRE1100461/1 (Unapproved)	28/01/2011 02:45:00 PM	28/01/2011 05:00:00 PM	Operational Note, Tank K401; MER; UNK; T10REME; LW B40 Merl; EME; 2
21		EME	Operational Note (KSPN)	CRE1100461/2 (Unapproved)	28/01/2011 07:52:58 PM	29/01/2011 08:51:58 PM	Operational Note, Tank K401; MER; UNK; T10REME; LW B40 Merl; EME; 2
22		EME	Interwinery Transfer (out of winery)	168545	03/02/2011 07:50:00 PM	04/02/2011 07:50:00 PM	Interwinery Transfer (out of winery) from Tank K401; 69,530 litres; EME
23		EME	Packdown Transfer of Interwinery Tra...	168545	03/02/2011 07:50:00 PM	04/02/2011 07:50:00 PM	Packdown Transfer of Interwinery Transfer (out of winery) from K401 to
24		EME	Packdown Transfer of Interwinery Tra...	168545	03/02/2011 07:50:00 PM	04/02/2011 07:50:00 PM	Packdown Transfer of Interwinery Transfer (out of winery) from K401 to
25		EME	Operational Note Transfer Out (KLTR)	CRE1100448/1 (Unapproved)	03/02/2011 11:00:00 PM	04/02/2011 02:55:00 AM	Operational Note Transfer Out (KLTR) of winery Tank=K401 69,530 litres
26		EME	Interwinery Transfer (out of winery)	17/02/2011	14/02/2011 12:00:00 PM	15/02/2011 12:00:00 PM	Interwinery Transfer (out of winery) from Tank K401; 4 litres; EME; Unkn

Warning	Tank	Area	Blend Code	Blend Description	Capacity	Volume	Wine Status Description	Grade	Wine Type	Wood
	K401	OLD 8	T10REME	LW B40 Merl	139,938	0	WINE CROSS FLOWED & BLENDED	D2	Dry Red	☐

Batch Code	Batch Description	MPF Code	Region	Variety	Analysis	Sweetness
K0EME01	Blend 1 BP 16231	EME [2010]	UNK, BBR, BRO, BTN, GUI, MIO, S...	MER, PEV	SG, Alc, TS02, FS02, TA, VA, pH, ...	Other

Figure 7.4: Tank Farm Map View Screenshot. The circles represent phyisical layout and contents (by colour). An Activities popup shows the transactions on a single tank

be acquired. But in the dynamic real world, this is not sufficient: workers can call in sick and pieces of equipment can become unavailable for maintenance or repair. Such variations need to be handled by a real-world system.

7.3.2 Functionality

The Tank Farm Optimiser is designed to deal with all the tank farm issues of tank selection, and resource scheduling, across the full range of scenarios, including receiving vintage intake, wine manufacturing operations, servicing bottling requests, as well as blend planning and management of bulk wine transfers (inter-winery, export and purchasing.) We describe some major features, with particular emphasis on optimisation and dealing with dynamic changes.

The Tank Farm Optimiser is arranged via a series of tabs that allow access to the main planning activities (intake, operation, bottling, bulk wine movement and blending) as well more generic informational screens (tank farm maps and schedule, capacity plan, the vintage plan, and the long term production plan). This arrangement allows customising of the interface for specific user roles: for example, an intake planner may just have the vintage plan, intake, and tank farm map/schedule tabs.

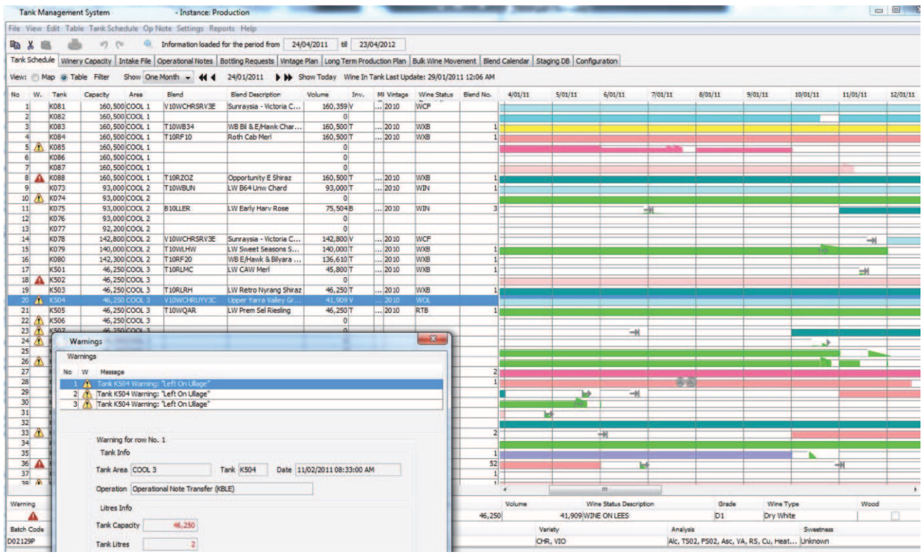
The Tank Farm Map and Table give two distinct visual representations of the contents and events of the tanks in the tank farm (see Figure 7.4 and 7.5).

In Figure 7.4 the Map view gives a graphical layout of the tanks on a specific date. The key is quite intuitive: the sizes of the tanks circles reflect the total capacity of the tanks; the colours reflect the type of content (this is customisable, and might represent the grape variety, such as Shiraz or Chardonnay); and the proportions that are coloured indicate how full the tanks are. Tanks that are all full or all white (empty) would represent a situation where no tank is left on ullage. Arrows between tank circles represent scheduled transfers, whilst warning markers indicate the presence of issues at some point in the tank's timeline (details of which are revealed by clicking on the system)

Simply clicking on a tank circle reveals other basic facts about the contents of the tank, displayed in the lower part of Figure 7.4, including the precise varieties, vintage years, exact volume, status of the wine (e.g. ready for bottling) and a succinct representation of analysis results (such as current measurements of the wine sweetness and alcohol levels). By opening up any tank,an activities screen for the tank is displayed (visible in the upper part of the figure). This screen displays the sequence of all transactions that relate to that tank in the model, such as transfers of fluids in or out, operations (such as additives), and synchronisations with updated content information.

The table view (see Figure 7.5) is similar to the map view, but the tanks are listed in a tabular form, rather than arranged by geographical location. Each row presents data for a single tank, and the various textual and numerical columns (visible on the left of the figure) give various attributes of the tank and its contents for a specific date (these columns are particularly useful for sorting and filtering for tanks having specific properties.) On the right of Figure 7.5 we see the timeline of the contents of the tank represented, providing a graphical view of the dynamic changes of the tank, both historical and planned for the future. The colour of each bar represents the type of content (using the same colour mapping as used in the map view colouring of the circles) and the height of the bar represents the proportion of the tank that is filled. Transfers are represented as icons on the bars. Scrolling the right side of this table allows the timeframe of these timelines to be altered. A wide range of filters can be applied to both views, and the selection context and filtering is maintained as the user switches between map and table views.

In the lower left of Figure 7.5 a warning icon on a row has been opened to a warning details panel, giving information about constraints that have been violated on the tank on that row. These constraints may be volume violations



Information loaded for the period from 24/04/2011 till 23/04/2012

24/01/2011 Wine In Tank Last Update: 29/01/2011 12:06 AM

No	W.	Tank	Capacity	Area	Blend	Blend Description	Volume	Inv.	MI	Vintage	Wine Status	Blend No.
1		K081	160,500	COOL 1	V10WCHRSRV3E	Sunraysia - Victoria C...	160,359	V	...	2010	WCF	
2		K082	160,500	COOL 1			0					
3		K083	160,500	COOL 1	T10WB34	WB Bil & E/Hawk Char...	160,500	T	...	2010	WXB	1
4		K084	160,500	COOL 1	T10RF10	Roth Cab Merl	160,500	T	...	2010	WXB	1
5	⚠	K085	160,500	COOL 1			0					
6		K086	160,500	COOL 1			0					
7		K087	160,500	COOL 1			0					
8	⚠	K088	160,500	COOL 1	T10RZOZ	Opportunity E Shiraz	160,500	T	...	2010	WXB	1
9		K073	93,000	COOL 2	T10WBUN	LW B64 Unw Chard	93,000	T	...	2010	WIN	1
10	⚠	K074	93,000	COOL 2			0					
11		K075	93,000	COOL 2	B10LLER	LW Early Harv Rose	75,504	B	...	2010	WIN	3
12		K076	93,000	COOL 2			0					
13		K077	92,200	COOL 2			0					
14		K078	142,800	COOL 2	V10WCHRSRV3E	Sunraysia - Victoria C...	142,800	V	...	2010	WCF	
15		K079	140,000	COOL 2	T10WLHW	LW Sweet Seasons S...	140,000	T	...	2010	WXB	1
16		K080	142,300	COOL 2	T10RF20	WB E/Hawk & Bilyara ...	136,610	T	...	2010	WXB	1
17		K501	46,250	COOL 3	T10RLMC	LW CAW Merl	45,800	T	...	2010	WXB	1
18	⚠	K502	46,250	COOL 3			0					
19		K503	46,250	COOL 3	T10RLRH	LW Retro Nyrang Shiraz	46,250	T	...	2010	WXB	1
20	⚠	K504	46,250	COOL 3	V10WCHRUYV3C	Upper Yarra Valley Gr...	41,909	V	...	2010	WOL	
21		K505	46,250	COOL 3	T10WQAR	LW Prem Sel Riesling	46,250	T	...	2010	RTB	1
22	⚠	K506	46,250	COOL 3			0					
23	⚠	K507	46,250	COOL 3			0					

Warnings

No	W	Message
1	⚠	Tank K504 Warning: "Left On Ullage"
2	⚠	Tank K504 Warning: "Left On Ullage"
3	⚠	Tank K504 Warning: "Left On Ullage"

Warning for row No. 1

Tank Info

Tank Area: COOL 3 Tank: K504 Date: 11/02/2011 08:33:00 AM

Operation: Operational Note Transfer (KBLE)

Litres Info

Tank Capacity: 46,250

Tank Litres: 2

Warning	Batch Code		Volume	Wine Status Description	Grade	Wine Type	Wood
⚠	D02129P	46,250	41,909	WINE ON LEES	D1	Dry White	☐

Variety	Analysis	Sweetness
CHR, VIO	Alc, TS02, FS02, Asc, VA, RS, Cu, Heat...	Unknown

Figure 7.5: Tank Farm Table View Screenshot. Timelines for the tanks are visible on the right, coloured by the wine variety

(e.g. an attempt to transfer more from a tank than it contains at the time that transfer is scheduled) or contents violations (e.g. the mixing of the contents of one tank into another tank, which would break configured rules of the varieties that may be mixed.)

The underlying model of tank transfer events is an effective way of dealing with the dynamic temporal aspects. For example, the tank model may show that a 20,000 tank T100 has 10,000 litres of 2009 Shiraz. Let us suppose that this is precisely what is required for us to service a bottling request for the next two days. A manual operator may be tempted to use this for the bottling; but instead, the optimiser might suggest to only take 4,000 from T100, whilst taking the remaining 6,000 from a small 6,000 litre tank, full of 2009 Shiraz. It seems like a worse solution, since T100 now continues to be left on ullage, with 4,000 litres. The explanation is that there is already a booking in the system that will take 6,000 from T100 in several days' time. In this way, the optimiser is looking ahead at the existing scheduled transfer bookings, and selecting to ensure that we do not take too much from a tank (leaving an already scheduled and approved transfer infeasible.)

The tank selection algorithm (either for source or destination) primarily works by evaluating each tank as a potential candidate. Firstly, infeasible tanks are rejected, and a reason for rejection is recorded for such tanks. For the feasible tank, a weighted set of penalties is calculated and recorded. The weighting covers the various requirements for tank selection listed before: minimising transfer distance, avoiding ullages, and avoiding fermenter tanks when fermentation is not required. There are also penalties that aim to improve the future efficiency of the tank farm, such as by encouraging clustering of tanks of the same type of wine in the same area. The highest weighted tank is the one selected. If that tank does not completely satisfy the requirement (e.g. it does not provide enough for a large export order) then the values are recalculated, as the previously second-ranked tank may not be the most appropriate given that the first tank is now being used. For example, it is quite likely that the second tank might be left on ullage, and so a smaller tank that more closely matches the outstanding requirement would now be preferable.

The optimisation of the schedule relies on tracking the resources, and placing earliest due tasks earlier that later tasks. However there can occur task order inversion, on the basis of availability of specific resources. Furthermore the algorithm assesses the costs of varied resources, aiming to use lower cost resources in preferences.

7.3.3 Results

Both the tank selection and the labour and resource optimisation features of the Tank Farm Optimiser system provide effective and timely solutions, and adapt well to real world, dynamic changes. An important aspect of the utility of the system is how it interacts with the workflows of the users and with their individual planning.

Tank Selection

As new orders and requests enter the system, via a batch loading system, they are automatically optimised together. This provides new tank selection allocations without disrupting existing allocations. This is important to help provide the users with stability in relation to the tank farm, and to preserve the additional planning information already added for existing allocations. When requests are revised, particularly in relation to volume, this situation is visually highlighted for users, but does not result in an automated re-optimisation. This is because the users may still wish to use existing tank allocations, and will merely add or remove tanks from the existing assignment.

A particularly important aspect of the use of the tank selection has been the presentation of the penalties and constraints involved with tank selection, when demanded by the user. Users have needed to be able to refer to such decision making data, in order to trust in the choices made by the Tank Farm Optimiser. We believe that this may be partly due to the traditional basis of the wine making process, where individual winemakers have been used to considerable personal control and responsibility with all aspects of the process; hence ther may be a relative unwillingness to give up some aspects of this control, particularly if the optimisation appears as a black box. Another important factor in presenting the optimisation reasoning, is that of enabling users to deal better with the effects of dynamic changes. For example, a user who is allocated a tank which appears "inferior" to that allocated to someone who books after them, can sense an aspect of unfairness - and perhaps suboptimality - in the system. However, once the optimisation records are presented, they can see that at the time of their booking, their selection was indeed optimal.

Since the incremental approach of allocating tanks is not guaranteed to be globally optimal, users also make use of other features that give extra flexibility. These include reoptimising selections that occurred previously, and manually stipulating tank selection, by-passing the optimiser. One of the noted results of manual bypassing of the optimiser is that the users' deliberate choices can work to undermine the global goals of the optimisation program - such as reducing the clustering of tanks for the same or similar wines.

Tank Farm Resources Scheduling

The labour and resource scheduling is carried out automatically, at the same time as the tank selection, when records needed for the tank operation files are loaded into the system.

Interestingly, although the Tank Farm Optimiser does not provide a rationale for its resource scheduling, this does not seem to be a major concern to users. This may be because they have less inherent interest in exactly when and by which cellar hand the work will be carried out; or it may be because in a visual presentation of the schedule, the interlinked and cross-dependent complexity of the problem is more evident, leading to less obvious substitute solutions.

As a result, the changes that are made to the schedule appear more in the nature of refinements and improvements, constructively using the existing schedule as a basis. Examples of such changes may arise in a situation where the users note that the setup time for a following task maybe reduced, or even eliminated, since it may be possible to use the equipment setup up for the preceding task and thus reflect a global optimisation that is not yet accessible to the Tank Farm Optimiser.

7.4 Algorithm for the Global Module

Previous sections of this chapter described the wine supply chain and discussed each silo in detail. Scheduling, planning and forecasting decisions for each of these silos produced by the systems were mentioned above. In this section we will refer to each of these systems as a *silo system.* These silo systems, apart from local optimisation, confer an additional advantage as they provide the ability to analyse various what-if scenarios. However, the challenge for the companies owning a vertically integrated supply chain, such as the one being considered here, is to have a decision support system that covers all the aspects of the supply chain – globally optimal decisions, taking into account intra-silo constraints of each particular silo system, as well as higher level inter-silo constraints that set the dependencies between the silos.

Additional challenge is to *reuse the existing decision support modules* of each of the developed silo systems, so the global optimisation module has to exploit and coordinate the execution of the existing local modules as the building blocks.

7.4.1 Algorithm

There are several goals for the decision support system to address in this supply chain: the maximisation of profit, the maximisation of customer satisfaction (for example, providing high quality product, delivering orders on time), and the possible minimisation of carbon emissions – this list can easily be extended with 5-10 additional KPIs. The need to address such a multiplicity goals really creates a multi-objective optimisation problem. To the best of our knowledge, no decision support system exists that addresses, from the global perspective, even one of the objective mentioned above. The algorithm described in the next subsection collects the solution quality measures from the solutions of each of the silos and links them into a single fitness via the linear combination:

$$fitness = k_1 f_1 + k_2 f_2 + k_3 f_3 + k_4 f_4 + k_5 f_5$$

where k_i are the coefficients and f_i are the fitness values of each of the silo solutions. This can be viewed as a problem with multiple objective but in this thesis we will be considering an aggregation of the objective into a single value.

The proposed algorithm that addresses the problem of optimal decision support for the wine supply chain is based on cooperative coevolution. It aligns very well with the algorithms discussed in previous chapters, especially in Chapters 4 and 5. Solutions in each of the five silo systems represent a species of a wine supply chain ecosystem. The evaluation of each of the solutions is based on how well this solution performs in cooperation with representative solutions of other species.

The evaluation of solutions is based on the supply chain simulation. It uses the simulation framework used for the experiment in Chapter 5. A chain of five silos is constructed in the framework, and each simulation silo implements a functionality based on the solution it is using. The first silo, responsible for maturity models, constructs a forecast for the second silo which is responsible for the vintage intake planning. It uses this forecast to construct the schedule from the genotype of the individual. This schedule is then fed into the third silo responsible for the crushing process. The crushing schedule (phenotype) is decoded from the crushing individual (genotype). The fourth silo, which is responsible for the tank farm, uses this information to decode its solution for the logistical decisions. The last silo (bottling) has information on customer orders as well as knowledge of the wine supply from the tank farm, and additional material requirements (such as bottles, labels, and corks) to construct a bottling schedule from the genotype provided by its individual.

A *Global Module* (GM) coordinates all the activities of the species. It first commands each silo system to initialise the individuals of its species. Then it asks for representative partner individuals from each and shares them between all the participant silo systems. From time to time the GM asks silo systems to provide partner representatives and shares them again. Each of these exchanges will be referred to as an *epoch*. These are done *asynchronously*, i.e a number of generations between two epochs is not fixed for each of the species. In the current implementation it is done based on time – each T seconds ($T = 30$ for this algorithm). This runs for $nEpoch$ times ($nEpoch = 100$ for this algorithm) after which it asks each of the silo systems for the best result and presents the user with the one with the highest fitness.

7.4.2 System Design and Communication Protocol

The cooperative coevolutionary global module needs to be designed keeping in mind that the systems at local silos need to have a minimal number of changes. There is need for a communication protocol that will allow the silo systems and a global module to communicate. In our implementation this communication is based on the Java RMI.

Each of the solutions produced by the silo systems implement the *Individual* interface which has the following methods:

- **Set<Individual> getPartnerIndividuals()** returns the set of all partnering individuals. The encoding of this set is a global solution.
- **double getFitness()** returns the fitness of individual based on cooperation with other partners
- **double getSelfFitness()** returns the fitness of individual based on local evaluation without cooperating with all the partners

Each of the silo systems produces a solution that implements this interface: *MMIndividual* for Maturity Models, *VipIndividual* for the Vintage Intake Planner, *CrushingIndividual* for the Crushing system, *TfIndividual* for the Tank Farm optimiser, and *BottlingIndividual* for the Bottling system.

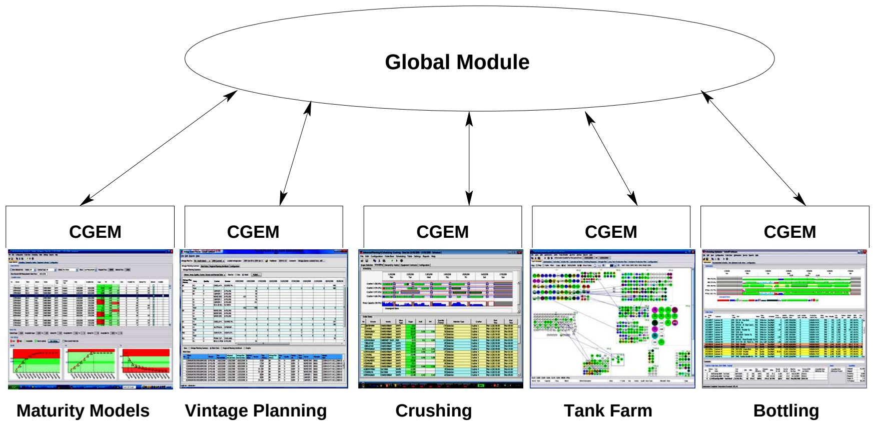


Figure 7.6: The design of the wine supply chain with global optimisation

The overall design of the supply chain under the control of global optimisation is shown in Figure 7.6. Each of the silo systems has an additional

Communication and Global Evaluation Module (CGEM) attached which is responsible for instantiation of communications with the *Global Module* and also evaluation the solution in conjunction with the partner solutions. When the system is deployed with the CGEM, it can switch to the *server mode* which lets the silo system to receive commands from the GM. GM can request the silo system through CGEM to initialise the session and optimisation, return the best results so far, and update partner representatives and certain variables. As this communication is based on Java RMI all silo systems can reside on different computers and are connected through the network. This turns the global supply chain optimisation to the distributed system and presents an additional advantage of utilising concurrency.

The process of communication involves the following steps:

1. **Initialise Session.** This is done when the GM invokes the *initSession()* method. When received the system initialises all variables for the session and returns its unique name so the GM verifies the identity of the silo.

2. **Initialise Optimiser.** This stage is only done when the response for *initSession()* is received from all the systems. Then it invokes the *initOptimisation()* method to initialise optimiser modules on each of the silo systems and then initialise the population (in case of non population based optimisers a single solution is created). The system responds with the *OK* or *FAIL* message.

3. **Request Partner Individuals.** After all systems return *OK* to initialise the optimiser, the next step is to request the partner individual from each of the silo systems via *getPartner()* remote method.

4. **Set Partner Individuals.** When all the partner individuals are received, the GM distributes all of them to all silo systems via the *setPartners(individuals)* remote method. If this is done the first time in the current session, this method kick starts the optimiser. CGEM then returns an *OK* message to confirm that partner individuals were set. After that GM waits for T seconds and calls step number 3 (Request Partner Individuals) again. This loop is repeated *nEpoch* times.

5. **Stop Optimisation.** The method *stopOptimisation()* stops the optimiser in silo systems, then the best so far individual is returned as the result of the call and then closes the session.

This process controls the execution of algorithms at each of the silo systems guided by the GM. After the *nEpoch* partner synchronisations the GM gathers the best individuals from all the silo systems and presents the user with the one with the highest fitness.

7.4.3 Results

After the GM was implemented, each of the silo systems was deployed with the CGEM module on separate computers. The systems were run and the server mode was turned on. The GM, run on different computers on the network, started to manage the optimisation execution of the silo systems. It was configured to run for 100 epochs and the result was produced within one hour. After a careful examination of the solution by the domain expert, it was concluded that "this solution can be used to provide the decision support process". It is hard to compare this solution with the current production solution as there is no system that would gather all the pieces of the local solution in one to produce a global solution.

An additional benefit of the global optimisation is the global what-if scenario analysis. The operator can change almost any setting of any of the silo systems as well as "freeze" a solution at a particular silo system and run a global optimisation around that fixed solution to produce the result. This would significantly improve the decision process, not only as to how to operate the supply chain, but also as to what short and long term decisions or investments could be made to improve it.

7.5 Summary

Previous sections described five components of the vertically integrated supply chain. The decision support of each system is coordinated by the global module which makes possible the cooperation between the components of the whole supply chain. The existence of the global module makes possible the existence of several significant benefits in optimising the vertically integrated wine supply chain – these include:

- better use of available production capacity.
- reduced risk of late deliveries due to production capacity issues, supply issues, or scheduling errors, and better visibility of potential risk.

- the ability of schedulers to quantify the relative merits of different schedules, and make informed decisions as to which scheduled to choose.
- higher confidence in production schedules may allow running at lower inventory levels.
- long-term planning from sales forecasts (e.g. assist with production capacity planning and production smoothing, supply planning for long lead-time items, and inventory planning, what-if scenarios for strategic and operational planning, testing the impact of changes on business rules, infrastructure investment, overtime or extra shifts).
- reporting on production (e.g. identification of capacity problems, identification of production or supply bottlenecks, high-level overview as well as information on specific orders).
- reduction in labour requirements through labour balancing, a reduction in labour-intensive operations, and a reduction in overall transfers.
- reduction in "free working space" on the tank farm, leading to increased tank utilisation, capacity, and throughput.
- process improvement in the area of work order handling, by reducing paper handling and data duplication.
- provision of centralised applications that are maintained and supported.
- provision of a scalable platform for future extensions.
- straightforward integration with other applications for prediction and optimisation.
- provision of integrated views (carrier, winery, etc.)
- provision of integrated inputs (e.g. for Grower Liaison Officers and Logistics Coordinators).
- provision of optimised capacity planning (i.e. automated "smoothing").
- automatic generation of robust, optimised production schedules that maximise service levels and utilisation, while minimising cost.
- faster feedback to production planners, management, sales, and other interested parties for placing orders or requesting changes.

There are also a number of flow-on benefits, e.g. less time required by planners to produce bottling plans, less chance of human error, identification of potential data problems, ability to handle dynamic changes to the schedule, at the same time minimisation of the impact on existing work orders near their production date. All the above listed benefits are possible due to the *global module* which coordinated optimisation in between the components.

An additional benefit of the design described in this chapter is that the global optimisation relies very little on the design of the systems that make silo based decisions and systems can be “plugged in” or “pulled out” of the global optimisation relatively easily.

To realise all these benefits, the Computational Intelligence method must play a central role in the development of the software. It would be very difficult, for example, to build a linear model of such a whole supply chain, representing all objectives, constraints, and dependencies – indeed, most standard software packages have failed in this complex environment over the last 15 years.

Chapter 8

Multi-silo Supply Chain with Complex Structure

Due to the nature and complexity of some supply chains, evolutionary computation algorithms are not sufficient to produce a good solution for them. This chapter presents a multi-silo supply chain with complex dependencies. A fuzzy-evolutionary algorithm is proposed to address the complexity of this supply chain.

8.1 Introduction

This chapter presents ways to synchronise and reconcile the drivers of multi-echelon supply chain networks as demonstrated on a real-world example of an Australian manufacturer of agricultural chemicals. Although the model presented in this chapter aims to represent a specific supply chain model of a particular manufacturer, the components on which the network is based represent supply chain entities as can be found in many other businesses. We present two approaches which generate optimised production and material procurement plans. The system will facilitate the planning process, which is currently a two week task for a planning team of two down to a matter of hours. The operators will have the opportunity to regenerate plans as soon as the environment changes, thus obtaining a response within a few minutes. The prompt generation of plans allows for strategic planning which could not be achieved by the previous mode of operation. The power of what-if-scenarios can be harnessed to benefit from early structural decisions of the supply chain, such as added production, storage or transportation capacities.

This chapter details two approaches on how to balance the output of producing entities such as plants and factories with storage facilities like tanks, stock piles or silos. The aim is to generate an optimised production plan for each site, including decision making about such aspects as the sourcing of raw material and production rates while honouring storage capacity constraints.

Although both approaches employ a simulation as part of the solution evaluation, the type of simulation differs. In the first approach, an Evolutionary Algorithm tries to find a sequence of events. An event is defined by the date it occurs and an impact it has on the simulation state. The events are stored in a priority queue and executed in chronological order. An event could, for instance, cause a material changeover or a wind down of the production rate of a plant. An actual simulation only occurs at these discrete events (Discrete Event Simulation); in between any two adjacent sample points (events), the system's state is assumed to be linearly changing, which means a tank's level can be inferred at any given time between two sequential events. In this particular implementation, the reduction of the plant's utilisation in order to comply with capacity constraints is performed deterministically by taking the excess production of a storage and propagating it back to its supplying nodes reducing their production proportionally to their share of the excess amount.

The second approach presented in this chapter does not use events to change the state of the system. As opposed to the previous approach, it tries to find the underlying rules that will balance the supply chain's producers and product changeovers. During the optimisation process, an evolutionary algorithm generates a rule base and compares its performance on the simulation run at a predefined interval.

The above two approaches are compared in terms of their overall performance, comprising violation of hard constraints such as storage capacity, total final product yield at the end of the run, and satisfaction of demand.

This chapter is structured as follows. After this introduction, the description of the problem is presented in detail. After a literature review in section 8.3, the approaches used to tackle the stated problem are discussed in section 8.4. Experimental results are given in the following section 8.5 and finally the chapter finishes with a conclusion.

8.2 The Problem

In many industries which base their operations on multi-echelon production systems, procurement planners and factory operators often face the same sort of problems when they create production plans for a short, mid-term planning horizon:

- What is the best supplier of raw materials (in terms of reliability, contractual bindings, availability, costs)?
- How much of it should be sourced in a given period?
- How much finished or intermediate product should be produced?
- When is the best time to schedule maintenance outages?

There are many more questions that can be considered. The bottom line is that the problem is rather complex, and involves trade-offs which cannot be made in isolation, that is, disregarding processes that happen further down the supply chain; indeed all decisions are heavily interdependent. Applying rule-of-thumb reasoning will not lead to an optimal or even to a feasible plan if hard-constraints such as capacity limits are violated.

In order to understand the complexity of the presented problem and the manual interaction involved, let us consider the following example. A factory plan is supposed to be developed that balances the production and storage capacities of the supply chain presented in figure 8.1.

The network is composed of three plants ($P_1 ... P_3$) producing material M_1, M_2 and M_3 (P_1 produces M_1, P_2 M_2 and P_3 M_3) at a rate of 50 units per time unit t and three storage tanks (S_1 and S_3). Assume that all tanks' capacity is 100 units and S_1 and S_3 are already filled to 80% of their maximum capacity. The planner tries to run the plants as hard as possible – that is, it is desired to run them on maximum production capacity. Running P_1 hard within a period of t means it produces 50 units of M_1 which are stored into S_1. Since there is a constant consumption of M_1 by plant P_3 an overflowing ot the tank does not occur. P_2 also produces its material M_2 and conveys it into its tank S_2. P_3 is sourcing its two input products and converting it at a ratio of 1:1 (which means we obtain 50 units at the end of t from P_3). Since storage tank S_3 is already filled by 80 units (assume no consumption at this point), an excess amount of 30 units has been produced at the end of the time period. Since the excess

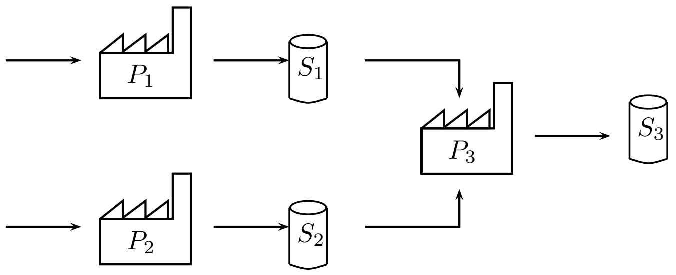
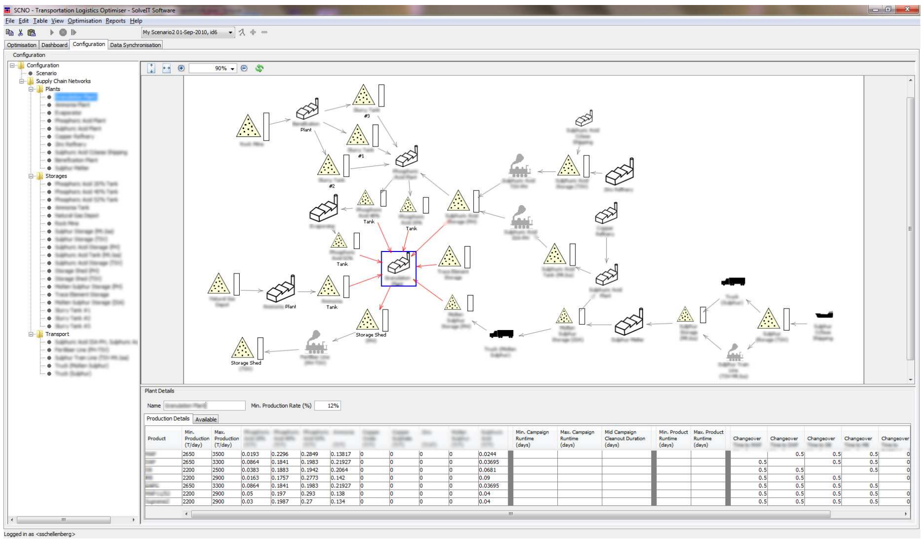


Figure 8.1: Simple supply chain example to demonstrate complexity of balancing the components involved.

amount cannot be stored elsewhere, the production of the plant feeding into S_3 has to be reduced. In this example, a utilisation of 40% would avoid the excess capacity to be created and storage S_3 to containing 100 units at the end of the period. Winding down the production in P_3 however, reduces the demand of M_1 which is produced in P_1. The planner has to go back to the producer of M_1 and also reduce the production rate of this plant, as the material cannot be stored in S_1 (which is also filled to 80% of its capacity). This process of propagating back the plant utilisation has to be done for every storage facility that is exceeding its capacity. In this case, only two plants were affected, but for multi-echelon networks, it is easy to imagine that the amount of work that has to be done once a storage constraints is violated is enormous. The dynamics of the environment in which the plan is going to be implemented, often forces the planners to re-create a plan multiple times (imagine outages of production plants). Figure 8.2 shows the screenshot of the system with all the confidential client information removed. This screenshot shows the number of silos in the supply chain and the complexity of relationships between them.

In essence, the process of creating such a plan can be very labour intensive in a dynamically changing environment. It becomes even more tedious when consideration has to be given to a long term planning horizon, which bears a lot of uncertainty towards its end. The benefit of long term planning, the ability to strategically plan by evaluating what-if-scenarios, becomes virtually impossible when plans are generated manually. For a given industry example, it took no less


SCNO - Transportation Logistics Optimiser - SolveIT Software
File Edit Table View Optimisation Reports Help
My Scenario2 01-Sep-2010, id6
Optimisation Dashboard Configuration Data Synchronisation
Configuration
Scenario
Supply Chain Networks
Plants
Storages
Transport
90%
Plant
Storage Shed
Tank
Logged in as <sschellenberg>

Plant Details

Name [illegible] Min. Production Rate (%) 12%

Production Details | Available

Product	Min. Production (T/day)	Max. Production (T/day)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Min. Campaign Runtime (days)	Max. Campaign Runtime (days)	Mid Campaign Cleanout Duration (days)	Min. Product Runtime (days)	Max. Product Runtime (days)	Changeover [illegible]	Changeover [illegible]	Changeover [illegible]	Changeover [illegible]	Changeover [illegible]
[illegible]	2650	3500	0.0193	0.2296	0.2849	0.13817	0	0	0	0	0.0244							0.5	0.5	0.5	0
[illegible]	2650	3300	0.0864	0.1841	0.1983	0.21927	0	0	0	0	0.03695						0.5		0.5	0.5	0
[illegible]	2200	2500	0.0383	0.1883	0.1942	0.2064	0	0	0	0	0.0681						0.5	0.5		0.5	0
[illegible]	2200	2900	0.0163	0.1757	0.2773	0.142	0	0	0	0	0.09						0.5	0.5	0.5		0
[illegible]	2650	3300	0.0864	0.1841	0.1983	0.21927	0	0	0	0	0.03695						0.5	0.5	0.5	0.5	
[illegible]	2200	2900	0.05	0.197	0.293	0.138	0	0	0	0	0.04						0.5	0.5	0.5	0.5	0
[illegible]	2200	2900	0.03	0.1987	0.27	0.134	0	0	0	0	0.04						0.5	0.5	0.5	0.5	0

Figure 8.2: Diagram, configuration and production details of the supply chain

than two weeks for the planning team to sketch a plan which took into account storage constraints and currently firmed orders. Running a what-if-scenario analysis to determine the impact of say a corroded storage tank which is likely to leak in the next couple of weeks would be almost pointless, as the event is more likely to occur by the time the planner finishes his what-if-evaluation.

The proposed system automates the planning process to the extent that constraints are entered via an intuitive user interface and a plan is delivered within minutes after the optimisation process has been started. This allows for strategic planning as well as a prompt update as soon as new orders are placed, thus impacting on the production schedule.

8.3 Related Work

The optimisation of supply chain networks has been an ongoing research topic for many years. The research work can be categorised into two major streams. Either a supply chain has to be built from scratch with optimal location of production facilities or storage and distribution centres, or an existing supply chain has to be managed in order to make decisions that pertain to factors such as the sourcing of raw materials and the amount of production. A combination of both is also possible in which case an existing supply chain is supposed to be structurally altered (e.g. by adding new sites or negotiating new supply contracts). Structural or management decisions can also be considered with regard to their planning horizon. While structural decisions impact on longer planning horizons, management decisions deal with day-to-day operations. The former decisions are of a strategic and the latter of tactical and operational nature. Depending on the planning horizon for which the supply chain has to be optimised, different categories of questions become important. While the factory manager is most likely be interested in a plan that tells the personnel at which rates the production is supposed to run for the coming weeks (short term, operational planning), a manager is interested in the bigger picture; questions about the impact of a new production site may be more relevant to him than the current factory plan in the plant (long term, strategic planning) [15].

An approach that addresses long and short term planning questions is presented by Truong and Azadivar [169]. In that paper, the authors describe a system that uses a hybrid technique of mixed integer programming, genetic algorithm and discrete event simulation to make the optimal decision about where to produce (internally or externally), production planning, transportation as well as strategic decisions on the location and capacity of facilities. While the genetic algorithm is employed to optimise sourcing policies and qualitative variables, mixed integer programming reduces computational costs by solving quantitative variables. The effectiveness of the obtained supply chain configuration is evaluated by a simulation.

In [170], a discrete-event simulation facilitates the evaluation of supply chain scenarios of a food supplier. The results obtained from the simulations suggest ways to improve the supply chain by changing inventory strategies. As a result, the stock level could be reduced, and this was beneficial to the freshness of the products, also allowing additional products to be introduced as shelf space was freed.

Other researchers concentrate only on isolated parts of the supply chain. Ding et al. [60] describes a hybrid simulation-optimisation approach with the objective of selecting the best supplier of a supplier portfolio (in a strategic way rather than for daily sourcing; the supplier found is used throughout the planning horizon). A genetic algorithm is employed to search for possible configurations of suppliers. As with the previous approach, a discrete-event simulation determines the key performance indicators (KPIs) that form the input for the evaluation function.

Another approach concentrating on parts of the supply chain in isolation is discussed by Xie and Petrovic [177]. They analyse research work done on the problem of stock allocation and conclude that, for most of the strategies given, the rules for cost optimal allocation policies are derived from mathematical models for the considered distribution systems. Their approach defines a new decision-making system for stock allocation that is based on fuzzy IF-THEN rules. Initially, the rule base is generated by domain experts, but they allow for alteration by changing the rule's weights. A simulation on a 2-echelon supply chain (one warehouse and multiple retailers) is run to demonstrate the effectiveness of this approach.

Fuzzy set theory is also applied in Wang and Shu [173]. Unlike Xie and Petrovic [177], Wang and Shu use fuzzy logic to model uncertainty such as demand, processing time and delivery of supplies. The objective of their work is to develop a supply chain model that minimises the inventory costs by meeting demand. A genetic algorithm tries to find the optimal order-up-to levels for all stock-keeping units. Again, this approach only looks at one objective (minimisation of inventory costs) and one method to achieve this objective (reduction of inventory).

The common denominator of the above papers is the application of a simulation (mostly DES) in order to obtain the properties of the supply chain and to evaluate the performance of the optimisation method. This observation is also supported by several surveys (such as [157, 74, 164]). Methods used to obtain inputs for the simulation are mainly genetic algorithms (GA). However, even though GAs seem to dominate, recent publications indicate the appearance of fuzzy logic systems as drivers for the simulation process.

Another aspect that is important to consider when building a solution, that is meant to be reusable, is the building blocks of the supply chain, that is, the model. Many attempts have been made to develop a unified supply chain model and terminology. Some of them (for example [162]) represent common terminology, or others (see [148]) emphasise configurability and propose event-discrete simulation as a means to analyse supply chains. This work, however, places its emphasis on business processes rather than the definition of reusable components as these processes are desirable in the creation of a programming model. Others (see [110, 61, 67]) use special modelling languages to express the complexity of business processes and automatically generate simulation models. Although all of them suggest that we employ simulation to analyse supply chain networks, they require expert knowledge of modelling languages like Rockwell Software's ARENA, one of the prevalent languages for model supply chain networks. The resulting models may be generated in a programming language that is incompatible with the rest of the system. We believe that a generic supply chain model can be developed that supports both flexibility and ease of use when creating the model without the necessity of acquiring expert knowledge of simulation languages. The ideas presented in this chapter are implemented in a framework which is employed to model the supply chain operations of a real-world business.

8.4 The Approach

In this section we describe two approaches that we developed to optimise the supply chain of the given business. Both approaches perform a simulation in order to determine the quality (or fitness) of the solution. The first approach generates a sequence of events and applies these to a discrete event simulation (DES). We call this an "Event-Based Optimisation" approach. A simulation is performed at any event that occurs (next-event time advance, [99]). In contrast, the second approach uses rules to make decisions for factory utilisation and sourcing. The "Rule-Based Optimisation" approach also uses a simulation to evaluate evolved solutions, but advances the time at a fixed interval (fixed-increment time advance [99]). In addition to the supply chain model that is being used, and the common parts of the evolutionary algorithm, this section describes both approaches in detail, and lists their advantages and disadvantages.

8.4.1 The Supply Chain Model

When modelling a real-world system, careful attention has to be paid to the level of abstraction of the model. On the one hand, a high level of detail improves the fidelity of the examined properties of the system, but on the other hand, the simulation process becomes computationally expensive, which prolongs the run-time of the optimisation process. A trade-off has to be made between reflecting as much as possible so as to draw the required conclusions from the system, and the minimisation of details. The model we employ for both of our optimisation approaches is described below.

The supply chain network can be thought of as a directed graph $G = (V, E)$ with $V = \{0, ..., n\}$ as a set of vertices and $E = \{0, ..., m\}$ as the set of edges. Each vertex/node can be of any of the three following types:

- plant
- storage
- switch

Although the type of node determines the different way nodes process material, the three node types have common properties. Each node contains a set of predecessors on which it depends with respect to its source products. These predecessors are defined by the incoming edges of the vertex. Figure 8.3 illustrates a very simple supply chain network. *Plant A* produces two products, stores both of them in separate storages and *Plant B* converts *Product A* into *Product C* which is finally stored in *Storage C*. Note, that in this simplified version, *Plant A* has no predecessors, whereas *Plant B*'s predecessor is *Storage B*. The conversion from a raw material into an intermediate or final product can be thought of as a chemical reaction with arbitrary input and output products.

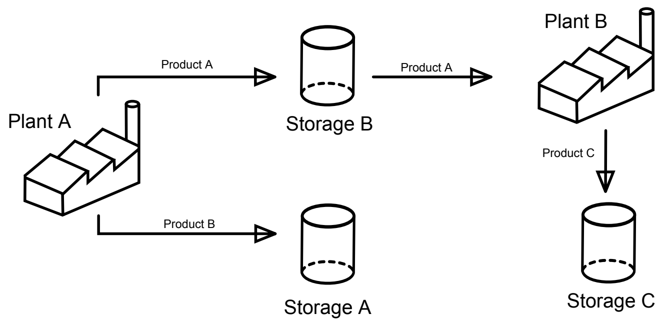


Figure 8.3: Supply Chain model showing material flow between supply chain nodes

A switch, the third kind of supply chain node, is not a physical entity, but it serves to route the material through the supply chain network. Due to the internal design and the paradigm to keep functional units as simple as possible, plants and storages are only allowed to source a material from one predecessor node. In case there is more than one predecessor node which outputs the same product, a switch has to be inserted which controls the material flow (see Figure 8.4). A switch can either implement a local heuristic to determine which successor to deliver to, or it may be controlled by the optimiser which evolves the routing as part of its usual individual reproduction process (the switch becomes part of an individual's genotype). An example of the former may be that *Switch A* is implemented in a way such that it always exhausts the capacity of a tank before it starts filling up another tank (see Figure 8.4, *Switch A*). The local deterministic heuristic incorporated into *Switch B* may reduce the storage's level, evenly draining all tanks at the same time.

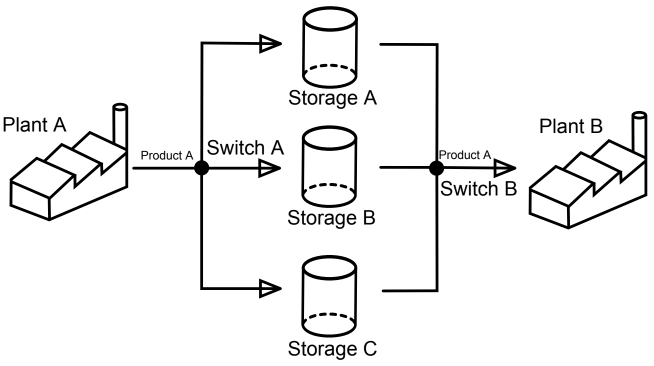


Figure 8.4: Supply Chain illustrating material flow routed by Switches

In addition to nodes having predecessors, nodes also contain output buffers (one per product) in which the products they produce or store and their quantity are temporarily kept. Switches do not contain output buffers, but delegate requests for material to their predecessors depending on their implemented routing logic.

The previously described node-predecessor relationship only works for continuous material flow such as for liquids that are pumped from a producing plant into a storage tank. In many supply chain networks the distribution of material is done via transportation means with limited capacity or infrequent transportation times (trucks, ships, air planes or railway). Therefore, an additional property of the edge between two nodes is a schedule defining the availability of the transportation means and its capacity. Only when a means of transportation available at the plant or storage, can it empty the node's buffers and store the material temporarily in its own buffer. Plants either have to shut down or store their products in adjacent tanks at times at which the transportation means is not available.

Given the supply chain network of the particular business, a simulation process can be performed that is mostly similar for both optimisation approaches. The only difference is in the way in which the decisions that change the state of the supply chain network are inferred.

Before the simulation is started, the supply chain nodes have to be initialised and then a list of the sample points (event queue) will be created. During the initialisation process of the nodes, the buffers of some storage entities are filled to simulate an opening stock. The next step is to determine the points of the planning horizon at which the system is sampled. These points are the set of all events that occur during the planning horizon, such as

- change of availability of a plant (plants may run at a reduced run rate due to partial maintenance or outages).
- product changeovers.
- arrival/departure of transportation means travelling between two nodes.

It is of paramount importance to the validity of the simulation to add all events that cause a discontinuous change of the supply chain network's state to this event queue. Otherwise a change that is non-linear would prevent any inference of nodes' properties in between two adjacent sample points. If desired, additional sample points can be added to verify the simulation's result.

Once the event queue is filled, the nodes are sorted by precedence. Nodes without predecessors occur first, whereas terminal nodes that base their buffer's material and quantity on the result of all previous components' production are located last in this list. The next step is to iterate through the list of predecessors (beginning with the least dependant node) and process the node. Processing a node breaks down into three steps:

1. pull resources (raw material, intermediate material, final product, etc) from predecessor.
2. apply conversion rule.
3. push converted material into output buffer(s).

The first step is straight-forward: since each nodes knows about its predecessors and the material it requires, it pulls the maximum amount of this material from each of its predecessor nodes. Switches forward the call to their pull method the appropriate predecessor. The pull step basically boils down to copying the content of the predecessor's output buffers and passing it on to the conversion step.

In the conversion step, the actual business logic for each plant is implemented. If we consider a chemical formula like $1A + 2B \rightarrow 2C + 1D$ with hypothetical elements A, B, C and D, the factory that processes this formula would source all the material it could get for Product A and B and determine the quantities of resulting Product C and D (taking into account the ratio of A:B=1:2). The amount of input material, which has actually been used during the conversion process, is reduced from the predecessor's output buffers to account only for the actual consumption and to determine overproduction. While *switches* do not implement a conversion routine, they pass on the incoming materials to their output buffers.

Finally in the last step, the produced material is stored in its output buffers to be available for the successor node's processing procedure. At the end of the sample cycle, that is once all nodes have been processed, the simulator captures the state of the system by storing the buffers which contain the quantity/products tuple for each node. The final buffer capacity allows for determining the plants' utilisation or the storages' remaining capacity.

The simulation terminates once the planning horizon's end date is reached.

8.4.2 Features of the Optimiser

The meta-heuristic used to optimise the supply chain network is a steady state Evolutionary Algorithm (EA); only one individual alteration occurs at each generational step. Evolutionary algorithms are well understood, are reliable, and they can be customised to solve a variety of problems of different domains. They have been applied to other problems of a similar level of difficulty and performed satisfactorily. This section recapitulates the general working principle of the EA and elaborates on the specific implementation that contributes to the success of the search process. A description of the operation of the EA follows.

The population contains a number of individuals (approx. 100 individuals) which encode a solution in their genotype. After initialising and evaluating the individuals of the population, the recombination process is started. As part of this process, an operator from the set of available operators is chosen to alter the individual. These operators are application specific as their operation strongly depends on the encoding chosen as well as the particular business problem they try to solve. Some operators may have the ability to consider certain business constraints to operate in a feasible search space which reduces the search space and leads to faster convergence. The specific operators used for both optimisation approaches are explained in the following sections.

After the operator changed the individual, the new individual is immediately evaluated. A self-tuning procedure compares the fitness value of the individual before and after the operation, and adjusts operator weights according to whether the new individual yields a better solution or not. The weight is taken into account at the previous step of selecting the operator. In case an operator performed poorly, it is less likely to be selected for evolving the next individuals (similar to roulette-wheel selection). After a defined number of generations, the weights are reset, so dominating operators have to again proove their performance. This is particularly important if the optimiser prematurely converges and the optimisation process gets stuck at a local optimum. The previously preferred operator would not contribute to improving the solution's quality which means other (potentially more explorative) operators come into play.

The evaluation procedure uses application specific measures to determine the fitness of the individual. For this particular optimisation problem, both optimisation approaches use an identical fitness evaluation function. Part of the evaluation is the simulation. Once finished, the system state is evaluated. Objectives of the optimisation process for supply chains include but are not limited to :

- maximisation of product yield.
- minimisation of product changeovers.
- adherence to a specified product ratio for the remaining planning period.
- satisfaction of demand.
- minimising transportation, inventory, backlogging costs.

A comprehensive list of measures of supply chains can be found at [75].

The optimisation process is terminated once a pre-defined number of generations is reached or if the search stagnates over a certain period of generations. The best individual found throughout the optimisation process is returned (keep-the-best strategy [118]).

Both techniques used in this chapter leverage off the same optimisation engine described in this section. Different business rules are encoded in the individual and the operators which make the EA reusable for other optimisation problems.

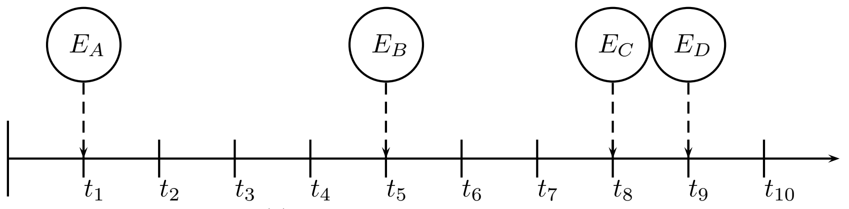
8.4.3 Event-Based Optimisation

The event-based optimisation approach tries to determine a sequence of events that, when applied to the simulation, result in an optimal solution. Events are defined by the date they occur and the action they perform, i.e. the state change they cause to the system. An example could be a changeover event in a factory which causes a factory to produce a different product, a change in the factory's utilisation, a change of the availability of the factory or a factory specific event such as a cleanout of storage tanks. The event-based approach has a very shallow hierarchy of representations. From the event sequence (which can be understood as the genotype) a conversion into the final solution is made by means of the simulation.

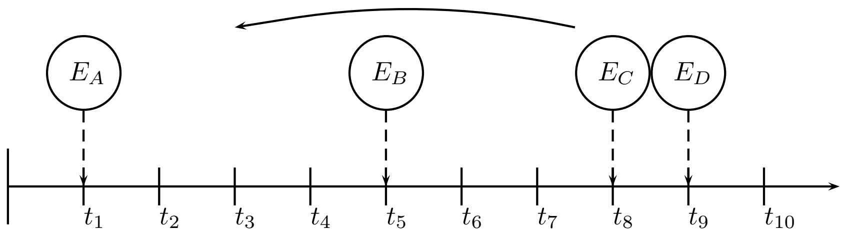
At the start of the optimisation run, each individual is initialised with a random sequence of events. The operators alter the event queue in different ways. They change the type of event (which changes the action they perform) or the date of their occurrence as illustrated in figure 8.5. Some of them insert a delay at a specific time which causes all subsequent events to be delayed. A cross-over operator randomly determines an event of the event queue of two individuals and swaps all of the following events with the other individual, similar to the classical cross-over operation for genetic algorithms.

Part of the evaluation process is to transform the genotype of the individual (the representation of the solution to which we apply the operators) into a representation that can be evaluated (the other part is to aggregate the fitness value components). The transformation is done by running the simulation according to the encoded sequence of events. At each of the event dates, a simulation is executed which samples the state of the system (fill level of storage tanks, excess level, utilisation, amount of raw material sourced). After the event date, the system state will change non-linearly (for instance, a different product is produced; a plant shuts down its operation). Once the first simulation pass is completed a repair function is triggered. Before the repair, the produced products of each plant are consumed by their subsequent storages, disregarding the current available storage capacity.

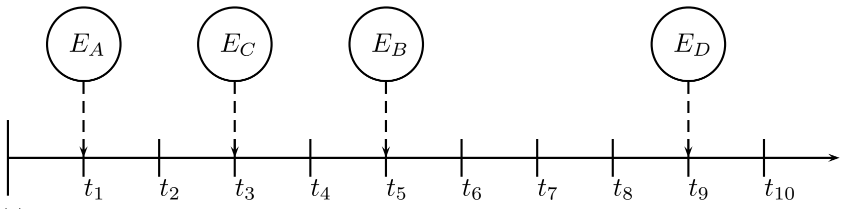
The repair function deterministically winds down the producers (the plants) in order to balance the production with the available storage capacity. This process starts at the last storage in the supply chain, i.e. the one that relies on all of the previous nodes, and works back to the first node in the supply chain (i.e. the one without any predecessor nodes). The excess amount of an overflowing storage is proportionally reduced from the previous producers in order to determine the appropriate production rate (planned utilisation). Each time a plant's utilisation is adjusted, a partial simulation has to be performed again as the minimised production has implications for other supply chain nodes



(a) Event queue before mutation



(b) Mutation operator in action changing date of E_C to t_3



(c) Event queue after mutation. E_C now alters the state at t_3 according to its encoded type.

Figure 8.5: Timeline of the simulated horizon illustrating the impact of the date mutation operator on the event queue (dashed lines indicate the time events occur, i.e. when their they cause a change of the state of the simulated system).

downstream in the supply chain network. The advantage of this deterministic method to determine the utilisation is that these kind of events do not fall into the search process and hence decrease the search space.

On the other hand, this way of adjusting the plant's utilisation is rather expensive as the simulation is rerun every time the utilisation is changed. An optimisation process not honouring the storage constraints, that is, with a disabled repair function was able to evaluate about 1500 individuals per minute, whereas a normal optimisation (repair enabled) could only evaluate a maximum of 150 to 200 individuals on the same machine and the same supply chain network (this comparison of course is not exact as violated storage constraints result in more raw material being available and this skews the production plan and the obtained product yield).

8.4.4 Rule-Based Optimisation

The approach proposed in this section is based on a combination of fuzzy logic and evolutionary algorithms. Rather than directly evolving decisions made during the simulation of the supply chain (in form of events), rules are generated which drive the decision making process.

Since the main focus of this thesis is the application of evolutionary algorithms, a very short introduction to Fuzzy Logic is given in this section. Thereafter, the application of fuzzy logic to the subject of this chapter, optimising supply chains, is demonstrated and the particular implementation is discussed by providing examples of encoding and decoding of the individual's genotype.

Fuzzy Logic

Fuzzy logic is based on the fuzzy set theory in which values are expressed as a degree of membership rather than as crisp inputs like discrete measurements. If we were to classify the age of a person having two categories (young and old), classical boolean logic would either map a given age to young or old. Let us say the cut-off point separating old from young is exactly forty years of age. In boolean logic, every person below 40 years would be young, whereas everyone greater or equal to forty is classified as old. Instead, in fuzzy logic an age can belong to multiple categories (or degrees of membership). The degree of membership is denoted by a number between 0 and 1. Given the two membership functions young and old, a person aged thirty years would be young to the degree of 0.8 and old to the degree of 0.3. A fuzzy term such as "age is young" is called a linguistic term ("age" is a linguistic variable and "young" a linguistic value). Those terms can be combined by fuzzy-and or fuzzy-or (or other fuzzy operators, or T-Norms [76]) and further extended to IF-THEN rules of the form "IF age IS old AND health is bad THEN health-insurance-premium IS high" (IF part is called *"antecedent"* and THEN part is named *"consequent"*). Many of these rules can make up a rule base.

The type of *fuzzy logic system* (FLS) used for this approach is called Mamdani-type FSL [112]. The fuzzy inference process can be decomposed into three major steps. First of all, the crisp input data obtained by measurements is transformed into fuzzy sets (fuzzification step) by determining the degree of membership of the fuzzy terms specified in the fuzzy rule that is applied. The next steps combine the degrees of the fuzzy sets by the given fuzzy-operators and summarise each term into a rule weight. This rule weight is used to determine the output of the rule by limiting the shape of the output function. The last steps, the defuzzification, combine all reshaped output functions and applies a defuzzification function in order to obtain a crisp output value. This function is usually centre of mass or a weighted average function.

Application of Fuzzy Logic to Supply Chain Optimisation

Translated to our actual problem of generating decisions to drive a supply chain simulation, the fuzzy rules determine in their consequent part when to schedule product changeovers (or which products are produced at which time) or the utilisation of the factories (utilisation of the facility). The advantage of using fuzzy logic to encode the rules driving supply chain decisions is that these natural language rules facilitate diagnostics and audit features of the system. A planner controlling the system can deduct the reasoning of the system by analysing the rule bases; this is more intuitive than the previous event-based approach.

Another downside of the event-based approach is also that the EA has to find the correct event for each time, although the conditions may be similar to a previous point in the plan at which the correct decision was made. Say a product changeover has to be triggered every time a storage tank is about to overflow, as the product is consumed from the overflowing tank as soon as the changed over product is produced. This changeover may not be triggered at another point in time as the EA has not yet generated such an event at this time. It may or may not randomly generate such an event at the particular time. The rule-based system however would have developed a rule that triggers a changeover upon reaching the maximal storage capacity of the tank, which means every time the condition holds (storage tank is high), the changeover is triggered. The application of rules makes the generated plan more predictive and coherent. In addition, the underlying logic can be investigated and manually fine tuned.

The individuals evolved in the evolutionary process encode the rule base. We decided to generate the rules by the EA as they may differ for different settings and different products; evolving the rule base by the EA allows adaptation to the current constraints and environment. There are many other means to combine evolutionary algorithms with fuzzy logic. They differ in the part of the fuzzy inference system that is subject to the optimisation. Genetic tuning, for instance, changes the database (shape or number membership functions, linguistic terms) of the fuzzy inference system. Other methods evolve the knowledge base or generate new knowledge base components. An elaborate taxonomy and survey of methods to combine genetic algorithms and fuzzy logic systems (GFS) can be found [78].

Consider a fuzzy rule of the structure (consider this example for explanatory purposes: `IF Level_Of_Tank IS high THEN Utilisation_Of_Plant is low`) consists of an antecedent part (`IF ...`) and a consequent part (`THEN ...`). The antecedent can contain multiple linguistic terms (`Level_Of_Tank IS high`) which can be combined by different operators (T-Norm). For our purposes, Fuzzy-And and Fuzzy-Or were sufficient. A linguistic term has two parts, the linguistic variable (`Level_Of_Tank`) and the linguistic value (`high`). The number of linguistic values can be arbitrary, but in order to reduce complexity, we opted for 5 linguistic values and triangle membership functions (see figure 8.7).

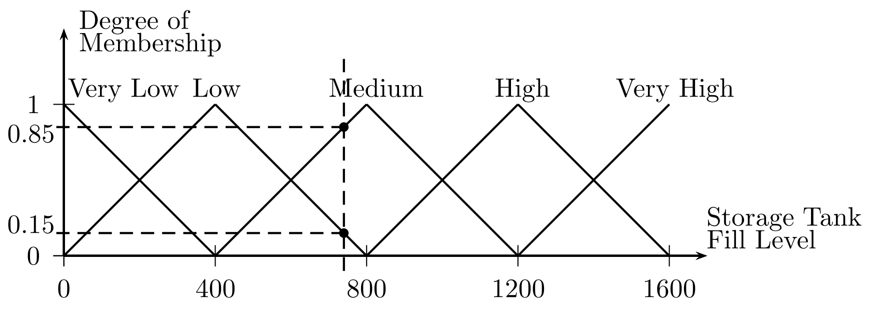


Figure 8.6: Overlay of 5 membership functions denoting the fill level of a tank. At a level of 740l, the tank level is a member of the "low" function by 0.15 and a member of the "medium" function by 0.85. For each storage, the parameters for each membership function are different as the maximum capacity may differ (and hence "very high" would relate to a different maximal level).

The possible linguistic variables are the level of each storage tank, the currently produced product of a plant and previous utilisation. A set of rules forms a rule base. For the proposed system, a number of rule bases is created, one rule base per possible consequent part. Since the linguistic variable for all the rules of a rule base are the same (that is, all rules in one rule base pertain to the utilisation of a specific plant), the only additional information that needs to be encoded in the chromosome is the linguistic value of the consequent part. Figure 8.6 displays the integer number vectors encoding each rule. A rule base contains up to n rules; the first index represents whether the rule is enabled, the second specifies the fuzzy operation which combines the linguistic terms (Fuzzy-and or Fuzzy-or) and the following pairs of integer values contain a pointer to a look-up-table of linguistic variables and linguistic values. The last cell holds the linguistic value for the rule base's consequent part.

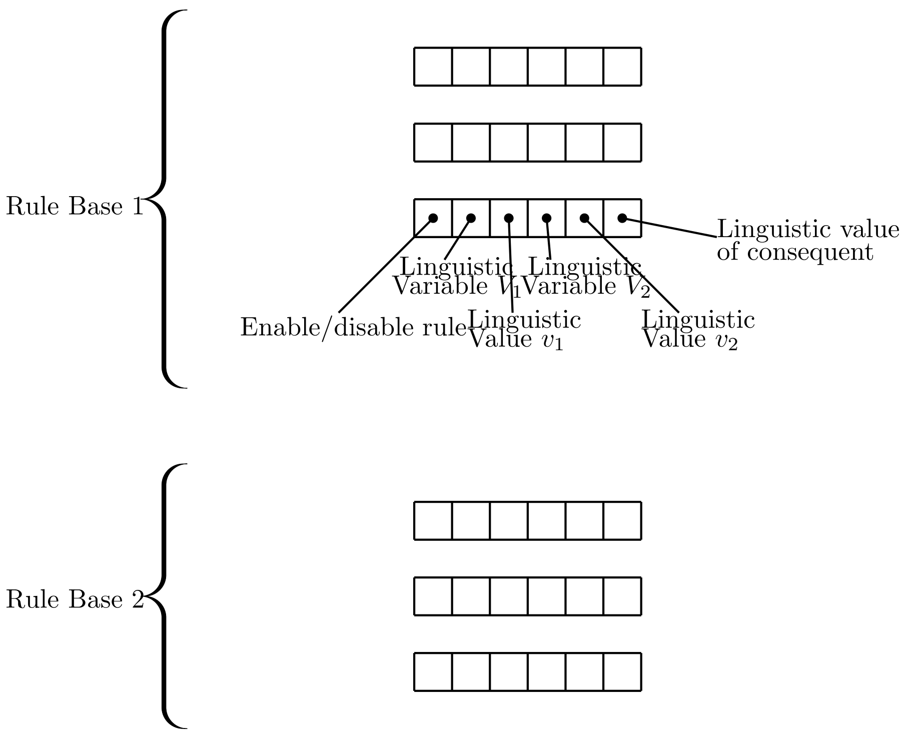


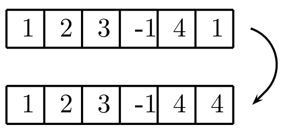
Figure 8.7: Encoding the rule bases. Elements in squares are of type integer number and stand either for booleans or indices in look-up-tables that hold linguistic variables and values.

Generation of Rules

As in the previous section, the same evolutionary algorithm with its self-calibrating capabilities is employed. The only difference is the specifics of the evaluation – that is how the simulation is carried out, and the set of operators used to evolve the individuals (as the encoding of the individuals). The evaluation function itself is exactly the same. It takes into account the maximal production yield, a penalty is deducted in case storage constraints are violated, and also a penalty is applied in case of a delay in providing enough final product to satisfy firmed orders.

Unlike the event-based optimisation, the rule-based approach samples the system at pre-defined intervals. At these sample points, all properties of the simulated system are evaluated and actions are derived from the current state. These actions are triggered by the rules described above. A rule may pertain to the fill level of a storage shed and cause a reduction of the feeding plant upon reaching a "high" fill level.

Different operators modify the genotype of the individual at each recombination step. A mutation operator randomly changes bits of the genes by honouring the feasible maximal possible integer number value at the position in the chromosome. The meaning of such a mutated chromosome changes in the decoding step which leads to different rules, and thus different decisions in the simulation step (see figure 8.8).



(a) Mutation of chromosome (before and after mutation)

```
Before:  IF Level(TankC) IS
high
THEN Util.(PlantA) IS low

After :  IF Level(TankC)
IS high
THEN Util.(PlantA) IS very
high
```

Index	Ling. Variable	Ling. Term
0	Level(TankA)	very low
1	Level(TankB)	low
2	Level(TankC)	medium full
3	Level(TankD)	high
4	Level(TankE)	very high
5	Level(TankF)	
7		
...		

(b) Phenotype decoding

Figure 8.8: Impact of mutation on decoded phenotype ("-1" means the term is not considered in the rule)

Another typical operator for genetic algorithms was also adapted and implemented. Two flavours of the single point crossover are employed. One of them cuts the individual rules of two parent individuals into two pieces (at a random crossover point) and swaps its right-hand part with the other parent. The other crossover operator swaps entire rules at once, by determining again a crossover point and replacing one part with the parent's rule set.

8.5 Experimental Results

We have tested the two approaches on two identical supply chain networks. The aim was to maximise production while honouring storage constraints. The EA was configured to terminate its search after a maximum of 5000 generations, or prematurely if the search would not yield any improvement within 1000 generations. By virtue of the system, the event-based approach runs a simulation whenever an event occurs. The rule-based approach was configured to sample the system at a fixed interval of one day and change the system state by applying its rules.

Both algorithms were able to generate feasible solutions without violating constraints. The quality of the averaged solutions of each approach, however, differed significantly. While the event-based approach managed to fill up the final product storage shed to $172,000$ tonnes (see figure 8.9), the rule-based approach exceeded this value by 37% ($233,000$ tonnes). In addition, the search procedure of the latter terminated much earlier (on average at about 1300 generations) while the event-based algorithm used up the full span of available generational cycles. Another interesting observation is that the time it takes to evaluate an individual is much less for the rule based approach (approx. 800 individuals per minute vs. 150 i/min). This and the premature termination caused the rule-based approach to find a (even better) solution after a few minutes of run-time only.

As already stated in the introduction to this chapter, the system presented is applied to a real-world problem and as such it is difficult to compare it to synthetic problems as they are usually used as a baseline in academia. The only plausible baseline can be obtained by comparing the factory planner's schedule and the expected product yield with the schedule generated by the system. Preliminary test results indicate a high degree of similarity between human and system generated factory schedule with respect to the length of product runs (or in other words, with the date of scheduled product changeovers) and accumulated product yield at the end of the planning horizon. Taking only these few measures into account, one can conclude that the model adequately represents the supply chain. The time it takes to planning personnel to generate a yearly plan is substantial. It takes at least 14 days of planning for a team of two to come up with a plan that does not violate any capacity constraints. A what-if-scenario analysis becomes virtually impossible. Unless we deal with strategic what-if-scenarios, the result of such a scenario would become obsolete

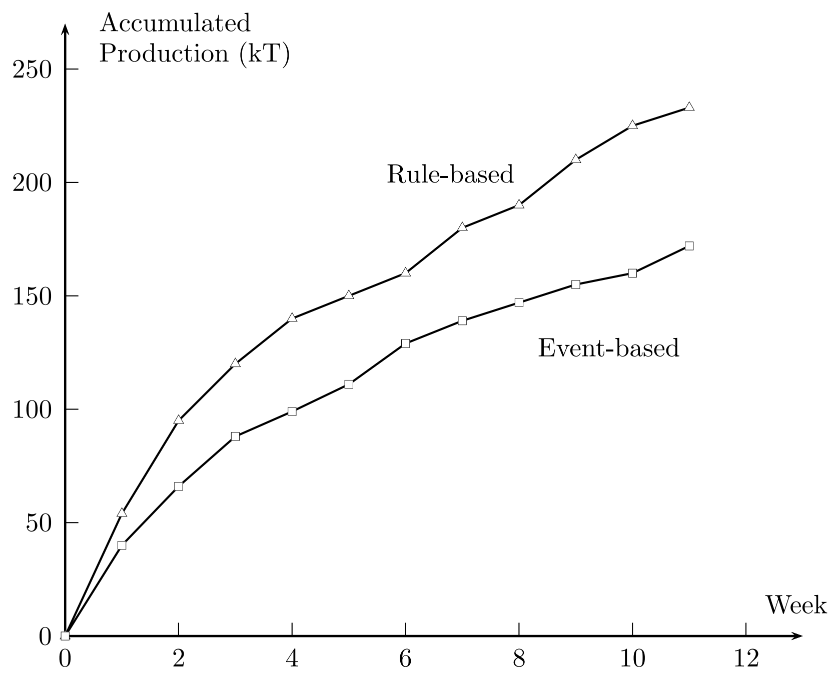


Figure 8.9: Total product yield of event-based and rule-based approach

by the time it is obtained. Using the proposed system, an optimal plan for an entire year could be create in less than 10 minutes for the event-based approach, and in about 2 minutes for the rule-based approach respectively (carried out on a standard Dual Core 1.6GHz computer optimising a 5-echelon supply chain).

Another observation worth mentioning is that, in some instances, the optimiser made decisions that were, by a human operator, hard to justify. These decisions dealt with production trade-offs that were done early in the planning period in order to benefit from an event that happened later, with the aim to increase the overall production. This may be a valid decision with respect to the evaluation function (higher production means a fitter individual), but, since we deal with a real-world environment, one has to consider the uncertainties that the future may bring (especially for long plans). The future benefit the optimiser was speculating on may in reality never materialise; this would result in an overall inferior plan (compared to one that would not have made the trade-off decision in the first place). As a result, we introduced a staged optimisation that partitions the planning horizon in periods which are optimised in isolation. Once a period has been optimised, the resulting stock is carried over into the next period, thus serving as an opening stock.

We ran a trial to determine the impact of partitioning the optimisation. To obtain a normalised result we limited each period's run-time according to its share on the overall planning horizon. Optimising the whole period at once took about 8 minutes. For the test case with two periods, a maximum optimisation time of 4 minutes was allocated for each of the periods. Essentially, this means we allocated the processing time evenly, as opposed to allowing the EA to exhaust the maximum of 2000 generations for each period (in which case the result would be skewed as the search space is only half the size, but the same amount of computational resources is applied to optimise). The simulation was run four times over the whole period and the planning horizon split into 2, 4 and 8 periods. Figure 8.10 confirms our initial assumption. The total yields diminished by about 25%, after comparing an average optimisation run over the entire period to the averaged result obtained by subsequent optimisation over 8 periods. This implies that the search space for an entire year is too vast for the EA to converge to an optimal solution. It seems to be better to abandon the advantages of a "global optimisation" of the entire horizon by optimising only reduced areas of the search space so as to eventually combining the solution.

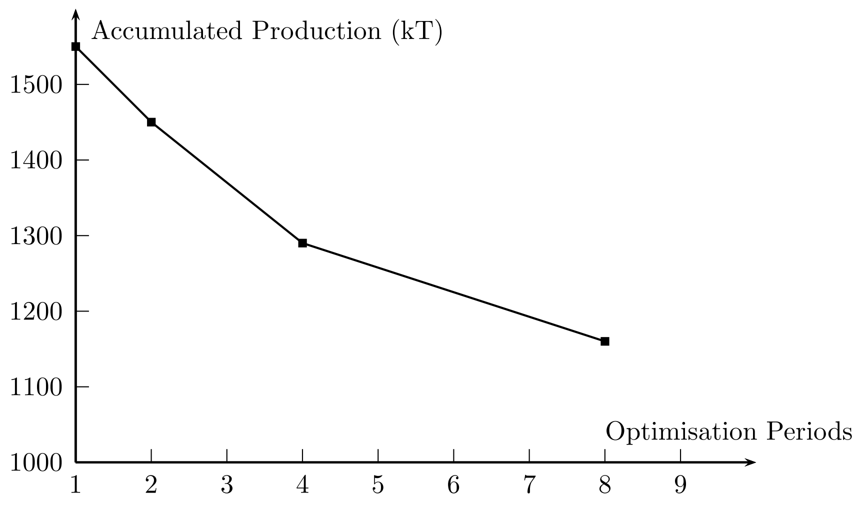


Figure 8.10: Decline of total yield when optimising periods in isolation

8.6 Summary

In this chapter, we presented two approaches to simulating and optimising multi-echelon supply chain networks. Production rates and optimal dates for product changeovers are determined by a hybrid discrete-event simulation, and a fuzzy-rule based decision making system, which are both incorporated into an Evolutionary Algorithm that was tailored to the problem domain of supply chain optimisation. The results obtained meet, and in some instances exceed manually created plans. The real improvement, however, is the improvement of speed at which the system is able to churn out high level production plans. While manual creation of these plans took about 10 to 14 days, the implemented system is able to produce a plan within minutes, allowing what-if-analysis and comparison of different scenarios.

In this chapter we looked at the application of an evolutionary algorithm to the problem of optimising the key decision points in the supply chain network of a major real-world agricultural chemicals company. Modelling the highly complex nature of that company's operations was the first part of the challenge of successfully accomplishing this endeavour. A balanced mix of discrete and continuous event simulation had to be used. Furthermore, the model was designed in such a way as to be amenable to use within the framework of an evolutionary algorithm.

Of key importance was the development of the ability to represent the decision points in the supply chain network simulation as entities that could be manipulated by evolutionary operators. This was achieved in different ways. For the event-based approach, this was done firstly by allowing the timing of decision events to be determined by values within the candidate individual representation, and also the types of those decision events. Thus, for example, a candidate individual could specify that the operation of "change from product A to product B" could be scheduled to happen in a particular part of the network on a given date and time. Secondly, the processing logic of "switching" nodes of the network could be manipulated by evolutionary operators. For example, the production of a particular chemical could employ materials from a variety of sources, with the existing complex business contractual rules associated with each source. In some cases, it might be easy to determine a set of rules to spell out the correct routing logic to use. In other situations, it may not be obvious, and thus it would be preferable to let the individual represent the decision making logic as part of the encoding, and then evaluate the performance of evolved logic as a component of the overall fitness evaluation process.

The rule-based approach abandons the idea of having isolated nodes that handle the sourcing. In contrast to the event-based system, a decision is made at each sample point based on the current state of the system. The resulting plan produces consistent and traceable decisions. Despite the fact that this approach adds the process of balancing supply and storage capacities to the search space (as opposed to running a deterministic repair function), it is able to find a solution much faster while generating even better solutions. The reasons for this observation may be that the search space of the event-based approach, that is all combinations of type and date of events, seems to be larger than the permutation of rules used in the rule-based approach. In addition, many infeasible solutions seem to be generated, which demands for repair by running a costly re-evaluation. Using rules to make decisions constricts the search space. However, only those potential solutions that are less viable seem to be neglected, as indicated by the better solutions obtained. The reason is less surprising when considering the nature of the problem. Changeovers, reduced plant production and sourcing of material are based on rules. A changeover occurs once a tank is reaching its maximum capacity, the production is reduced upon downstream bottlenecks and a sourcing decision depends on minimal procurement costs. Trying to find these decisions without understanding their natural cause is more expensive and results in many lost opportunities compared to a supply chain that is driven by a condition-decision scheme such as we presented to it.

The results presented in this chapter also illustrate the trade-off that is frequently accepted by business managers in practice. By running the software in a global mode over a large time-frame, a relatively high optimal result could be achieved, but the validity of such a result could be called into question by human managers who would rightly point out that the further out into the future assumptions are made about supply chain conditions, the less reliable those assumptions would be. Hence we chose to apply the simulation/evolution algorithm over the whole time-frame in phases, starting with a short term phase of a few months, and then looking further into the future. This gave our approach the benefit of seeking higher levels of optimisation in the short term, wherein knowledge of conditions is quite firm, and then freezing those results and progressively expanding the scope of inclusion to seek out optimisation further into the future.

As constructed, the software application arising from the rule-driven simulation model and the evolutionary algorithm presented in this chapter, was able to provide invaluable insight and speculative modelling ("what-if") capabilities to managers of the client company, allowing them to find ways to optimise their supply chain network, and of course maximise production. This is the litmus test of the value of this application, and it is a rewarding application of the power of evolutionary computation to a real-world business.

In addition to our work presented in this chapter, several promising ideas may improve the results obtained. As an example, we aim to expand the linguistic terms that can be taken into account by adding future availability of plants as well as past utilisation to smoothen the plant's overall utilisation.

A promising method to improve the evolution of the rule bases is to employ a co-evolutionary approach in which rule bases would be developed in isolation. An instance of an EA would only concentrate on its designated rule base (i.e. one EA instance could be employed per plant to evolve rules for its utilisation, one EA instance for sourcing) passing on the best of its rule bases to form the overall solution rule bases.

This chapter presents only the current state of our endeavour to find a common model and methodology for optimising supply chain networks which cater for many business cases and industries. We expect the current method to be fine tuned and extended to allow maximal generalisation and applicability.

Chapter 9

Concluding Remarks and Future Work

In this thesis we studied complex real-world problems involving multiple interconnected components usually set in a time-changing environment. Many problems from classical operations research that were studied by the EC community (such as scheduling, vehicle routing, travelling salesman, inventory management, stock cutting, and packing among other problems) represent just isolated components of today's many real-world problems. Decisions in each isolated component may not lead to the optimal overall decisions and require a "holistic treatment" [3]. In this thesis we selected the supply chain domain to study problems that consist of several interacting components. We investigated different types of interactions between the components, created a framework based on cooperative coevolution and showed that cooperation between the components during the optimisation process lead to better quality solutions.

9.1 Main contributions

Initially, we observed issues of time varying constraints within a single silo, based on a case study from wine bottling in a mass-production environment. An important consideration that was emphasised was that complexities arising due to time-variability have to be dealt with adequately in order to ensure that the solution produced is usable in an actual business environment. We explored different issues, such as manual assignments and the ability to create schedules that seamlessly mesh with the existing ones, arising in the production environment. Methods for dealing with time-varying constraints as well as detailed description of the algorithm were presented in Chapter 3.

Then we moved to a basic two-silo experiment (see Chapter 4) where each silo was modelled based on the classical problem from operations research. We described details of the model and employed two techniques to it: sequential optimisation of silos and cooperative coevolutionary approach with silo decisions evolving based on the overall quality of solutions. An important part of both algorithms is that they "reuse" algorithms that address problems in local silos as building blocks of the algorithm. They create a global optimisation layer that manages the cooperation and execution of the silo algorithms. Several variations of cooperative algorithms were investigated; the compared results have shown that the cooperation between silos led to better quality solutions. An additional experiment was conducted to determine the optimal parameter set for population sizes.

We extended the two-silo experiment to investigate different types of flows between multiple components of a two-echelon supply chain: one-to-many and many-to-one (see Chapter 5). As in the previous two-silo experiment, the existing silo algorithms were combined under the management of the global module, based on sequential and cooperative coevolutionary approaches. The same algorithms for JSSP and VRP were used as building blocks for the local algorithms in each of the silos, thus proving the extensibility of the framework to tackle even more complex supply chain models. Although additional enhancements to the algorithm were incorporated (for example, a special operator that permits avoidance of convergence to the Nash equilibrium), as with the two-silo model, the cooperative coevolutionary approach yielded better results.

One of the reasons for the exploration of the experimental supply chains, was to define some common material flow types between the silos, and to explore whether the algorithms used for these supply chains can be carried to the real-world problems. These common patterns explored from these problems were applied (with some customisations) to the real-world problems discussed in the previous chapters.

We described a multi-silo two-echelon supply chain based on an existing Australian company that produces sheet steel. Despite the multiple silos in each echelon, the components have one-to-one flow: one silo of the first echelon connected to only one silo of the second echelon. The problem is described in detail and an algorithm based on variation of the on-the-fly partner generation

(FLY) technique that we investigated (see Chapter 4) has been used. The comparison of the solution generated by the proposed algorithm with the current production solution created by the company's planning expert team resulted in a considerable profit increase.

The supply chain presented next comes from the mining industry (Chapter 6). Many mining companies struggle with the problem of the planning and scheduling of equipment in iron ore mines, and are subject to numerous time-varying constraints and conflicting objectives. The supply chain is modelled as multi-silo two-echelon with inter- and intra-echelon dependencies. The interactions within one echelon (ore processing echelon) occur due to the ability to move ore from one stockpile to another or from stockpiles to the crusher. A detailed mathematical model of the problem was described, followed by the three stage algorithm that addressed the problem. We outlined main functionality, screens, an analysis and the what-if tools implemented in the system. The analysis of the produced solution becomes much easier with the help of reports that show the KPIs of the produced decisions.

A more complex five-echelon wine supply chain is presented in Chapter 7. The described supply chain consisted of the following silos: maturity models, vintage intake planning, crushing, tank farm and bottling. Each of the five silos were described in detail in this chapter, as well as a brief description of the algorithms used to address the optimisation in each silo. The bottling silo of this supply chain has been described in detail in Chapter 3.

Although each of the silo problems of the wine supply chain present a large number of difficulties, the main goal was to combine all the existing local algorithms under the framework of the global module that provides optimisation across the whole supply chain. The global module is based on cooperative coevolution. Since the systems and algorithms at local silos are very complex, the communication architecture was revised. A special communication protocol was designed which allowed not only the bridging of separate systems but also produced advantages of distributed computing (each system can reside on a separate machine on the local network). The solution produced by this system has been evaluated by the domain expert as a good quality set of decisions.

When dealing with the supply chains with more complex dependencies often evolutionary algorithms are not sufficient. The discussion of multi-silo supply chains in Chapter 8 led us to a real-world problem in the business of the production of agricultural chemicals. A hybrid algorithm combining elements from evolutionary computation and fuzzy logic was employed to address all the complexities of the described supply chain. In addition to the algorithms, the architectural decisions of a discrete event simulator are discussed.

We emphasised the importance of the software design decisions made during the implementation of the algorithms and frameworks, in particular the supply chain simulation framework used for the evaluation of the quality of solutions and the design of the cooperative coevolutionary global module and its communication protocol (see Chapter 7). The software architecture of the implemented algorithms or their components often gets very little attention in papers describing optimisation problems. Part of the reason is because the researchers are mainly interested in the algorithm description and the results. However, the the correct architectural decisions involved in the implementation of algorithms leads to less error in experiments, and provides reusability which lets researchers do many more experiments in less time.

9.2 Future work

The research that has been conducted in this thesis left a few issues which are worth exploring. We mentioned that usually supply chains have several, often conflicting, objectives. Cooperative coevolutionary approaches described in this thesis can be extended to tackle the multi-objective optimisation of supply chains. A very difficult issue of cooperative coevolution is its convergence to the Nash equilibrium [141], especially since there is a high level of dependency between the components. In this thesis we used a special shuffling operator which temporarily evolves part of the population using a local fitness function; however, we believe it might be worthwhile to study other methods of shuffling populations in order to break out of the Nash Equilibrium.

Many of the algorithms described in this thesis use cooperative coevolutionary approaches to address the problem of global optimisation. The global optimisation module runs several optimisations simultaneously and coordinates the evaluation. A very promising idea for the global optimisation is the use of approaches based on *hierarchical optimisation*. Very often it is almost impossible to solve a problem at the required level of detail with the optimise-all-at-once approach. Instead, it is possible to zoom out of the problem to look at it at a very high level without paying much attention to the details. In this way the search space is decreased at the price of the precision or loss of some potentially good solutions. However, it is possible to look at a more detailed model of the problem using the created high level solution as a guide. With this hierarchical structure it is possible to do a staged optimisation for each level of the hierarchy in a top-down manner.

This approach is widely used in mine planning where a life of mine plan is first created, then, based on it, planners create a long term plan. After that a medium term, a short term and finally operational plans are created based on the predecessor plan. Another example of this approach can be seen in the way governments plan their budget. At a very high level they allocate a certain percentage to each industry (such as military, education and healthcare) and then people in charge of each industry make further decisions on how to optimally spend it.

Another enhancement to the optimal decision support in supply chains can be the addition of the prediction component. One of the drivers of a supply chain is customer demand. The ability to forecast such demand based on historical data can lead to more optimal scheduling and planning decisions. In addition to the pull factor (customer demand), a supply chain can also be affected by the push factors, such as the supply of raw materials, which also can be forecasted and used in the optimisation process.

Bibliography

[1] Benchmark data sets for dynamic vehicle routing problems. `http://www.fernuni-hagen.de/WINF/inhalte/benchmark_data.htm`.

[2] *Evolutionary Computation, IEEE Transactions on.* Number 99. 2012.

[3] R. L. Ackoff. The future of operational research is past. *JORS*, 30:93–104, 1979.

[4] C. H. Aikens. Facility location models for distribution planning. *European Journal of Operational Research*, 22(3):263–279, December 1985.

[5] A. Akaike and K. Dagdelen. A strategic production scheduling method for an open pit mine. In *Proc. 28th Internat. Appl. Comput. Oper. Res. Mineral Indust. (APCOM) Sympos.,, SME, Littleton, CO*, pages 729–738, 1999.

[6] J. Alander. Indexed bibliography of genetic algorithms papers of 1996. 3:8+, 1998.

[7] E. Alba. Solving the vehicle routing problem by using cellular genetic algorithms. *Evolutionary Computation in Combinatorial*, 152(2):11–20, 2004.

[8] F. Altiparmak, M. Gen, L. Lin, and T. Paksoy. A genetic algorithm approach for multi-objective optimization of supply chain networks. *Computers & Industrial Engineering*, 51(1):196 – 215, 2006. Special Issue on Computational Intelligence and Information Technology: Applications to Industrial Engineering.

[9] N. Astashkin. Scheduling of mining operations. *Journal of Mining Science*, 8:424–429, 1972. 10.1007/BF02497857.

[10] P. Augerat, J. Belenguer, E. Benavent, A. Corbern, D. Naddef, and G. Rinaldi. Computational results with a branch and cut code for the capacitated vehicle routing problem. *Tech. Rep. 1 RR949 -M, ARTEMIS-IMAG, Grenoble France*, 1995.

[11] N. Azi, M. Gendreau, and J. Potvin. An exact algorithm for a single-vehicle routing problem with time windows and multiple routes. *European Journal Of Operational Research*, 178(3):755–766, 2007.

[12] M. Bader-El-Den and S. Fatima. Genetic programming for auction based scheduling. In *Proceedings of the 13th European conference on Genetic Programming*, EuroGP'10, pages 256–267, Berlin, Heidelberg, 2010. Springer-Verlag.

[13] R. Baldacci, N. Christofides, and A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115:351–385, 2008. 10.1007/s10107-007-0178-5.

[14] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Oper. Res.*, 52:723–738, October 2004.

[15] R. H. Ballou. *Business Logistics: Supply Chain Managemen.* Prentice Hall, fifth edition, 2003.

[16] J. F. Bard, G. Kontoravdis, and G. Yu. A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, 36:250–269, May 2002.

[17] J. Beasley. Route first–cluster second methods for vehicle routing. *Omega*, 11(4):403–408, 1983.

[18] J. E. Beasley. Route first–Cluster second methods for vehicle routing. *Omega*, 11(4):403 – 408, 1983.

[19] J. Berger and M. Barkaoui. A new hybrid genetic algorithm for the capacitated vehicle routing problem. *The Journal of the Operational Research Society*, 54(12):pp. 1254–1262, 2003.

[20] A. Bley, N. Boland, C. Fricke, and G. Froyland. A strengthened formulation and cutting planes for the open pit mine production scheduling problem. *Comput. Oper. Res.*, 37:1641–1647, September 2010.

[21] R. B. Boulton, V. L. Singleton, L. F. Bisson, and R. E. Kunkee. *Principles and Practices of Winemaking.* Springer, 1998.

[22] J. Branke. The moving peaks benchmark. `http://www.aifb.uni-karlsruhe.de/~jbr/MovPeaks/movpeaks/`.

[23] J. Branke. Evolutionary approaches to dynamic optimization problems - a survey -. pages 134–137, 1999.

[24] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Proc. of the Congress on Evolutionary Computation CEC99*, pages 1875–1882. IEEE, 1999.

[25] J. Branke. *Evolutionary Optimization in Dynamic Environments.* Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[26] E. K. Burke and A. J. Smith. A memetic algorithm to schedule planned maintenance for the national grid. *J. Exp. Algorithmics*, 4:1, 1999.

[27] E. Busnach, A. Mehrez, and Z. Sinuany-Stern. A production problem in phosphate mining. *Journal of the Operational Research Society*, 36:285–288, 1985.

[28] L. Caccetta and L. M. Giannini. Optimization techniques for the open pit limit problem. In *Australas. Inst. Min. Metall*, pages 57–63, 1986.

[29] L. Caccetta and L. M. Giannini. Application of operations research techniques in open pit mining. In *Asian-Pacific Operations Research : APORS88 (Byong-Hun Ahn Ed.)*, pages 707–724, 1990.

[30] L. Caccetta and S. P. Hill. Optimization techniques for open pit mine scheduling. In *International Congress on Modelling and Simulation*, 1999.

[31] L. Caccetta and S. P. Hill. An application of branch and cut to open pit mine scheduling. *Journal of Global Optimization*, 27:349–365, 2003. 10.1023/A:1024835022186.

[32] K. E. Caggiano, P. L. Jackson, J. A. Muckstadt, and J. A. Rappold. Optimizing Service Parts Inventory in a Multiechelon, Multi-Item Supply Chain with Time-Based Customer Service-Level Agreements. *OPERATIONS RESEARCH*, 55(2):303–318, 2007.

[33] D. Caglar, C.-L. Li, and D. Simchi-Levi. Two-echelon spare parts inventory system subject to a service constraint. *IIE Transactions*, 36(7):655–666, 2004.

[34] I. Capar and F. Ulengin. A taxonomy for supply chain management literature, 2003.

[35] A. Carlisle and G. Dozier. Adapting particle swarm optimization to dynamic environments. In *International Conference on Artificial Intelligence*, pages 429–434, Las Vegas, NV, USA, 2000.

[36] P. Chandra and M. L. Fisher. Coordination of production and distribution planning. *European Journal of Operational Research*, 72(3):503–517, 1994.

[37] S.-H. Chen. *Evolutionary Computation in Economics and Finance.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[38] R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms—i: representation. *Comput. Ind. Eng.*, 30(4):983–997, 1996.

[39] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. *Combinatorial Optimization*, page 431448, 1979.

[40] G. Chryssolouris and V. Subramaniam. Dynamic scheduling of manufacturing job shops using genetic algorithms. *Journal of Intelligent Manufacturing*, 12(3):281–293, 2001.

[41] A. J. Clark. A dynamic, single-item, multi-echelon inventory model. 1958.

[42] A. J. Clark and H. Scarf. Optimal policies for a multi-echelon inventory problem. *Manage. Sci.*, 50(12 Supplement):1782–1790, 2004.

[43] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *OPERATIONS RESEARCH*, 12(4):568–581, 1964.

[44] D. W. Coit and A. E. Smith. Solving the redundancy allocation problem using a combined neural network/genetic algorithm approach. *Computers & Operations Research*, 23(6):515 – 526, 1996.

[45] J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *The Journal of the Operational Research Society*, 52(8):pp. 928–936, 2001.

[46] J. F. Cordeau, G. Laporte, and A. Mercier. Improved tabu search algorithm for the handling of route duration constraints in vehicle routing problems with time windows. *The Journal of the Operational Research Society*, 55(5):pp. 542–546, 2004.

[47] J.-F. Cordeau, G. Laporte, M. W. Savelsbergh, and D. Vigo. Chapter 6 vehicle routing. In C. Barnhart and G. Laporte, editors, *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 367 – 428. Elsevier, 2007.

[48] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms, second edition, 2001.

[49] C. Cotta and J. julin Merelo. Where is evolutionary computation going? a temporal analysis of the ec community, 2007.

[50] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 183–187, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc.

[51] K. Dagdelen and T. Johnson. Optimum open pit mine production scheduling by lagrangian parameterization. In *Proc. 19th Internat. Appl. Comput. Oper. Res. Mineral Indust. (APCOM) Sympos., SME, Littleton, CO*, pages 127–141, 1986.

[52] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[53] C. Darwin. *On the origin of species.* New York :P.F. Collier,, 1859.

[54] L. Davis. Applying adaptive algorithms to epistatic domains. In *IJCAI'85: Proceedings of the 9th international joint conference on Artificial intelligence*, pages 162–164, San Francisco, CA, USA, 1985. Morgan Kaufmann Publishers Inc.

[55] L. Davis. Job shop scheduling with genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 136–140, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc.

[56] L. Davis. Embracing complexity. toward a 21st century supply chain solution.
Web-resource: http://sdcexec.com/online/printer.jsp?id=9012, 2008.

[57] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, Ann Arbor, MI, USA, 1975. AAI7609381.

[58] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* John Wiley & Sons, Inc., New York, NY, USA, 2001.

[59] S. Dimitrakopoulos, R. Ramazan. Uncertainty-based production scheduling in open pit mining. *Transactions - Society for Mining Metallurgy and Exploration Incorporated*, 316:106–112, 2004.

[60] H. Ding, L. Benyoucef, and X. Xie. Simulation optimization in manufacturing analysis: a simulation-optimization approach using genetic search for supplier selection. In *WSC '03: Proceedings of the 35th conference on Winter simulation*, pages 1260–1267. Winter Simulation Conference, 2003.

[61] H. Ding, L. Benyoucef, X. Xie, C. Hans, and J. Schumacher. "one" a new tool for supply chain network optimization and simulation. In *WSC '04: Proceedings of the 36th conference on Winter simulation*, pages 1404–1411. Winter Simulation Conference, 2004.

[62] O. Ergun, J. B. Orlin, and A. Steele-Feldman. Creating very large scale neighborhoods out of smaller ones by compounding moves. *Journal of Heuristics*, 12(1-2):115–140, 2006.

[63] M. Fischetti, P. Toth, and D. Vigo. A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operations Research*, 42(5):pp. 846–859, 1994.

[64] M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981.

[65] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley, New York, USA, 1966.

[66] R. Fukasawa, H. Longo, J. Lysgaard, M. P. D. Aragão, M. Reis, E. Uchoa, and R. F. Werneck. Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem. *Mathematical Programming*, 106(3):491–511, 2006.

[67] S. Ganapathy, S. Narayanan, and K. Srinivasan. Logistics: simulation based decision support for supply chain logistics. In *WSC '03: Proceedings of the 35th conference on Winter simulation*, pages 1013–1020. Winter Simulation Conference, 2003.

[68] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1976.

[69] B. E. Gillett and L. R. Miller. A Heuristic Algorithm for the Vehicle-Dispatch Problem. *Operations Research*, 22(2):340–349, 1974.

[70] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[71] D. E. Goldberg and J. R. Lingle. Alleles, loci, and the traveling salesman problem. In *First International Conference on Genetic Algorithms and Their Applications*, 1985.

[72] B. González, E. J. M. Winter, Gabriel, and B. Galván. Minimum-cost planning of the multimodal transport of pipes with evolutionary computation. *Int. J. Simul. Multidisci. Des. Optim.*, 3(3):401–405, 2009.

[73] J. J. Grefenstette. Genetic algorithms for changing environments. In *Parallel Problem Solving from Nature 2*, pages 137–144. Elsevier, 1992.

[74] S. S. Group. Simulation in the uk manufacturing industry, 1991.

[75] A. Gunasekaran, C. Patel, and E. Tirtiroglu. Performance measures and metrics in a supply chain environment. *International Journal of Operations & Production Management*, 21(1):71–87, Feb. 2001.

[76] P. Hájek. *Mathematics of Fuzzy Logic (Trends in Logic).* Springer, first edition, 1998.

[77] F. Hanssmann. Optimal inventory location and control in production and distribution networks. *Operations Research*, 7(4):483–498, 1959.

[78] F. Herrera. Genetic fuzzy systems: taxonomy, current research trends and prospects. *Evolutionary Intelligence*, 1(1):27–46, Mar. 2008.

[79] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Phys. D*, 42(1-3):228–234, 1990.

[80] E. G. Hinkelman. *Dictionary of International Trade 6th Edition - handbook of the Global Trade Community.* World Trade Press, sixth edition, 2005.

[81] D. S. Hochbaum and A. Chen. Performance analysis and best implementations of old and new algorithms for the open-pit mining problem. *Operations Research*, 48(6):894–914, November/December 2000.

[82] J. H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.

[83] J. H. Holland. Properties of the bucket brigade. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 1–7, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc.

[84] O. Holthaus. Scheduling in job shops with machine breakdowns : an experimental study. *Computers & industrial engineering*, 36(1):Computers & industrial engineering, 1999.

[85] G. S. Hornby and T. Yu. A survey of practitioners of evolutionary computation. In *Studies in Computational Intelligence*, pages 283–297, 2008.

[86] P. Husbands and F. Mill. Simulated coevolution as the mechanism for emergent planning and scheduling. In R. Belew and L. Booker, editors, *Proceedings of the Fourch International Conference on Genetic Algorithms*, pages 264–270. Morgan Kaufmann, 1991.

[87] W. Hustrulid and M. Kuchta. *Open Pit Mine Planning And Design.* Number v. 1 in Balkema proceedings and monographs in engineering, water and earth sciences. Taylor & Francis, 2006.

[88] A. K. Jain and H. A. Elmaraghy. Production scheduling/rescheduling in flexible manufacturing. *International Journal of Production Research*, 35(1):281–309, 1997.

[89] D. Jakobovi, L. Jelenkovi, and L. Budin. Genetic programming heuristics for multiple machine scheduling. In M. Ebner, M. ONeill, A. Ekrt, L. Vanneschi, and A. Esparcia-Alczar, editors, *Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 321–330. Springer Berlin Heidelberg, 2007.

[90] D. Jakobović and L. Budin. Dynamic scheduling with genetic programming. In P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 73–84, Budapest, Hungary, 10 - 12 Apr. 2006. Springer.

[91] Y. Jin, S. Member, and J. Branke. Evolutionary optimization in uncertain environments - a survey. *IEEE Trans. on Evol. Computation*, 2005.

[92] K. Kawahata. *A new algorithm to solve large scale mine production scheduling problems by using the Lagrangian relaxation method.* PhD thesis, Colorado School of Mines, 2006.

[93] J. Koza. *Genetic Programming: A Paradigm For Genetically Breeding Populations Of Computer Programs To Solve Problems.* Stanford University Computer Science Department technical report, 1990.

[94] K. Krishnakumar. Micro-genetic algorithms for stationary and non-stationary function optimization. In *Proc. of the SPIE, Intelligent Control and Adaptive Systems*, pages 289–296, 1989.

[95] E. Kutanoglu and I. Sabuncuoglu. Routing-based reactive scheduling policies for machine failures in dynamic job shops. *International Journal of Production Research*, 39(14):3141–3158, 2001.

[96] P. Laarhoven. Job shop scheduling by simulated annealing. *Operations research*, 40:113, 1992.

[97] G. Laporte and Y. Nobert. A branch and bound algorithm for the capacitated vehicle routing problem. *OR Spectrum*, 5:77–85, 1983. 10.1007/BF01720015.

[98] K. K. Lau, M. J. Kumar, and N. R. Achuthan. Parallel implementation of branch and bound algorithm for solving vehicle routing problem on nows.

In *Proceedings of the 1997 International Symposium on Parallel Architectures, Algorithms and Networks*, ISPAN '97, pages 247–, Washington, DC, USA, 1997. IEEE Computer Society.

[99] A. Law. *Simulation Modeling and Analysis (McGraw-Hill Series in Industrial Engineering and Management).* McGraw-Hill Science/Engineering/Math, 2006.

[100] A. Le Bouthillier and T. G. Crainic. A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Comput. Oper. Res.*, 32:1685–1708, July 2005.

[101] C. Lee and J. Choi. A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights. *Computers & Operations Research*, 22(8):857 – 869, 1995.

[102] H. Lee, J. M. Pinto, I. E. Grossmann, and S. Park. Mixed-integer linear programming model for refinery short-term scheduling of crude oil unloading with inventory management. *Industrial & Engineering Chemistry Research*, 35(5):1630–1641, 1996.

[103] H. Lerchs and I. Grossmannn. Optimum design of open pit mines. In *Canad. Inst. Mining Bull*, pages 47–54, 1965.

[104] J. Levine and F. Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *The Journal of the Operational Research Society*, 55(7):705–716, 2004.

[105] K.-H. Liang, X. Yao, C. Newton, and D. Hoffman. A new evolutionary approach to cutting stock problems with and without contiguity. *Computers & Operations Research*, 29(12):1641 – 1659, 2002.

[106] Y. Lizotte and J. Elbrond. Optimum scheduling of overburden removal in open pit mines. *CIM Bulletin*, 75:154–163, 1982.

[107] S. Luke, K. Sullivan, and F. Abidi. Large scale empirical analysis of cooperative coevolution. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, pages 151–152, New York, NY, USA, 2011. ACM.

[108] Y. T. M. Gen and E. Kubota. Solving job-shop scheduling problem using genetic algorithms. In *16th Int. Conf. on Computer and Industrial Engineering*, pages 576–579, 1994.

[109] P. Machado, J. Tavares, F. B. Pereira, and E. Costa. Vehicle routing problem: Doing it the evolutionary way. In *In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, page 690. Morgan Kaufmann Publishers, 2002.

[110] G. T. Mackulak, F. P. Lawrence, and T. Colvin. Effective simulation model reuse: a case study for amhs modeling. In *WSC '98: Proceedings of the 30th conference on Winter simulation*, pages 979–984, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[111] A. M. Madureira, C. Ramos, and S. C. Silva. Using genetic algorithms for dynamic scheduling. In *I14th Annual Production and Operations Management Society Conference (POMS 2003)*, 2003.

[112] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1–13, 1975.

[113] E. Marchiori and A. Steenbeek. An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. In *Scheduling, in Real World Applications of Evolutionary Computing. Springer-Verlag, Lecture Notes in Computer Science*, pages 367–381. Springer, 2000.

[114] C. H. Martin, D. C. Dent, and J. C. Eckhart. Integrated production, distribution, and inventory planning at libbey-owens-ford. *Interfaces*, 23(3):68–78, 1993.

[115] F. Martinelli. Stochastic comparison algorithm for discrete optimization with estimation of time-varying objective functions. *J. Optim. Theory Appl.*, 103(1):137–159, 1999.

[116] J. M.D. and A. H.-M. Scheduling with neural networks the case of the hubble space telescope. *Computers & Operations Research*, 19(3-4):209–240, 1992.

[117] E. Mezura-Montes. *Constraint-Handling in Evolutionary Optimization.* Springer Publishing Company, Incorporated, 1st edition, 2009.

[118] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs (3rd, revised and extended ed.).* Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[119] Z. Michalewicz. The emperor is naked: Evolutionary algorithms for real-world applications. *ACM Ubiquity*, 2012.

[120] Z. Michalewicz. Quo vadis, evolutionary computation? on a growing gap between theory and practice. In *Proc. of the Congress on Evolutionary Computation*, 2012.

[121] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics.* Springer, enlarged 2nd edition, Dec. 2004.

[122] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4:1–32, 1996.

[123] D. E. Moriarty and R. Mikkulainen. Efficient reinforcement learning through symbiotic evolution. *Mach. Learn.*, 22(1-3):11–32, 1996.

[124] R. Morrison and K. De Jong. A test problem generator for non-stationary environments. In *Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99.*

[125] D. Naso, M. Surico, B. Turchiano, and U. Kaymak. Genetic algorithms for supply-chain scheduling: A case study in the distribution of ready-mixed concrete. *European Journal of Operational Research*, 177(3):2069 – 2099, 2007.

[126] A. M. Newman, E. Rubio, R. Caro, A. Weintraub, and K. Eurek. A review of operations research in mine planning. *Interfaces*, 40(3):222–245, May/June 2010.

[127] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan. A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2012.

[128] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan. Evolving reusable operation-based due-date assignment models for job shop scheduling with genetic programming. In *EuroGP*, pages 121–133, 2012.

[129] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1):160–175, May 1996.

[130] R. K. Oliver and M. D. Webber. Supply-chain management: Logistics catches up with strategy. *Reprint from Outlook (1982) in Logistics: Chapman and Hall*, 1982.

[131] M. Osanloo, J. Gholamnejad, and B. Karimi. Long-term open pit mine production planning: a review of models and algorithms. *International Journal of Mining Reclamation and Environment*, 22:3–35, 2008.

[132] D. Petrovic and D. Alejandra. A fuzzy logic based production scheduling/rescheduling in the presence of uncertain disruptions. *Fuzzy sets and systems*, 157(16):2273–2285, 2006.

[133] E. Pettit and K. M. Swigger. An analysis of genetic-based pattern tracking and cognitive-based component tracking models of adaptation. In M. R. Genesereth, editor, *AAAI*, pages 327–332. AAAI Press, 1983.

[134] J. Picard. Maximum closure of a graph and applications to combinatorial problems. *Management Science*, 22:1268–1272, 1976.

[135] M. Pinedo. Scheduling : theory, algorithms, and systems. 2002.

[136] S. Ponnambalam and M. Mohan Reddy. A ga-sa multiobjective hybrid search algorithm for integrating lot sizing and sequencing in flow-line scheduling. *International Journal of Advanced Manufacturing Technology*, 2003.

[137] E. Popovici. Sequential versus parallel cooperative coevolutionary algorithms for optimization. IEEE Press, 2006.

[138] E. Popovici, A. Bucci, R. P. Wiegand, and E. D. De Jong. Coevolutionary principles. *Handbook of Natural Computing*, pages 1–53, 2010.

[139] E. Popovici and K. De Jong. Understanding cooperative co-evolutionary dynamics via simple fitness landscapes. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 507–514, New York, NY, USA, 2005. ACM.

[140] E. Popovici and K. A. D. Jong. Relationships between internal and external metrics in co-evolution. In *Congress on Evolutionary Computation*, pages 2800–2807, 2005.

[141] M. Potter. *The Design and Analysis of a Computational Model of Cooperative Coevolution*. PhD thesis, George Mason University, 1997.

[142] M. A. Potter and K. A. D. Jong. A cooperative coevolutionary approach to function optimization. In *Proceedings from the Third Conference on Parallel Problem Solving from Nature*, pages 249–257. Springer-Verlag, 1994.

[143] M. A. Potter and K. A. D. Jong. Evolving neural networks with collaborative species. In *Proceedings of the 1995 Summer Computer Simulation Conference*, pages 340–345, 1995.

[144] M. A. Potter and K. A. D. Jong. The coevolution of antibodies for concept learning. In *Proceedings from the Fifth Parallel Problem Solving from Nature*, pages 530–539. Springer-Verlag, 1998.

[145] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985 – 2002, 2004.

[146] D. F. Pyke and M. A. Cohen. Performance characteristics of stochastic integrated production-distribution systems. *European Journal of Operational Research*, 68(1):23 – 48, 1993.

[147] D. F. Pyke and M. A. Cohen. Multiproduct integrated production–distribution systems. *European Journal of Operational Research*, 74(1):18 – 49, 1994.

[148] M. Rabe, F.-W. Jaekel, and H. Weinaug. Reference models for supply chain design and configuration. In *WSC '06: Proceedings of the 38th conference on Winter simulation*, pages 1143–1150. Winter Simulation Conference, 2006.

[149] T. Ray and R. A. Sarker. Optimum oil production planning using an evolutionary approach. In *Evolutionary Scheduling*, pages 273–292. 2007.

[150] I. Rechenberg. *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, 1973.

[151] M. Reimann, K. Doerner, and R. F. Hartl. D-Ants: Savings Based Ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4):563–591, Apr. 2004.

[152] J.-M. Rendu. Geostatistical simulations for risk assessment and decision making: The mining industry perspective. *International Journal of Mining, Reclamation and Environment*, 16, 2002.

[153] C. D. Rosin and R. K. Belew. Methods for competitive co-evolution: Finding opponents worth beating. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 373–381, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[154] B. Samanta, A. Bhattacherjee, and R. Ganguli. *A genetic algorithms approach for grade control planning in a bauxite deposit.* Taylor & Francis, 2005.

[155] L. Schönemann. Evolution strategies in dynamic environments. In *Evolutionary Computation in Dynamic and Uncertain Environments*, pages 51–77. 2007.

[156] H. P. Schwefel. *Numerical optimization of computer models.* Wiley & Sons, Chichester, 1981.

[157] M. Semini, H. Fauske, and J. O. Strandhagen. Applications of discrete-event simulation to support manufacturing logistics decision-making: a survey. In *WSC '06: Proceedings of the 38th conference on Winter simulation*, pages 1946–1953. Winter Simulation Conference, 2006.

[158] D. Simchi-Levi, E. Simchi-Levi, and M. Watson. Tactical planning for reinventing the supply chain. In T. Harrison, H. Lee, and J. Neale, editors, *The Practice of Supply Chain Management: Where Theory and Application Converge*, pages 13–30. Springer, 2003.

[159] M. L. Smith and T. W. Youa. Mine production scheduling for optimization of plant recovery in surface phosphate operations. *International Journal of Mining Reclamation and Environment*, 9:41–46, 1995.

[160] L. Song, X. Li, and A. Garcia-Diaz. Multi-echelon supply chain simulation using metamodel. In *WSC '08: Proceedings of the 40th Conference on Winter Simulation*, pages 2691–2699. Winter Simulation Conference, 2008.

[161] H. Stadtler and C. Kilger. *Supply Chain Management and Advanced Planning.* Springer, 2008.

[162] Supply Chain Council. Supply-Chain Operations Reference-model Version 9.0, 2008. `http://www.supply-chain.org/` [2010-01-15].

[163] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, January 1993.

[164] S. Terzi and S. Cavalieri. Simulation in the supply chain context: a survey. *Comput. Ind.*, 53(1):3–16, 2004.

[165] D. J. Thomas and P. M. Griffin. Coordinated supply chain management. *European Journal of Operational Research*, 94(1):1–15, October 1996.

[166] R. Tinós and S. Yang. Genetic algorithms with self-organizing behaviour in dynamic environments. pages 105–127. 2007.

[167] B. Tolwinski and R. Underwood. A scheduling algorithm for open pit mines. *Journal of Mathematics Applied in Business and Industry*, 7:247–270, 1996.

[168] P. Toth and D. Vigo. *The Vehicle routing problem.* Society for Industrial and Applied Mathematics, 2001.

[169] T. H. Truong and F. Azadivar. Simulation optimization in manufacturing analysis: simulation based optimization for supply chain configuration design. In *WSC '03: Proceedings of the 35th conference on Winter simulation*, pages 1268–1275. Winter Simulation Conference, 2003.

[170] J. G. A. J. van der Vorst, A. J. M. Beulens, and P. van Beek. Modelling and simulating multi-echelon food systems. *European Journal of Operational Research*, 122(2):354–366, April 2000.

[171] F. E. Vergara, M. Khouja, and Z. Michalewicz. An evolutionary algorithm for optimizing material flow in supply chains. *Comput. Ind. Eng.*, 43(3):407–421, 2002.

[172] C. J. Vidal and M. Goetschalckx. Strategic production-distribution models: A critical review with emphasis on global supply chain models. *European Journal of Operational Research*, 98(1):1–18, April 1997.

[173] J. Wang and Y.-F. Shu. Fuzzy decision modeling for supply chain management. *Fuzzy Sets and Systems*, 150(1):107 – 127, 2005.

[174] L. Wang and D.-Z. Zheng. A modified genetic algorithm for job-shop scheduling. *International Journal of Advanced Manufacturing Technology*, 2002.

[175] R. P. Wiegand. *An analysis of cooperative coevolutionary algorithms.* PhD thesis, George Mason University, Fairfax, VA, USA, 2004. Director-Jong, Kenneth A.

[176] H. Wong, B. Kranenburg, G. van Houtum, and D. Cattrysse. Efficient heuristics for two-echelon spare parts inventory systems with an aggregate mean waiting time constraint per local warehouse. *OR Spectrum*, 29(4):699–722, 2007.

[177] Y. Xie and D. Petrovic. Fuzzy-logic-based decision-making system for stock allocation in a distribution supply chain: Research articles. *Int. J. Intell. Syst. Account. Financ. Manage.*, 14(1–2):27–42, 2006.

[178] T. Yamada and C. R. Reeves. Solving the c_{sum} permutation flowshop scheduling problem by genetic local search, 1998.

[179] S. Yang. Explicit memory schemes for evolutionary algorithms in dynamic environments. In *Evolutionary Computation in Dynamic and Uncertain Environments*, pages 3–28. 2007.

[180] X. Yu, K. Tang, and X. Yao. An immigrants scheme based on environmental information for genetic algorithms in changing environments. In *IEEE Congress on Evolutionary Computation*, pages 1141–1147, 2008.

[181] Q. Yun and T. Yegulalp. Optimum scheduling of overburden removal in open pit mines. *CIM Bulletin*, 75:22–31, 1982.

[182] G. Zhou, H. Min, and M. Gen. A genetic algorithm approach to the bicriteria allocation of customers to warehouses. *International Journal of Production Economics*, 86(1):35–45, October 2003.

[183] K. Zielinski, P. Weitkemper, R. Laur, and K.-D. Kammeyer. Parameter study for differential evolution using a power allocation problem including interference cancellation. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1857–1864, 2006.