A multi-objective optimisation suite for Tecnomatix Plant Simulation

by

Toussaint Bamporiki

Thesis presented in fulfilment of the requirements for the degree of Master of Engineering (Industrial Engineering) in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof. JF Bekker

December 2018

Dedicated to my father.

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2018

Acknowledgements

- I would like to thank my supervisor, Prof. James Bekker, for his guidance and support. Prof, your dedication to your work and more importantly to your students, has always been inspiring. Thank you very much for everything.
- I also would like to thank my father, Jean-Marie Bamporiki, to whom I have dedicated this work. Dad, your commitment to your children and your faith in education is what kept me going throughout this project. I simply could not have done this without you. Thank you for always believing in me.
- Finally, I would like to thank my family (my siblings) and my friends, including my USMA research group peers. The support I received from all of you guys during this journey was much appreciated. Much more than I could, probably, ever be able to let you know. Thank you.

Abstract

This thesis presents the development of an optimisation suite for a commercial, discrete-event simulation software package. It is demonstrated in this work that the capabilities of the simulation software are limited in the context of stochastic multi-objective optimisation problems and can, thus, be improved using existing knowledge in the literature. The suite developed in this work utilises, therefore, modern and more effective techniques from the literature to tackle stochastic multi-objective optimisation problems. Its purpose is that of being a third-party multi-objective optimisation solver that can be integrated with the commercial discrete-event simulation software in order to assist it in its limitations. The suite is validated using well-known problems in the literature and the relevance of the solution approach proposed in this thesis is demonstrated.

Opsomming

Hierdie tesis handel oor die ontwikkeling van 'n optimeringsuite vir 'n kommersiële sagtewarepakket wat diskrete gebeure simuleer (oftewel "DES"-sagteware). Die studie toon dat die funksies van die DES-sagteware beperk is in die konteks van stogastiese optimeringsprobleme met veelvuldige doelwitte, en dat dit met behulp van bestaande kennis in die literatuur verbeter kan word. Daarom gebruik die suite wat in die studie ontwikkel is moderne en doeltreffender tegnieke uit die literatuur om stogastiese optimeringsprobleme met veelvuldige doelwitte die hoof te bied. Die doel is dat die suite as 'n derdepartyoplosser van optimeringsprobleme met veelvuldige doelwitte moet dien wat by die kommersiële DES-sagteware geïntegreer kan word en sodoende die beperkinge daarvan te bowe kan kom. Die suite word met bekende probleme in die literatuur gestaaf en die relevansie van die voorgestelde oplossingsbenadering word aangetoon.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

Acronyms

ACO	Ant colony optimisation
ANOVA	Analysis of variances
ASP	Associated stochastic problem
BAP	Buffer allocation problem
CDF	Cumulative distribution function
CEM	Cross-entropy method
CI	Confidence interval
COM	Component object model
COMPASS	Convergent Optimization via Most Promising Area Stochastic Search
COvS	Continuous optimisation via simulation
DLL	Dynamic-link library
DOvS	Discrete optimisation via simulation
DV	Decision variable
ELA	Entrance lane assignment
GA	Genetic algorithm

GUI	Graphical user-interface
HRH	High-level Relay Hybrids
HTH	High-level Teamwork Hybrids
IPC	Inter-process communication
ISC	Industrial strength COMPASS
IZ	Indifference-zone
LFC	Least favourable configuration
LRH	Low-level Relay Hybrids
LS	Local search
LTH	Low-level Teamwork Hybrids
MCP	Mechanised car park
MOO	Multi-objective optimisation
MOO CEM	Cross-entropy method for multi-objective optimisation
MOSO	Multi-objective simulation optimisation
NN	Neural network
OCBA	Optimal computing budget allocation
ODF	Operation dependent failure
OF	Objective function
OvS	Optimisation via simulation
PAD	Priority choice between arrival and departure service
PB	Parking bay
PBA	Parking bay allocation

PD	Park-drive
PDF	Probability density function
R&S	Ranking and selection
SA	Simulated annealing
SAR	Simulation allocation rules
SO	Simulation optimisation
SS	Scatter search
TPS	Tecnomatix Plant Simulation
TS	Tabu search
VTC	Vehicle transfer car
WIP	Work-in-progress

Chapter 1

Introduction

This chapter serves as an introduction to the thesis. Background information for the research is presented, followed by a full description of the problem this study will attempt to solve. The thesis objectives and the research methodology are also discussed. The chapter concludes with a description of the structure of the document.

1.1 Background

Many problems that industrial engineers must solve often require that multiple objectives be simultaneously optimised while searching for the best decisions. These problems occur across various industries and with varying levels of complexity.

Consider, for instance, the following simple example: A company may want to improve (maximise) the performance of a product while trying to minimise cost at the same time (Yang, 2010). It can be seen here that the two objectives the company is trying to achieve are in conflict, as high performance often comes at a cost. The problem may be complicated further, however, if one or both the objectives were subject to a random factor (sometimes referred to as "noise"). For instance, performance in this case may be dependent on the reliability of a component in the product that is subject to random variations. This noise element must be taken into account while the problem is being solved, to ensure that the solution is valid. When randomness is part of a problem, the problem is said to be *stochastic*, as opposed to being *deterministic*. In such cases, computer simulation is often strongly recommended as the solution tool for the problem. Additionally, if the complexity of the problem were such that it could not

be described analytically, computer simulation is again strongly recommended (Law & Kelton, 2000). Using simulation, the noise in the problem is dealt with by means of numerous observations (on the potential decisions to be made) supported by statistics-based data. And in cases where there are no analytical descriptions (or analytical description is difficult to do), a simulation model is used to serve as a *black-box* evaluator that adequately mimics the behaviour of the *real* problem.

In general, problems as the one just described are referred to as *multi-objective optimisation* (MOO) problems. The conflicting objectives in an MOO problem make it difficult to isolate a single best solution to the problem. This is because a solution (*i.e.* a decision or set of decisions) that optimises one or some objectives does not necessarily optimise the rest of them, in fact, improvement in one dimension (*i.e.* objective) in this case is often synonymous with deterioration in at least one other dimension. Thus, if no particular preference is attributed to any objective, it becomes important to identify all (or as many as possible) optimal (or near-optimal) options that exist in order to have knowledge of the different alternatives available, so as to make a more informed decision. The set of optimal options or solutions in this case form what is referred to in the literature as the *Pareto optimal set*.

Finding the Pareto optimal set in many real-life situations is not an easy task as the solution space to a problem can be very large. Moreover, especially when computer simulation is being used, this can become a time-consuming and impractical process if every potential solution is to be evaluated. In such cases, efficient techniques are needed to intelligently search the solution space in order to evaluate, mostly, promising options only. Combining these techniques together with simulation is known in the literature as *simulation optimisation* (SO); an umbrella term for techniques used to optimise stochastic simulation problems (Amaran *et al.*, 2014).

There are many optimisation methods used today to optimise simulation processes. The survey by Amaran *et al.* (2014) presents a considerable number of such methods (*e.g.* response surface methodology, gradient-based methods, direct search *etc.*); and among them are random search methods or *metaheuristics*.

The term metaheuristic generally refers to approximate algorithms for optimisation that are not specifically expressed for a particular problem. Ant colony optimisation, genetic and evolutionary algorithms, simulated annealing and tabu search (in alphabetical order) are typical representatives of the class of metaheuristic algorithms (Blum

et al., 2011). Most metaheuristic algorithms are nature-inspired as they have been developed based on some abstraction of nature (Yang, 2010).

An important question, nonetheless, is what algorithm to use when solving a problem? According to Yang (2010), this depends on many factors. Among them he lists: the type of problem, the solution quality, the available computing resource, the time limit before which a problem must be solved as well as the balance of advantages and disadvantages of each algorithm. This thesis focuses on the first two factors listed.

1.2 Problem description

As already mentioned in the previous section, many solution approaches exist that can help assist a decision-maker in dealing with stochastic optimisation problems. The most efficient and practical ones are generally those that involve the use of optimisation libraries or suites that implement various algorithms, including metaheuristics. Many such optimisation suites are, in effect, powerful tools in practice and are sometimes embedded in discrete-event simulation software products to form integral units that can solve stochastic optimisation problems with more efficiency and with more convenience relative to other existing methods. Nonetheless, these solution approaches (*e.g.* optimisation suites) are sometimes limited in their effectiveness when attempting to handle stochastic optimisation problems in the multi-objective context.

One example of such a product is the commercial, discrete-event simulation software package Tecnomatix Plant Simulation (TPS). TPS has been proven to be a powerful tool at the disposal of an industrial engineer when conducting complex simulation studies (Bamporiki & Bekker, 2017). The software package also provides for a built-in optimisation library for stochastic optimisation problems. The library embedded in TPS is, however, best suited for stochastic optimisation problems in the single objective context. In effect, although the optimisation suite has a solution approach that can be used to solve MOO problems, it is not the most effective approach there is and better approaches exist that are more effective.

The goal in this thesis is to equip TPS with a multi-objective optimisation suite that would allow the simulation software to handle stochastic multi-objective optimisation problems more effectively. The MOO suite is thus to be developed as a third-party

library to be integrated with TPS, and be ready for use whenever the need to solve a MOO stochastic problem with TPS arises.

1.3 Thesis scope and objectives

The purpose of this thesis is to develop an optimisation product that should enable Tecnomatix Plant Simulation to deal with stochastic multi-objective optimisation problems more effectively.

In order to successfully develop this product (*i.e.* the MOO suite), the following objectives are to be pursued in this thesis:

1. To do a comprehensive literature study on the topics pertaining to this study, including:
   - Multi-objective optimisation,
   - Simulation optimisation and SO in the MOO context, and
   - Solution approaches in the literature for SO and MOO problems (including metaheuristics).
2. To design and develop the optimisation suite. This will require:
   - Understanding the concept and the workings of third-party libraries incorporated within simulation software products,
   - Knowledge of how to design and develop such libraries, and
   - Knowledge of how to create user-interfaces for such libraries.
3. To incorporate the developed optimisation suite with Tecnomatix Plant Simulation. This will require a good understanding of the workings of TPS in addition to the knowledge that is needed for Objective 2.
4. To validate the optimisation suite by demonstrating its workings on well-known problems.

In as far as will be possible, considering the vastness of the MOO and SO fields, as well as the knowledge that the student/author will acquire, the optimisation suite to

be developed will attempt to be as effective a tool as it possibly could be, in order to successfully achieve the purpose of this thesis.

This study will only rely on existing algorithms in the literature for MOO and SO problems. The focus will be placed on understanding them for effective implementation and possible hybridisation purposes. The modification of existing algorithms for the purpose of this study falls outside the thesis scope.

1.4 Research methodology

The methodology procedure to be followed in this thesis, in order to develop the optimisation suite to be integrated with Tecnomatix Plant Simulation, is as follows:

1. Rigorously study the existing literature with respect to all the topics mentioned in Objective 1 to acquire a comprehensive understanding of the knowledge that is available.

2. Develop knowledge in computer applications and software: their design, development and implementation. Here if need be, experts in the field will be consulted for assistance and short courses will be followed, in order to successfully achieve Objectives 2 and 3.

3. Select a number of algorithms for the optimisation suite based on the knowledge acquired in the literature.

4. Code and test the workings of the selected algorithms using an appropriate language or platform.

5. Integrate the optimisation suite into Tecnomatix Plant Simulation and ensure that it works as expected; thus completing all the objectives and successfully accomplishing the purpose of the thesis.

1.5 Structure of the document

The present chapter introduces the workings of this document. It provides background information that has ultimately led to the problem at hand, and it fully describes the

problem itself. Moreover, it also specifies the objectives of the thesis as well as the research methodology to be followed in order to successfully complete the project.

In **Chapter** 2, a literature study on multi-objective optimisation and simulation optimisation is presented. The focus in the chapter is placed on the study of existing solution approaches and the directions being suggested by experts in the SO and MOO fields for future developments.

Chapter 3 provides a study of Tecnomatix Plant Simulation's current capabilities (and limitations) in the SO and MOO context. The chapter also serves as a motivation for the product to be developed in the succeeding chapters of the thesis.

The development process of the optimisation suite begins in **Chapter** 4 where an architectural design is presented and a solution approach proposed, following the knowledge acquired in the literature and the results obtained in the previous chapter. The algorithms selected for the optimisation suite are also fully described in the chapter.

Having established the conceptual works of the optimisation product and having supported the reasoning behind the solution approach it utilises, the content of **Chapter** 5 is the actual development and implementation of the optimisation suite. Here, the techniques used to integrate the third-party library with TPS are fully described. Also, the user-interface for TPS is presented and described in great detail.

In **Chapter** 6, the MOO suite is validated using problems in the literature with known solutions.

Having been validated, the optimisation suite is now ready to be tested further using case study problems. **Chapter** 7 is used for this purpose. Specifically, the solution approach proposed in this study is tested and its relevance is demonstrated.

Finally, **Chapter** 8 concludes the research. A summary of the work is provided, followed by a description of the shortcomings experienced in the project and a proposal for future works.

Chapter 2

Literature study

Decision-making under uncertainty and in the presence of conflicting objectives is an important field of study in industrial engineering. Industrial engineers and/or business leaders in practice are expected to guide the operations of various systems/problems by making decisions under such conditions. The literature, as will be seen shortly, is not short of techniques that can assist decision-makers in attempting to solve or find solutions to problems under these circumstances. However, many "elegant" and tractable solution approaches are often limited in the face of uncertainty and conflicting objectives. Researchers continue to strive nonetheless in their quest for improving existing techniques and finding new ways of tackling these problems more effectively and where possible, with better efficiency.

In this chapter, stochastic multi-objective optimisation problems are discussed. The focus is put on the solution approaches that currently exist in the literature and in practice for these problems, as well as on the direction being taken and suggested by researchers with regards to future developments.

2.1 Multi-objective optimisation

In general, a multi-objective optimisation problem is formulated as follows, without loss of generality:

$$\text{Minimise } \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_k(\mathbf{x})]^T \tag{2.1}$$

Subject to

$$g_j(\mathbf{x}) \leq 0, j = 1, 2, ..., N_g, \tag{2.2}$$

$$h_i(\mathbf{x}) = 0, i = 1, 2, ..., N_h \tag{2.3}$$

where k is the number of conflicting objective functions, N_g is the number of inequality constraints, and N_h is the number of equality constraints. $\mathbf{x} \in X$ is a vector of decision variables and X is the *feasible decision or solution space* formally defined as $\{\mathbf{x} \mid g_j(\mathbf{x}) \leq 0, j = 1, 2, ...N_g \text{ and } h_i(\mathbf{x}) = 0, i = 1, 2, ...N_h\}$. Similarly, $\mathbf{f}(\mathbf{x}) \in Y$ is a vector of objective functions and Y is the *feasible objective space* formally defined as $\{\mathbf{f}(\mathbf{x}) \mid \mathbf{x} \in X\}$. For each element in X, there exists an equivalent element in Y (Deb, 2005).

Though (2.1) says "Minimise (or Maximise)" $\mathbf{f}(\mathbf{x})$, not all components of $\mathbf{f}(\mathbf{x})$ follow, necessarily, the same optimisation direction. In effect, the example presented in Section 1.1, showed that the performance and cost objectives had opposite optimisation directions (*i.e.* performance was maximised while cost was minimised). Nonetheless, it is possible through the *duality principle* (Deb, 2005), to use the same optimisation direction for all the objectives in $\mathbf{f}(\mathbf{x})$. According to this principle, if one desires to solve, say, the example in Section 1.1 by using a technique that uses a minimisation approach, one must multiply the performance objective by -1. The objective must then, of course, be converted back to its original form once the problem is solved.

Multi-objective optimisation problems as described here have more than one optimal solution. These are often referred to as Pareto optimal solutions. The reason why this is the case is due to the existing conflict between the objectives, causing the candidate solutions (*i.e.* the decision vectors) to "score" unevenly on the different objectives. It becomes, therefore, difficult to declare one single solution as the ultimate best (see Section 1.1) but rather a set, the Pareto optimal set. The set of Pareto optimal solutions (or Pareto set for short), consequently, consists of all decision vectors for which the corresponding objective vectors cannot be improved in a given dimension (*i.e.* objective function) without worsening another. In other words, they form a set of trade-offs (Chankong & Haimes, 1983).

Throughout this study, the terms *system design* (or simply *design*) as well as *scenario* will be used interchangeably in addition to decision vector and solution, to refer to $\mathbf{x}$.

Following are definitions from Coello Coello (2009) that formally describe the Pareto optimality (minimisation) concept in a deterministic context:

Definition 2.1: Given two vectors $\mathbf{u} = (u_1, u_2, ..., u_k)^T, \mathbf{v} = (v_1, v_2, ..., v_k)^T \in Y$ it is said that $\mathbf{u} \leq \mathbf{v}$ if $u_i \leq v_i$ for $i = 1, 2, ..., k$, and that $\mathbf{u} < \mathbf{v}$ if $\mathbf{u} \leq \mathbf{v}$ and $\mathbf{u} \neq \mathbf{v}$.

Definition 2.2: Given two vectors $\mathbf{u}, \mathbf{v} \in Y$, it is said that $\mathbf{u}$ *dominates* $\mathbf{v}$ (denoted by $\mathbf{u} \prec \mathbf{v}$) if $\mathbf{u} < \mathbf{v}$.

Definition 2.3: It is said that a vector of decision variables $\mathbf{x}^* \in X$ is Pareto optimal if there does not exist another $\mathbf{x} \in X$ such that $\mathbf{f}(\mathbf{x}) \prec \mathbf{f}(\mathbf{x}^*)$.

Definition 2.4: The Pareto optimal set S_p is defined by: $S_p = \{\mathbf{x} \in X \mid \mathbf{x} = \mathbf{x}^*\}$.

Definition 2.5: The *Pareto front* S_pf, which is the set of all Pareto optimal solutions' equivalents in the objective space, is defined by: $S_pf = \{\mathbf{f}(\mathbf{x}) \in Y \mid \mathbf{x} \in S_p\}$.

The decision vectors in S_p are called *non-dominated* and there is no $\mathbf{x}$ in X such that $\mathbf{f}(\mathbf{x})$ dominates $\mathbf{f}(\mathbf{x}^*)$. The dominance concept it illustrated in Figure 2.1, where the red solutions are considered to be non-dominated and the blue ones dominated. The red solutions form, therefore, the Pareto front.

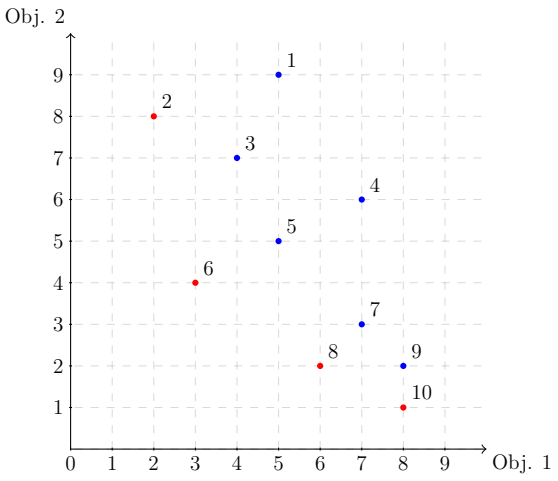


Figure 2.1: An example of Pareto optimal solutions for two minimised objectives.

The goal, when solving a MOO problem, is therefore to obtain for (2.1) the Pareto optimal set S_p by identifying in X all the decision vectors $\mathbf{x}^*$ that satisfy the constraints

(2.2) and (2.3), if they exist.

Goldberg (1989) developed a *Pareto ranking* algorithm that finds S_p with respect to a user-specified threshold t_h, when given a set of N decision vectors $\mathbf{x}_i$ $(i = 1, 2, ..., N)$ and their respective $\mathbf{f}(\mathbf{x})$ values. t_h is an integer value that allows the algorithm to include in S_p, $\mathbf{x} \notin S_p$ that are dominated by, at most, t_h $\mathbf{x}$(s) $(\mathbf{x} \in S_p)$. Now consider $\mathbf{W}$, a matrix with N rows and $n + m + 1$ columns, where n is the number of decision variables in $\mathbf{x}$ and m is the number of objective functions $(m > 1)$. Goldberg (1989)'s algorithm, thus, is as presented in Algorithm 1.

Algorithm 1 Pareto ranking algorithm (minimisation)

1: Input: $\mathbf{W}$ and t_h.
2: Set $j = n + 1$.
3: Sort the working matrix $\mathbf{W}$ with the values in column j in *descending* order.
4: Set $r_p = 1$.
5: Set $r_i = r_p$.
6: If $\mathbf{W}(r_p, j + 1) \geq \mathbf{W}(r_i + 1, j + 1)$, increment the rank value in $\mathbf{W}(r_p, n + m + 1)$.
7: Increment r_i.
8: If $\mathbf{W}(r_p, n + m + 1) < t_h$ and $r_i < N$ return to Step 6.
9: Increment r_p.
10: If $r_p < N$ return to Step 5.
11: Increment j.
12: If $j < n + m - 1$, return to Step 3, otherwise return the rows in $\mathbf{W}$ with rank value not exceeding t_h as the non-dominated members of S_p.

The reality in practice, however, is that S_p can only be approximated as in many cases it is hard to know with certainty whether the true set was obtained. In effect, many real-world problems are such that X is very large and cannot be fully explored practically. Moreover, the problems are often subject to stochastic elements, meaning that the true values of $\mathbf{f}(\mathbf{x}) \in Y$ can only be estimated.

Although this work focuses on methods for approximating the *entire* Pareto set, it is important to state that in some cases this may not be necessary. There exist situations in practice where the decision-maker does already have particular preferences for some objectives over others *prior* to the problem being solved. For example, a decision-maker in the example considered in Section 1.1, may desire a solution whereby performance maximisation is given more importance or more "weight" relative to cost

minimisation. While the Pareto set would, in principle, have such a solution as one of the trade-offs, computational effort could be reduced significantly by focusing solely on finding the unique solution that matches the "preference" of the decision-maker via an appropriate method. Literature is not short of methods for solving MOO problems in this way. In particular, these methods are generally classified into two main groups often referred to as *Scalarisation* and *Constraint* methods. The interested reader can refer to Marler & Arora (2004), where a comprehensive survey on different methods for solving multi-objective optimisation problems is presented. Nevertheless, according to Li *et al.* (2015), it is not always easy to assign fair weights to various objectives, that truly reflect the decision-maker's bias. Moreover, the complexity of some problems may not allow these methods to work correctly (more detail about this will be provided in Chapter 3). So though these methods may be effective in certain cases, using techniques that attempt to find the entire Pareto set is ultimately the ideal approach. In this study, the author refers to such techniques as *Pareto approach* methods/techniques or MOO methods/techniques that use the Pareto approach. This is done to distinguish them from MOO methods that focus on finding single optimal solutions.

So far in this chapter, most of the discussion has been limited to the deterministic context. This is a context whereby it is assumed that there is no random, or stochastic element affecting the correct analysis of a problem. In the next section, simulation optimisation is introduced. The simulation optimisation field is concerned with methods for solving stochastic optimisation problems using simulation (*i.e.* discrete-event simulation, for the purpose of this study).

The simulation optimisation field is vast and has been researched very actively over many years. The oldest contribution towards the SO field in this literature study dates back as far as the year 1954, while the newest contribution is from 2018. It is in this particular field that some of the most significant advances in solution approaches for real-world optimisation problems are being developed.

2.2 Simulation optimisation

The term simulation optimisation is an umbrella term for techniques used to optimise stochastic simulation problems (Amaran *et al.*, 2014) or simply *SO problems*. The

term SO problem is used here to refer to optimisation problems solved with computer simulation for reasons mentioned in Section 1.1.

In their work, Fu *et al.* (2000) distinguished between two kinds of approaches for solving SO problems: one where a constraint set (possibly unbounded and uncountable) is provided, over which an algorithm seeks improved solutions, and another where a fixed set of alternatives is provided *a priori* and the so called *ranking and selection* (R&S) procedures are used to determine the best alternative. According to Fu *et al.* (2000), the focus in the first approach is on the searching mechanism, whereas in the second approach, statistical considerations are paramount.

In a similar way, Yoon & Bekker (2017) have also distinguished between SO problems based on their solution space size which, in the words of the researchers, determines the fundamental approaches needed to solve them. They have categorised, on one hand, SO problems with a relatively small solution space (*small-scale* SO problems) for which R&S procedures are sufficient to find the best solutions and, on the other hand, SO problems with a large solution space (*large-scale* SO problems) for which intelligent search mechanisms, with or without the partnership of R&S procedures, are needed for seeking the optimal or near-optimal solutions.

Both researches are in agreement regarding how to approach SO problems. It is clear that the size of the solution space matters.

2.2.1 Decision variables and solution space size

Given that potential solutions to an SO problem are not definitive nor known in advance, it is important to study the size of the solution space of the problem at hand in order to solve it accordingly. The size of a solution space can be determined by the nature of the decision variables of interest; that is, whether the decision variables are discrete, continuous or mixed; as well as by the boundaries over which the decision variable values are allowed to be selected. Decision variables that can be defined in this manner are often referred to as quantitative decision variables. Besides them, another type also exists that is sometimes referred to as categorical or qualitative (Law & Kelton, 2000) (see for example the problem in Section 3.1).

SO problems with qualitative decision variables are generally small in scale (*i.e.* the size of their solution space is generally small). SO problems with quantitative decision variables, on the other hand, can be either small or large in scale. When the

potential solutions to be evaluated are known in advance and no searching mechanism is needed, then the problem can again be treated as a small-scale problem, despite having quantitative decision variables. When none of the previous applies, then the problem *should* be treated as a large-scale problem if "actual" optimality is to be attained or approximated.

Simulation problems (*i.e.* stochastic problems solved with discrete-event simulation) are generally treated as small-scale problems in simulation studies. Optimisation in this case is reduced to the identification or selection of the best solution(s) out of all potential solutions being considered. But unless such problems are truly small-scale problems, then the solutions found are not "truly" optimal. In effect, when a problem that should be treated as a large-scale problem is reduced to a small-scale one, the approach being taken for the problem is fundamentally wrong. Hence, large-scale and small-scale problems must be differentiated and solved accordingly.

2.2.2 Solution approaches for SO problems in the literature

It is important to mention that in a large portion of the literature on SO, those specifically on large-scale SO, there is a clear separation between solution approaches (or algorithms) used when decision variables are continuous and when they are discrete. In other words, after the size of the space has been determined as being large, it is the nature of the decision variables that dictates which approach (*i.e.* search mechanism) is to be used to solve the problem.

Hong & Nelson (2009a) actually divide SO problems into three categories rather than simply two because of this, with each category requiring distinctive solution approaches. In the first category, the solution space has a small number of solutions (often less than 100, according to the researchers) and the decision variables are numerical or categorical. (This category is identical to the small-scale SO category described earlier.) In the second and third categories, the solution space is large. In the second category in particular, decision variables are exclusively continuous. Problems in this category are also referred to as *continuous optimisation via simulation* (COvS) problems. (Optimisation via simulation (OvS) is another term for simulation optimisation in the literature.) Finally, in the third category, decision variables are exclusively discrete. Problems in this category are also known as *discrete optimisation via simulation* (DOvS) problems. As mentioned already, for each of these categories, the researchers

present in their work a number of solution approaches that are distinctively different from each other (some of them will be discussed in Section 2.5.2). An earlier and similar work in the literature by Andradottir (1998) also presents a review of methods for solving SO problems by distinguishing them as done by Hong & Nelson (2009a).

In this study, however, the author is interested in a class of search mechanisms that is not limited by the nature of decision variables. In other words, the algorithms in this class can be used for both discrete and continuous large-scale problems without the need to be distinctively different for each case. The reason for this choice will be made known as the study progresses. The earlier distinction of SO problems as simply being small or large (in solution space scale) in order to determine the solution approaches to be used to solve them is therefore, *somewhat*, justified for the purpose of this work.

Before discussing small-scale and large-scale SO problems further, SO problems with multi-objectives are first introduced and discussed next.

2.3 Multi-objective simulation optimisation

Multi-objective simulation optimisation (MOSO) problems are MOO problems subject to noise (or stochastic behaviours) or simply SO problems with multiple, conflicting objectives. They are often formulated as, without loss of generality,

$$\text{Minimise } (E[f_1(\mathbf{x},\xi)], E[f_2(\mathbf{x},\xi)], ..., E[f_k(\mathbf{x},\xi)])^T \tag{2.4}$$

$$\text{Subject to}$$

$$\mathbf{x} \in X \tag{2.5}$$

where the expression $f_i(\mathbf{x},\xi), i = 1, 2, ..., k$ represents the *varying* or changing values that objective i can take on when system design $\mathbf{x}$ is selected in the presence of random element ξ, which is responsible for the noise or randomness in the system. $E[f_i(\mathbf{x},\xi)]$ is the expected value of objective i. Because it is difficult to obtain the *true* value of $E[f_i(\mathbf{x},\xi)]$ due to ξ, it is sufficient in practice to rather seek for an estimate of the true value that can be obtained with *enough confidence*, when a number of n simulation replications (or observations) are made.

Consider the notation $f_{ij}(\mathbf{x},\xi)$ where $j = 1, 2, ..., n$ represent the j^{th} observation made for objective i, then

$$\hat{E}[f_i(\mathbf{x},\xi)] = \frac{1}{n}\sum_{j=1}^{n} f_{ij}(\mathbf{x},\xi), \tag{2.6}$$

is an estimate value for objective i.

Due to the use of estimates (or sample means) in the case of MOSO problems, the Pareto optimal set obtained is sometimes called the "observed Pareto set" or the "approximate Pareto set". In this work, it will simply be referred to as *Pareto set*. The term "observed" in this study is thus implied as all the problems considered are MOSO problems, unless stated otherwise. Similarly, note that all the definitions in Section 2.1 apply here in the stochastic sense *e.g.* $f_i(\mathbf{x}) = E[f_i(\mathbf{x},\xi)]$ *etc.*

The MOSO problem as defined in this section represents the framework of all problems that will be considered in this thesis, with expression (2.5), however, applicable for the case of large-scale problems only; and $k = 2$.

2.4 Small-scale SO problems

Small-scale SO problems are problems whose potential solutions are known or preselected (see Section 2.2.1). Such problems can be solved with ranking and selection procedures. There are also other methods used in the literature to solve these problems which will be briefly mentioned in this section. The focus in this study, however, is on ranking and selection.

2.4.1 Ranking and selection

R&S procedures are statistical methods developed to select the best system design or a subset that contains the best system design from a set of n competing alternatives (Goldsman & Nelson, 1994). Efficient R&S procedures also aim, in the process, to minimise the total number of simulation replications required while preserving a desired confidence level. Two important R&S procedures dominate the literature: the *indifference-zone* (IZ) methods and the *optimal computing budget allocation* (OCBA) methods. They are discussed in this section.

R&S procedures (or algorithms) find their origin in the 1950s within the statistics community. Bechhofer (1954) was the first to introduce the concepts of indifference-zone and probability of correct selection P(CS). His work aimed to improve on the then

(and possibly still) popular method of analysis of variances (ANOVA) "deficiencies". Following his contribution, R&S drew the attention of the simulation community due to its potential usefulness in stochastic simulation output analysis and many researchers have since built upon the foundations laid by Bechhofer (1954). In particular, Dudewicz & Dalal (1975) then Rinott (1978) further improved on Bechhofer's work, proposing more efficient IZ methods. Rinott's method (Rinott, 1978) particularly, which is discussed later in this section, is one of the simplest and well-known R&S procedures and will be used in this study to illustrate the basic concept behind IZ methods (Kim & Nelson, 2007).

2.4.1.1 Indifference-Zone methods

The main idea behind IZ methods is to guarantee, with a probability of at least P^*, that the system design ultimately selected is the best (Bechhofer, 1954). Kim & Nelson (2007) provide a comprehensive survey on recent advances on the topic and they discuss, in detail, a number of IZ methods. In Yoon & Bekker (2017), which is another survey, a procedure by Chen & Lee (2009) is presented that attempts to use the IZ concept in the MOO context. The study, however, remains an empirical study and does not guarantee the probability of correct selection requirement $\mathrm{P(CS)} \geq P^*$ for the final Pareto optimal set (Yoon, 2018). This was achieved in Yoon (2018), where the researcher presents a new IZ multi-objective R&S procedure with $\mathrm{P(CS)} \geq P^*$ guaranteed.

IZ methods make use of a parameter δ, which is set by the experimenter or the decision-maker to be the smallest actual difference that is important to detect. If the difference between the estimated means of any two system designs is within δ, then the difference between them is viewed as being, for practical purposes, insignificant; meaning that the decision-maker is indifferent (hence the name *indifference-zone*) in selecting or ignoring these system designs depending on how they compare with other competing system designs outside the IZ. To illustrate how the IZ methods work, the following two-stage IZ method, namely *Procedure R* by Rinott (1978) is repeated here (Algorithm 2).

The constant h^* in Step 4 is the solution to the following double integral equation:

$$\int_0^\infty \left[\int_0^\infty \frac{h^*}{\sqrt{(n_0-1)(\frac{1}{x}+\frac{1}{y})}} f(x)\mathrm{d}x \right]^{k-1} f(y)\mathrm{d}y = P^*, \tag{2.7}$$

Algorithm 2 Procedure R

1: Select the probability requirement P^*, the indifference-zone value δ^*, and the first-stage sample size $n_0 \geq 2$.
2: Run n_0 simulations for each system i $(i = 1, ..., k)$.
3: Calculate sample variances $S_i^2(n_0)(i = 1, ..., k)$.
4: Let $N_i = \max\left\{n_0, \left\lceil \left(\frac{h^* S_i(n_0)}{\delta^*}\right)^2 \right\rceil\right\}$, where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x, and h^* is the solution to (2.7).
5: Run additional $N_i - n_0$ simulation replications for system i $(i = 1, ..., k)$.
6: Compute the overall sample means $\bar{X}_i(N_i)(i = 1, ..., k)$ and present system b as the best system, where $b = \arg\min_i \bar{X}_i(N_i)$.

where f denotes the probability density function (pdf) of the χ^2 distribution with $n_0 - 1$ degrees of freedom.

Procedure R as well as other IZ methods use the *least favourable configuration* (LFC) assumption, which prevents them from taking advantage of the sample mean information (Yoon, 2018), making them more conservative than they should be. Yoon (2018) developed a more efficient IZ method based on Procedure R, the $\mathcal{MY}$ procedure, which follows the *Bayesian* probabilistic approach, instead of the LFC assumption, for its probability of correct selection formulation. The procedure is presented in Algorithm 3.

2.4.1.2 Optimal Computing Budget Allocation methods

OCBA methods have been developed to address the efficiency issue related to the many simulation replications that are often utilised during R&S procedures. OCBA methods follow the Bayesian probabilistic theory. The main idea here is to *maximise* the probability of correct selection P(CS) by intelligently controlling the number of simulation replications based on the mean and variance information in the face of limited computing budget (Lee *et al.*, 2010). OCBA has also been successfully adapted for multi-objective problems. Lee & Goldsman (2004), for example, incorporated the concept of Pareto optimality in OCBA and used the method to find non-dominated system designs.

Many OCBA methods exist in the literature for single and multi-objective problems. The survey by Lee *et al.* (2010) lists a number of them and points to further references

Algorithm 3 Procedure $\mathcal{MY}$

1: Select the probability requirement $P^* = 1 - \alpha$, the indifference-zone value δ^*, and the first-stage sample size $n_0 \geq 2$. Let $I = \{1, 2, ..., M\}$ be the set of systems in competition, and let $\beta = \frac{\alpha}{M-1}$.

2: Simulate n_0 replications for all M systems, and calculate sample means $\bar{X}_i(n_0)$ and sample variances $S_i^2(n_0)$. Let $N_i = n_0$ $(i = 1, ..., M)$, and let $b = \arg\min_i \bar{X}_i(N_i)$.

3: Delete system i $(i \neq b)$ from I if

$$N_i \geq \left\lceil \left(\frac{hS_i(N_i)}{\delta_i} \right)^2 \right\rceil \text{ and } N_b \geq \left\lceil \left(\frac{hS_b(N_b)}{\delta_i} \right)^2 \right\rceil, \tag{2.8}$$

and delete system b from I if

$$N_b \geq \left\lceil \left(\frac{hS_b(N_b)}{\delta_i} \right)^2 \right\rceil \text{ for all } i \neq b, \tag{2.9}$$

where $\delta_i = \max\{\delta^*, \bar{X}_i(N_i) - \bar{X}_b(N_b)\}$, and $\lceil x \rceil$ denotes the smallest integer greater than or equal to x, and h is the solution of:

$$\int_0^\infty \left[\int_0^\infty \frac{h}{\sqrt{(N_i - 1)\frac{1}{x} + (N_b - 1)\frac{1}{y}}} f_1(x)\mathrm{d}x \right] f_2(y)\mathrm{d}y = 1 - \beta, \tag{2.10}$$

where f_1 and f_2 denote the pdf of the χ^2 distribution with $N_i - 1$ and $N_b - 1$ degrees of freedom, respectively.

4: If $|I| = 0$, then stop and present system b as the best system. Otherwise, go to Step 5.

5: Take one additional observation X_{i,N_i+1} from each system $i \in I$, and set $N_i \leftarrow N_i + 1$ $(\forall_i \in I)$. Set $I = \{1, 2, ..., M\}$ and update $\bar{X}_i(N_i)$, $S_i^2(N_i)$ and $b = \arg\min_i \bar{X}_i(N_i)$, go to Step 3.

for more.

2.4.2 Other algorithms for small-scale SO

Besides R&S methods, there are other procedures available for solving small-scale SO problems. These are often referred to as *multiple comparison procedures*. In these procedures a number of simulation replications are performed on all the potential designs, and conclusions are made by constructing confidence intervals on the performance metric (Amaran *et al.*, 2014). (See also Tekin & Sabuncuoglu (2004) and Rosen *et al.* (2008).)

2.5 Large-scale SO problems

It was said earlier in this chapter that the focus in solving large-scale SO problems was on the search mechanisms used to explore the vast, and sometimes complex, solution spaces (Fu *et al.*, 2000). It was also said in Chapter 1 that techniques capable of finding good enough solutions in reasonable computational time were favoured in practice. These are the techniques that were alluded to by the author in Section 2.2.2. In effect, many large-scale SO problems can be expensive to run in terms of time, money or resources (Amaran *et al.*, 2014). The use of efficient techniques or search mechanisms in solving these problems is therefore key.

Though the literature has a number of techniques for solving large-scale SO problems as discussed in Section 2.2.2, *metaheuristics* seem to be preferred in practice (Amaran *et al.* (2014), Fu (2002)). For more details on reasons why that is the case, the reader can refer to Fu (2002), where the researcher contrasts between the focus of researchers in the SO field and the techniques being adopted in practice. Nevertheless, it is widely known that many of the solution approaches that are specifically devised to handle large SO problems in the research community (see Andradottir (1998) and Hong & Nelson (2009a)) are often limited in practice. A brief discussion on these methods is provided in this section.

Metaheuristic algorithms such as the genetic algorithm (GA) (briefly discussed in Section 3.2.1), the simulated annealing (SA), the tabu search (TS), cross-entropy method (CEM) and the ant colony optimisation (ACO), however, have been proven to be effective search mechanisms for many practical large-scale complex deterministic

problems, including those with multi-objectives. This logically makes them good candidates for large-scale SO problems as well, despite some of their own limitations. In the next section, an attempt to formally define metaheuristics is made and the different metaheuristics mentioned above are discussed in more detail.

2.5.1 Metaheuristics

Metaheuristics are a class of approximate solution methods that have developed dramatically since their inception in the early 1980s. They are designed to attack complex (deterministic) optimisation problems where classical heuristics and optimisation methods have failed to be effective and efficient (Osman & Laporte, 1997).

The literature has a number of *formal* definitions for the word metaheuristic (see for example Blum *et al.* (2008)). There does not seem to be a consensus on a singular definition for the word, possibly due to the generality of the metaheuristic concept.

Most definitions seem to include many important aspects of the workings of many metaheuristics. However, the more one learns about new metaheuristics (which there are a large number of), the more one realises how challenging it is to cover, in a single concise definition, what a metaheuristic is exactly. The following formal definition was thus selected as it tries not to be very specific and, in the author's opinion, captures well the broadness of the concept (Dorigo *et al.* (2006)):

> A metaheuristic is a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems. In other words, a metaheuristic is a general-purpose algorithmic framework that can be applied to different optimisation problems with relatively few modifications.

Most metaheuristics are created to address, in an approximative way, deterministic optimisation problems for which no exact algorithms exist to solve the problems efficiently *i.e.* in a practical manner. Metaheuristics are able to do this because they are not problem structure-dependent (at least not as much as many methods in the research community), a characteristic that makes them robust heuristics according to Hong & Nelson (2009a). Rather, they rely on simple principles of nature that they are able to model in generic mathematical frameworks and apply to a variety of optimisation problems. But why nature? According to Yang (2010), nature has evolved over

millions of years and has found perfect solutions to almost all the problems she met. We can thus learn the success of her problem-solving (ability) and develop nature-inspired heuristic algorithms.

Metaheuristics are generally globally convergent; meaning that if iterated long enough, under the right user-defined parameters, they may converge to the optimum (or optima, in the case of MOO problems). But in any case, they guarantee at least good solutions in a reasonable amount of computational time.

For the purpose of this study, the metaheuristics presented next are believed to be good candidates for the SO context, due to their effectiveness in solving deterministic problems. They are discussed in some detail, narratively and using pseudo-codes, and additional references are provided for more information. A brief discussion on other methods (non-metaheuristics) available for SO problems is also provided at the end of the section.

2.5.1.1 Simulated annealing

The simulated annealing algorithm is believed to be the oldest among the metaheuristics. According to Weise (2009), Kirkpatrick *et al.* (1983) pioneered the utilisation of SA for global optimisation in the early 1980s after being inspired by the work of Metropolis *et al.* (2002). The algorithm developed was initially applied to various combinatorial (discrete) optimisation problems and since then, there have been extensive studies on the topic.

The SA algorithm mimics the annealing process in material science where a material (*e.g.* metal or glass) is strengthened through heat treatment that is followed by a carefully controlled cooling process. This allows the material to reach a stable state whereby its defects are removed and its strength is increased (Radin (1998), Bandyopadhyay *et al.* (2008), Gendreau & Potvin (2010)).

Let X be the solution space and $f : X \to Y$ be an objective function defined on the solution space. The goal is, without loss of generality, to find a global minimum $\mathbf{x}^* \in X$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $(\mathbf{x} \in X)$. Now, define $N(\mathbf{x})$ as a set of solutions constituting the neighbourhood function for $\mathbf{x}$. Associated with every solution or system design $(\mathbf{x} \in X)$, therefore, are neighbouring solutions $N(\mathbf{x})$ that can be attained from $\mathbf{x}$ in a single iteration or a single move. Algorithm 4 illustrates how the metaheuristic works (Eglese, 1990).

Algorithm 4 Simulated annealing metaheuristic

Select an initial state $\mathbf{x} \in X$, an initial temperature $T > 0$.
Set temperature change counter $t = 0$.
while $n < N(t)$ **do**
    Generate state $\mathbf{x}'$, a neighbour of $\mathbf{x}$.
    Calculate $\delta = f(\mathbf{x}') - f(\mathbf{x})$.
    if $\delta < 0$ **then**
        $\mathbf{x} \leftarrow \mathbf{x}'$
    else
        if random$(0, 1) <$ exp$(-\delta/T)$ **then**
            $\mathbf{x} \leftarrow \mathbf{x}'$
        end if
    end if
    $n \leftarrow n + 1$.
end while
$t \leftarrow t + 1$.
$T \leftarrow T(t)$.
Until stopping criterion is true.

Applications of SA are numerous and the range of problems the algorithm is able to solve is vast. The reader is referred to Gendreau & Potvin (2010), Weise (2009) and Osman & Laporte (1997) for more detail. There are also many MOO variants of the SA algorithm. As an example, Bandyopadhyay *et al.* (2008) adapted the SA algorithm for MOO problems. The researchers proposed AMOSA, a simulated annealing-based multi-objective optimisation that finds a set of trade-off solutions.

2.5.1.2 Tabu search

According to Weise (2009), Glover (1986) initially introduced the basic ideas of tabu search and later in future works (Glover (1989), Glover (1990)), developed it into a general framework.

TS is one of many metaheuristics devised to overcome the limitations of traditional local search (LS) heuristics by using extended search strategies where traditional LS would normally stop. According to Blum *et al.* (2008), the basic idea of TS is the explicit use of search history, both to escape from local optima and to implement a

strategy for exploring the search space.

TS introduces into the LS scheme the concept of memory, in the form of the so-called *tabu list* (Blum *et al.*, 2008) (a list that, momentarily, remembers a number of prohibited candidate solutions) to help avoid the local optima trap.

Suppose a function $f(\mathbf{x})$ is to be minimised over some domain. TS-based algorithms can be generalised in two main steps, namely, the initialisation and the search step (Gendreau & Potvin, 2010). Consider the following notation (Hertz & de Werra (1990), Gendreau & Potvin (2010)): $\mathbf{x}$ is the current or incumbent solution, $\mathbf{x}^*$ the best-known solution, f^* the performance of $\mathbf{x}^*$, $N(\mathbf{x})$ the neighbourhood of $\mathbf{x}$, $\mathbf{x}'$ the admissible subset of $N(\mathbf{x})$ *i.e.* non-tabu candidate solutions, and T the tabu list. Algorithm 5 illustrates how the metaheuristic works.

Algorithm 5 Tabu search metaheuristic

Initialisation:
Construct initial solution $\mathbf{x}_0$.
Set $\mathbf{x}^* \leftarrow \mathbf{x}_0$, $f^* \leftarrow f(\mathbf{x}_0)$, $T \leftarrow \emptyset$.
Search:
while termination condition is not met **do**
    Select $\mathbf{x} = \arg\min_{\mathbf{x}' \in N(\mathbf{x})}[f(\mathbf{x}')]$.
    if $f(\mathbf{x}) < f^*$ **then**
        $f^* \leftarrow f(\mathbf{x})$, $\mathbf{x}^* \leftarrow \mathbf{x}$
    end if
    Record $\mathbf{x}$ in T and delete the oldest entry if necessary.
end while

According to Hertz & de Werra (1990), TS is one of the most efficient metaheuristics for handling large optimisation problems. Hertz (1991) used TS to solve a large-scale timetabling problem. In Toth & Vigo (2003), TS is used for a wide class of combinatorial optimisation problems while Caballero *et al.* (2007) adapted a metaheuristic for multi-objective combinatorial optimisation problems based on TS to solve a multi-objective location routing problem.

2.5.1.3 Cross-entropy method

The cross-entropy method was motivated by an adaptive algorithm for estimating probabilities of rare events in complex stochastic networks (Rubinstein, 1997). It was soon

realised that a simple cross-entropy modification of Rubinstein (1997) could be used for solving difficult optimisation problems as well (Rubinstein, 1999).

The CEM involves an iterative procedure where each iteration can be broken down into two phases (de Boer *et al.*, 2005). Before the iterative procedure, however, the CEM associates with each optimisation problem a rare event estimation problem, the so-called *associated stochastic problem* (ASP) (Kroese *et al.*, 2006). After the ASP is defined, the two iterative phases are as follows:

1. Generate a random data sample according to a specified mechanism.
2. Update the parameters of the random mechanism based on the data to produce a "better" sample in the next iteration.

So the algorithm first samples randomly from a chosen probability distribution over the space of decision variables. For each sample, a corresponding function evaluation is obtained. Based on the function values observed, a predefined percentile of the best samples is picked. A new distribution is then built around this "elite set" of points via a fitting method such as the *maximum likelihood ratio estimator* and the process is repeated. Algorithm 6 illustrates how the metaheuristic works (Amaran *et al.*, 2014).

Algorithm 6 Cross-entropy method metaheuristic

Requirement: θ, an initial set of parameters for a pre-chosen distribution $p(\mathbf{x};\theta)$ over the set of decision variables; s, a number of simulations to be performed; e, the number of elite samples representing the top δ percentile of the s samples.
while not converged or within simulation budget **do**
    for $i = 1 \rightarrow s$ **do**
        Sample $\mathbf{x}_i$ from $p(\mathbf{x};\theta)$.
        $t_i \leftarrow \text{simulate}(\mathbf{x}_i)$.
    end for
    $E \leftarrow \emptyset$.
    for $i = 1 \rightarrow e$ **do**
        $E_i \leftarrow \arg\min_{i \notin E} t_i$.
    end for
    $p(\mathbf{x};\theta) \leftarrow \text{fit}(\mathbf{x}_E)$.
end while

The CEM is often classified as a *model-based* metaheuristic. These are metaheuristics that attempt to build a probability distribution over the space of solutions and use it to guide the search process (Amaran *et al.*, 2014).

In the literature, Alon *et al.* (2005) applied the CEM to the well-known buffer allocation problem in a SO context. Bekker & Aldrich (2011) adapted the CEM for MOO and validated the proposed algorithm to known test problems. In Bekker (2012), the algorithm in Bekker & Aldrich (2011) is integrated with the Arena software package and used to solve MOSO problems.

2.5.1.4 Ant colony optimisation

Inspired by the research done by Deneubourg *et al.* (1983) on real ants, Dorigo *et al.* (1996) developed the ant colony optimisation algorithm (Weise, 2009).

ACO is one of many swarm intelligence methods. Swarm intelligence is a relatively new approach to problem-solving that takes inspiration from the social behaviours of insects and of other animals (Dorigo *et al.*, 2006).

ACO is a set of search algorithms that takes inspiration from the foraging behaviour of real ants. Most ant species' way of foraging enables them to find the shortest paths between food sources and their nests. When foraging, a swarm of ants communicates indirectly in their local environment by the laying of scent chemicals or *pheromone*, creating trails that link the food source with their nest (Yang, 2010). The first members of the colony that find their way to the food source do it randomly by trying different routes. Future members, however, are able to decide on what routes to follow thanks to the pheromone deposited by the members of the colony gone before them. The higher the pheromone concentration on a route, the higher the probability it will be selected by an ant. Experiment shows that as time progresses, the shortest route will start to have higher traffic density, causing a gradual increase on its pheromone concentration while the pheromone of the other routes experiencing low traffic evaporates progressively. Eventually, the great majority of ants in the colony converge into a single route, the shortest one.

In ACO algorithms, artificial ants are stochastic solution construction procedures that build candidate solutions for the problem under consideration by exploiting artificial pheromone information that is adapted based on the ants' search experience (Gendreau & Potvin, 2010). The pheromone trails are simulated via a parameterised

probabilistic model that is called the pheromone model. It consists of a set of model parameters whose values are called the pheromone values. These values act as the memory that keeps track of the search process. The basic ingredient of ACO algorithms is a constructive heuristic that is used to, probabilistically, construct solutions using the pheromone values. Algorithm 7 illustrates how the metaheuristic works (Dorigo *et al.*, 2006).

Algorithm 7 Ant colony optimisation metaheuristic

1: Set parameters.
2: Initialise pheromone trails.
3: **while** termination condition is not met **do**
4: *Construct ant solutions.*
5: *Apply local search* (optimal).
6: *Update pheromones.*
7: **end while**

ACO algorithms are often classified as both model- and *population-based* metaheuristics; population-based because they use a set of solutions rather than a single solution at each iteration. In the literature, ACO is mostly used for discrete optimisation problems, though variants of the metaheuristic for continuous problems also exist. In Merkle *et al.* (2002), the researchers use ACO to solve a resource-constrained scheduling problem whereas Bella & McMullen (2004) use a variant of the algorithm to solve a vehicle-routing problem. Efforts have also been made to adapt ACO for MOO problems and variants of the algorithm for this purpose can be found in Gendreau & Potvin (2010).

2.5.2 Other search mechanisms

It was mentioned in Section 2.5.1 that metaheuristics are generally devised for deterministic problems. There are other search mechanisms, however, that have been specifically designed for SO problems. The main distinguishing aspect of these techniques is that, contrary to metaheuristics, they all have a "noise handling strategy" in the form of *simulation allocation rules* (SAR) embedded in their algorithmic procedures. Despite such an advantage, nonetheless, most of these algorithms are generally less robust than metaheuristics. Unlike metaheuristics that can be easily adapted to

different problems (*e.g.* problems with decision variables with different nature), here algorithms can be too problem-specific, a factor that often limits their use in practice.

There is thus a clear distinction between SO methods for continuous problems and SO methods for discrete problems here. In the literature, these methods are better known as COvS and DOvS (as seen before in Section 2.2.2) algorithms.

2.5.2.1 COvS algorithms

COvS methods have been researched intensively in the past, arguably more than DOvS. COvS algorithms include, among others, the *stochastic approximation*, the *gradient estimation* and the *sample path optimisation* methods. For more detail, the reader is referred to Andradottir (1998), Tekin & Sabuncuoglu (2004) and Hong & Nelson (2009a), which give reviews on the topic.

2.5.2.2 DOvS algorithms

In recent years, research on DOvS methods have been trying to close the gap on some of the advances made in the COvS field. There are many DOvS algorithms that can be found in the literature (Andradottir (1998), Hong & Nelson (2009a), Yoon & Bekker (2017)). Among them is the *Convergent Optimization via Most Promising Area Stochastic Search* (COMPASS) algorithm due to Hong & Nelson (2009b).

COMPASS solves DOvS problems by implementing an adaptive region called the most promising area, where preferable solutions can be found with high probability. It has been shown that, in both constrained and unconstrained search space, the algorithm asymptotically converges to one of the local optima (Li *et al.*, 2015). Li *et al.* (2015) also successfully adapted the algorithm for MOO problems with the Pareto approach. Algorithm 8 illustrates how the method works (Hong & Nelson, 2009a).

Despite the "non-metaheuristics" having *some* statistical features in the form of their SAR, these are often not as efficient nor as effective as those used in R&S procedures. There are therefore efforts being made, in this particular field, to further improve the noise-handling ability of these large-scale SO algorithms. For example, Xu *et al.* (2010) suggested that OCBA be used as the SAR for COMPASS to improve the efficiency of the algorithm. Efforts to improve SO methods are thus continuously being made to make the algorithms more effective.

Algorithm 8 COMPASS algorithm

1: Build the most promising area in each iteration around the current sample best solution based on geometry.
2: Sample new solutions from the most promising area in each iteration.
3: Simulate new solutions and solutions that define the most promising area a little bit more.
4: Calculate the cumulative sample average for each active solution, and choose the solution with the best cumulative sample average. Go to Step 1.

Metaheuristics are also continuously being improved in various ways. One way to make them more effective is to combine them with other algorithms in order to benefit from the synergy. This concept is known as *hybridisation* and is discussed in the following section.

2.6 Hybrid metaheuristics

The concept of hybrid metaheuristics has been commonly accepted only in recent years, even if the idea of combining different metaheuristic strategies and algorithms dates back to the 1980s (Blum *et al.*, 2008).

According to Blum *et al.* (2011), quite an impressive number of algorithms have been reported in the literature that do not purely follow the paradigm of a single traditional metaheuristic. On the contrary, they combine various algorithmic components, often originating from algorithms of other research areas on optimisation.

Blum *et al.* (2008) and Talbi (2013) distinguish between two main categories of hybrids: the first consists of metaheuristics that are combined with other metaheuristics while the second consists of metaheuristics that are combined with other techniques in fields such as operations research and artificial intelligence. For the purpose of this thesis, the author is interested in the second category. Talbi (2013) presents a taxonomy of hybrid metaheuristics in an attempt to provide a common terminology and classification mechanism of these algorithms. The researcher (*i.e.* Talbi (2013)) argues that the taxonomy could be used to classify any hybrid. In an earlier work (Talbi, 2002), the researcher collected a comprehensive number of hybrids developed up to that point in time and demonstrated that they could all be classified using the taxonomy.

The taxonomy has two parts, namely, a hierarchical classification and a flat classification. The focus in this study is placed on the hierarchical classification (Figure 2.2). The flat classification, which provides additional detail on the hierarchical classification, can be left out without harm for the purpose of this work. A discussion about Figure 2.2 from Talbi (2013) is presented next.

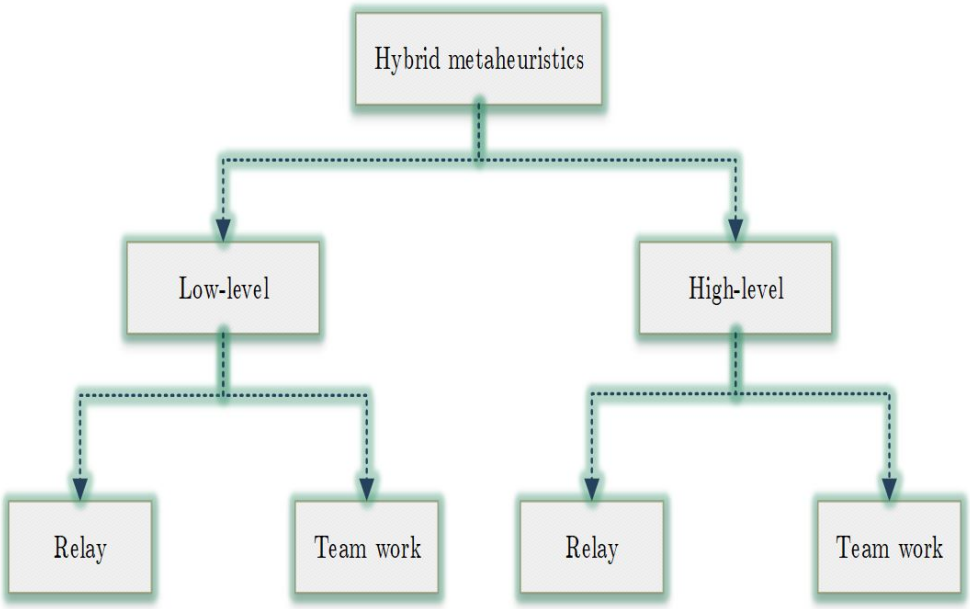


Figure 2.2: Hierarchical classification of hybrid metaheuristics.

The hierarchy structure has two levels. At the top, one can distinguish between low- and high-level hybridisations. The *low-level* hybridisation addresses the functional composition of a single optimisation method. In this hybrid class, a given function of a metaheuristic is replaced by another algorithmic procedure (could be another metaheuristic) so as to make the hybrid better. In *high-level* hybrids, on the other hand, the algorithms being brought together remain self-contained and there is no direct relationship to their internal workings.

At the bottom, we distinguish between the *relay* and the *teamwork* hybridisations. In relay hybrids, a set of algorithms is applied one after the other, each using the output of the previous as its input, acting in a pipeline fashion; while in teamwork hybrids, algorithms cooperate to simultaneously carry out a search in a solution space.

Four classes are derived from this hierarchical taxonomy and they are discussed in the following subsections.

2.6.1 Low-level Relay Hybrids (LRH)

This class of hybrids groups algorithms in which a given heuristic is embedded into a trajectory or single point-based metaheuristic (*S-metaheuristic*). The idea here is to enhance the performance of the S-metaheuristic by adding to it an aspect or aspects of the heuristic (possibly another S-metaheuristic) that can make the principal S-metaheuristic more efficient.

For example, Martin *et al.* (1992) enhanced SA's performance by combining it with a *multiple-start local search* heuristic. Multiple-start local search heuristics seek for the global optimum by restarting the search at different starting points every time the algorithm reaches a local optimum. All local optima are subsequently compared after a user-defined number of restarts and the best solution is chosen as the approximate global optimum. It is this ability to sample from different local optima that makes the heuristic a good hybrid candidate for the SA algorithm. Recalling the description of the SA algorithm in Section 2.5.1.1, the benefit of embedding the heuristic in SA is that the neighbouring solution $\mathbf{x}' \in N(x)$ is no longer generated at random but with the help of the heuristic, thus making the search mechanism more effective.

2.6.2 Low-level Teamwork Hybrids (LTH)

In this hybrid class, S-metaheuristic algorithms are embedded into population-based metaheuristics (*P-metaheuristics*). It is known that P-metaheuristics are powerful in the exploration of the solution space and sometimes weak in its exploitation. On the other hand, S-metaheuristics are known to be powerful optimisation methods in terms of exploitation and rather weak in terms of exploration. The two classes can thus be complementary and the goal of this hybrid class is to exploit this complementarity.

In the work of Huang & Liao (2008), ACO is combined with a variant of TS to create a hybrid algorithm that merges the exploration strength of ACO and the exploitation capabilities of TS. More specifically, considering the description of the ACO algorithm (Section 2.5.1.4), once all artificial ants have constructed their own solutions, the best one is first improved by the embedded local search (TS) algorithm (*i.e.* given that the best solution is $\mathbf{x}$ then $N(\mathbf{x})$ is exploited first) before the pheromone trails are updated.

2.6.3 High-level Relay Hybrids (HRH)

In this class, self-contained metaheuristics are executed in a sequence. For example, the initial solution of a given S-metaheuristic may be generated by some other algorithm, and its results fed to yet a different algorithm and so on. Contrary to the low-level hybrids, here the internal workings of each algorithm remain intact.

The *industrial strength COMPASS* (ISC) (Xu *et al.*, 2010) that will be discussed in Section 2.7.3 can be considered as an HRH hybrid. The hybrid, developed to solve large-scale discrete SO problems, uses three self-contained algorithms that operate in a sequential manner, each one contributing with its own strength in order to enhance the overall performance of the hybrid. The process is initiated by a search done by a P-metaheuristic, then a non-metaheuristic exploits the P-metaheuristic results and finally, an R&S procedure is used to ensure correct selection.

2.6.4 High-level Teamwork Hybrids (HTH)

Finally, the HTH class involves several self-contained algorithms performing a search in parallel, and cooperating to find an optimum. The advantage of parallel computing is that it reduces the elapsed time to obtain the same solution as reached with a sequential scheme. Additionally, the technology is also likely to find solutions of better quality than sequential computing (Falco *et al.*, 1997).

In their work, Falco *et al.* (1997) parallelised the SA algorithm using a framework that, according to the researchers, can be used for other metaheuristics as well. The hybrid is based on a set of SA sequential processes arranged in a given topology and on the exchange of good solutions among *neighbouring* SA processes only.

Although the discussion above only used single-objective algorithms as examples, the taxonomy is also applicable to their MOO variants. For more detail and additional references the reader is referred to Talbi (2013).

2.7 Optimisation suites for SO problems

Hybrid metaheuristics are commonly used in optimisation suites for SO problems where they serve as search engines, the principle features of these products. One reason why hybrids are popular with optimisation suites is perhaps the existing need in practice for

as efficient and as effective decision-supporting tools as possible, capable of handling a wide range of problems.

In Amaran *et al.* (2014), the reader can find a comprehensive list of current commercial optimisation suites, including their vendors and the optimisation techniques they utilise. In the following sections, one of the listed products in Amaran *et al.* (2014), namely, *OptQuest*, is briefly discussed. According to Hong & Nelson (2009a), OptQuest is a good representative of optimisation suites in practice. It is a product that is widely used, being integrated into 13 simulation products. Another optimisation suite, a non-commercial this time, will also be discussed.

2.7.1 General discussion on optimisation suites

The goal of optimisation suites is to optimise complex systems, which are those that cannot be easily formulated as mathematical models and solved with classical optimisation tools. Many real-world optimisation problems in business, engineering and science are indeed too complex to be given tractable mathematical formulations (Laguna & Marti, 2003).

Optimisation suites achieve their goal by orchestrating the simulation of a sequence of system designs so that the best design (or designs in the MOO context) is *hopefully* obtained *i.e.* the design that is as close as possible to optimum. This process is referred to, in this study, as the *SO process* and is illustrated in Figure 2.3 (Law & Kelton, 2000). SO processes are generally time consuming. Ideally, it is desirable for an SO process to both spend the least possible amount of computational time, and guarantee convergence in the form of optimality and correct selection. Because of the use of metaheuristics, the former desire is often satisfied while the latter, not so much. In effect, statistical considerations in optimisation suites are often not as rigorous as they should be, especially in the case of commercial optimisation suites. Efforts, however, in the research community are being made to improve this. The industrial strength COMPASS, an academic optimisation suite, tries to provide better statistical considerations in its SO process. For Fu (2002), ideally, the SO process must move from the one in Figure 2.3 to the one in Figure 2.4.

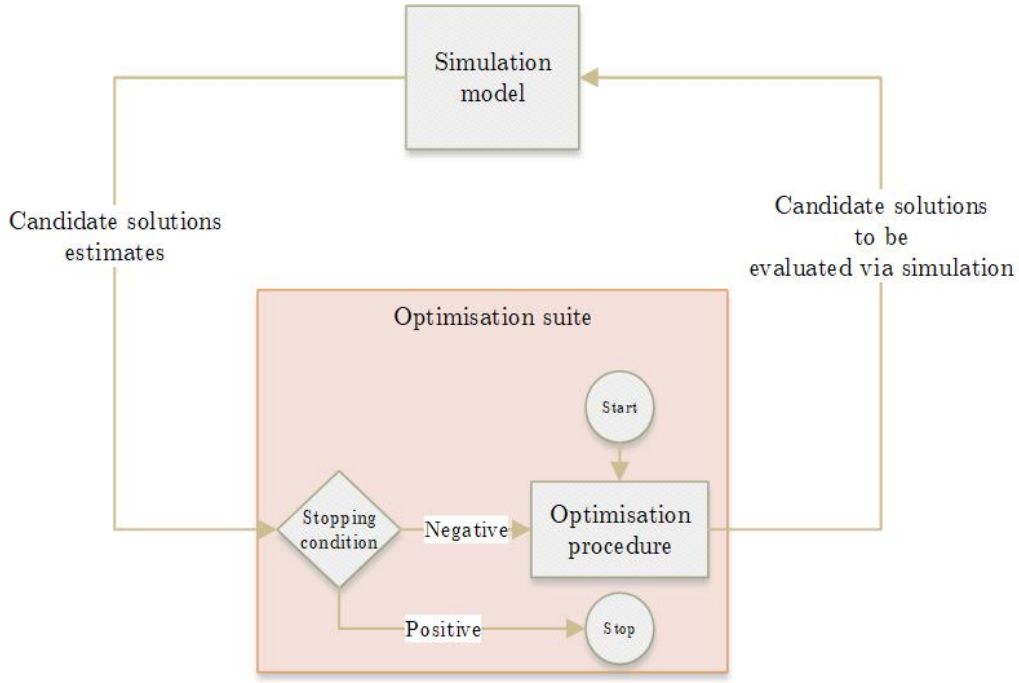


Figure 2.3: A typical SO process.

2.7.2 OptQuest: A commercial suite

The OptQuest suite (Laguna, 2011) is a powerful hybrid metaheuristic optimisation suite that uses a variety of algorithms including the CEM, GA, and more. Though it is not made available to the general public how these algorithms are all orchestrated, many papers do give an explanation of how the suite default algorithm works. OptQuest default algorithm is a hybrid that uses the implementations of three metaheuristics, namely, the *scatter search* (SS), which serves as the main search strategy, the tabu search and the *neural network* (NN).

The SO process works as follows (Eskandari *et al.*, 2011): The SS, which is a P-metaheuristic, generates a starting set of designs and designates a subset of best designs to be reference solutions or points. Then the algorithm forms a linear combination of subsets of current reference points and generates new points. In the next step, the SS algorithm selects a combination of the best solutions, uses them as starting points for a new application of its search mechanism and repeat these steps until a specified number

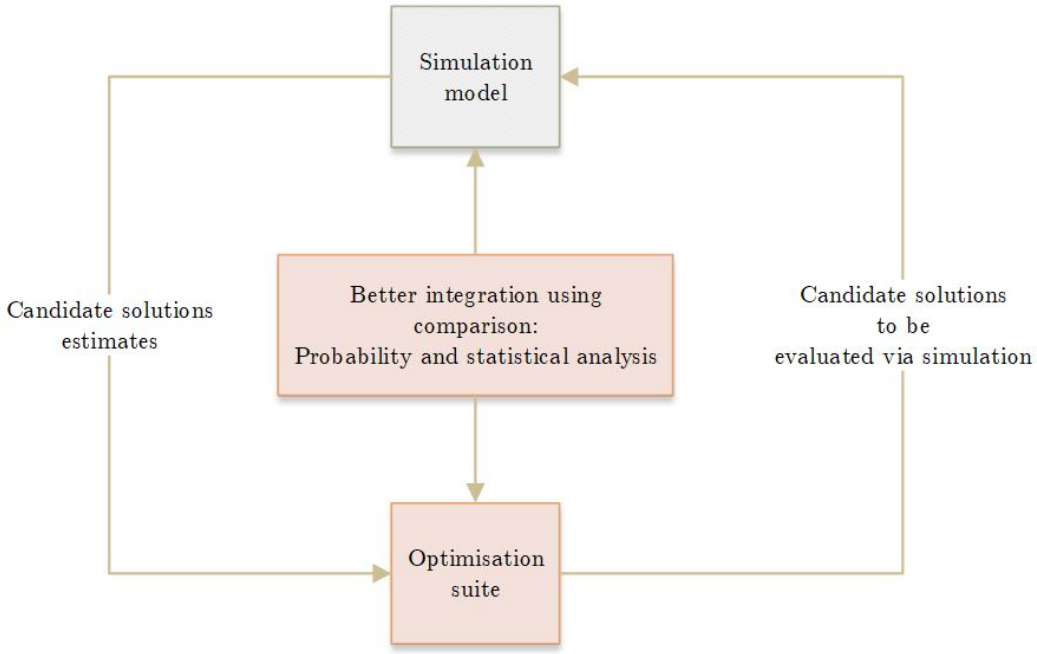


Figure 2.4: SO process future needs.

of iterations or until it reaches a stopping criteria. The TS uses adaptive memory to prohibit the search from re-investigating solutions that have already been evaluated and to guide the search to a globally optimal solution while the NN is used to screen out solutions that are likely to be poor without allowing the simulation to evaluate them (a sort of SAR).

OptQuest has two available stopping rules; one lets the SO process run until a user-specified number of configurations have been completed, while another lets the process run until a user-specified amount of wall-clock time has elapsed. In addition, the user must also specify the population size, which is the number of system configurations simultaneously being considered (Law & Kelton, 2000).

From the author's assessment, the OptQuest default algorithm described here can be classified as a combination of LTH (as TS is embedded in SS) and HRH as the hybrid formed by SS and TS is self-contained and feeds its results to NN, which is also self-contained.

2.7.3 Industrial Strength COMPASS: An academic suite/solver

The industrial strength COMPASS (Xu *et al.*, 2010) divides the optimisation process into three stages: a global stage that explores the entire feasible region and identifies several regions possibly with competitive locally optimal solutions; a local stage that exploits the local information and finds a locally optimal solution for each of the regions identified in the global stage; and a clean-up stage that selects the best solution among all identified locally optimal solutions and estimates the true value of the selected solution.

The ISC hybrid uses a niching genetic algorithm for the global stage, the COMPASS algorithm for the local stage, and a R&S procedure for the clean-up stage. It also defines meaningful and testable transition rules between the stages. As mentioned in Section 2.6.3, it is the author's opinion that the ISC algorithm is an HRH hybrid.

2.8 Chapter summary

In this chapter, the author provided a literature study on simulation optimisation (SO), pertaining to the purpose of this thesis. Specifically, an effort was made to try to cover the major SO solution approaches that are available today.

Multi-objective problems were also discussed and it was discovered that many existing algorithms (metaheuristics mostly) do have MOO variants.

In the next chapter, the author presents a study conducted on Tecnomatix Plant Simulation current SO techniques for both small- and large-scale SO problems.

Chapter 3

Solving SO problems with Tecnomatix Plant Simulation

In this chapter, the current SO capabilities (and limitations) of Tecnomatix Plant Simulation are demonstrated by briefly discussing two problems that have been solved by the author using the software, namely, the mechanised car park (MCP) problem and the buffer allocation problem (BAP).

As it was mentioned in Chapter 1, TPS has been proven to be a powerful tool for conducting complex simulation studies. In the first problem that will be discussed, one of the goals was to demonstrate that such a problem could, indeed, be solved with TPS as the problem had previously been solved using a different software package (Bekker & Viviers, 2008). The goal was successfully achieved and the solution to the problem deemed valid.

Elements in this chapter were first presented in an article co-authored by the author for a conference proceedings (Bamporiki & Bekker, 2017).

3.1 The mechanised car park problem

Parking shortage issues are not new. The mechanised car park concept, however, was a relatively new approach for tackling the issue at the time it was first considered in 2005 (Bekker & Viviers, 2008). The concept was still in its design phase when a study was conducted as a simulation study to evaluate the effectiveness of the approach from a business standpoint. The study was successfully completed and its results are

documented in Bekker & Viviers (2008). In 2016, a similar study was to be conducted, this time on a different and more modern simulation software package and with a new set of goals. The goals of the new study were essentially twofold: To solve the MCP problem again (*i.e.* find operational policies that would "optimise" the system's outputs), and to demonstrate in the process the capabilities of the new simulation software. The study is the subject of this section and is briefly discussed below.

A mechanised car park is an automated parking system that provides parking services whereby the only request made to a client is to park and lock the car at an entrance parking lot on ground level. The car is then taken and stored by the main dynamic mechanisms of the system: a hoist and a vehicle transfer car (VTC). When the client returns, the system retrieves and delivers the car back to its owner using the same mechanism.

The system structure can be described as a parking garage consisting of shelves with closely packed cages (parking bays) where cars are stored. It is a multilevel structure that consists of m horizontal levels and three vertical layers, namely, the front, the back and the middle layer. The back layer is populated with parking bays (PBs). The front layer is similar to the back layer with the only exception that some of its PBs are lost in order to provide for n number of hoist shafts. The middle layer serves as a canal for VTCs and connects the front and back layers. Figure 3.1 illustrates the concept. The n PDs are park-drive areas where the cars are dropped off and retrieved from the MCP by clients whereas the n queue lanes are the different entrances to the MCP where clients wait should the PDs be occupied. (See Figure 3.1.)

The MCP problem, thus, consists of finding efficient ways of operating the system such that the system's outputs (*i.e.* the MCP problem objective functions) are optimised. This is achieved by effectively utilising the system's resources (*i.e.* hoists and VTCs) through a set of operational policies. The goal is to maximise the system's throughput (T_R) *i.e.* the total number of cars stored in the MCP over a period of time, and minimise the system's waiting time (W_T) *i.e.* the average time a client spends in a queue before being serviced. The MCP problem is complex. Its main *modelling* challenge, for example, lies in the fact that the same resources must handle both storage and retrieval tasks, and must thus be capable of telling the difference between the two. Moreover, the system is subject to the stochastic element, ξ, caused by clients' random arrivals and returns to the MCP.

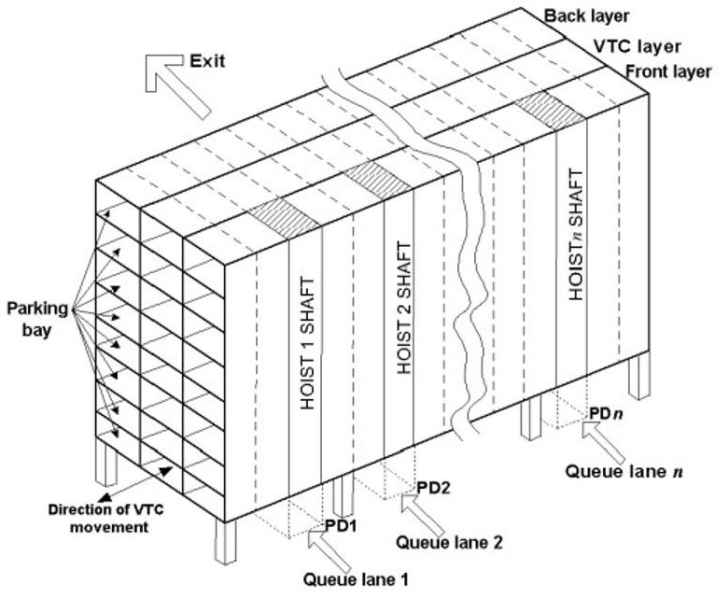


Figure 3.1: Schematic drawing of a mechanised car park (Bekker & Viviers, 2008).

The system designs $\mathbf{x}$ in this problem are, therefore, the different set of policies to be generated by the analyst. $\mathbf{x}$ in this case is a decision vector such that $\mathbf{x} = (x_1, x_2, x_3)$ where x_1 is the *parking bay allocation* (PBA) or storage policy, x_2 is the *entrance lane assignment* (ELA) policy and x_3 is the *priority choice between arrival and departure service* (PAD) policy. These policies are identified to be the main factors capable of influencing the MCP's outputs (*i.e.* $T_R(\mathbf{x}, \xi)$) and $W_T(\mathbf{x}, \xi)$). Since only a relatively small number of system designs can be generated, the MCP problem can be considered as a small-scale SO problem. Furthermore, the set of decision variables in this problem is an example of categorical/qualitative decision variables (see Section 2.2.1).

It is important to mention here that although this problem has two objective functions, it is not a multi-objective problem (as defined in Section 2.1) because the two objectives are not in conflict. In effect, maximising $T_R(\mathbf{x}, \xi)$ equates to minimising $W_T(\mathbf{x}, \xi)$ and *vice versa*.

3.1.1 Solving a small-scale SO problem with Tecnomatix

It is possible to do R&S with TPS using one of the software's statistical tools for output analysis *i.e.* analysis of variance (ANOVA). The software itself does not, directly,

provide for an R&S procedure such as those presented in Chapter 2. However, a simple procedure can be devised using TPS ANOVA. The procedure is briefly described below.

Once all system designs are identified, an initial number of replications r (*e.g.* 10) is used to run the simulation model with a confidence level of 95% (the default value in TPS). TPS then outputs results for all system designs (*i.e.* the estimated objective function values) with their respective *confidence intervals* (CI). If the analyst is satisfied with the largest CI *half-width* (h) observed, then the ANOVA results (which are provided automatically by the software) are used to select the best system design. If, on the other hand, the analyst/user is not happy with h, they reduce it to a desired value h^*. With this, a new number of replications r^* can be calculated using the formula by Law & Kelton (2000):

$$r^* = r \left\lceil \frac{h}{h^*} \right\rceil^2 . \tag{3.1}$$

The simulation model is then run again and the ANOVA results are used to determine the best solution. ANOVA (as done by TPS) does a pairwise comparison between all system designs of interest. The analysis is based on the hypothesis (H_0) that the means of all r or r^* observations made on any two system designs (*e.g.* μ_i and μ_j, $i \neq i$) being compared are equal (*i.e.* $H_0 : \mu_i = \mu_j$, $i \neq j$). A probability (p) is then calculated to indicate evidence against this hypothesis. If p is equal or smaller than a certain threshold value (often 5%) then this is considered strong evidence against the hypothesis and the two system designs being compared are said to differ statistically significantly. Otherwise, they are said to be, statistically-wise, identical. With this information, the analyst can then select the best solution with confidence. In cases where more than one objective function is being considered, a separate ANOVA is made for each. The result of ANOVA in TPS is presented in a table format containing all p-values, from which the analyst must make a selection based on the observed p-values.

3.1.2 Specifics of the MCP problem solved

In the MCP problem that was solved with TPS, the number of levels was $l = 11$ and the number of hoist shafts was $n = 6$. In total, the system had 440 PBs with 40 PBs in each level.

10 PBA, 3 ELA and 1 PAD (first come first served) policies were generated, amounting to an overall of 30 system designs that were evaluated for efficiency. It is important to note that during the study, it was found that PBA policies, in general, had way more influence on the system's outputs than ELA policies. The approach that was used to generate PBA policies will now be discussed briefly.

It was discovered that the MCP could be viewed as a matrix (Figure 3.2) with each cell representing a PB. Thus, various ways of searching this matrix for available PBs could be created with TPS, taking into account the limited number of resources (*i.e.* hoists and VTCs). Ten search patterns were created to look for unoccupied PBs in the matrix (PBA1-PBA10). These were combined with three ELA policies (ELA1-ELA3) to generate a total of 30 system designs. All system designs used a first come, first served approach with regards to the PAD policy.

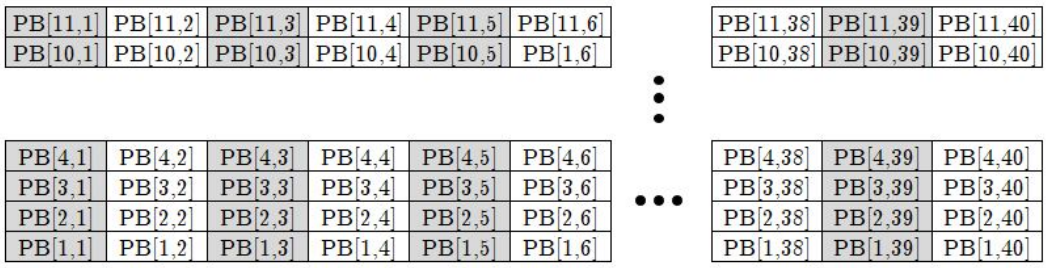


Figure 3.2: Schematic view of the MCP as a matrix.

3.1.3 Results and limitations

An extract of the actual results that were obtained for the problem is presented here and the current limitations of the software are discussed.

Table 3.1 presents the top nine results that were obtained. It can be seen that some of these results are very close to each other. This is, in effect, where R&S procedures draw their importance as these numbers are mere estimates. The closer their true values, the higher the chances that the observed better one is not the actual better one (Teng *et al.*, 2010). For example, though System design 1 appears to be numerically the best, we learn from Table 3.2 that actually, the difference between System designs 1 and 2 is not statistically significant. Thus, the analyst can either choose to go for a higher r (*i.e.* a lower h), or, because the desired h (*i.e.* h^*) was already selected

Table 3.1: Top nine results for the MCP problem.

System design	PBA Policy	ELA Policy	T_R	W_T
1	3	1	5775	04:58.4
2	3	2	5766	05:02.2
3	3	3	5751	05:08.8
4	5	1	5171	08:17.9
5	5	3	5168	08:20.7
6	5	2	5153	08:25.2
7	9	3	4812	08:42.3
8	9	1	4804	08:42.3
9	9	2	4795	08:46.3

Table 3.2: T_R ANOVA results.

System design	2	3	4	5	6	7	8	9
1	0.089	0	0	0	0	0	0	0
2		0.008	0	0	0	0	0	0
3			0	0	0	0	0	0
4				0.534	0.003	0	0	0
5					0.015	0	0	0
6						0	0	0
7							0.241	0.013
8								0.198

in this case, be indifferent in choosing between the two designs (with regards to T_R). Note that a similar table to Table 3.2 also exists for W_T and may give the analyst more information about Designs 1 and 2. For example, the W_T ANOVA results may indicate a significant difference between Designs 1 and 2, in which case, the analyst would no longer be indifferent in choosing between them but rather choose the better of the two.

A major drawback in this R&S procedure is that r^* replications are allocated to all system designs. Although this guarantees that all designs have h values that are within the desired h^*, the approach is not efficient as it can be computationally expensive. (This is an example of a procedure that uses the LFC assumption mentioned in Section 2.4.1.1). There are more efficient techniques in the literature (see Section 2.4.1) that

have the ability to intelligently allocate simulation replications among system designs while still maintaining the desired confidence level. Moreover, even though one may consider the results obtained here as good, it follows from the work done by Bechhofer (1954) that ANOVA is not the ideal approach for R&S purposes. (See Section 2.4.1.)

In the next subsection, the second problem is discussed. The problem demonstrates Tecnomatix Plant Simulation capabilities but also limitations in handling large-scale SO problems, especially those with multi-objectives.

3.2 The buffer allocation problem

Production systems are often organised with machines connected in series and separated by buffers. This arrangement is often called a flow line or a production line (Gershwin & Schor, 2000). The buffer allocation problem is a well-known problem in the design of production lines.

The basic setting of the BAP is the following (Bekker (2012), Alon *et al.* (2005)). Consider a production line consisting of m machines in series, numbered $1, 2, ..., m$. Jobs are processed by all machines in sequential order. The processing time at machine i has, generally, a fixed distribution with rate μ_i, $i = 1, 2, ..., m$. The machines are assumed to be unreliable and their failures are operation dependent failures (ODFs) with Poisson distributions having parameters λ_i while their repair times are exponential with parameters β_i, $i = 1, 2, ..., m$. All processing, failure and repair times are assumed to be independent of each other.

The machines are separated by $m-1$ storage areas or niches in which jobs *i.e. work-in-progress* (WIP) can be stored (Figure 3.3). The total number of storage spaces, or buffer spaces, is unknown and must be minimised and the required number of buffer spaces can be determined by estimating the WIP.

Figure 3.3: A typical series of m machines with $m-1$ niches.

When a machine breaks down, this can have consequences for other machines upstream or downstream in the production line. Particularly, an upstream machine can become blocked when its successor has failed, while a downstream machine can eventually become starved if its predecessor has failed.

The objective functions in this case are the throughput rate $T_R(x, \xi)$ (to be maximised) and WIP (to be minimised), denoted by $W_P(x, \xi)$. The values of the objective functions are estimated by means of simulation models of the BAPs as the system is subject to the stochastic element, ξ, caused by the machines' failures as well as repair and processing times. Also, because these objectives are conflicting, the problem is a multi-objective simulation-optimisation problem and can be formulated as

$$\text{Minimise } (E[-T_R(\mathbf{x}, \xi)], E[W_P(\mathbf{x}, \xi)])^T$$
$$\text{Subject to}$$
$$\mathbf{x} \in \mathbb{N}.$$

$\mathbf{x} = (x_1, ..., x_{m-1})$ is the decision vector, where x_i is the number of buffer spaces at niche i. Because x_i is a discrete number and there are potentially an infinite number of alternatives for $\mathbf{x}$, this BAP is a combinatorial optimisation problem and thus has a large solution space, so it is a large-scale SO problem.

3.2.1 Solving a large-scale SO problem with Tecnomatix

TPS uses a built-in optimisation suite/solver that enables SO. The suite uses a genetic algorithm metaheuristic for the task. Though there are many variants of the GA in the literature, their core principle is similar. The author briefly explains how GAs work from Konak *et al.* (2006).

GAs are inspired by the biological principle of evolution with the survival of the fittest being a fundamental property. In nature, weak and unfit members of species within their environment are faced with extinction by natural selection while stronger ones have greater chances of surviving by passing their genes on to future generations via reproduction. GAs base their search mechanism on these principles.

In the GA terminology, $\mathbf{x} \in \mathbb{N}$ is called a chromosome (Konak *et al.*, 2006). Chromosomes are made of units called genes. In many variants of the GA, genes are binary

digits and chromosomes correspond to unique $\mathbf{x} \in \mathbb{N}$. Thus, the actual values of $\mathbf{x}$ are encoded in binary form during the search process and decoded again to evaluate the fitness of the results found (*e.g.* $TR(\mathbf{x}, \xi)$). GAs operate iteratively with a collection of chromosomes, called *population*. In every iteration, fitter chromosomes exchange their genes with one another to form new chromosomes that are carried in the following iteration. In this way, future populations have progressively fitter chromosomes as the search progresses. When the search is carried out for long enough, convergence usually occurs, meaning that one chromosome has achieved a level of fitness that can hardly be improved. An iteration in GA terms is also referred to as a *generation*.

When making use of the optimisation suite, the analyst must specify the desired number of generations as well as the size of a population via the suite user-interface (the *GAWizard*). Also, the analyst specifies the objective functions of interest, the optimisation direction (min or max), the solution space to be searched and the desired number of observations r^* to be applied to each chromosome that must be evaluated. Note that unlike in the previous problem, here r^* is selected "blindly".

For multi-objective problems, TPS GA does not use the Pareto approach. Instead, MOO problems are treated as single-objective problems by using the weighting sum approach (a scalarisation method) (see Marler & Arora (2004)). The solution found by the optimisation suite is therefore a single solution and not a Pareto set.

3.2.2 Specifics of the BAP solved

In the BAP that was solved with TPS, the number of machines was $m = 5$. All repair and processing times were assumed to be exponentially distributed whereas machine failures followed a Poisson distribution. Table 3.3 summarises the information about all machines.

Table 3.3: Machines information for the BAP.

Machine	**1**	**2**	**3**	**4**	**5**
Processing times (μ_i)	60 min	55 min	50 min	46 min	43 min
ODFs (λ_i)	20	20	20	20	20
Repair times (β_i)	120 min	120 min	120 min	120 min	120 min

The input parameters in the optimisation suite were as follows. The optimisation direction was chosen to be *maximisation* and the objective functions were, consequently,

entered as $T_R(\mathbf{x}, \xi)$ and $-W_P(\mathbf{x}, \xi)$. For the purpose of the experiment, both objectives were given weights indicating that they were equally important to the analyst (*i.e.* 0.5). (Note that the selected weights must add up to 1.) The number of generations was chosen to be 20 while the population size was made 20. To create the feasible solution space X, x_i was constrained such that $1 \leq x_i \leq 20$ and r^* was made, arbitrarily, 15.

3.2.3 Results and limitations

The single solution obtained by the suite was $\mathbf{x} = (6, 6, 4, 2)$ with $T_R = 85.77$ and $W_P = 18$.

To test the quality of the result obtained, the model was run further with more constrained feasible solution spaces while using the same weights. The results obtained are summarised in Table 3.4.

Table 3.4: GA solutions for more constrained solution spaces.

Solution space	**Solution obtained**	T_R	W_P
$1 \leq x_i \leq 15$	$\mathbf{x} = (5, 5, 3, 2)$	85.26	15
$1 \leq x_i \leq 10$	$\mathbf{x} = (6, 8, 5, 3)$	85.63	22
$1 \leq x_i \leq 8$	$\mathbf{x} = (6, 7, 5, 3)$	85.97	21
$1 \leq x_i \leq 5$	$\mathbf{x} = (5, 5, 3, 2)$	85.26	15

Comparing the first solution obtained to those in Table 3.4, one can easily observe that increases in total buffer spaces (W_P) do not change, essentially, the throughput values. Thus, for practical purposes, it would be ideal for a decision-maker to know about the solutions in Table 3.4, especially those with better W_P results then the original solution. Observe also that except for the solution at solution space $1 \leq x_i \leq 10$, every other solution in Table 3.4 is not dominated by the original solution obtained.

Moreover, because in the original run the solution space was the largest, it is possible that at some point during the search mechanism, the algorithm evaluated solutions in Table 3.4. Assuming that it did, why output the original solution and not, for example, the solution at solution space $1 \leq x_i \leq 15$? which may arguably be a better option. And assuming that it did not, then this is a problem that Branke *et al.* (2008) have predicted for MOO algorithms that use a weighting sum method when faced with a

"non-convex problem". And it is not often easy to check for convexity in the case of SO problems (Branke *et al.*, 2008).

Additionally, the author also ran the BAP model with different weight selections while keeping the solution space size unchanged (*i.e.* $1 \leq x_i \leq 20$). The experiment is summarised in Table 3.5.

Table 3.5: BAP results for different weights selection.

T_R weight	W_P weight	**Solution**	T_R	W_P	Note
0	1	NA	NA	NA	Negative fitness value found
0.2	0.8	NA	NA	NA	Negative fitness value found
0.4	0.6	(5, 5, 4, 2)	63.72	16	
0.5	0.5	(6, 6, 4, 2)	85.77	18	
0.6	0.4	(10, 8, 7, 4)	108.32	29	
0.8	0.2	(13, 10, 9, 7)	155.93	39	
1	0	(18, 11, 9, 12)	205.46	50	

Note that all the solutions in Table 3.5 are non-dominated. Also note that the optimisation procedure was unable to handle negative fitness values while processing, in which case it returned an error message and, consequently, no results.

Assuming that the original solution obtained by the suite is Pareto-optimal (which it is supposed to be), using an MOO algorithm with the Pareto approach would, in principle, output all the non-dominated solutions in Tables 3.4 and 3.5; hence, giving insight to the decision-maker on the quality of potential solutions as well as a wider range to choose from. But as will be shown in Chapter 7, the original solution obtained in this section is actually not Pareto-optimal; meaning that this BAP is probably a non-convex problem.

Furthermore, it can also be seen that there are no statistical considerations (R&S) involved when dealing with large-scale SO problems in TPS. This is another drawback. In Fu *et al.* (2000), the researchers state that statistics must come into play (when solving large-scale SO problems) if any convergence results are to be rigorously established for search algorithms.

3.3 Chapter summary

In this chapter, the author demonstrated the current capabilities and limitations of TPS with respect to both small and large-scale SO problems in the MOO context.

The goal of this thesis is to develop a product that will allow TPS to deal with MOSO problems better than it currently does. As far as the author is aware (see Chapter 2), there is at present no optimisation suite that uses MOO algorithms with the Pareto approach and rigorous statistical techniques (R&S) in solving large-scale MOSO problems, not even OptQuest. ISC does have R&S, but ISC does not do MOO.

The product to be developed in this thesis (in the succeeding chapters), and its solution approach to MOSO problems, is thus a step forward towards achieving the ideal sought by Fu *et al.* (2000) (Figure 2.4) in the MOO context.

Chapter 4

Solution architecture and selected algorithms

Existing solution techniques for SO problems in the literature; and how a simulation software package, namely, Tecnomatix Plant Simulation, utilises some of them in the MOO context were studied in Chapters 2 and 3, respectively. The results obtained in Chapter 3 showed that TPS SO capabilities were limited and could be improved.

The goal in this thesis is to develop a solution approach, in the form of an optimisation suite that addresses TPS limitations, as illustrated in Chapter 3. Specifically, the goal is to develop an optimisation suite that uses a metaheuristic approach that deals with MOSO problems more effectively; that is, a metaheuristic approach that utilises the Pareto approach. Moreover, the goal is also to provide the optimisation suite with a rigorous statistical R&S technique that can be used in both small- and large-scale MOSO context.

In this chapter, the author presents and describes the architectural design of the solution approach as well as the algorithms that were selected for the optimisation suite to be developed.

4.1 Solution architecture

The author desired the optimisation suite to be accessible from TPS as a third-party library containing the necessary functions (*i.e.* the solution algorithms) that would control the SO process of a user-defined MOSO problem of the form of the framework

discussed in Section 2.3. The author also wanted to make the suite a stand-alone, callable module that would link with TPS at run time in a similar way as OptQuest. The suite to be developed was named the *multi-objective optimisation solver* library, shortened as the MOOSolver library or simply MOOSolver. Similar to the TPS current optimisation suite and similar to OptQuest, the user would provide MOOSolver with the necessary parameters of the simulation model to be solved. The flow diagram in Figure 4.1 illustrates a high-level architectural design of the concept and shows the inter-process communication between MOOSolver and TPS.

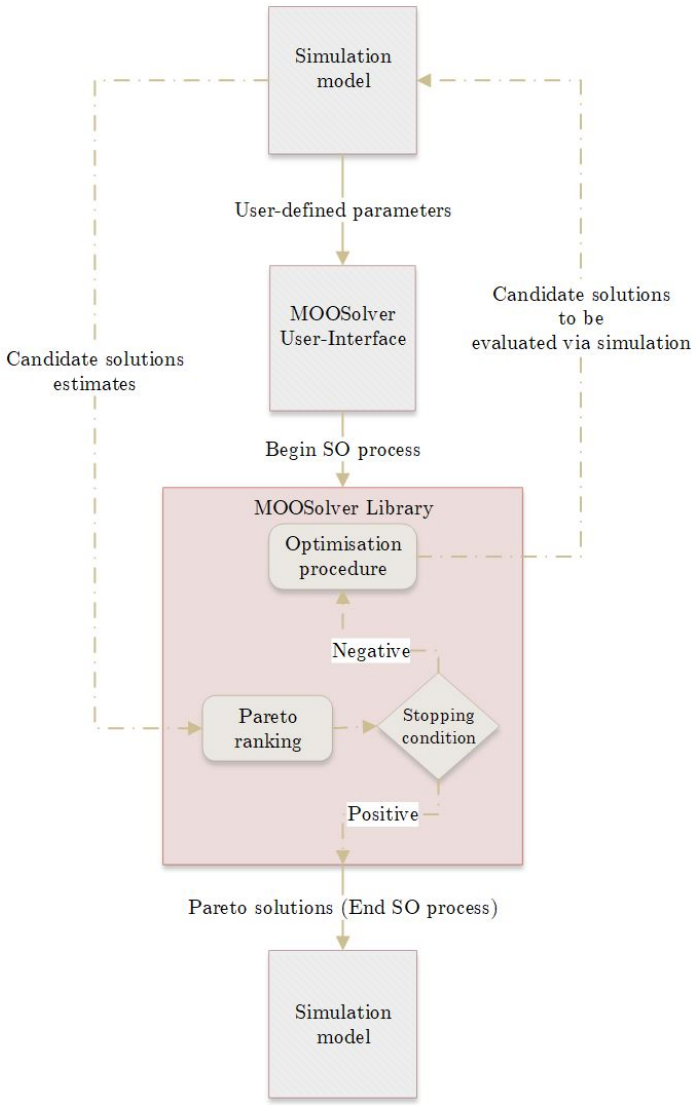


Figure 4.1: Architectural design of the SO process for TPS using MOOSolver.

4.1.1 Large-scale approach

As was suggested by Fu *et al.* (2000) regarding future developments of optimisation suites (see Section 2.7), MOOSolver will attempt to solve large-scale MOSO problems by including rigorous statistical analysis in the SO process. This, however, will be done as a hybrid approach of type HRH (Section 2.6.3). In other words, the metaheuristic responsible for the search mechanism and the R&S responsible for the statistical analysis are to be self-contained and the results of the metaheuristic are to be "fed" to the R&S procedure after the search process is complete. For reasons that will be discussed in Section 4.2.2.2, the process was made an interactive one with the user involved in selecting the systems to be fed to the R&S procedure once the search process is complete.

The author believes that, ideally, the hybrid SO process should be of type LRH (Section 2.6.1), in other words, using the R&S procedure as an integral part of the metaheuristic (an approach that is beyond the current scope of this work as it entails tampering with the inner workings of the algorithms). However, using the HRH approach in an interactive manner turns out to have its own advantages. In effect, giving the user the opportunity to first see all the good solutions found by the metaheuristic before the R&S procedure is used, allows the user to select from the approximate Pareto set (which can be overwhelmingly large sometimes), solutions or systems that they may have biased interest in. Only these preferred solutions are then fed to the R&S procedure for further statistical analysis. This approach (using R&S on the user-preferred solutions) can save a tremendous amount of computational time in practice as R&S procedures can take long to process. In fact, the computational time factor is a challenge that the author anticipates for an LRH approach *i.e.* how does one practically integrate R&S in an MOO metaheuristic? But as already mentioned, this is beyond the scope of the present work.

The interactive HRH approach proposed in this thesis, hence reduces the large-scale problem to a small-scale one that uses preselected solutions that are known to be good. Because the metaheuristic cannot guarantee the statistical soundness of the approximated Pareto set, the R&S is used to *clean* the potential "noise" in the user-preferred systems by providing better estimates and guaranteeing correct selection. In

effect, some solutions that present themselves as members of the Pareto set may no longer be, after running the R&S procedure.

When solving a large-scale problem with MOOSolver, therefore, the user would provide the following parameters to the optimisation suite:

1. The number of decision variables.
2. The simulation model decision variables as well as the nature of the decision variables.
3. The decision variables' limits that will create a feasible solution space for the problem.
4. The objective functions as well as the optimisation direction of each objective.

   Note that unlike the current TPS optimisation suite, MOOSolver will incorporate the duality principle in its MOO algorithms so that the user would not have to do it manually (see Section 3.2).
5. The number of observations to be used per solution during the search mechanism.
6. And finally, the algorithms' parameters (for both the metaheuristic and the R&S procedure), which will be discussed in Section 4.2.

4.1.2 Small-scale approach

MOOSolver will solve small-scale MOSO problems using a modern, more effective and more efficient indifference-zone based R&S procedure. The procedure is presented in Section 4.2.2.

The SO process will require the user to provide the suite with the following parameters:

1. The system designs or scenarios to be compared.
2. The initial number of observations.
3. The indifference-zone values of the respective objective functions.

4. And finally, just as in the previous section, the decision variables, the number of decision variables, the objective functions as well as their respective optimisation directions.

In what follows, the algorithms that were selected for MOOSolver are presented and described in great detail. The information given in the succeeding section is important for the effective use of the final product to be developed.

4.2 Selected algorithms

For the purpose of this project, two algorithms were selected for the suite. One for the search mechanism and the other for rigorous statistical R&S analysis.

The selected search algorithm is the cross-entropy method for multi-objective optimisation (MOO CEM) (Bekker, 2012) while the statistical, ranking and selection algorithm is the $\mathcal{MMY}$ procedure (Yoon, 2018). MOO CEM is believed to be a good search mechanism for large-scale MOSO problems which are often computationally demanding. MOO CEM is, in effect, known to be a relatively fast converging metaheuristic (Bekker, 2012). $\mathcal{MMY}$, on the other hand, is believed to be an effective and an efficient modern R&S procedure, ideal for MOSO problems.

4.2.1 The MOO CEM metaheuristic

The MOO CEM is a multi-objective adaptation of the CEM metaheuristic described in Section 2.5.1.3. The algorithm was developed by Bekker (2012). The author dedicates this section to the full description of the MOO CEM as it is the principal algorithm of the MOOSolver library. The description is based on the works of Bekker & Aldrich (2011).

The algorithm adapts the CEM for MOO problems by using a number of key concepts. The first one is that of a *Working matrix*. The MOO CEM metaheuristic uses a Working matrix that consists of N rows and $n+m+1$ columns, where N is an arbitrary number of solutions (*i.e.* the population size), n is the number of decision variables (DVs) and m is the number of objectives. Sample values of the first DV are stored in column 1, the second DV in column 2, and so on up to column n. The objective function values for objective 1 are stored in column $n+1$, for objective 2 in column

$n+2$, and for objective m in column $n+m$. The last column is used to store the Pareto ranking value of each solution. The structure is shown in Table 4.1.

Table 4.1: Structure of the working matrix.

Decision variables				Objectives				Rank
x_{11}	x_{12}	$\cdots$	x_{1n}	f_{11}	f_{12}	$\cdots$	f_{1m}	ρ_1
$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
x_{N1}	x_{N2}	$\cdots$	x_{Nn}	f_{N1}	f_{N2}	$\cdots$	f_{Nm}	ρ_N

The metaheuristic generates new populations by creating a sequence of probability density functions (pdfs) for each DV in every iteration (this is the "specified" mechanism alluded to in phase one of the CEM in Section 2.5.1.3). To form a sample vector $\mathbf{x}_i$ from a pdf $h_i(\cdot;\hat{\mathbf{V}}_{t-1})$, a truncated normal distribution is used for each decision variable. For the n DVs defined over ranges $[l_i, u_i]$, l_i is the lower limit and u_i the upper limit of DV $x_i, 1 \leq i \leq n$. The truncated normal distribution h_{Ti}, defined in the range $[l_i, u_i]$ and having mean μ_i and variance σ_i^2, is given by

$$h_{Ti}(x) = \begin{cases} 0, & x < l_i, \\ \frac{h_n(x)}{\int_{l_i}^{u_i} h_n(x)d(x)}, & l_i \leq x \leq u_i, \\ 0, & x > u_i. \end{cases}$$

The function $h_n(x)$ is the normal pdf defined on $-\infty < x < \infty$.

Using truncated distributions makes it easy to contain the search. As required by the CEM, an arbitrarily large value for σ_i (*i.e.* the standard deviation) is initially assigned, using $\sigma_i = 10 \cdot (u_i - l_i)$. The first n columns of the working matrix are filled with sample values from each applicable truncated normal distribution.

Next, each of the objective functions is evaluated using the row vectors $X_{1i}, ..., X_{Ni}$ of Table 4.1. This yields two or more performance vectors $f_j(\mathbf{x})$ with $1 < j \leq m$.

The best combinations of objective functions are found by doing a Pareto-ranking using Algorithm 1 in Section 2.1.

The values of the decision variables in the non-dominated set (*i.e.* the *elite matrix*) provided by Algorithm 1 are used to construct a *histogram* for each decision variable. The histogram concept is the second key concept used to adapt the CEM into a MOO algorithm. The histograms provide guiding information for the MOO CEM algorithm and are maintained while the algorithm is searching for non-dominated solutions.

The concept works as follows: for a decision variable that is defined in the range $[l_i, u_i]$, the lower boundary of the first class is set equal to l_i, and the upper boundary of the last class is set equal to u_i. Next, the upper boundary of the first class is set equal to the minimum value of the decision variable x_i in the elite matrix, *i.e.* $min(Elite(\cdot, i))$. The lower boundary of the last class is equal to the maximum value of the decision variable in the elite matrix, namely $max(Elite(\cdot, i))$. A number of equal-sized classes are formed between these two boundaries using $(max(Elite(\cdot, i)) - min(Elite(\cdot, i)))/r$ if r of these classes are formed, resulting in a total number of $r + 2$ classes (see Figure 4.2).

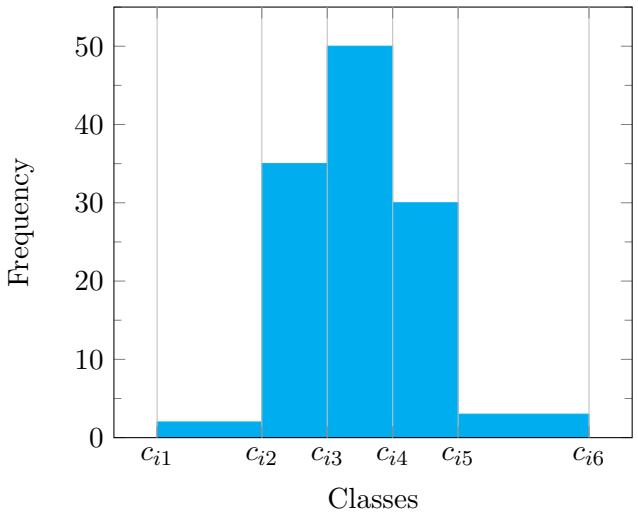


Figure 4.2: Example of a histogram for decision variable x_i and $r = 3$.

The class boundaries for the histogram of decision variable x_i are recorded in a vector $\mathbf{C}_i = \{c_{i1}, c_{i2}, ..., c_{i(r+2)}, c_{i((r+2)+1)}\}$, with $c_{i1} = l_i$ and $c_{i((r+2)+1)} = u_i$. Note that $\mathbf{C}_i$ contains $r + 3$ elements because the histogram has $r + 2$ classes, and that the class widths of the first class ($[c_{i1}, c_{i2}]$) and the last class ($[c_{i(r+2)}, c_{i((r+2)+1)}]$) can be different from each other and from the widths of the r classes.

The elite matrix has the same columns as the working matrix shown in Table 4.1, and the values in column $i, 1 \leq i \leq n$ are used to determine frequency values for decision variable x_i. The decision variable values are classified according to the following rule: x_{ij} belongs to the class $[c_{ik}, c_{i(k+1)})$ if $c_{ik} \leq x_{ij} < c_{i(k+1)}$, $1 \leq k \leq r+2$. The histogram frequency values are recorded in a vector $\mathbf{R}_i = \{\tau_{i1}, \tau_{i2}, ..., \tau_{i(r+1)}, \tau_{i(r+2)}\}$, where τ_{i1} is

equal to the frequency count of decision variable x_i in the range $[c_{i1}, c_{i2})$, τ_{i2} represents the count in range $[c_{i2}, c_{i3})$, and so on.

In preparation for the next iteration of the algorithm, the new population of possible solutions is formed proportionally according to the class frequencies for each decision variable: suppose the elite matrix contains E_r rows and there are τ_{ik} occurrences in class $[c_{ij}, c_{i(j+1)})$ for a given decision variable x_i, then $\lfloor N\tau_{ik}/E_r \rfloor$ values will be created from this class range for this variable (the population size is N and $1 \leq k \leq r+2$). When the proportional numbers do not add up to N due to the rounding down of the proportion calculation, the small difference is arbitrarily added to the last class. When generating observations from a class range $[c_{ij}, c_{i(j+1)}]$, temporary μ'_{ik} and σ'_{ik} values are used. These values are associated with the specific histogram class ranges, so for the class $[c_{ik}, c_{i(k+1)})$ for decision variable x_i, the parameter estimators are $\mu'_{ik} = c_{ik} + U(c_{i(k+1)} - c_{ik})$, whereas $\sigma'_{ik} = (c_{i(k+1)} - c_{ik})$, $1 \leq k \leq r+2$ and U is a uniformly distributed random number.

To prevent premature convergence, the histogram frequencies are adjusted during each iteration t with a preset probability of typically $0.1 - 0.3$. To do so, the maximum frequency over all classes is determined for a given decision variable. The frequency in each class is then subtracted from this value, resulting in an inverted histogram as shown in Figure 4.3. This ensures that search ranges that were given small proportions of population candidate allocations receive higher proportions of allocations, while search ranges with high proportions of population allocations receive fewer allocations after frequency inversion.

The algorithm will readjust the frequencies according to the rankings returned by the candidates so that a class that does not contribute to the elite matrix effectively becomes eliminated as the search progresses. The histogram concept also allows for the accommodation of discontinuous search spaces.

To ensure exploitation in the MOO context, the process described above is repeated o_l times as an *outer loop* of the algorithm. This is the third concept used to adapt the CEM for MOO. After each loop, the elite matrix is ranked again and the number of classes of the histograms is incremented. Increasing the number of classes as the search progresses makes it possible to maintain good combinations of decision variable values as the resolution of the decision variable spaces becomes finer.

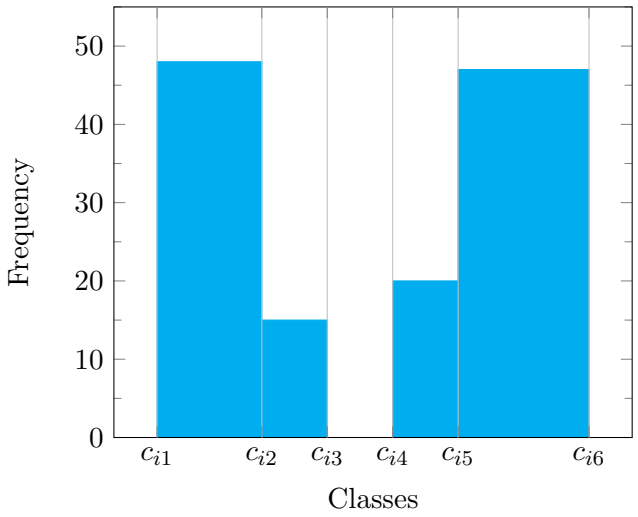


Figure 4.3: The inverted histogram of Figure 4.2.

The parameter vectors (μ_i, σ_i) are updated using (4.1) and (4.2) respectively, and the values in the DV columns of the elite matrix.

$$\hat{\mu}_{i,t} = \alpha_m \tilde{\mu}_{i,t} + (1 - \alpha_m)\hat{\mu}_{i,t-1} \tag{4.1}$$

$$\hat{\sigma}_{i,t} = \alpha_m \tilde{\sigma}_{i,t} + (1 - \alpha_m)\hat{\sigma}_{i,t-1} \tag{4.2}$$

where t is the iteration count index and $\alpha_m = 0.7$ in all cases. This process is continued until the σ_i-value of each decision variable has decreased below a common threshold *epsilon* (ϵ). On algorithm termination, the elite matrix should contain the solutions members of the Pareto set, as well as their associated objective functions values.

To support exploration and exploitation of the search, the initial ranking threshold t_h is relaxed and a value of two is selected. When a new loop starts and a new population is formed, the elite matrix is trimmed and the threshold is set to one. When the algorithm terminates, the existing elite matrix is refined a final time, and the threshold then used is zero, which means all solutions selected are non-dominated. This constitutes the fourth and final concept. The algorithm is presented in pseudo-code as Algorithm 9.

Algorithm 9 MOO CEM metaheuristic

1: Set *Elite* $= \emptyset$, $t = 1$, $o_l = 1$.
2: Initialise decision variable vectors $\mathbf{x}_i = \emptyset, 1 \leq i \leq n$.
3: For each decision variable x_i, $1 \leq i \leq n$, initialise a histogram class vector $\mathbf{C}_i = \{c_{i1}, c_{i2}, ..., c_{i(r+2)}, c_{i((r+2)+1)}\}$ and histogram frequency vector $\mathbf{R}_i = \{\tau_{i1}, \tau_{i2}, ..., \tau_{i(r+1)}, \tau_{i(r+2)}\}$.
4: Set $i = 0$.
5: Set $k = 0$.
6: Increment k.
7: Do for frequency element τ_{ik} in $\mathbf{R}_i$.
8: Generate a class-based $\tilde{\mathbf{v}}'$ in the range $[c_{ik}, c_{i(k+1)})$, $1 \leq k \leq r + 2$.
9: Generate a subsample $\mathbf{y}$ according to the pdf $h_{Ti}(\mathbf{x}_i, \tilde{\mathbf{v}}')$ with $\mathbf{x}_i \in [c_{ik}, c_i(k + 1))$ and $|\mathbf{y}| = \tau_{ik}$, $1 \leq k \leq r + 2$.
10: Append $\mathbf{y}$ to $\mathbf{x}_i$.
11: If $k < r + 2$ return to Step 6.
12: Increment i.
13: If $i \leq n$, return to Step 5.
14: Compute the Nm performance values using $\mathbf{x}_i$, $1 \leq i \leq n$.
15: Rank the performance values using the Pareto ranking of Algorithm 1 with a relaxed $t_h = 2$ to obtain an updated elite matrix *Elite*.
16: Form new histogram class vectors $\mathbf{C_i}$ and histogram frequency vectors $\mathbf{R_i}$ based on *Elite*, $1 \leq i \leq n$.
17: Use the values in *Elite* and compute $\tilde{\mathbf{v}}_{it}$ for all $1 \leq i \leq n$.
18: Smooth the vectors $\tilde{\mathbf{v}}_{it}$ for all $1 \leq i \leq n$, using (4.1) and (4.2).
19: If all $\sigma_{i,t} > \epsilon$ or less than the allowable number of evaluations has been done, increment t and reiterate from Step 4.
20: Rank the elite matrix using the Pareto ranking of Algorithm 1 with $t_h = 1$.
21: Increment o_l.
22: If o_l is less than the allowable number of loops, return to Step 2.
23: Rank the elite matrix using the Pareto ranking of Algorithm 1 with $t_h = 0$ to obtain the final elite matrix.

4.2.2 The $\mathcal{MMY}$ procedure

Procedure $\mathcal{MMY}$ is the multi-objective variant of the $\mathcal{MY}$ procedure presented in Section 2.4.1.1. It is the first MOO R&S procedure with the indifference-zone approach in the literature, that guarantees correct selection following the Bayesian probabilistic theory (Section 2.4.1.1). The algorithm was selected to equip MOOSolver with a modern, efficient multi-objective R&S procedure.

Table 4.2 provides the notation used in procedure $\mathcal{MMY}$, which is presented in Algorithm 10.

Table 4.2: Notation for procedure $\mathcal{MMY}$.

M	the number of systems in the problem;
S	the feasible solution set, *i.e.*, $S = \{1, ..., M\}$;
I	the set of systems that are still in competition;
H	the number of objectives;
K	the objective set, *i.e.*, $K = \{1, ..., H\}$;
N_i	the total number of simulation replications assigned to system i;
$\bar{X}_{ik}(N_i)$	the sample mean of system i for objective k based on N_i observations;
S_p	the observed Pareto set based on $\bar{X}_{ik}$ ($i \in S$ and $k \in K$);
S_p^c	the observed non-Pareto set based on $\bar{X}_{ik}$ ($i \in S$ and $k \in K$);
n_0	the number of simulation replications at the first stage;
δ_k^*	the indifference-zone value for objective k;
P^*	the minimum required value for P(CS).

Following are some definitions used in the procedure. Let

$$\delta_{ijk} = \max\{\delta_k^*, \bar{X}_{jk}(N_j) - \bar{X}_{ik}(N_i)\} \tag{4.3}$$

and $\lceil x \rceil$ denotes the smallest integer greater than x. Consider a pair of systems (i, j) where system i is observed as non-dominated and system j can be any other system in S. This pair (i, j) ($i \in S_p$ and $j \in S, j \neq i$) is relevant to Steps 4 and 5 in Algorithm 10.

For each pair (i, j) ($i \in S_p$ and $j \in S, j \neq i$), let

$$K_1 = \{k \mid |\bar{X}_{jk} - \bar{X}_{ik}| \leq \delta_k^*, k \in K\} \tag{4.7}$$

Algorithm 10 Procedure $\mathcal{MMY}$

1: Select the probability of correction requirement $P^* = 1 - \alpha$, the indifference-zone value δ_k^* for each objective $k \in K$, and the first-stage sample size $n_0 \geq 10$. Set $I = \{1, 2, ..., M\}$ and $\beta = \frac{\alpha}{M}$.

2: Simulate n_0 replications for all M systems, and calculate sample means $\bar{X}_{ik}(n_0)$ and sample variances $S_{ik}^2(n_0)$ ($i \in S$ and $k \in K$). Let $N_i = n_0$.

3: Observe the Pareto set S_p and the non-Pareto set S_p^c based on the sample means $\bar{X}_{ik}(N_i)$ ($i \in S$ and $k \in K$) using Algorithm 1 without the indifference-zone concept.

4: For each system $i \in S_p$ and $j \in S$ ($j \neq i$) with $K_1 = K$, check if the following two conditions are met:

$$N_i > \left\lceil \max_k \left(\frac{h_1 S_{ik}(N_i)}{\delta_{ijk}} \right)^2 \right\rceil \text{ and } N_j > \left\lceil \max_k \left(\frac{h_1 S_{jk}(N_j)}{\delta_{ijk}} \right)^2 \right\rceil, \tag{4.4}$$

where h_1 is the solution to (4.9), and K_1 is defined in (4.7).

5: For each system $i \in S_p$ and $j \in S$ ($j \neq i$) with $K_1 \neq K$, check if the following two conditions are met:

$$N_i > \left\lceil \left(\frac{h_2 S_{ik'}(N_i)}{\delta_{ijk'}} \right)^2 \right\rceil \text{ and } N_j > \left\lceil \left(\frac{h_2 S_{jk'}(N_j)}{\delta_{ijk'}} \right)^2 \right\rceil, \tag{4.5}$$

where k' is defined in (4.8) and h_2 is to solution to (4.10).

6: Delete system i from I if conditions (4.4) or (4.5) are satisfied for all $j \in S$ ($j \neq i$).

7: For each system $j \in S_p^c$, find system $i \in S_p$ as defined in (4.11). Check if the following two conditions are met:

$$N_i > \left\lceil \max_k \left(\frac{h_3 S_{ik}(N_i)}{\delta_{ijk}} \right)^2 \right\rceil \text{ and } N_j > \left\lceil \max_k \left(\frac{h_3 S_{jk}(N_j)}{\delta_{ijk}} \right)^2 \right\rceil, \tag{4.6}$$

where h_3 is the solution to (4.12).

8: Delete system j from I if conditions in (4.6) are satisfied.

9: If $|I| = 0$, then stop and present the current Pareto set S_p as the final solution set. Otherwise, for each system $i \in S_p \cap I$, that is, systems in S_p that were not deleted from I in Step 6, add system $j \in S$ ($j \neq i$) to I if it does not satisfy conditions (4.4) or (4.5). Similarly, for each system $j \in S_p^c \cap I$, that is, systems in S_p^c that were not deleted from I in Step 8, add the corresponding system $i \in S_p$ to I if it does not satisfy (4.6). Go to Step 10.

10: Take one additional observation X_{i,k,N_i+1} from each system $i \in I$, and set $N_i \leftarrow N_i + 1$ ($\forall_i \in I$). Set $I = \{1, 2, ..., M\}$ and update $\bar{X}_{ik}(N_i)$ and $S_{ik}^2 ik(Ni)$ for all $i \in S$ and $k \in K$, and go to Step 3.

and

$$k' = \arg\max_{k \in K} \Phi \left(\frac{\bar{X}_{jk}(N_j) - \bar{X}_{ik}(N_i)}{\sqrt{\frac{S^2_{ik}(N_i)}{N_i} + \frac{S^2_{jk}(N_j)}{N_j}}} \right) \tag{4.8}$$

where Φ denotes the cumulative distribution function (cdf) of the standard normal distribution. Note that K_1 and k' should be defined for every pair of (i, j) ($i \in S_p$ and $j \in S, j \neq i$). Step 4 in Algorithm 10 deals with (i, j) pairs when $K_1 = K$, that is, system i and j are observed to be indifferent to each other, while Step 5 considers the case when $K_1 \neq K$.

The constants h_1 (in Step 4) and h_2 (in Step 5) are the solution to the following equations, respectively:

$$\left[\int_0^\infty \left[\int_0^\infty \Phi \left(\frac{h_1}{\sqrt{(N_i - 1)\frac{1}{x} + (N_j - 1)\frac{1}{y}}} \right) f_1(x) \mathrm{d}x \right] f_2(y) \mathrm{d}y \right]^H = 1 - \gamma, \tag{4.9}$$

and

$$\int_0^\infty \left[\int_0^\infty \Phi \left(\frac{h_2}{\sqrt{(N_i - 1)\frac{1}{x} + (N_j - 1)\frac{1}{y}}} \right) f_1(x) \mathrm{d}x \right] f_2(y) \mathrm{d}y = 1 - \gamma, \tag{4.10}$$

where $\gamma = \frac{\beta}{M-1}$, and f_1 and f_2 denote the pdf of the χ^2 distribution with $N_i - 1$ and $N_j - 1$ degrees of freedom, respectively. The system that dominates system j with the maximum probability can be found with

$$\begin{aligned} i &= \arg\max_{i' \in S_p} P(i' \prec j) \\ &\approx \arg\max_{i' \in S_p} \prod_{k=1}^{H} \Phi \left(\frac{\bar{X}_{jk}(N_j) - \bar{X}_{i'k}(N_{i'})}{\sqrt{\frac{S^2_{i'k}(N_{i'})}{N_{i'}} + \frac{S^2_{jk}(N_j)}{N_j}}} \right). \end{aligned} \tag{4.11}$$

Note that such i should be defined for every $j \in S_p^c$. This pair of systems (i, j) $(i \in S_p, j \in S_p^c)$ is considered in Step 7 in Algorithm 10. The constant h_3 in the same step of the algorithm is the solution to

$$\left[\int_0^\infty \left[\int_0^\infty \Phi\left(\frac{h_3}{\sqrt{(N_i-1)\frac{1}{x}+(N_j-1)\frac{1}{y}}}\right) f_1(x)dx\right] f_2(y)dy\right]^H = 1-\beta, \qquad (4.12)$$

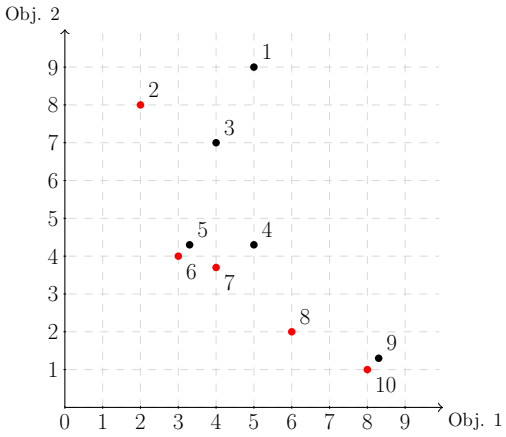
where $\beta = \frac{\alpha}{M}$, and f_1 and f_2 denote the pdf of the χ^2 distribution with $N_i - 1$ and $N_j - 1$ degrees of freedom, respectively.

4.2.2.1 The relaxed Pareto set approach

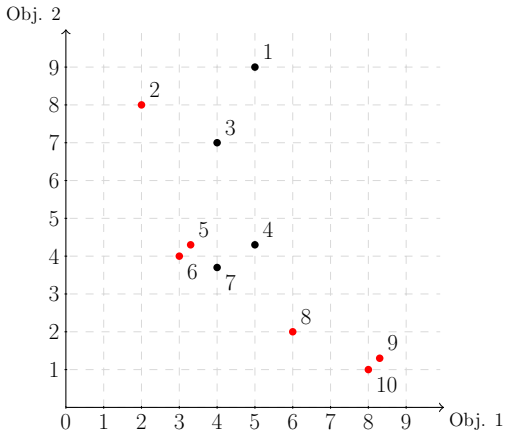
An important particularity of the $\mathcal{MMY}$ procedure is that it outputs an approximate *relaxed* Pareto set with respect to the indifference-zone values selected by the user. In effect, one would expect an IZ-based MOO R&S procedure to include the IZ concept during its Pareto ranking step; however, this is not the case in procedure $\mathcal{MMY}$ as pointed out in Step 3 of Algorithm 10. The relaxed Pareto set approach is, in effect, one that is not strict about the IZ concept with regards to solutions that should be considered as non-dominated. Yoon (2018) found in her work that using a strict IZ regime can cause the R&S algorithm to run indefinitely when comparing a pair of systems whose true means are very close. Hence, the researcher (*i.e.* Yoon (2018)) proposed going instead for a relaxed Pareto set in order to avoid this. This approach therefore makes the procedure, one that is safer and more flexible in practice.

Moreover, the researcher (*i.e.* Yoon (2018)) also argues in her work that allowing the final Pareto set to *possibly* consider as Pareto-optimal solutions, non-dominated solutions (under no IZ regime) that would otherwise be seen as dominated under a strict IZ regime, gives the decision-maker a more comprehensive final set.

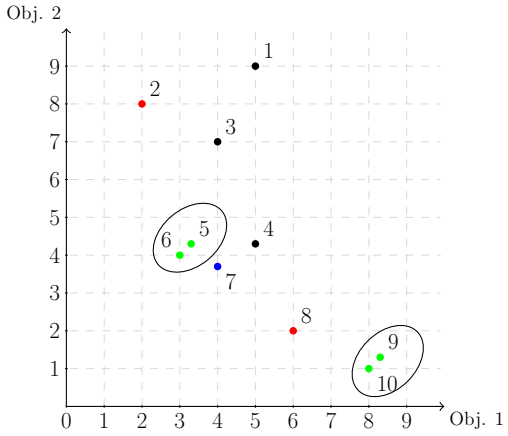
To illustrate the concept, consider Figure 4.4 where the solutions in red are non-dominated solutions and those in black are dominated ones. The IZ value for both objectives in this figure is selected as 0.5. A relaxed Pareto set (Figure 4.4(c)) contains all non-dominated systems that do not have indifferent systems (solutions in red), at least one system from a group of indifferent systems (solutions in green); and regarding systems classified as non-dominated without the IZ concept, but dominated under the IZ regime (solutions in blue), a relaxed Pareto set may or may not contain them.



(a) Pareto set without IZ.



(b) Pareto set with IZ.



(c) Relaxed Pareto set.

Figure 4.4: Pareto set examples (Yoon, 2018).

4.2.2.2 $\mathcal{MMY}$ implementation challenge

The major challenge in implementing the $\mathcal{MMY}$ procedure is the h_i values calculation ($i = 1, 2, 3$). h_i values are solutions to double integral equations that are time-consuming to solve as they are solved numerically. It is therefore impractical to dynamically calculate these values when the procedure is being executed. Instead, these values are calculated beforehand and saved in tables (*i.e.* an h_1 table contains h_1 values *etc.*) which the procedure can then look-up during execution. Generating these tables, however, is another challenge. It can be observed in (4.9), (4.10) and (4.12), that they all account for the total number of systems M being considered by the procedure. This means that every h_i table is unique to a specific number of total systems being considered by the procedure. In other words, for a problem where *e.g.* $M = 8$, the procedure will look-up a different set of h tables than one where *e.g.* $M = 9$.

Now, consider using the results of a metaheuristic that uses the Pareto approach directly as inputs to the R&S procedure. This would mean that the R&S procedure must have a set of h tables for every possible number of solutions in the approximated Pareto set obtained by the metaheuristic. This is in effect a difficult thing to do, in addition to being impractical as the set of solutions could be relatively very large. This is what led the author to consider the interactive approach and actually limit the number of systems that the user can select from the approximate Pareto set. In this study, this number was limited to up to 10, meaning that nine set of h tables had to be calculated beforehand (*i.e.* $1 < M \leq 10$). Note that the author kept P^* constant with a value of 90.

4.3 Chapter summary

In this chapter, the solution approach proposed for the purpose of this study was described. Additionally, the selected algorithms for the MOO optimisation suite were also presented and described in detail.

In the next chapter, the development and implementation of the MOO optimisation suite are presented.

Chapter 5

Development and implementation

In this chapter, the author describes the development and implementation of the MOO-Solver library. In the opening section of the chapter, a discussion on the suite itself is presented. This is followed by discussions on the interfaces that were used to integrate the suite with TPS. First, a discussion on the C-Interface inter-process communication (IPC) technology (also simply referred to as C-Interface or C for short) is presented. A subsection on the limitations of the C-Interface then follows in order to put the next section into context, which is a discussion on the COM-Interface IPC technology (also simply referred to as COM-Interface or COM for short). In the last section, the author presents the MOOSolver user-interface for TPS.

5.1 MOOSolver: A Dynamic-link Library solver for MOSO problems

Following the architectural design in the previous chapter, MOOSolver was developed and implemented as a dynamic-link library (DLL). A DLL is a module that contains functions and data that can be used by another module (an application or another DLL). Unlike in the case of statically linked libraries, the programs or applications that call a DLL are connected to it at runtime rather than at linking or compiling time.

DLLs can define two kinds of functions: exported and internal. The exported functions are intended to be called by other modules, as well as from within the DLL

where they are defined. Internal functions, on the other hand, are typically intended to be called only from within the DLL where they are defined.

The MOOSolver DLL was developed in C++ using Microsoft Visual Studio 2017. According to the C++ resources network website (www.cplusplus.com): C++ is designed to be a compiled language, meaning that it is generally translated into machine language that can be understood directly by the system, making the generated program highly efficient.

The MOSO problems for which MOOSolver is intended are expected to be computationally intensive. A program written in a "highly efficient" language is thus important if these problems are to be solved effectively.

5.1.1 The C-Interface

An interface defines a set of methods (or functions) that an object can support, without dictating anything about the implementation. The interface marks a clear boundary between code that calls a method and the code that implements the method (Microsoft, 2010).

According to the Help guide of the Tecnomatix Plant Simulation software, the features and functionalities of the software can be extended considerably by integrating functions programmed in C/C++. The integration is made possible through the C-Interface, which comes as part of the TPS software package.

The C-Interface makes it possible to load an external DLL and call the functions in the DLL from the simulation software. Moreover, the C-Interface also makes it possible to manipulate the simulation software from within the DLL. A two-way interaction can thus exist between the two platforms.

The opportunity provided by the C-Interface was exploited in the manner illustrated by the diagram in Figure 5.1 in order to implement the simulation optimisation process designed in Chapter 4. The diagram shows in greater detail how the inter-process communication between MOOSolver and TPS was made possible via the C-Interface set of functions. Positioned on the boundary between the two platforms in the diagram are the interfacing functions that were used. The functions are colour-coded to indicate the platform where they are defined (or implemented). In what follows, the author explains in further details each step in Figure 5.1.

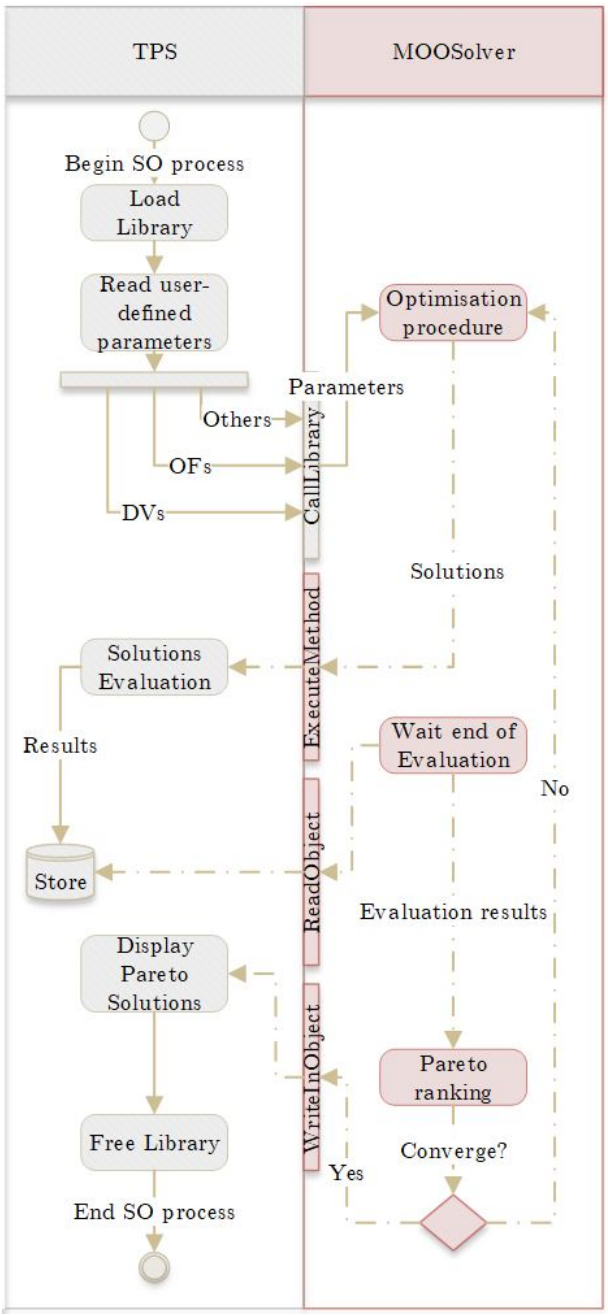


Figure 5.1: The C-Interface inter-process communication procedure.

Begin SO process: This is the starting point of the SO process. After a model has been developed and all the necessary information about it has been provided to MOOSolver, the user initiates the SO process.

Load library: After the SO process has been initiated by the user, a method named *callback* takes over. Callback loads the MOOSolver library by locating it and establishing a connection between TPS and the suite.

It is through the callback method that TPS communicates with MOOSolver. Callback is part of a set of objects in TPS created and dedicated to facilitate the inter-process communication between TPS and MOOSolver via the C-Interface. These objects form part of the MOOSolver user-interface in TPS and will be mentioned one by one as this section progresses.

Read user-defined parameters: This step of the process reads the information about the simulation model as defined by the user. The information is then passed to MOOSolver via the *CallLibrary* C-Interface function.

Optimisation procedure: Upon receiving the simulation model parameters, the suite triggers the optimisation procedure. This is where, in the case of large-scale problems, the metaheuristic begins its execution and seeks for a set of candidate solutions. When found, the elements in the set are passed to TPS via the *ExecuteMethod* function to be evaluated.

Now, evaluation via simulation (that is, evaluation via the use of a discrete-event simulation software package) can be done without running *actual* simulations if the model happens to be, for some reason, deterministic with an analytical formulation of its objective functions. In such a case, the elements to be evaluated are evaluated by executing the method where the objective functions are defined (as illustrated in the diagram). The goal, however, is to use MOOSolver for MOSO problems that require actual simulation runs during the evaluation process. A C-Interface function, namely *RunSimulation*, exists in effect for this very purpose. However, as will be explained shortly, the function in question is unable to fulfil its task properly in the context of the SO process as designed in Section 4.1. The function is thus of no use here. The author proceeds first with the description of the C procedure before discussing the present issue (*i.e.* How to evaluate non-deterministic problems) further. (This will be done in Section 5.1.2.)

Evaluation process: This step includes the evaluation of the solutions found by the metaheuristic (or those provided by the user in the case of small-scale problems) as well as the halting of the SO process while the evaluation process is in progress. After evaluation, the results are stored in a *temporary store*, which is another object

(a Table/Array object) in TPS dedicated for MOOSolver. These results can then be accessed by the MOOSolver library via the *ReadObject* C-Interface function.

Pareto ranking: After the solutions have been evaluated, they are ranked using Algorithm 1. The non-dominated solutions are retained, while the dominated ones are discarded according to the rules of the algorithm being executed (*i.e.* MOO CEM or $\mathcal{MMY}$).

A convergence check then follows to determine whether the SO process should be terminated. There are two events, whichever occurs first, that typically terminate the process. The first event happens when a stopping condition has been reached such as, in the case of the metaheuristic, the maximum number of evaluations allowed, and in the case of the R&S procedure, the depletion of the simulation budget. The second event, on the other hand, happens when the quality of newly found solutions begin to have little to no difference after the algorithm has run sufficiently long in the metaheuristic case, and in the R&S case, this happens when the necessary number of simulation runs have been made to all solutions being compared. The second condition is governed by the user-defined parameter ϵ (see Section 4.2.1) in the metaheuristic case, while in the R&S case, it is done by the user-defined indifference-zone values (see Section 4.2.2).

Display Pareto solutions: When the convergence check is positive, the SO process is set to stop. Otherwise, the process goes back to the *Optimisation procedure* step. Before the process ends, the final Pareto set (the approximated one, that is) is displayed in a table format (using another dedicated TPS object) via the *WriteInObject* C-Interface function.

Free library: The SO process is terminated after the principal connection established between TPS and MOOSolver via the C-Interface is finally closed. This is done by *freeing* the MOOSolver library in the callback method.

5.1.2 Limitations of the C-Interface

The goal of integrating the suite with the simulation software is so that simulation runs can be executed from the suite (on non-deterministic problems) as part of the SO process. In other words, simulation runs are to be an integral part of the SO process to support the optimisation procedure while it is in progress (please refer to Figure 4.1).

The issue, however, is that a TPS application cannot be commanded to execute a simulation run while other methods are being executed within the same application. More specifically, while the method calling MOOSolver in TPS (*i.e.* the callback method) is processing, it is not possible to, simultaneously, execute simulation runs from the suite. What happens instead is, the simulation runs are postponed and executed only after all methods processing in TPS are completed (including the method calling the suite). This is a limitation that makes it impossible to have a valid SO process using the design in Figure 4.1. The design requires, in effect, that simulation runs be executed while the calling method is still processing.

In order to implement the designed architecture, therefore, this obstacle had to be overcome somehow. This was achieved by using a COM-Interface within the existing C-Interface to open a second, supporting, TPS application. This approach was suggested to the author by a TPS expert via the TPS forum in the Siemens community website. In summary, when using the COM technology, an *instance* of the TPS application of interest (the discrete-event simulation model of interest, to be more precise) can be opened as a background process, and simulation runs can be executed via this supporting application rather than the original one. This is possible because the supporting application exists as an independent module or *object* and is unaware of the existing, ongoing, C-Interface IPC procedure between the original TPS application and MOOSolver.

5.1.3 The COM-Interface

The *component object model* technology (COM) is defined by Microsoft (2010) as a binary-interface standard for software components. It is a language-neutral way of implementing methods that can be used in environments different from the one in which they were created, even across machine boundaries. It is used to enable interprocess communication object creation in a large range of programming languages.

Unlike in the C-Interface case, COM only offers a one-way communication process between the two platforms of interest. A critical aspect of COM is, in effect, how *client* and *server* platforms interact. A COM client is whatever application or object gets a *pointer* to a COM server and uses its services by calling the methods (or functions) of its interfaces. A COM server, on the other hand, is any application or object that provides services to a COM client; these services are in the form of interface implementations

(*i.e.* the COM-Interface functions) that can be called by any client that is able to get a pointer to these interfaces on the server object. The client object, in this study, is MOOSolver while the server object is TPS.

It is thus possible to manipulate TPS from the suite (the client) but not the other way round. Using the COM-Interface set of functions, MOOSolver can execute simulation runs by opening TPS in the background and loading an instance of the model of interest. This model being opened using the COM technology is totally independent of its original version being executed using the C-Interface. As a result, both IPC procedures can execute simultaneously and simulations run while the callback method in TPS is still processing.

The background process reads the solutions from the optimisation procedure and controls the evaluation process. Results from the evaluation process can then be accessed by the primary process once the simulation runs are completed.

The procedure described above is illustrated in Figure 5.2 and following are detailed descriptions of each step in the figure.

Initialise COM library: When using COM, a strict protocol must be followed by the client object in order to successfully establish a connection with the server object. Any process that uses COM must both initialise and un-initialise the COM library. The COM technology implements some important services in this library, among which are a small number of fundamental functions that facilitate, for example, the creation of the server object and a standard mechanism to allow, *e.g.*, the client object to control how memory is allocated within its process, particularly memory that needs to be passed between cooperating objects (*i.e.* client and server) so that it can be freed properly (Microsoft, 2010).

Create instance of TPS as an object: After the COM library has been successfully initialised, the client object (*i.e.* TPS) is created in the background (*i.e.* TPS opens in the background) and its COM-Interface set of functions is now available to the client object (*i.e.* MOOSolver). The first COM-Interface function to be used is *LoadModel*, to which the ID of the model being currently used via C is passed in order to open another instance of it. If successful, the model opens in the background.

Read solutions and run simulations: In these steps, MOOSolver reads the solutions to be evaluated from the original model and writes them to the background model via the function *SetValue*. Once that is completed, MOOSolver resets and runs

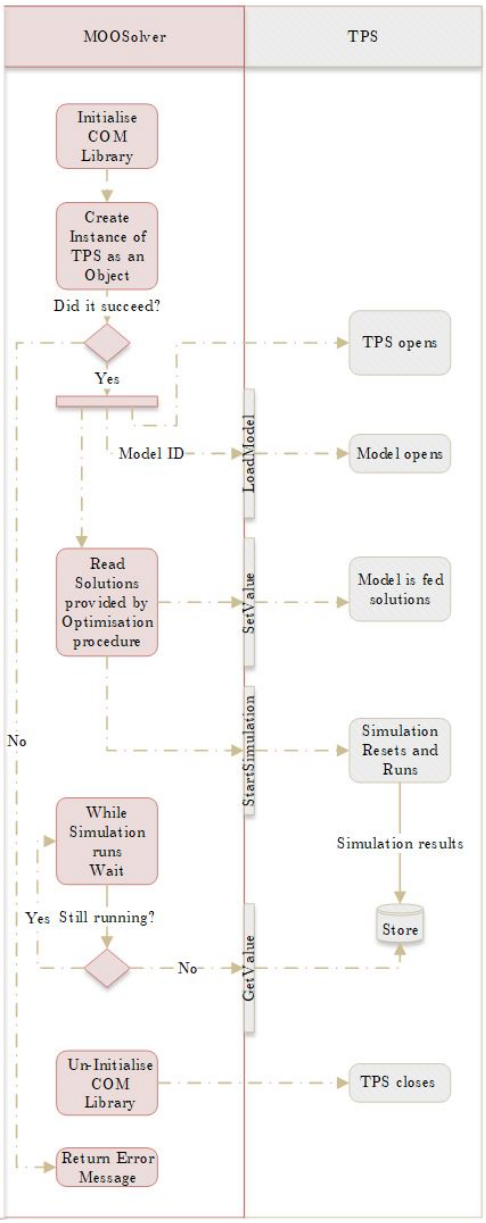


Figure 5.2: The COM-Interface inter-process communication procedure.

the background model via the COM-Interface functions *ResetSimulation* and *StartSimulation*, respectively.

While simulations run wait: The procedure is paused while simulations are being run in the background. When the simulations are complete, MOOSolver gets access to the simulations results via the *GetValue* function. Once MOOSolver has the

results, the main procedure (*i.e.* the C-Interface IPC procedure) proceeds.

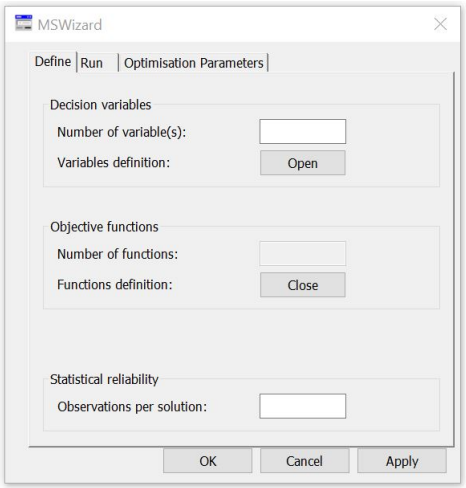
Un-initialise COM library: It is important to un-initialise the COM library once the SO process is complete. This step includes *realising* (*i.e.* freeing) the COM library as well as *closing* the supporting model and TPS application in the background. This step is critical as, if not done properly, the processes in the background can remain open long after the main model and application have been closed; an incident that is undesirable for computers.

Note that the procedures described in Figures 5.1 and 5.2 are hidden from the user. The user simply initiates the entire SO process via a user-interface in TPS and waits for the final results. This user-interface is the subject of the next section.

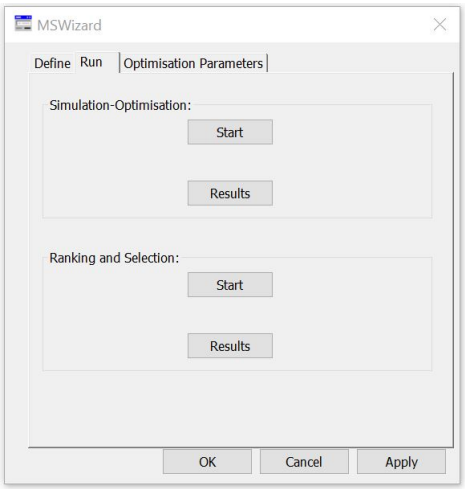
5.2 MOOSolver: The user-interface for TPS

The MOOSolver user-interface for TPS is a *Dialog* object which allows user interaction with the suite. Many elements that form part of this object have already been described in the previous section where they were referred to as dedicated TPS objects for MOOSolver. This section focuses mainly on the graphical user-interface (GUI) and its elements. The GUI is the principal medium through which the user provides information about the simulation model to the suite. It is also the principal medium through which the user receives results from the suite. Unlike some of the other elements of the user-interface defined earlier (*e.g.* the callback method and the temporary stores), which are principally used in the background (*i.e.* outside the user's sight), the GUI and its features are for direct interaction between the user and the MOOSolver library. The design of the GUI was inspired by that of the TPS built-in optimisation suite GUI, namely, the *GAWizard*. The MOOSolver GUI for TPS was subsequently named the *MSWizard*.

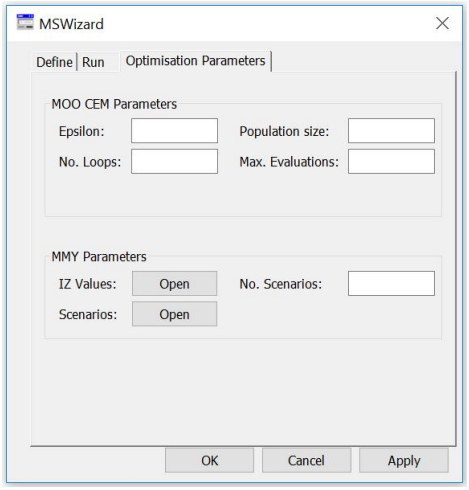
This section is divided into two parts; in the first part, the author describes the GUI features used to input information about the simulation model as well as those used to input user-selected parameters for the algorithms. In the second part, the GUI features used to present (to the user) output results obtained by the suite are described.



(a) Define tab.



(b) Run tab.



(c) Optimisation Parameters tab.

Figure 5.3: MSWizard GUI principal tabs.

5.2.1 GUI input features

The MSWizard has three main tabs named *Define*, *Run* and *Optimisation Parameters*. Screen-shots of the wizard can be seen in Figure 5.3. The following paragraphs describe the functions of each tab.

In the first tab (Figure 5.3(a)), the user defines the simulation model parameters. The tab has three sections called *group boxes*. In the group box *Decision variables*, the user specifies the number of decision variables (DVs) in the model as well as their

respective paths (*i.e.* location) within the simulation model (see Figure 5.4). In the same table where the DV paths are specified, the user also provides all DVs respective boundaries as well as the DVs respective nature (*i.e.* whether they are discrete or continuous); the DVs boundaries are the limits (or constraints) that create a feasible solution space for the problem, following the MOSO framework of Chapter 2.

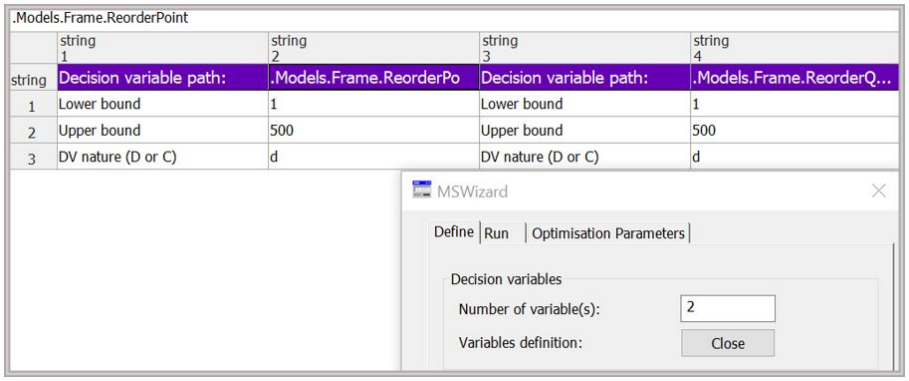


Figure 5.4: Decision variables definition table.

In the second group box, the user defines the objective functions (OFs) following a similar approach to that of the first group box (see Figure 5.5). In addition to the paths of each OF, the user also specifies their respective optimisation directions.

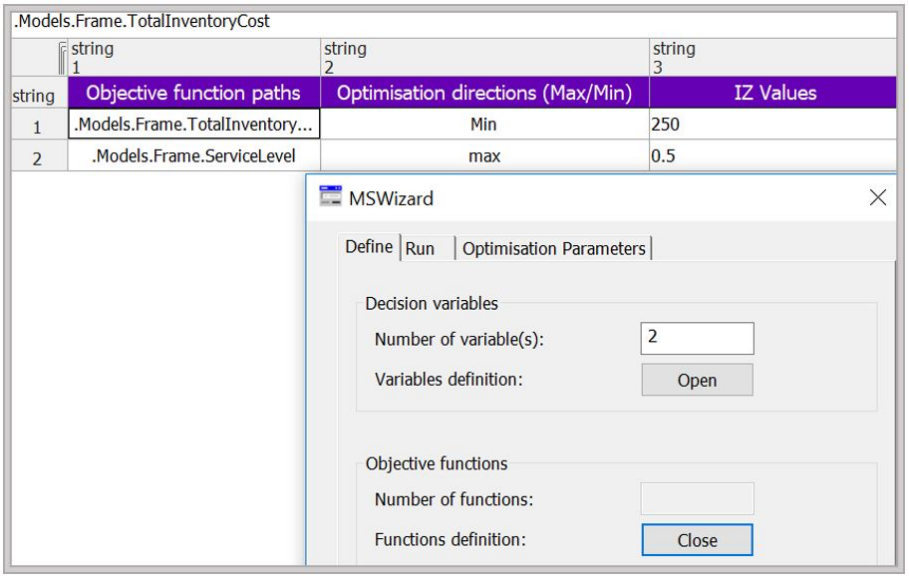


Figure 5.5: Objective functions definition table.

The last group box is for the number of observations that the user wants MOOSolver to use for each solution that will be evaluated (*i.e.* a "blindly" selected n^* similar to that of Section 3.2.1).

In the second tab (Figure 5.3(b)), the user runs the algorithms once all the parameters have been correctly inserted. The first group box contains the button for starting the MOO CEM while the second group box contains the button for starting the $\mathcal{MMY}$ procedure. Before running the algorithms however, the user must specify additional parameters in the third and final tab.

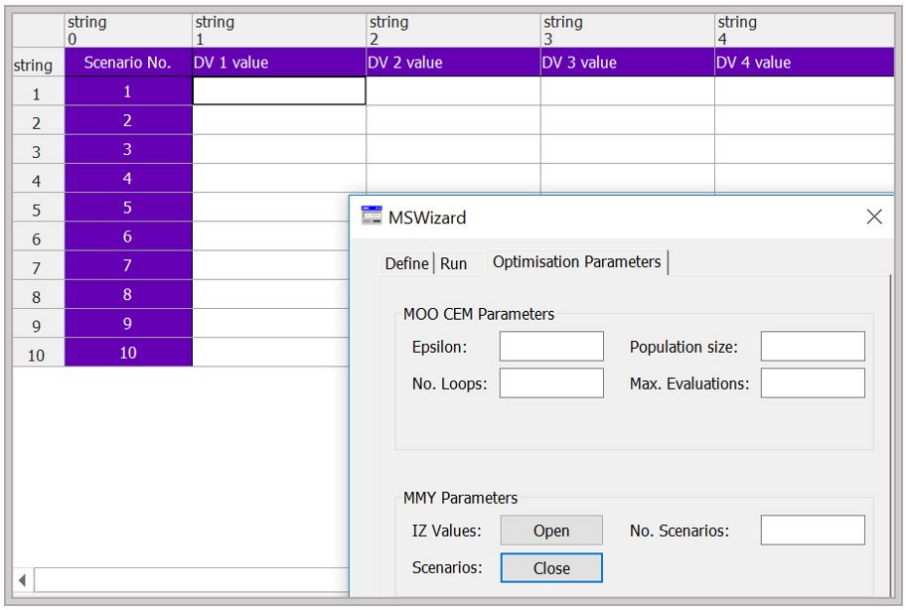


Figure 5.6: $\mathcal{MMY}$ Scenarios table.

In the final tab (Figure 5.3(c)), the user specifies the MOO CEM parameters as well as the $\mathcal{MMY}$ parameters. For the MOO CEM, the user must select the values for epsilon, the maximum number of evaluations, the number of outer loops as well as the population size. (Please refer to Section 4.2.1 for descriptions on these parameters pertaining to their respective roles within the metaheuristic.) For the $\mathcal{MMY}$, the user must specify the scenarios/systems or solutions to be compared as well as the respective IZ values of the objective functions (entered in the OFs definition table (Figure 5.5)). As already mentioned, MOOSolver allows the user to select up to 10 scenarios (see Figure 5.6) for reasons discussed in Section 4.2.2.2. Note that the columns in Figure

5.6 are to follow the order according to which the DVs were defined. In other words, if DV 1 was defined before DV 2 in the columns of the DVs definition table (Figure 5.4), then the values for DV 1 in the scenarios table must be in the first column and those for DV 2 in the second column etc. Note also that the $\mathcal{MMY}$ procedure has a preselected probability of correct selection value (90%) as was specified in Section 4.2.2.

5.2.2 GUI output features

The results obtained by MOOSolver are presented to the user in tables. In this section, the author describes the table formats for both the MOO CEM and the $\mathcal{MMY}$.

Table 5.1: MOO CEM output table format.

Decision variables			**Objectives**		
x_{11}	$\cdots$	x_{1n}	f_{11}	$\cdots$	f_m
$\vdots$		$\vdots$	$\vdots$		$\vdots$
x_{N1}	$\cdots$	x_{Nn}	f_{N1}	$\cdots$	f_{Nn}

The output presented to the user at the end of the MOO CEM run, is a table that contains information about the approximate Pareto set obtained by the metaheuristic. The table has the format illustrated in Tale 5.1, where n is the number of decision variables in solution $\mathbf{x}_i, i = 1, ..., N$, N the total number of solutions in the approximate Pareto set and m the number of objective functions, which in this study is always 2.

As the table can be overwhelmingly large, a better way of presenting the approximate Pareto set to the user is by plotting a Pareto front that contains the results to all the solutions in the Pareto set. Examples of Pareto fronts obtained by MOOSolver can be observed in Chapters 6 and 7. All the points in a Pareto front are results to specific solutions in the Pareto set and are hence, sometimes, referred to as solutions themselves. Using Pareto fronts is also a way of exploiting the fact that the number of objectives MOOSolver can handle at present is 2. In effect, when the number of objectives grows past 2 and 3, it becomes difficult to generate insightful graphical or visual representations of the Pareto set; the user, in this case must settle for the use of tables.

The output presented to the user at the end of the $\mathcal{MMY}$ run, on the other hand, is a table that contains information about the scenarios or solutions compared to each other

by the procedure. The table format is illustrated in Table 5.2, where n is the number of decision variables in system $\mathbf{x}_i$, $i = 1, ..., M$, M the total number of preselected systems fed to the procedure and m the number of objective functions, which is again always 2 in this study.

Table 5.2: $\mathcal{MMY}$ output table format.

Decision variables			**Objectives**			**Variances**			**Rank**	**Runs**	**Status**	**ID**
x_{11}	$\cdots$	x_{1n}	f_{11}	$\cdots$	f_{1m}	S^2_{11}	$\cdots$	S^2_{1m}	ρ_1	R_1	St_1	ID_1
$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_{M1}	$\cdots$	x_{Mn}	f_{M1}	$\cdots$	f_{Mm}	S^2_{M1}	$\cdots$	S^2_{Mm}	ρ_M	R_M	St_M	ID_M

Additionally, S^2_{ij} is the variance information of objective function j of system $\mathbf{x}_i$, $j = 1, ..., m$; ρ_i is the Pareto rank of system $\mathbf{x}_i$ indicating the total number of systems in the set that dominate system $\mathbf{x}_i$. An integer value greater than 0 in this column indicates that system $\mathbf{x}_i$ is dominated by ρ_i systems in the set and is thus not a correct selection. R_i is the total number of observations that was run for system $\mathbf{x}_i$ during the execution of the procedure while St_i is the status of system $\mathbf{x}_i$ indicating whether the necessary number of observations were run for the system. Since the procedure has a limited simulation budget to be run for every system (for practical purposes), if the budget were exhausted before the procedure could guarantee the statistical soundness of the system (with respect to the parameters used by the procedure in Step 1 of Algorithm 9), the Status column would indicate it. When a value of 0 is displayed for system $\mathbf{x}_i$, this indicates that the procedure was able to run the necessary number of observations for the system. Any other value (typically 1 or 2, which have themselves specific meanings in the context of the procedure implementation) indicates that the system needed more observations than allowed (in order to guarantee statistical soundness). Finally, ID_i is the ID of system $\mathbf{x}_i$ with respect to the order in which the system was initially fed to the procedure by the user (via the Scenarios table). Since the order in which the systems are given to the procedure changes during the execution of the procedure, using IDs to identify each system once the runs are complete is more convenient than doing so with the help of decision variables.

5.3 Chapter summary

In this chapter, the development and implementation of the multi-objective optimisation suite were presented. In particular, the inter-process communication procedures used to integrate the suite with TPS were fully described. Moreover, the user-interface for TPS and its features were also described in great detail. In the next chapter, the product developed in this chapter is validated.

Chapter 6

Validation

In the previous chapter, the development and implementation of MOOSolver was presented. MOOSolver, which is an optimisation suite for MOSO problems, was developed as a dynamic-link library and integrated with Tecnomatix Plant Simulation using, simultaneously, C and COM inter-process communication technologies. It was also mentioned in Chapter 4 that two algorithms were selected for the suite, namely, the MOO CEM metaheuristic and the $\mathcal{MMY}$ procedure.

In this chapter, the suite is validated by using known MOO test problems as well as a variant of the buffer allocation problem discussed in Chapter 3. The MOO test problem will help validate the implementation of the MOO CEM and the BAP, that of the $\mathcal{MMY}$ procedure. Valid results also mean, automatically, that the IPC procedures used to integrate MOOSolver with TPS, namely C and COM, were both successful.

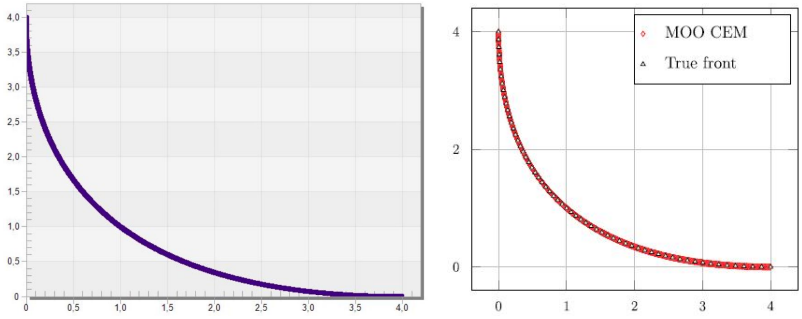
6.1 MOO test problems

The author used known MOO test problems to validate the implementation of the MOO CEM metaheuristic as well as the C-Interface inter-process communication procedure. The MOO test problems were modelled as deterministic models in TPS. In total, five test problems were used from Coello Coello *et al.* (2007) and they are presented in Table 6.1. The results obtained by MOOSolver for the five problems are shown in Figures 6.1 and 6.2 where they were compared with results obtained by a MATLAB implementation of the MOO CEM together with the known true results of the test problems.

Table 6.1: Standard MOO test functions.

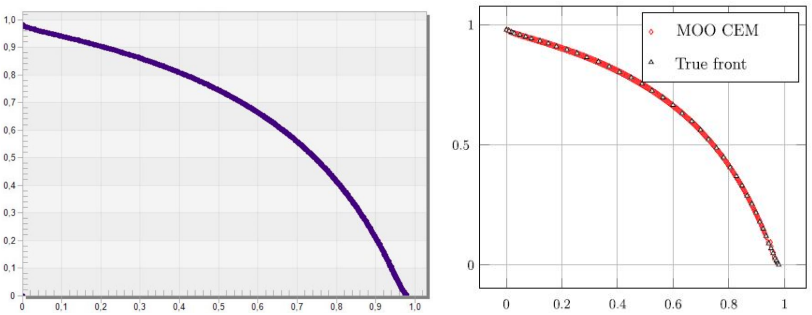
Function	Definition	Constraints
MOP1	$f_1(x) = x^2$	$-10^5 \leq x \leq 10^5$
(Min)	$f_2(x) = (x-2)^2$	
MOP2	$f_1(\mathbf{x}) = 1 - \exp\left(-\sum_{i=1}^{n}\left(x_i - \frac{1}{\sqrt{n}}\right)^2\right)$	$-4 \leq x_i \leq 4$
(Min)	$f_2(\mathbf{x}) = 1 - \exp\left(-\sum_{i=1}^{n}\left(x_i + \frac{1}{\sqrt{n}}\right)^2\right)$	$i = 1, ..., n, n = 3$
MOP3	$f_1(x,y) = -[1 + (A_1 - B_1)^2 + (A_2 - B_2)^2]$	$-\pi \leq x, y \leq \pi$
(Max)	$f_2(x,y) = -[(x-3)^2 + (y+1)^2]$	$A_1 = 0.5\sin 1 - 2\cos 1 + \sin 2 - 1.5\cos 2,$
		$A_2 = 1.5\sin 1 - \cos 1 + \sin 2 - 0.5\cos 2,$
		$B_1 = 0.5\sin x - 2\cos x + \sin y - 1.5\cos y,$
		$B_2 = 1.5\sin x - \cos x + \sin y - 0.5\cos y,$
MOP4	$f_1(\mathbf{x}) = \sum_{i=1}^{n-1}\left(-10\exp(-0.2)\sqrt{x_i^2 + x_{i+1}^2}\right)$	$-5 \leq x_i \leq 5$
(Min)	$f_2(\mathbf{x}) = \sum_{i=1}^{n-1}\left(\lvert x_i\rvert^a + 5\sin(x_i)^b\right)$	$i = 1, 2, 3, a = 0.8, b = 3$
MOP6	$f_1(x,y) = x$	$0 \leq x, y \leq 1$
(Min)	$f_2(x,y) = (1+10y)\left[1 - \left(\frac{x}{1+10y}\right)^{\alpha} - \frac{x}{1+10y}\sin(2\pi q x)\right]$	$q = 6, \alpha = 2$

The comparisons in Figures 6.1 and 6.2 show that the MOO CEM implementation in MOOSolver is valid. Additional (and more rigorous) tests for the metaheuristic can be found in Bekker & Aldrich (2011). For the purpose of this study, the results obtained in this chapter are considered sufficient validation material as the algorithm used here (Algorithm 9) is the same as the one in Bekker & Aldrich (2011). More importantly, the results in this section validate the C-Interface IPC procedure (Figure 5.1) developed to integrate the suite with the discrete-event simulation software.



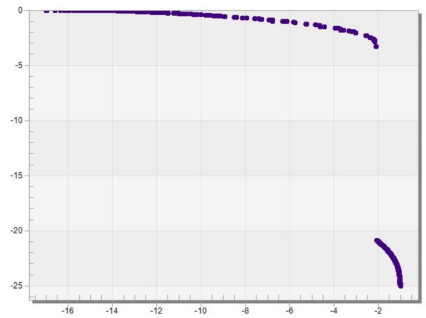
(a) MOP1 by MOOSolver. (b) MOP1 in MATLAB.

Figure 6.1: Comparison between MOO test problems results obtained by MOOSolver (a) and results obtained in MATLAB together with the known results (b).

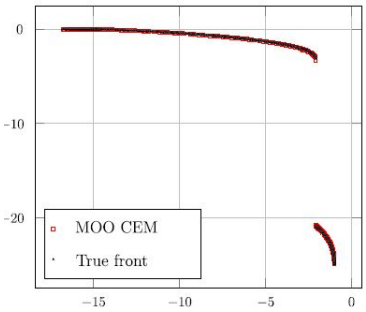


(a) MOP2 by MOOSolver.

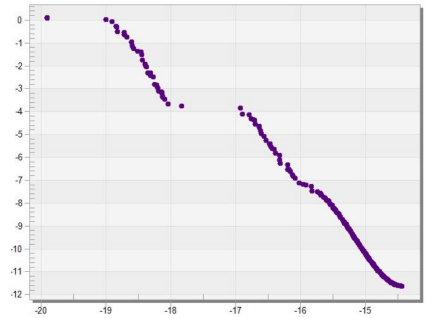
(b) MOP2 in MATLAB.



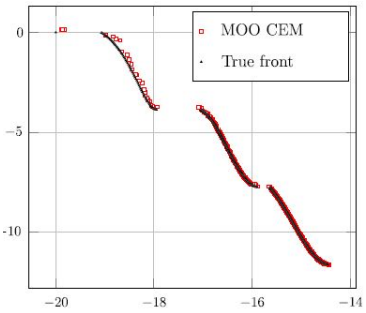
(c) MOP3 by MOOSolver.



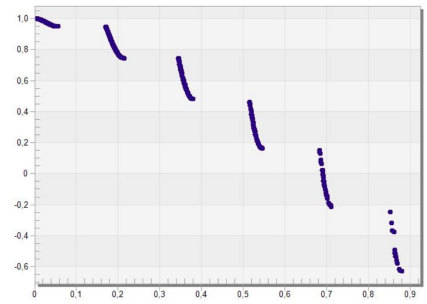
(d) MOP3 in MATLAB.



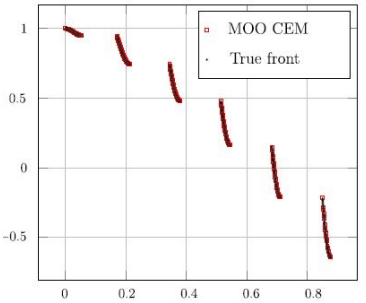
(e) MOP4 by MOOSolver.



(f) MOP4 in MATLAB.



(g) MOP6 by MOOSolver.



(h) MOP6 in MATLAB.

Figure 6.2: Comparison between MOO test problems results obtained by MOOSolver (a, c, e, and g) and results obtained in MATLAB together with the known results (b, d, f and h).

6.2 The buffer allocation problem

In her work, Yoon (2018) solved the BAP as a small-scale SO problem with preselected solutions using a MATLAB implementation of the $\mathcal{MMY}$ algorithm. She used the BAP to validate the procedure. The same BAP is solved here with MOOSolver and a similar validation approach is used. The results obtained here are compared to those in Yoon (2018). Successful comparison would thus validate the $\mathcal{MMY}$ implementation in MOOSolver and, consequently, the COM-Interface IPC procedure as well.

The validation process in Yoon (2018) was as follows: Ten scenarios were preselected to be compared using the $\mathcal{MMY}$ procedure. To ensure the validity of the results obtained by the procedure, 10 000 observations were run per scenario before using the procedure and the means obtained from these observations were considered as true means. The Pareto ranking algorithm (Algorithm 1) was subsequently used to determine the true relaxed Pareto set, which would be used to validate the estimated relaxed Pareto set to be obtained by the $\mathcal{MMY}$ procedure. The same true relaxed Pareto set is used in this study to validate the MOOSolver implementation of the procedure.

6.2.1 Specifics of the BAP solved in Yoon (2018)

It was assumed for this variant of the BAP that a total of, say, n_t buffer spaces was available to be arranged in the 4 niches considered for the BAPs in this study (*i.e.* the number of machines in series for all the BAPs in this study is $m = 5$, see Figure 3.3). The problem had therefore a total number of $\binom{n_t+m-2}{m-2}$ feasible solutions. With n_t selected as 6 in this case, the total number of potential solutions amounted to $\binom{9}{3} = 84$. From the 84, 10 were selected and the true mean values to their respective objective functions were obtained in the manner described in the previous paragraph. Because the total number of storage spaces used in this case was known and kept constant, objective W_P of Chapter 3 was calculated differently here. Instead of being the sum of buffer spaces used in a solution like in Chapter 3, here it is rather the average work-in-progress rate; still to be minimised. The selected IZ values for T_R and W_P were 0.2 and 0.12, respectively. Additional information about the problem is summarised in Table 6.2, and observations on the BAP model were made over a period of 100 days. The system was treated as a terminating system.

Table 6.2: Machines information for the BAP.

Machine	**1**	**2**	**3**	**4**	**5**
Processing times (μ_i)	60 min	55 min	50 min	46 min	43 min
ODFs (λ_i)	100	100	100	100	100
Repair times (β_i)	120 min	120 min	120 min	120 min	120 min

6.2.2 Results and validation

The ten solutions preselected and their *estimated* true mean performances are shown in Tables 6.3 and 6.4, respectively.

Table 6.3: Selected solutions in the BAP.

System	1	2	3	4	5
(x_1, x_2, x_3, x_4)	(1,1,1,3)	(1,1,2,2)	(1,2,1,2)	(1,2,2,1)	(2,1,1,2)
System	6	7	8	9	10
(x_1, x_2, x_3, x_4)	(2,1,2,1)	(2,2,1,1)	(3,1,1,1)	(1,3,1,1)	(1,1,3,1)

Table 6.4: Estimated true means in the BAP.

System	1	2	3	4	5
(T_R, W_P)	(16.42, 1.65)	(16.66, 1.76)	(16.97, 2.02)	(17.14, 2.13)	(17.04, 2.42)
System	6	7	8	9	10
(T_R, W_P)	(17.28, 2.55)	(17.48, 2.83)	(17.23, 3.19)	(17.18, 2.37)	(16.73, 1.86)

From Table 6.4, the true relaxed Pareto set can thus be obtained following the definition in Section 4.2.2.1. Figure 6.3 illustrates the true relaxed Pareto set for the BAP in this section. It follows from the figure that, except Systems 5 and 8, every other system in the set can be considered as relaxed Pareto-optimal.

With this information at hand, the $\mathcal{MMY}$ in MOOSolver could then be tested for validation. The result obtained by the suite is summarised in Table 6.5. The table shows that every system in the set preselected can be considered as relaxed Pareto-optimal except for Systems 5 and 8. The status column indicates, in effect, that the necessary number of observations was made for all considered systems. The reader can also observe that the highest number of observations made on a system during the

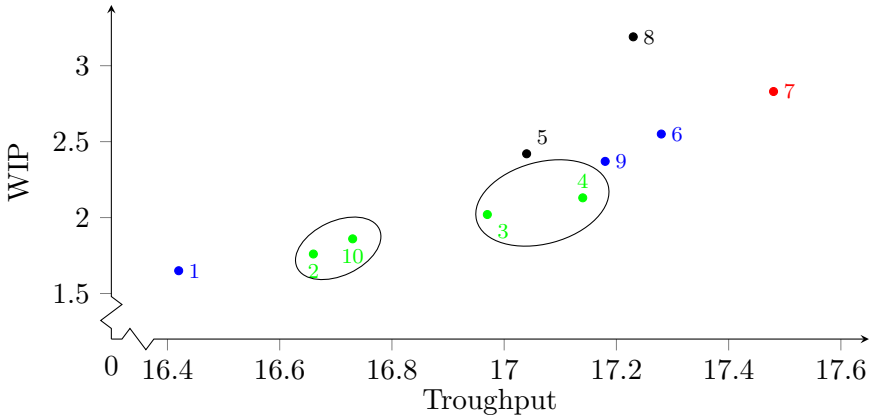


Figure 6.3: The true relaxed Pareto set for the BAP (Yoon, 2018).

execution of the procedure was 143 (made on System 9), which is way less than 10 000. With the lowest number of observations made being 15, one can only imagine how inefficient using an LFC-based approach (see Section 3.1.3) would be in this case. The result in Table 6.5 is in effect valid when compared to the true relaxed Pareto set. This, therefore, validates the implementation of the $\mathcal{MMY}$ procedure in MOOSolver and shows the efficiency of the procedure. Consequently as a result, the COM-Interface IPC procedure (Figure 5.2) is also valid.

Table 6.5: BAP result as obtained by MOOSolver.

Solution	T_R	W_P	$S^2_{T_R}$	$S^2_{W_P}$	**Rank**	**Runs**	**Status**	**ID**
(1,1,1,3)	16.44	1.49	0.06	0.04	0	36	0	1
(1,1,2,2)	16.69	1.62	0.07	0.05	0	37	0	2
(1,1,3,1)	16.76	1.71	0.06	0.09	0	37	0	10
(1,2,1,2)	17.00	1.85	0.08	0.07	0	41	0	3
(2,1,1,2)	17.08	2.18	0.07	0.05	2	77	0	5
(1,2,2,1)	17.17	1.96	0.08	0.09	0	41	0	4
(3,1,1,1)	17.18	2.82	0.08	0.05	3	15	0	8
(1,3,1,1)	17.22	2.14	0.07	0.15	0	143	0	9
(2,1,2,1)	17.31	2.32	0.07	0.07	0	41	0	6
(2,2,1,1)	17.50	2.55	0.08	0.10	0	41	0	7

6.3 Chapter summary

In this chapter, the author validated the multi-objective optimisation suite developed in the previous chapters. To do this, standard deterministic MOO test problems were used as well as a variant of the buffer allocation problem in the literature with known solutions. The BAP is a stochastic problem.

In the next chapter, the MOO suite is used to solve well know problems from the literature as well as in practice and the solution approach proposed in Section 4.1.1 is tested.

Chapter 7

Case studies

In the previous chapter, the optimisation suite was successfully validated using standard MOO deterministic test problems as well as a variant of the buffer allocation problem, which is a MOO stochastic problem.

In this chapter, the author uses the suite to solve MOSO case studies. The purpose of this chapter is to demonstrate that MOOSolver can, in effect, handle problems that are modelled within the framework described in Chapter 2. The first problem is the same buffer allocation problem presented in Chapter 3 where it was solved with the TPS built-in optimisation suite whereas the second problem is a known inventory-management problem. The problems are solved using the HRH approach proposed in Section 4.1.1.

7.1 The buffer allocation problem

This problem was described in Section 3.2 where it was solved using the TPS built-in optimisation suite. In this section, the problem is solved again using the MOOSolver library. The results obtained by the MOO suite are then compared to those obtained in Chapter 3.

7.1.1 Specifics of the problem solved

All the specifics of the BAP in Chapter 3 are also used here. The small difference, however, is in the feasible solution space. Whereas in Chapter 3 the solution space was made $1 \leq x_i \leq 20$, here it is made $0 \leq x_i \leq 20$. In effect, in the original version

(Bekker, 2012) of the BAP used in this study, Bekker (2012) considered the possibility of having no buffer at all at x_i. Using the optimisation suite in TPS, however, the GA could not, for some reason, handle such a space and kept returning error messages. The author had to, therefore, adjust the space slightly from $0 \leq x_i \leq 20$ to $1 \leq x_i \leq 20$. Despite the small difference, it is believed that the results from both chapters can still be compared.

The MOO CEM parameters were selected as follows: The maximum evaluations value was made 5 000, the number of outer loops was made 100, epsilon was made 1 and the population size was made 100.

The number of observations per solution was, arbitrarily, selected as 15 (assumed to be high enough) and observations were made over a simulation period of 10 days (all similar to the BAP in Chapter 3). The system was treated as a terminating system.

7.1.2 Results and discussion

After running the MOO CEM, the approximate Pareto front shown in Figure 7.1 was obtained with a total of 78 solutions. This section will be divided into two parts. In the first part, the author will compare the results obtained by MOOSolver with those in Section 3.2.3 while in the second part, the results obtained by the MOO suite will be analysed further using the proposed interactive HRH approach.

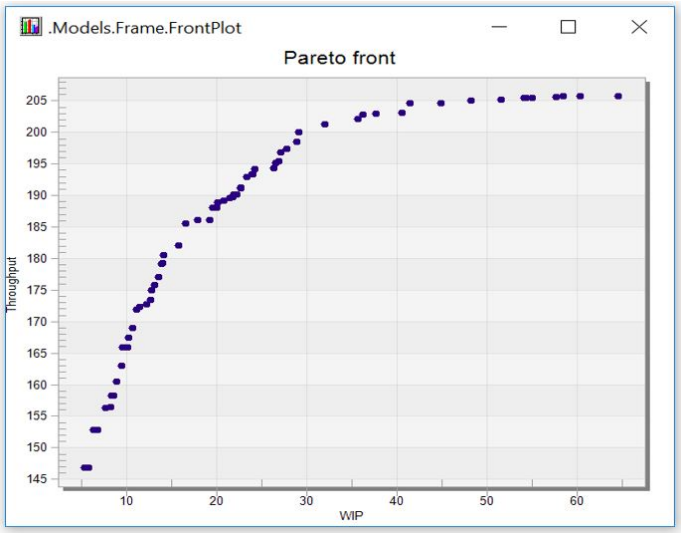


Figure 7.1: Pareto front obtained by MOOSolver for the BAP.

7.1.2.1 Pareto vs. Weighting sum approaches

Table 7.1: Comparing solutions obtained in Chapter 3 with similar and better solutions from the approximate Pareto set obtained by MOOSolver.

Weighting sum solutions	**Pareto solutions**
$\mathbf{x}_1 = (5,5,4,2)\ T_R = 64.74\ W_P = 16$	$\mathbf{x}_{11} = (4,4,3,2)\ T_R = 180.60\ W_P = 13$ $\mathbf{x}_{12} = (5,4,3,2)\ T_R = 182.07\ W_P = 14$ $\mathbf{x}_{13} = (5,5,3,2)\ T_R = 185.53\ W_P = 15$
$\mathbf{x}_2 = (6,6,4,2)\ T_R = 87.77\ W_P = 18$	$\mathbf{x}_{21} = (6,5,5,2)\ T_R = 188.93\ W_P = 18$ $\mathbf{x}_{22} = (6,6,4,3)\ T_R = 189.73\ W_P = 19$
$\mathbf{x}_3 = (10,8,7,4)\ T_R = 108.80\ W_P = 29$	$\mathbf{x}_{31} = (8,8,5,4)\ T_R = 196.80\ W_P = 25$ $\mathbf{x}_{32} = (9,5,8,4)\ T_R = 197.47\ W_P = 26$ $\mathbf{x}_{33} = (9,5,7,6)\ T_R = 198.47\ W_P = 27$ $\mathbf{x}_{34} = (8,10,6,4)\ T_R = 200.00\ W_P = 28$ $\mathbf{x}_{35} = (10,8,8,4)\ T_R = 201.33\ W_P = 30$
$\mathbf{x}_4 = (13,10,9,7)\ T_R = 155.93\ W_P = 39$	$\mathbf{x}_{41} = (1,1,1,1)\ T_R = 146.90\ W_P = 4$ $\mathbf{x}_{42} = (1,2,1,1)\ T_R = 152.80\ W_P = 5$ $\mathbf{x}_{43} = (2,2,1,1)\ T_R = 156.27\ W_P = 6$ $\mathbf{x}_{44} = (2,3,1,1)\ T_R = 161.20\ W_P = 7$ $\mathbf{x}_{45} = (3,3,2,1)\ T_R = 169.80\ W_P = 9$ $\mathbf{x}_{46} = (3,3,2,2)\ T_R = 172.27\ W_P = 10$
$\mathbf{x}_5 = (18,11,9,12)\ T_R = 205.60\ W_P = 50$	$\mathbf{x}_{51} = (13,9,6,6)\ T_R = 202.07\ W_P = 34$ $\mathbf{x}_{52} = (13,10,8,8)\ T_R = 204.67\ W_P = 39$ $\mathbf{x}_{53} = (14,13,9,10)\ T_R = 205.20\ W_P = 46$

The BAP in this chapter was first presented in Section 3.2, where it was solved as a large-scale SO problem using the TPS built-in optimisation suite, which uses a weighting sum approach in solving MOO problems. Having analysed the results in that section, it was proposed that using a MOO technique that utilises the Pareto approach would be a better choice than using one that utilises a weighting sum one. In this section, the benefit of using the Pareto approach is demonstrated by comparing the results obtained by MOOSolver to those obtained in Section 3.2.

To compare the results in both chapters, the author selected from the approximate Pareto set obtained by MOOSolver, solutions whose T_R or W_P values were similar to those obtained in Table 3.5 of Chapter 3. Because the Pareto approach outputs a large number of solutions, every solution obtained by the weighting sum approach

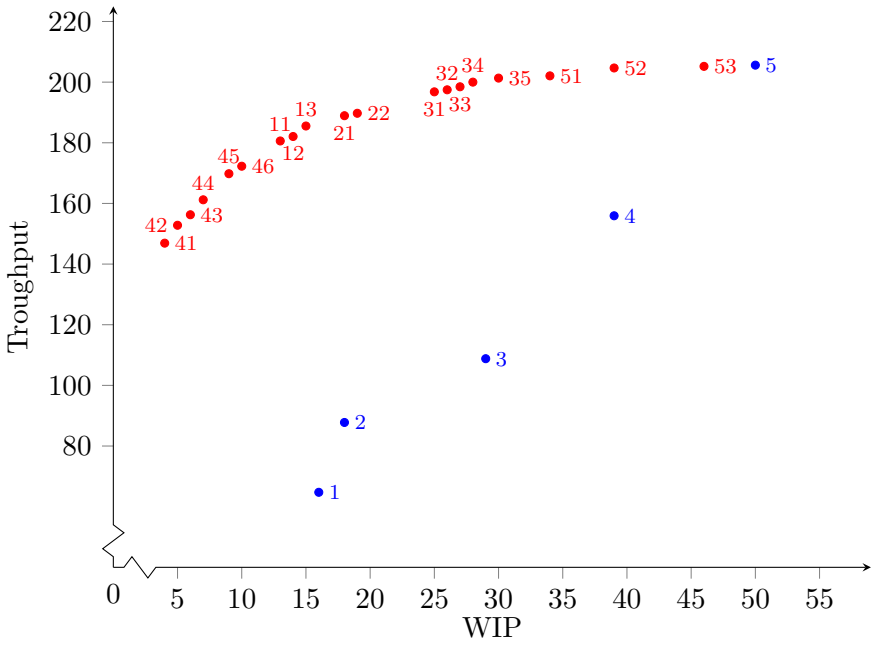


Figure 7.2: Visualisation of the comparison made in Table 7.1.

in Table 3.5 is placed against similar solutions obtained by the Pareto approach with respect to either the solutions' T_R or W_P performances. The idea here is to showcase possible missed opportunities in the form of Pareto solutions obtained by the Pareto approach, which the weighting sum approach could not find, given the weights that were selected. Table 7.1 illustrates the comparison, while Figure 7.2 provides a better visualisation of the comparison. (Note that the solutions were "numbered" to improve their visualisation.)

The comparison in Table 7.1 (and Figure 7.2) shows the superiority of the Pareto approach in this case. Even though the approach does not give decision-makers the convenience of selecting weights that exhibit their preferences, it provides them instead with a set of relatively high-quality solutions they can choose from, which in the author's assessment, is a better option to have (better than the weight selection convenience). In addition, it can be argued that decision-makers may know their preferences better once they actually know the alternatives they have.

In the next section, it is assumed that having seen the approximate Pareto set, a decision-maker can easily select from it a subset they prefer. This set is subsequently analysed further for more accurate estimates and a guaranteed correct selection.

7.1.2.2 Further analysis of the MOOSolver results

In this section, the results obtained by MOOSolver using the MOO CEM are analysed further with the $\mathcal{MMY}$ procedure. The large-scale approach proposed in Section 4.1.1 is therefore used. Just as a reminder to the reader, the goal of the approach is to reduce the large-scale MOSO problem into a small-scale one that preselects a number of good solutions (as obtained by the metaheuristic). These preselected solutions or scenarios are those the decision-maker has a biased interest towards, relative to the other solutions in the approximate Pareto set. It is therefore assumed that the decision-maker desires to have better result estimates for these solutions as well as a guarantee of making a correct selection out of the set.

Two experiments will be conducted in this section in order to test the interactive HRH approach. In the first experiment, it is assumed that the decision-maker has an interest in solutions that achieve a W_P performance no larger than 50 and a T_R value of at least 180. Figure 7.3 illustrates the decision-maker's bias.

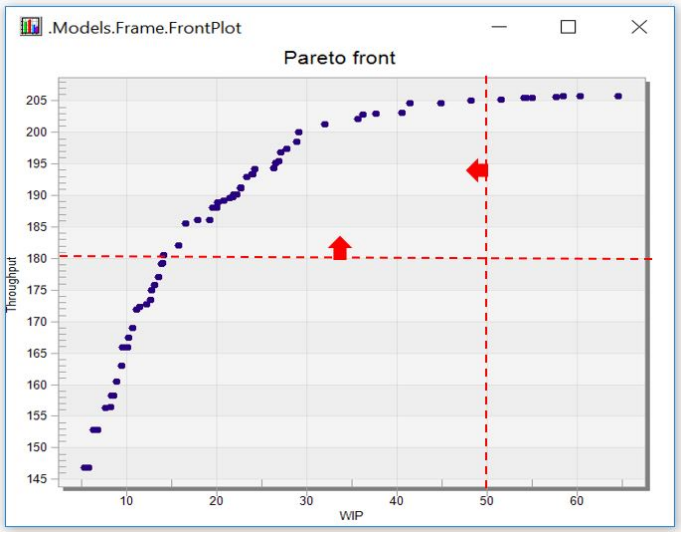


Figure 7.3: The decision-maker's assumed preference in the first experiment for the interactive HRH approach in the BAP.

From the set of possible solutions from which the decision-maker can choose, therefore, it is assumed that those presented in Table 7.2 are the ones they select as scenarios to be analysed further by the $\mathcal{MMY}$ procedure in MOOSolver.

Table 7.2: Decision-maker's preselected scenarios from Figure 7.3 and their respective results before using the $\mathcal{MMY}$ procedure.

ID	Solution	T_R	W_P
1	(4,4,3,2)	180.60	13
2	(5,5,3,2)	185.53	15
3	(5,6,4,4)	188.07	19
4	(6,6,4,4)	190.20	20
5	(8,7,4,4)	194.20	23
6	(8,9,5,4)	197.13	26
7	(9,5,7,6)	198.47	27
8	(13,9,6,6)	202.07	34
9	(16,9,9,5)	203.07	39
10	(14,13,9,10)	205.20	46

With this selection, it is also assumed that the decision-maker chooses IZ values δ^* of 3.5 and 2 for throughput (T_R) and total sum buffers (W_P), respectively. The $\mathcal{MMY}$ procedure is now ready to be run after specifying explicitly to the suite via the wizard ($\mathcal{MMY}$ Parameters group box) that the number of scenarios selected is 10 (see Figure 5.6). Table 7.3 illustrates the results returned by MOOSolver for this first experiment.

Table 7.3: $\mathcal{MMY}$ results by MOOSolver in the first experiment for the interactive HRH approach in the BAP.

Solution	T_R	W_P	$S^2_{T_R}$	$S^2_{W_P}$	Rank	Runs	Status	ID
(14,13,9,10)	198.28	46	138.84	0.00	0	217	0	10
(16,9,9,5)	196.53	39	121.43	0.00	0	191	0	9
(13,9,6,6)	195.69	34	119.67	0.00	0	189	0	8
(9,5,7,6)	190.80	27	115.44	0.00	1	206	0	7
(8,9,5,4)	191.47	26	105.73	0.00	0	206	0	6
(8,7,4,4)	188.67	23	93.02	0.00	0	173	0	5
(6,6,4,4)	185.25	20	113.13	0.00	0	203	0	4
(5,6,4,4)	184.23	19	100.26	0.00	0	210	0	3
(5,5,3,2)	179.28	15	91.19	0.00	0	231	0	2
(4,4,3,2)	175.36	13	92.69	0.00	0	146	0	1

The first difference that can be noticed in Table 7.3 when compared to Table 7.2 is the different estimate values for T_R. Table 7.3 provides, of course, better estimates for the selected solutions. Thus, before going further, the decision-maker can already discard solutions whose estimates now fall outside their preferred limits (illustrated here in Figure 7.3). Observe for example that Solutions 1 and 2 have now T_R values that are less than the user-preferred threshold in this first experiment. Moreover, according to the **Rank** column of Table 7.3, Solution 7 should not be selected by the decision-maker either as it is now dominated (in the relaxed Pareto set sense). The reader can also observe that the variance values in column $S^2_{W_P}$ are all 0. This is because W_P in this variant of the BAP is calculated as the total sum of buffers used in a solution, which remains constant in every observation made on the solution. This is in contrast to the W_P in Chapter 6 which is calculated as the average WIP in the system and thus varies with every observation, as shown in the variance column ($S^2_{W_P}$) of Table 6.5. Also observe the relatively high number of runs for this experiment in the **Runs** column, probably indicating that the IZ values selected in this case are relatively small. Nonetheless, the values in the **Status** column inform the decision-maker that correct selection with accuracy of at least 90% is guaranteed (see Section 5.2.2).

In the second experiment, the decision-maker's assumed preference is made as shown in Figure 7.4. The IZ values are kept the same as in the first experiment ($\delta^* = [3.5, 2]$) and the preselected scenarios are as shown in Table 7.4.

Table 7.4: Decision-maker's preselected scenarios from Figure 7.4 and their respective results before using the $\mathcal{MMY}$ procedure.

ID	**Solution**	T_R	W_P
1	(7,6,3,5)	190.93	21
2	(8,10,6,4)	200.00	28
3	(8,9,5,4)	197.13	26
4	(8,6,5,4)	194.47	23
5	(8,8,5,4)	196.80	25
6	(8,6,4,4)	193.40	22
7	(8,7,4,4)	194.20	23

The second experiment tells the decision-maker that, only Solutions 2, 3 and 5 in Table 7.4 are within their preferred limits (Figure 7.4) with respect to the solutions

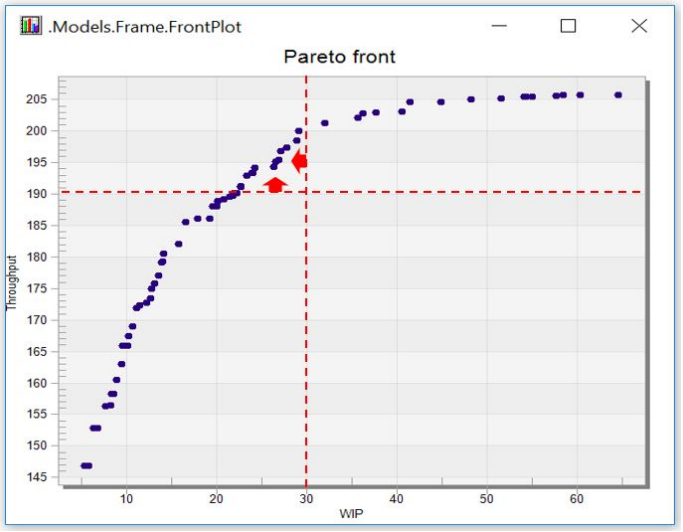


Figure 7.4: The decision-maker's assumed preference in the second experiment for the interactive HRH approach in the BAP.

Table 7.5: $\mathcal{MMY}$ results by MOOSolver in the second experiment for the interactive HRH approach in the BAP.

Solution	T_R	W_P	$S^2_{T_R}$	$S^2_{W_P}$	**Rank**	**Runs**	**Status**	**ID**
(8,10,6,4)	192.42	28	122.78	0.00	0	191	0	2
(8,9,5,4)	191.54	26	104.28	0.00	0	191	0	3
(8,8,5,4)	190.51	25	92.92	0.00	0	162	0	5
(8,6,5,4)	189.14	23	87.39	0.00	0	155	0	4
(8,7,4,4)	188.53	23	86.74	0.00	0	155	0	7
(8,6,4,4)	188.33	22	98.95	0.00	0	155	0	6
(7,6,3,5)	185.14	21	98.30	0.00	0	155	0	1

new estimate values. Note also that this is all with respect to the chosen IZ values (in both experiments) as they are the parameters that control how many observations are necessary per solution in order to guarantee correct selection. This means that had the decision-maker chosen larger IZ values, perhaps the discarded solutions (1, 4, 6 and 7) would have remained within the desired limits. But in this case they are not. Nonetheless, they still do form part of correct selection from what is indicated in the **Rank** column. In effect, for this experiment, all the preselected scenarios are relaxed Pareto-optimal.

This concludes the BAP case study. In this section, the author was able to demonstrate the relevance of the solution approach proposed in this thesis as well as the benefit of using the Pareto approach when solving MOO problems. In the following section, another problem is considered and the interactive HRH approach is tested further.

7.2 The (s, S) inventory problem

In this section, a variant of the well-known (s, S) inventory problem described in detail by Bashyam & Fu (1998), is considered. The problem is adjusted for the purpose of this study (based on the work of Bekker & Aldrich (2011)) to conform with the MOSO framework described in Chapter 2.

Consider a system in which a single, discrete commodity is sold to customers who arrive according to a Poisson process, with a rate of arrival λ. The inter-arrival times are thus exponentially distributed with mean β. Assume the demand of customer c is distributed according to a given distribution and all demands are processed according to an exponential distribution with a mean of μ. The manager of this process will wait until the inventory is consumed below the reorder point s, and then reorder a quantity S. A lead time before delivery follows a given distribution, during which customers still demand the commodity. Figure 7.5 shows typical characteristics of the process described.

Table 7.6: Notation for the (s, S) inventory problem.

I_t	the inventory level at time t when customer c arrives;
S_L	the service level;
D_c	the number of units demanded by customer c;
N_c	the total number of customers arriving in period $[0, T]$;
I_c	the total inventory cost during period $[0, T]$;
S_c	the number of units that cannot be supplied to customer c.

Now, consider the notation in Table 7.6. When the inventory reaches zero and the replenishment has not arrived, a stockout period follows during which customers cannot be served. All demands during that period are considered lost sales, which must be avoided from a profit point of view.

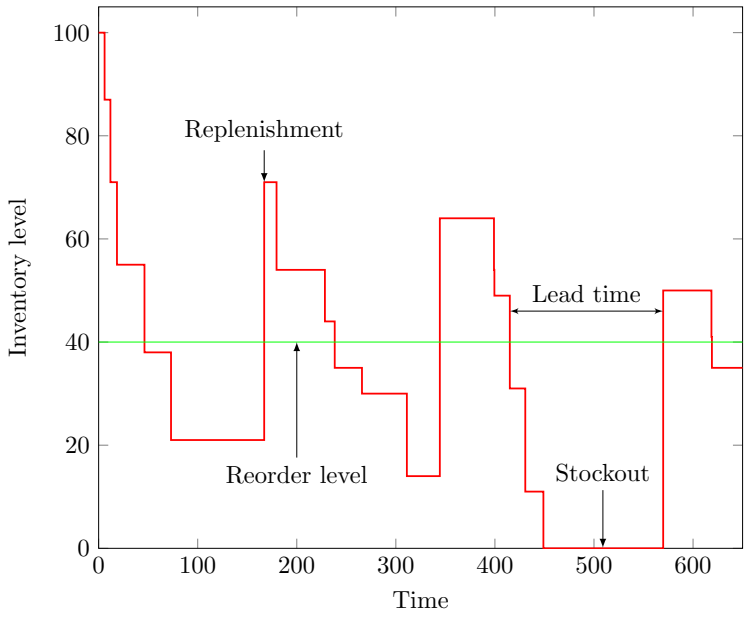


Figure 7.5: Typical characteristics of the (s, S) inventory process (Bekker, 2012).

The service level is given by

$$S_L = \frac{\sum_{c=1}^{N_c} D_c - \sum_{c=1}^{N_c} |S_c|}{\sum_{c=1}^{N_c} D_c} \cdot 100\% \tag{7.1}$$

while stockout follows from

$$S_c = 0 : I_t \geq D_c, \tag{7.2}$$

$$S_c = I_t - D_c : I_t < D_c. \tag{7.3}$$

It is assumed that the holding area is infinite and the supplier reliable, *i.e.* each time an order is placed, the correct number of units is received after the lead time has elapsed. If $I_t \geq D_c$ when customer c places an order, the customer is satisfied and considered a happy customer otherwise they are dissatisfied and considered a lost opportunity. Backlogs are not allowed. When the replenishment quantity arrives, I_t is adjusted according to $I_t + S$. The decision variables in this problem are s and S, and the performance measures (objectives) are the total inventory cost I_c over period $[0, T]$ and the service level S_L.

7.2.1 Specifics of the problem solved

The specifics of the problem considered in this study are as follows: The arrival rate is $\lambda = 3$ customers every hour, the inter-arrival time has thus a mean of $\beta = 20$ min. The demand of customer c is distributed according to $\lfloor Weibull(1, 8) \rfloor$ and demands are processed according to $\mu = 18$ min. Lead time before delivery follows a triangular distribution (14,12,20) in hours. Additionally, carrying inventory incurs a cost. Holding cost is taken as ZAR 10/unit/unit time, and the administration fee of a reorder is taken as ZAR 100.

The MOSO question is thus: Given that the decision variables are arbitrarily limited as: $0 < s \leq 500$ and $0 < S \leq 500$. For what values of s and S will I_c be at a minimum while S_L is at maximum, in the presence of element ξ caused by customers' arrivals, demands and order lead times. The problem can be formulated as

$$\text{Minimise } E[I_c((s, S), \xi)],$$
$$\text{Maximise } E[S_L((s, S), \xi)]$$
$$\text{Subject to}$$
$$0 < s \leq 500 \text{ and } 0 < S \leq 500.$$

The manager of this process will want to service all customers while carrying as little inventory as possible. Note that the objectives are conflicting, and their values are measured in different units.

The problem was modelled in TPS and solved using MOOSolver. The $[0, T]$ period was taken to be 50 days while the number of observations per solution was made, arbitrarily, 10 (10 was assumed to be high enough). The simulation model was treated as a terminating system.

The MOO CEM parameters were selected as follows: The maximum evaluations value was made 1 800, epsilon was made 2.5, the number of outer loops was made 10 and the population size was made 30.

7.2.2 Results and discussion

After running the MOO CEM, the approximate Pareto front shown in Figure 7.6 was obtained with a total of 317 solutions. In this section, two experiments will be con-

sidered where the IZ values of the OFs will be varied. It will be assumed that the decision-maker's preference with respect to the approximate Pareto set obtained will remain the same in both experiments. The goal here is again to test the proposed interactive HRH approach, but this time, through the analysis of the change in the results obtained by the $\mathcal{MMY}$ procedure when the IZ values are varied for the same set of selected solutions.

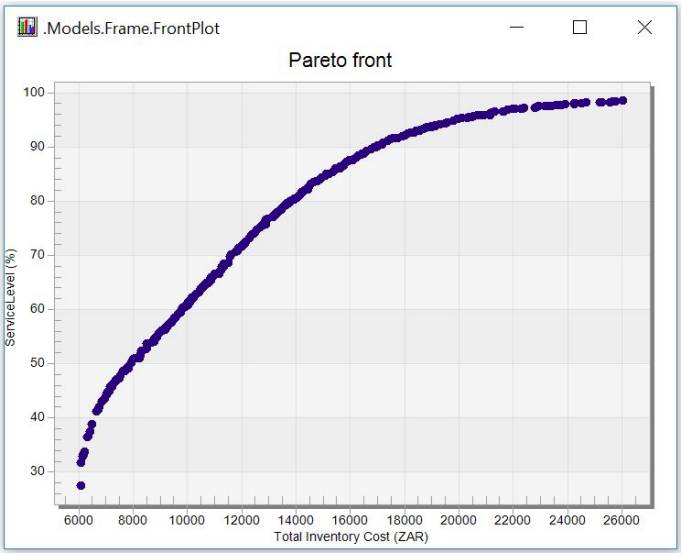


Figure 7.6: Pareto front obtained by MOOSolver for the (s, S) inventory problem.

The decision-maker's preference in this problem is as shown in Figure 7.7. In other words, they are only interested in solutions of which the total inventory cost does not exceed ZAR 20 000 and with service level at least 85%. The specific solutions selected from the subset are shown in Table 7.7.

In the first experiment, the IZ values are chosen as $\delta^* = [350, 2]$ for the total inventory cost and the service level, respectively. The results obtained after running the $\mathcal{MMY}$ procedure are shown in Table 7.8.

The results obtained in this experiment indicate that all the solutions selected are relaxed Pareto-optimal. Additionally, except for Solution 7 whose S_L estimate now falls outside the limits illustrated in Figure 7.7, all other solutions are within the preference limits set by the decision-maker. It can also be observed that it did not take many observation runs in this experiment to guarantee correct selection; this is probably because the IZ values selected were relatively large.

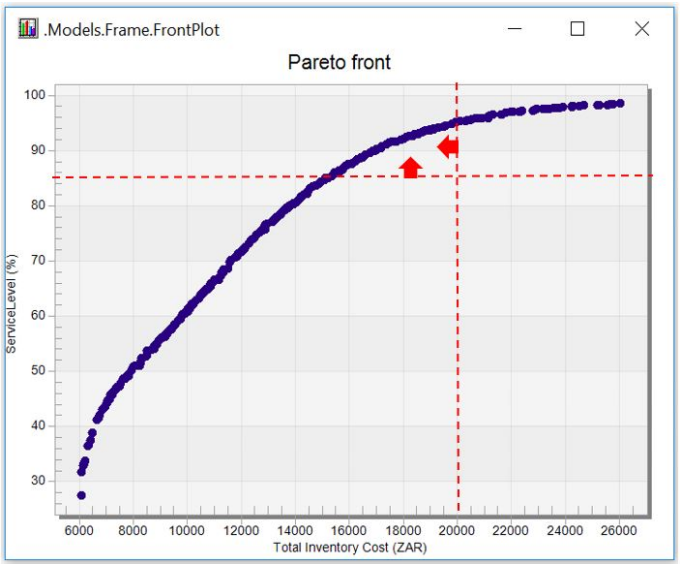


Figure 7.7: The decision-maker's assumed preference in the two experiments for the interactive HRH approach in the (s, S) inventory problem.

Table 7.7: Decision-maker's preselected scenarios from Figure 7.7 and their respective results before using the $\mathcal{MMY}$ procedure.

ID	**Solution**	I_c	S_L
1	(400, 415)	19 983.37	95.23
2	(319, 384)	16 082.11	87.63
3	(340, 350)	16 510.60	88.85
4	(388, 366)	18 677.84	93.30
5	(387, 393)	19 116.79	93.95
6	(349, 381)	17 220.49	90.78
7	(304, 366)	15 229.99	85.12
8	(368, 387)	18 119.63	92.54
9	(349, 353)	16 923.18	89.98

In the second experiment, the IZ values are made smaller. The respective values of 250 and 0.5 are now selected for the total inventory cost and the service level, respectively. The results obtained by the MOO suite are presented in Table 7.9.

The interesting fact about the results in Table 7.9 is that despite the significant increase in the number of runs for most solutions, the new estimates obtained are not

Table 7.8: $\mathcal{MMY}$ results by MOOSolver in the first experiment for the interactive HRH approach in the (s, S) inventory problem.

Solution	I_c	S_L	$S^2_{I_c}$	$S^2_{S_L}$	**Rank**	**Runs**	**Status**	**ID**
(400, 415)	19 916.67	94.99	107 867.70	1.07	0	22	0	1
(387, 393)	19 074.36	93.74	89 341.15	0.98	0	19	0	5
(388, 366)	18 611.53	93.15	85 465.04	0.93	0	18	0	4
(368, 387)	18 115.15	92.24	113 963.96	1.22	0	22	0	8
(349, 381)	17 254.48	90.59	61 037.42	1.33	0	18	0	6
(349, 353)	16 889.57	89.79	78 010.87	1.14	0	18	0	9
(340, 350)	16 477.48	88.66	62 945.17	1.33	0	24	0	3
(319, 384)	15 999.15	87.32	64 612.75	2.24	0	24	0	2
(304, 366)	15 247.81	84.97	57 848.71	2.62	0	17	0	7

Table 7.9: $\mathcal{MMY}$ results by MOOSolver in the second experiment for the interactive HRH approach in the (s, S) inventory problem.

Solution	I_c	S_L	$S^2_{I_c}$	$S^2_{S_L}$	**Rank**	**Runs**	**Status**	**ID**
(400, 415)	19 792.22	94.60	159 104.45	1.23	0	105	0	1
(387, 393)	18 900.63	93.31	142 068.94	1.34	0	118	0	5
(388, 366)	18 318.51	92.50	173 160.90	1.67	0	186	0	4
(368, 387)	17 955.93	91.72	136 602.77	1.77	0	131	0	8
(349, 381)	17 043.05	89.70	104 588.56	1.96	0	145	0	6
(349, 353)	16 706.73	89.11	114 825.09	2.00	0	160	0	9
(340, 350)	16 309.58	88.03	99 679.23	2.15	0	171	0	3
(319, 384)	15 898.34	86.61	87 061.41	1.99	0	124	0	2
(304, 366)	15 153.71	84.38	61 931.82	2.10	0	44	0	7

very different from the ones in the previous experiment. In fact, the relaxed Pareto set is the same. The reader can see, nonetheless, how a change in IZ values influences the computational effort in order to guarantee correct selection. Just for interest, the IZ values are reduced further in an attempt to reach the simulation budget limit in MOOSolver and illustrate the kind of output results the decision-maker can, typically, expect in such a situation. To do so, the IZ values are made 100 and 0.25 for I_c and S_L, respectively. Given the number of observation runs that were needed in the second experiment with $\delta^* = [250, 0.5]$, this should make MOOSolver reach its current limit.

The results of this test are shown in Table 7.10.

Table 7.10: $\mathcal{MMY}$ results by MOOSolver in the (s, S) inventory problem for, relatively, very small IZ values.

Solution	I_c	S_L	$S^2_{I_c}$	$S^2_{S_L}$	**Rank**	**Runs**	**Status**	**ID**
(400, 415)	19 728.34	94.48	138 262.73	1.01	0	250	1	1
(387, 393)	18 845.56	93.34	141 759.60	1.38	0	250	2	5
(388, 366)	18 301.64	92.47	167 480.00	1.73	0	250	2	4
(368, 387)	17 915.15	91.70	129 755.29	1.76	0	250	2	8
(349, 381)	17 033.30	89.71	106 781.61	2.13	0	250	1	6
(349, 353)	16 708.90	89.13	109 951.38	2.06	0	250	2	9
(340, 350)	16 298.12	88.03	101 501.87	2.09	0	250	1	3
(319, 384)	15 864.93	86.55	86 376.63	2.10	0	250	1	2
(304, 366)	15 137.62	84.33	74 629.82	2.22	0	130	0	7

The results of Table 7.10 are a good example of a case where the number of simulation runs required to guarantee correct selection exceeds the budget. Except for Solution 7, every other solution in Table 7.10 needs additional observation runs, given the relatively small choice of IZ values. This is indicated by the **Runs**, as well as the **Status** columns. When the value in column **Runs** is 250 for a solution (250 is the current limit allowed by MOOSolver) and the **Status** column (of the same solution) has a value greater than 0, this is an indication that the budget limit was reached before the procedure could ensure the "statistical soundness" of the given solution (with respect to the IZ values selected) *relative* to the other solutions in the set. In effect, the procedure does pairwise comparisons whereby each solution is compared to every other solution in the set. Thus, in order for the **Status** column to show a value of 0 (indicative of guaranteed correct selection) for a particular solution, the solution must "pass" all the pairwise comparison "tests". In Table 7.10, only Solution 7 achieves this. Solution 7 is therefore, for argument sake, the only solution that can be selected with a guarantee of at least 90% that its estimated values for I_c and S_L are what they appear to be in the table *in relation* to all the other estimated values (again, given the selected IZ values in this case). As was mentioned in Section 5.2.2, the values 1 and 2 in the **Status** column have technical meanings in the implementation context of the $\mathcal{MMY}$

procedure for MOOSolver. It is sufficient for the user or the decision-maker to know that a value greater than 0 in this column indicates "no correct selection guaranteed".

7.3 Chapter summary

In this chapter, the author solved two well-known problems in the literature, namely, the buffer allocation problem and the (s, S) inventory problem. These problems were solved using the multi-objective optimisation suite, namely MOOSolver, developed in the previous chapters. In the BAP case, it was demonstrated using the results obtained by the suite that the algorithm used for large-scale problems provides higher quality solutions than the current approach used in Tecnomatix Plant Simulation. And in the case of both problems, namely the BAP and the (s, S) inventory problem, the author demonstrated the relevance of the approach proposed in this study for large-scale MOSO problems.

Chapter 8

Summary and conclusions

This chapter concludes the research presented in this document. The chapter is divided into three parts; in the first part, a summary of the thesis is presented. In the second part, the shortcomings experienced during the project are described while in the last part, proposals for future work are made.

8.1 Thesis summary

The aim in this study was to develop a MOO optimisation suite for Tecnomatix Plant Simulation in order to strengthen the simulation software capabilities in handling stochastic MOO problems.

To do this, the literature was studied first in **Chapter** 2. The idea here was to gain as much information as needed regarding the existing solution approaches for stochastic MOO problems. It was found that the stochastic optimisation field is, in effect, vast, with many existing solution techniques. Nonetheless, it was also found that many techniques in the literature were not perfect and that there was potential for improvements and contributions, especially in the MOO context.

In **Chapter** 3, the limitations of Tecnomatix Plant Simulation were demonstrated by solving two problems in both the small-scale and the large-scale SO contexts. The results obtained served as a confirmation that there was, indeed, a gap between what was being done by the software and what exists in the literature. This motivated further the need for developing the MOO suite that is the main subject of this thesis.

The conceptual design of the MOO suite as well as its proposed solution approach

was subsequently described in **Chapter** 4. The selected algorithms used in the proposed solution approach were described in full detail in this chapter.

In **Chapter** 5, the development and the implementation of the optimisation suite was presented. The focus of the chapter was placed on the integration process used to merge the suite and the discrete-event simulation software. Additionally, the user-interface developed specifically for TPS was described in great detail.

In **Chapter** 6, the product developed in the last two chapters was validated using problems that have known solutions. The goal here was, essentially, twofold. One, to ensure that the inter-process communication procedures used in the previous chapter was successfully implemented and two, to also ensure that the selected algorithms were implemented correctly. The results obtained in this chapter showed that the MOO suite development and implementation were valid.

Having a valid MOO tool for stochastic problems, it was now to be tested using well-known problems in practice and in the literature; this was done in **Chapter** 7. The goal of the chapter was also to test the solution approach proposed in this study and its relevance; the goal was successfully achieved.

8.2 Thesis shortcomings

The initial idea for the thesis was the development of a hybrid MOO suite that included more than one metaheuristic. In effect, the word hybrid was understood in this case as the "bringing together" of different, separate, metaheuristics to be implemented within the optimisation suite and be used in order to compare their respective results to each other. Having done the literature study, however, more was learned concerning the hybrid metaheuristic field and the aims of the study were subsequently adjusted from *using many metaheuristics* to *combining metaheuristics with additional techniques* in order to make the optimisation suite an effective tool for MOSO problems.

Nonetheless, the author had considered using more than one metaheuristic in the context of having each of them handle problems of different decision variables nature. This could not be done, as can be observed in the study, due to time constraints. In effect, the optimisation suite uses a metaheuristic that specialises in the optimisation of problems with continuous decision variables. But as was mentioned in this study also (Chapter 2), metaheuristics are flexible algorithms and they can be used for various

kinds of problems without the need to be modified greatly. This was thus a perfect example of this, as the selected metaheuristic (Chapter 4) was used to solve problems with discrete decision variables (Chapter 7) and yet still achieved quality results. It is nonetheless important to, at least, explore the possibility of using specialised metaheuristics for discrete problems (on discrete problems) and this thesis fell short on doing so.

8.3 Future work propositions

The stochastic multi-objective optimisation field is a very interesting one with enough room for contributions and improvements.

It is the author's opinion that many solutions in the literature and in practice are imperfect. This thesis is a good illustration of this reality. And even though the work presented in this thesis made an attempt to add onto the MOSO field, it still has aspects that can/could be improved or built upon. This section looks into such aspects, which are presented as potential future works.

1. First, the literature showed that there are many existing metaheuristics (and other search mechanisms for that matter). Adding additional metaheuristics (perhaps even exploring the potential of non-metaheuristics) to the suite and providing solid rationales for doing so is thus a definite possible, valuable, contribution to add onto the works that was started by this thesis. This proposal can also be supported by the discussion in the previous section.

2. Then, multi-objective optimisation is often limited to two objectives in the literature. Exploring a possible increase in the number of objectives the suite can handle would make an interesting topic of study. This will of course pose problems of efficiency, and demand effective implementations to ensure the practicality of the solution approaches. Multi-objective optimisation is in effect by itself a computationally demanding task, and when combined with simulation, the efficiency challenge increases all the more. This is certainly not an easy one, but as the author has learned throughout this project, a true engineer thrive in the face of "limitations" and "problems". The most important things are to know the final destination and be persistent in journey.

3. Finally, better integration of rigorous statistical techniques and metaheuristics in the simulation optimisation context (*i.e.* using a LRH approach in place of the HRH proposed in this work, as was already proposed in this study) is another challenging and interesting area of study in the SO field that will certainly benefit from future contributions.

8.4 Chapter summary

This chapter provided a close to this research. A summary of the work accomplished was presented, followed by the shortcomings of the thesis. Finally, areas for potential future works based on the study done in this thesis were suggested.

References

Alon, G., Kroese, D., Raviv, T. & Rubinstein, R. (2005). Application of the cross-entropy method to the buffer allocation problem in a simulation-based environment. *Annals of Operations Research*, **134**, 137–151. 25, 42

Amaran, S., Sahinidis, N.V., Sharda, B. & Bury, S.J. (2014). Simulation optimization: a review of algorithms and applications. *Annals of Operations Research*, **12**, 301–333. 2, 11, 19, 24, 25, 32

Andradottir, S. (1998). A review of simulation optimization techniques. *Proceedings of the 1998 Winter Simulation Conference*, 158–158. 14, 19, 27

Bamporiki, T. & Bekker, J. (2017). Solving stochastic multi-objective optimisation problems with simulation. *28th Southern African Institute for Industrial Engineering (SAIIE28) annual conference proceedings*, 1–13. 3, 36

Bamporiki, T. & Bekker, J. (2018). Development of a discrete-event, stochastic multi-objective metaheuristic simulation optimisation suite for a commercial software package. *South African Journal of Industrial Engineering*, **29**, 12–25. 118

Bandyopadhyay, S., Saha, S., Maulik, U. & Deb, K. (2008). A simulated annealing-based multiobjective optimization algorithm: Amosa. *IEEE Transaction on evolutionary computation*, 269–283. 21, 22

Bashyam, S. & Fu, M. (1998). Optimization of (s, S) inventory systems with random lead times and a service level constraint. *Management Science*, **44**, 243–256. 94

Bechhofer, R. (1954). A single-sample multiple decision procedure for ranking means of normal populations with known variances. *The Annals of Mathematical Statistics*, 16–39. 15, 16, 42

Bekker, J. (2012). *Applying the cross-entropy method in multi-objective optimisation of dynamic stochastic systems*. Ph.D. thesis, Stellenbosch: Stellenbosch University. 25, 42, 52, 87, 95

Bekker, J. & Aldrich, C. (2011). The cross-entropy method in multi-objective optimisation: An assessment. *European Journal of Operational Research*, **211**, 112–121. 25, 52, 80, 94

Bekker, J. & Viviers, L. (2008). Using computer simulation to determine operations policies for a mechanised car park. *Simulation Modelling Practice and Theory*, 613–625. 36, 37, 38

Bella, J. & McMullen, P. (2004). Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, **18**, 41–48. 26

Blum, C., Aguilera, M., Roli, A. & Sampels, M. (2008). *Hybrid Metaheuristics: An emerging approach to optimisation*. Springer. 20, 22, 23, 28

Blum, C., Puchinger, J., Raidl, G. & Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied soft computing*, **11**, 4135–4151. 2, 28

Branke, J., Deb, K. & Miettinen, K., eds. (2008). *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Springer. 45, 46

Caballero, R., Gonzalez, M., Guerrero, F., Molina, J. & Paralera, C. (2007). Solving a multiobjective location routing problem with a metaheuristic based on tabu search. application to a real case in andalusia. *European Journal of Operational Research*, 1751–1763. 23

Chankong, M. & Haimes, Y. (1983). *Multiobjective decision making: theory and methodology*. Wiley. 8

Chen, E.J. & Lee, L.H. (2009). A multi-objective selection procedure of determining a Pareto set. *Computers & Operations Research*, **36**, 1872–1879. 16

Coello Coello, C.A. (2009). Evolutionary multi-objective optimization: some current research trends and topics that remain to be explored. *Frontiers of Computer Science in China*, **3**, 18–30. 9

Coello Coello, C.A., Lamont, G.B. & Van Veldhuizen, D.A. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2nd edn. 79

de Boer, P., Kroese, D., Mannor, S. & Rubinstein, R. (2005). A tutorial on the cross-entropy method. *Annals of Operations Research*, 9–67. 24

Deb, K. (2005). *Multi-objective optimization using evolutionary algorithms*. Wiley. 8

Deneubourg, J., Pasteels, J. & Verhaeghee, J. (1983). Probabilistic behaviour in ants: A strategy of errors? *Journal of theoretical biology*, **105**, 259–271. 25

Dorigo, M., Maniezzo, V. & Colorni, A. (1996). The ant system: Optimisation by a colony of cooperative agents. *IEEE Transactions on systems, man and cybernetics-part B: Cybernetics*, **26**, 1–13. 25

Dorigo, M., Birattari, M. & Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 28–39. 20, 25, 26

Dudewicz, E.J. & Dalal, S.R. (1975). Allocation of observations in ranking and selection with unequal variances. *Sankhyā: The Indian Journal of Statistics, Series B*, 28–78. 16

Eglese, R. (1990). Simulated annealing: A tool for operational research. *European Journal of Operational Research*, **46**, 271–281. 21

Eskandari, H., Mahmoodi, E., Fallah, H. & Geiger, C. (2011). The optquest callable library. *Proceedings of the 2011 Winter Simulation Conference*, **59**, 2363–2373. 33

Falco, I., Balio, R. & Tarantino, E. (1997). An analysis of parallel heuristics for task allocation in multicomputers. *Computing*, **59**, 259–275. 31

Fu, M., Andradottir, S., Carson, J., Glover, F., Harrell, C., Ho, Y. & Robinson, S. (2000). Integrating simulation and optimisation: research and practice. *Proceedings of the 2000 Winter Simulation Conference*, 610–616. 12, 19, 46, 47, 50

Fu, M.C. (2002). Optimization for simulation: Theory vs. practice. *INFORMS Journal on Computing*, **14**, 192–215. 19, 32

GENDREAU, M. & POTVIN, J. (2010). *Handbook of metaheuristics*. Springer. 21, 22, 23, 25, 26

GERSHWIN, S. & SCHOR, J. (2000). Efficient algorithms for buffer space allocation. *Annals of Operations Research*, **93**, 117–144. 42

GLOVER, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, **13**, 533–549. 22

GLOVER, F. (1989). Tabu search-part 1. *ORSA Journal on Computing*, **1**, 190–206. 22

GLOVER, F. (1990). Tabu search-part 2. *ORSA Journal on Computing*, **2**, 1–30. 22

GOLDBERG, D. (1989). *Genetic Algorithms in search, optimization and machine learning*. Addison-Wesley Publishing Company. 10

GOLDSMAN, D. & NELSON, B. (1994). Ranking, selection and multiple comparison in computer simulation. *Proceedings of the 1994 Winter Simulation Conference*, 192–199. 15

HERTZ, A. (1991). Tabu search for large scale timetabling problems. *European Journal of Operational Research*, 39–47. 23

HERTZ, A. & DE WERRA, D. (1990). The tabu search metaheuristic: How we use it. *Annals of Mathematics and Artificial Intelligence*, 111–121. 23

HONG, L. & NELSON, B. (2009a). A brief introduction to optimization via simulation. *Proceedings of the 2009 Winter Simulation Conference*, 151–158. 13, 14, 19, 20, 27, 32

HONG, L. & NELSON, B. (2009b). Discrete optimization via simulation using compass. *Operations research*, **36**, 115–129. 27

HUANG, K. & LIAO, C. (2008). Ant colony optimization combined with taboo search for the job shop scheduling problem. *Computers & Operations Research*, **35**, 1030–1046. 30

KIM, S.H. & NELSON, B.L. (2007). Recent advances in ranking and selection. In *Proceedings of the 2007 conference on Winter Simulation*, 162–172. 16

Kirkpatrick, S., Gelatt, C. & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, **220**, 671–680. 21

Konak, A., Coit, D. & Smith, A. (2006). Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering and System Safety*, 992–1007. 43

Kroese, D., Porotsky, S. & Rubinstein, R. (2006). The cross-entropy method for continuous multi-extremal optimization. *Methodology and Computing in Applied Probability*, **8**, 383–407. 24

Laguna, M. (2011). Optquest: Optimization of complex systems. 1–16. 33

Laguna, M. & Marti, R. (2003). The optquest callable library. *Optimization Software Class Libraries*, 193–218. 32

Law, A.M. & Kelton, W.D. (2000). *Simulation modeling and analysis*. McGraw-Hill, 3rd edn. 2, 12, 32, 34, 39

Lee, L. & Goldsman, D. (2004). Optimal computing budget allocation for multi-objectve simulation models. *Proceedings of the 2004 Winter Simulation Conference*, 586–594. 17

Lee, L.H., Chen, C., Chew, E.P., Li, J., Pujowidianto, N.A. & Zhang, S. (2010). A review of optimal computing budget allocation algorithms for simulation optimization problem. *International Journal of Operations Research*, **7**, 19–31. 17

Li, H., Lee, L.H., Chew, E.P. & Lendermann, P. (2015). MO-COMPASS: a fast convergent search algorithm for multi-objective discrete optimization via simulation. *IIE Transactions*, **47**, 1153–1169. 11, 27

Marler, R. & Arora, J. (2004). Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, **26**, 369–395. 11, 44

Martin, O., Otto, S. & Felten, E. (1992). Large-step markov chains for the tps incorporating local search heuristics. *Operations Research Letters*, **11**, 219–224. 30

Merkle, D., Middendorf, M. & Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *IEEE transactions on evolutionary computation*, **6**, 333–346. 26

Metropolis, N., Rosenbluth, A., Rosenbluth, A., Teller, A. & Teller, E. (2002). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 1087–1092. 21

Microsoft (2010). What is a com interface? *www.msdn.microsoft.com/en-us/library/windows/desktop/ff485850(v=vs.85).aspx. [Accessed May 2018]*. 65, 69, 70

Osman, I. & Laporte, G. (1997). Metaheuristics: A bibliography. *Annals of Operational Research*, **63**, 513–628. 20, 22

Radin (1998). *Optimization in Operation Research*. Prentice Hall. 21

Rinott, Y. (1978). On two-stage selection procedures and related probability-inequalities. *Communications in Statistics – Theory and methods*, **7**, 799–811. 16

Rosen, S.L., Harmonosky, C.M. & Traband, M.T. (2008). Optimization of systems with multiple performance measures via simulation: Survey and recommendations. *Computers & Industrial Engineering*, **54**, 327–339. 19

Rubinstein, R. (1997). Optimization of computer simulation models with rare events. *European Journal of Operations Research*, **99**, 89–112. 23, 24

Rubinstein, R. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability,*, 127–188. 24

Talbi, E. (2002). Hybrid metaheuristics. *Journal of heuristics*, **8**, 541–564. 28

Talbi, E. (2013). *Hybrid Metaheuristics*. Springer. 28, 29, 31

Tekin, E. & Sabuncuoglu, I. (2004). Simulation optimization: A comprehensive review on theory and applications. *IIE Transactions*, **36**, 1067–1081. 19, 27

Teng, S., Lee, L. & Chew, E. (2010). Integration of indifference-zone with multi-objective computing budget allocation. *European Journal of Operational Research*, **203**, 419–429. 40

Toth, P. & Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 333–346. 23

Weise, T. (2009). *Global Optimization Algorithms: Theory and Application*. 21, 22, 25

Xu, J., Nelson, B.L. & Hong, J. (2010). Industrial strength COMPASS: A comprehensive algorithm and software for optimization via simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, **20**, 3. 27, 31, 35

Yang, X. (2010). *Engineering optimization: An introduction with metaheuristic applications*. Wiley. 1, 3, 20, 25

Yoon, M. (2018). *New multi-objective ranking and selection procedures for discrete stochastic simulation problems*. Ph.D. thesis, Stellenbosch: Stellenbosch University. 16, 17, 52, 61, 62, 82, 84

Yoon, M. & Bekker, J. (2017). Multi-objective simulation optimisation on discrete sets: A literature review. *International journal of operation research (Submitted)*. 12, 16, 27

Appendix A

Additional tests for the MOO CEM metaheuristic

In this appendix, the author provides additional tests for the MOO CEM metaheuristic. In the first section, a Chi-square goodness-of-fit test is performed on the standard MOO test problem MOP4 (see Table 6.1). In the second section, results for different tests (using different MOO CEM parameters) for the buffer allocation problem of Chapter 7 are presented.

A.1 The Chi-square goodness-of-fit test for MOP4

The purpose of this test is to support the validation of the MOO CEM implementation in MOOSolver provided in Chapter 6.

In this section, the author reports on the results of 101 runs of the MOO suite in solving the MOP4. To perform the Chi-square test (which would validate, or not, the results being reported), the *hyperareas* (HAs) for each run were calculated. (The HAs were calculated as the areas under the obtained Pareto fronts in each run.) These were then compared to the known hyperarea (*i.e.* the reference HA) for the MOP4.

To perform the Chi-square goodness-of-fit test, the following equation is used

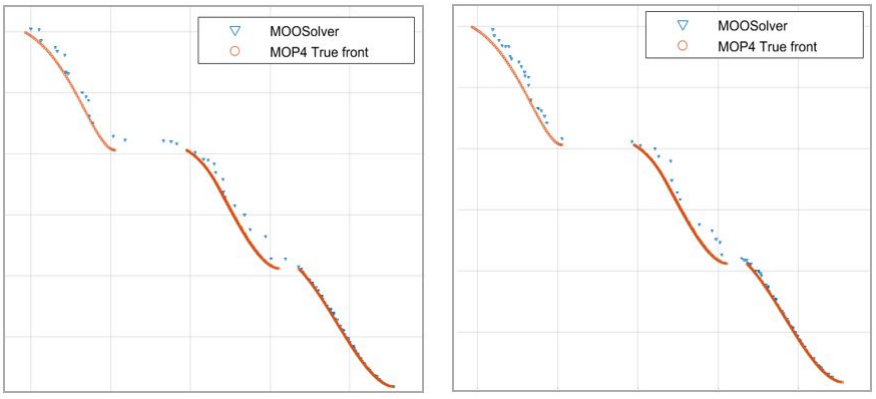
$$\chi^2_{calc} = \sum_{i=1}^{k} \left[\left(\frac{O_i - E_i}{E_i} \right)^2 \right],$$

where k is the number of runs made, which is 101; O_i is the reference hyperarea (denoted as Ref. HA in Table A.1) in run i, $i = 1, 2, ..., k$. It is the known HA for MOP4 and is constant in all runs; and E_i is the HA for run i, $i = 1, 2, ..., k$, obtained

by running the metaheuristic via MOOSolver. χ^2_{calc} is the *calculated* Chi-square value which must be compared to a *critical* Chi-square (χ^2_{crit}) value. χ^2_{crit} is obtained by using k (to calculate the degrees of freedom *i.e.* $k-1$), the Chi-square distribution (*i.e.* the upper critical one-tailed values) and a significance level α, which is usually 5%. It follows from the Chi-square distribution that $\chi^2_{crit} = 124.34$ in this case.

Next, a null hypothesis H_0 is made, which is that the results obtained by the MOO CEM implementation in MOOSolver for MOP4 (with the selected optimisation parameters) are a valid approximation of that of the known result to the problem. According to the Chi-square goodness-of-fit test, if $\chi^2_{calc} > \chi^2_{crit}$, H_0 must be rejected. Otherwise, there is no sufficient evidence to reject H_0.

The results from the 101 runs, which are recorded in Table A.1 (in the HA columns), were used to perform the test. χ^2_{calc} was calculated as 38.516, which is less than 124.34. There is thus no sufficient evidence to reject H_0, and therefore the implementation of the algorithm can be deemed valid (in the MOP4 context) based on the test result. For the reader's interest, the Pareto fronts for some of the runs (randomly selected) are illustrated in Figures A.1 and A.2, where they are compared to the known Pareto front. Note that the known Pareto front has approximately 870 solutions.

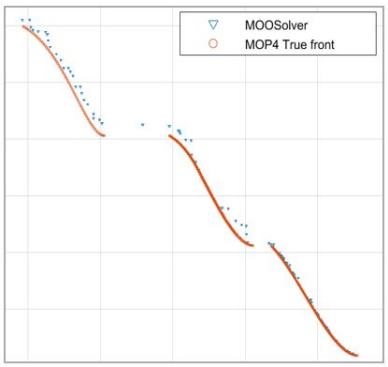


(a) Run no. 3 (130 solutions). (b) Run no. 12 (113 solutions).

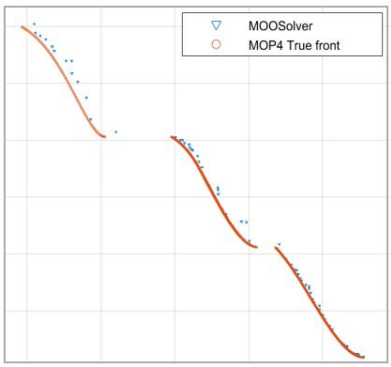
Figure A.1: Selected Pareto fronts for the MOP4 test problem solved with MOOSolver.

Table A.1: Hyperarea results for the MOP4 test problem.

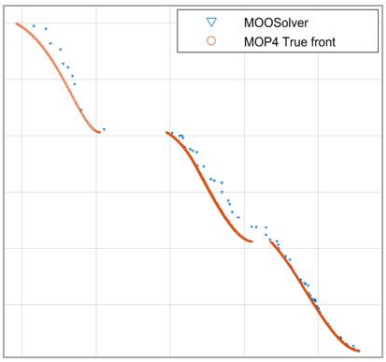
Run no.	HA	Ref. HA	Max. Ev.	Pop. Size	Epsilon	**Run no.**	HA	Ref. HA	Max. Ev.	Pop. Size	Epsilon	**Run no.**	HA	Ref. HA	Max. Ev.	Pop. Size	Epsilon
1	**27.357**	26.199	5000	100	0.1	**35**	**23.336**	26.199	5000	100	0.1	**69**	**24.931**	26.199	5000	100	0.1
2	**24.303**	26.199	5000	100	0.1	**36**	**25.384**	26.199	5000	100	0.1	**70**	**25.855**	26.199	5000	100	0.1
3	**26.062**	26.199	5000	100	0.1	**37**	**24.990**	26.199	5000	100	0.1	**71**	**25.427**	26.199	5000	100	0.1
4	**24.863**	26.199	5000	100	0.1	**38**	**25.776**	26.199	5000	100	0.1	**72**	**25.456**	26.199	5000	100	0.1
5	**25.534**	26.199	5000	100	0.1	**39**	**25.654**	26.199	5000	100	0.1	**73**	**26.665**	26.199	5000	100	0.1
6	**25.945**	26.199	5000	100	0.1	**40**	**26.483**	26.199	5000	100	0.1	**74**	**25.980**	26.199	5000	100	0.1
7	**23.871**	26.199	5000	100	0.1	**41**	**26.113**	26.199	5000	100	0.1	**75**	**25.455**	26.199	5000	100	0.1
8	**44.473**	26.199	5000	100	0.1	**42**	**43.633**	26.199	5000	100	0.1	**76**	**23.658**	26.199	5000	100	0.1
9	**25.578**	26.199	5000	100	0.1	**43**	**26.142**	26.199	5000	100	0.1	**77**	**25.174**	26.199	5000	100	0.1
10	**22.616**	26.199	5000	100	0.1	**44**	**25.035**	26.199	5000	100	0.1	**78**	**25.421**	26.199	5000	100	0.1
11	**25.734**	26.199	5000	100	0.1	**45**	**25.476**	26.199	5000	100	0.1	**79**	**26.515**	26.199	5000	100	0.1
12	**24.559**	26.199	5000	100	0.1	**46**	**25.455**	26.199	5000	100	0.1	**80**	**26.277**	26.199	5000	100	0.1
13	**26.841**	26.199	5000	100	0.1	**47**	**26.486**	26.199	5000	100	0.1	**81**	**23.966**	26.199	5000	100	0.1
14	**25.755**	26.199	5000	100	0.1	**48**	**25.575**	26.199	5000	100	0.1	**82**	**25.588**	26.199	5000	100	0.1
15	**23.952**	26.199	5000	100	0.1	**49**	**26.099**	26.199	5000	100	0.1	**83**	**44.740**	26.199	5000	100	0.1
16	**25.184**	26.199	5000	100	0.1	**50**	**23.834**	26.199	5000	100	0.1	**84**	**25.659**	26.199	5000	100	0.1
17	**25.597**	26.199	5000	100	0.1	**51**	**25.629**	26.199	5000	100	0.1	**85**	**25.801**	26.199	5000	100	0.1
18	**25.692**	26.199	5000	100	0.1	**52**	**22.778**	26.199	5000	100	0.1	**86**	**25.785**	26.199	5000	100	0.1
19	**26.586**	26.199	5000	100	0.1	**53**	**25.835**	26.199	5000	100	0.1	**87**	**24.172**	26.199	5000	100	0.1
20	**25.693**	26.199	5000	100	0.1	**54**	**45.434**	26.199	5000	100	0.1	**88**	**24.257**	26.199	5000	100	0.1
21	**26.247**	26.199	5000	100	0.1	**55**	**24.388**	26.199	5000	100	0.1	**89**	**23.492**	26.199	5000	100	0.1
22	**25.880**	26.199	5000	100	0.1	**56**	**25.200**	26.199	5000	100	0.1	**90**	**24.737**	26.199	5000	100	0.1
23	**25.228**	26.199	5000	100	0.1	**57**	**21.602**	26.199	5000	100	0.1	**91**	**25.701**	26.199	5000	100	0.1
24	**25.782**	26.199	5000	100	0.1	**58**	**25.076**	26.199	5000	100	0.1	**92**	**25.012**	26.199	5000	100	0.1
25	**25.051**	26.199	5000	100	0.1	**59**	**26.206**	26.199	5000	100	0.1	**93**	**26.317**	26.199	5000	100	0.1
26	**26.691**	26.199	5000	100	0.1	**60**	**25.004**	26.199	5000	100	0.1	**94**	**26.425**	26.199	5000	100	0.1
27	**24.853**	26.199	5000	100	0.1	**61**	**26.340**	26.199	5000	100	0.1	**95**	**25.690**	26.199	5000	100	0.1
28	**23.774**	26.199	5000	100	0.1	**62**	**24.291**	26.199	5000	100	0.1	**96**	**22.923**	26.199	5000	100	0.1
29	**24.383**	26.199	5000	100	0.1	**63**	**26.298**	26.199	5000	100	0.1	**97**	**25.047**	26.199	5000	100	0.1
30	**25.853**	26.199	5000	100	0.1	**64**	**23.838**	26.199	5000	100	0.1	**98**	**25.932**	26.199	5000	100	0.1
31	**25.171**	26.199	5000	100	0.1	**65**	**27.167**	26.199	5000	100	0.1	**99**	**26.160**	26.199	5000	100	0.1
32	**25.134**	26.199	5000	100	0.1	**66**	**25.277**	26.199	5000	100	0.1	**100**	**21.5511**	26.199	5000	100	0.1
33	**24.901**	26.199	5000	100	0.1	**67**	**25.656**	26.199	5000	100	0.1	**101**	**24.548**	26.199	5000	100	0.1
34	**25.459**	26.199	5000	100	0.1	**68**	**25.566**	26.199	5000	100	0.1			**Mean HA**	**26.019**	**Std HA**	**3.935**



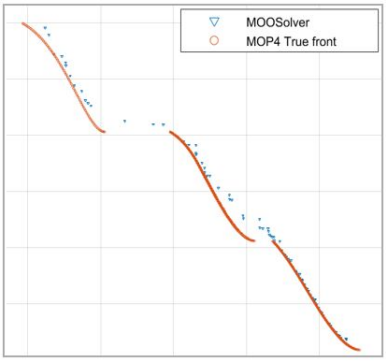
(a) Run no. 40 (124 solutions).



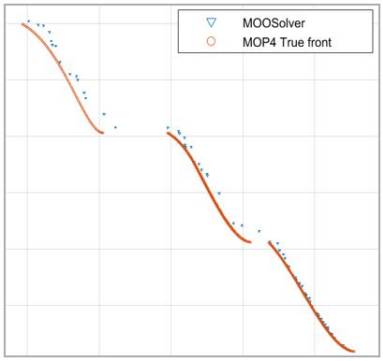
(b) Run no. 47 (86 solutions).



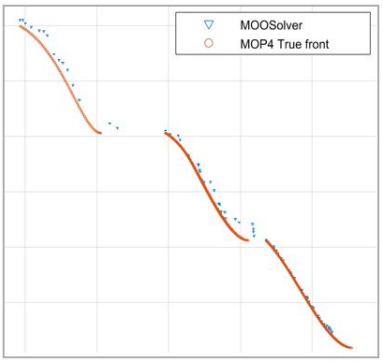
(c) Run no. 66 (84 solutions).



(d) Run no. 81 (95 solutions).



(e) Run no. 90 (78 solutions).



(f) Run no. 95 (80 solutions).

Figure A.2: Selected Pareto fronts for the MOP4 test problem solved with MOOSolver.

A.2 Results for the MOO CEM parameters test performed for the buffer allocation problem

Tables A.2 and A.3 illustrate the results that were obtained (*i.e.* run times and number of solutions) by varying the MOO CEM parameters (as well as the number of observations) for different runs of the buffer allocation problem of Chapter 7. In particular, Table A.2 contains the results of Tests 1, 2 and 3 while Table A.3 contains the results for Test 4.

The tests in Table A.2 focus on the impact of changing the number of observations as well as the number of loops while the maximum number of evaluations and other parameters are kept constant. Test 1 uses a value of 50 for the number of loops parameter whereas Test 2 and 3 use 20 and 80, respectively. Judging from the run times of the different tests, it was concluded that a value of 20, for the number of loops parameter, is too little for this problem.

Table A.2: Test results for the BAP using different parameters of the MOO CEM.

	Test 1.0	**Test 1.1**	**Test 1.2**	**Test 2.0**	**Test 2.1**	**Test 2.2**	**Test 3.0**	**Test 3.1**	**Test 3.2**
No. Observations	5	15	25	5	15	25	5	15	25
Epsilon	1	1	1	1	1	1	1	1	1
Alpha	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
No. Loops	50	50	50	20	20	20	80	80	80
Pop. Size	50	50	50	50	50	50	50	50	50
Inverse probability	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
Max. Evaluation	10000	10000	10000	10000	10000	10000	10000	10000	10000
Run time	03:55 min	14:39 min	20:03 min	02:12 min	05:26 min	09:22 min	04:16 min	13:35 min	23:24 min
No. Solutions	86	111	117	79	98	87	108	108	117

The tests in Table A.3, on the other hand, focus on the impact of changing the maximum number of evaluations while the other parameters are kept constant. Test 4 was used, in particular, to verify whether convergence toward a potential true Pareto front was occurring, as the number of evaluations allowed was being increased. Selected Pareto fronts from this test were compared and the comparison is illustrated in Figure A.3. Figure A.3 shows that convergence does, in effect, occur. Note, for example, how the Pareto fronts for Test 4.4 and Test 4.5 are virtually the same. Test 4.4 parameters were used to solve the BAP in Chapter 7.

A.2 Results for the MOO CEM parameters test performed for the buffer allocation problem

Table A.3: Test results for the BAP using different parameters of the MOO CEM.

	Test 4.0.0	Test 4.0.1	Test 4.0.2	Test 4.0.3	Test 4.1	Test 4.2	Test 4.3	Test 4.4	Test 4.5
No. Observations	15	15	15	15	15	15	5	15	15
Epsilon	1	1	1	1	1	1	1	1	1
Alpha	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
No. Loops	100	100	100	100	100	100	100	100	100
Pop. Size	100	100	100	100	100	100	100	100	100
Inverse probability	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Max. Evaluation	100	200	500	800	2000	3000	4000	5000	10000
Run time	NA	26 sec	52 sec	01:05 min	02:58 min	04:06 min	05:45 min	06:21 min	12:56 min
No. Solutions	NA	28	35	33	50	61	70	78	105

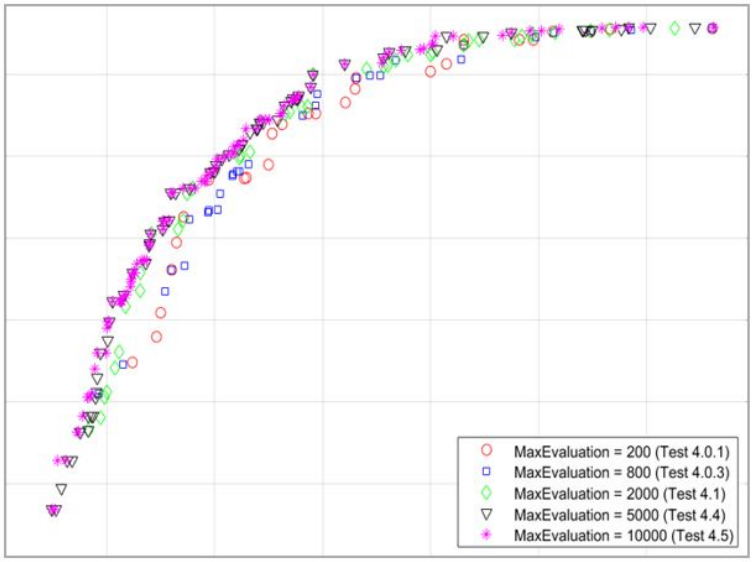


Figure A.3: Comparison of Pareto fronts obtained by varying the MOO CEM's Maximum evaluation parameter for the BAP (Bamporiki & Bekker, 2018).

Appendix B

How to build a MOOSolver-ready model in Tecnomatix Plant Simulation

In this appendix, directives on how to build a discrete-event, multi-objective simulation model in Tecnomatix Plant Simulation *that is ready* to serve as the simulation evaluator for the simulation optimisation process conducted by MOOSolver, are provided.

It is assumed that, at this point, the user has validly built a model in TPS under the MOOSolver problem framework presented in Chapter 2 and desires to utilise the simulation optimisation services of the MOOSolver library to find approximate Pareto solutions to their problem.

Before providing the model parameters to the MOO suite graphical user-interface for TPS (MSWizard), a few things must be checked and/or done in order to allow for a valid SO process. They are presented in a stepwise manner below.

B.1 Step one: The EventController object

- Ensure that their is an EventController object on the same frame as the model (Figure B.1).
- Please do not rename the object (it should not, actually, be possible to rename this particular object).
- Use the object to verify that the model returns correct values for the objective functions. The user should run the object once or twice to ensure that their model

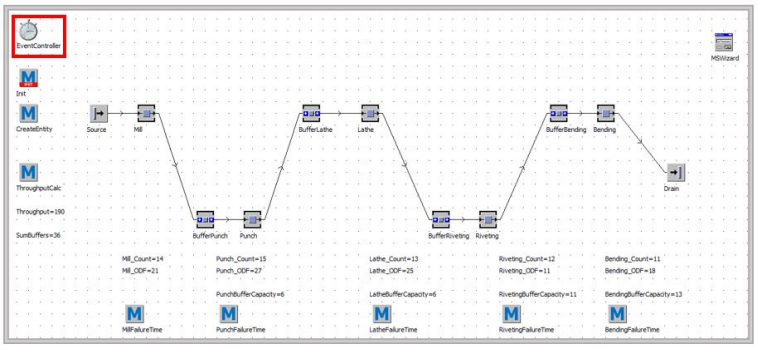


Figure B.1: The EventController object in the frame.

works as they desire.

B.2 Step two: Decision variables and Objective functions

- Ensure that the decision variables and objective functions are Variable objects.
- Ensure also that they are located on the same frame as the model.

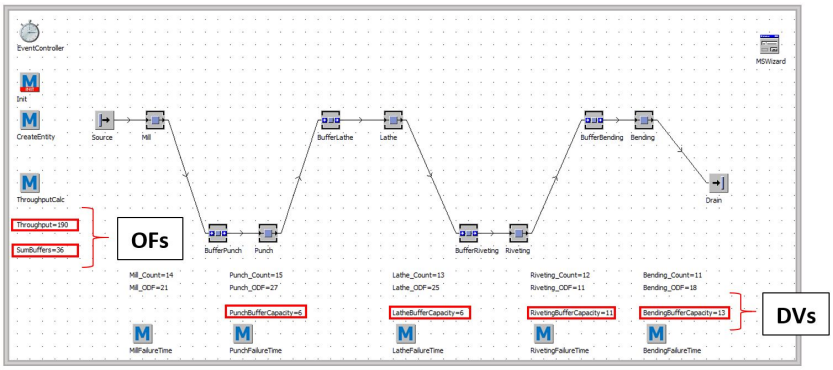


Figure B.2: DVs and OFs in the frame.

- Next, ensure that the **data types** for both the DVs and OFs variable objects in the frame are **real** (Figure B.3). This is not to say, in the DVs case for example, that the nature of the DVs are continuous. This is simply done to facilitate the inter-process communication between TPS and MOOSolver. In effect, the MOOSolver code works with variables of type real during the optimisation procedure. Since there is a direct exchange of information between the suite and TPS at this particular phase of the SO process, all the variables involved during the process in

TPS (DVs and OFs) must have data types that are consistent with that which is used by MOOSolver. As for the DVs nature, they are specified via the MSWizard (see next appendix) and the suite takes them into account during the SO process to do the necessary conversions.

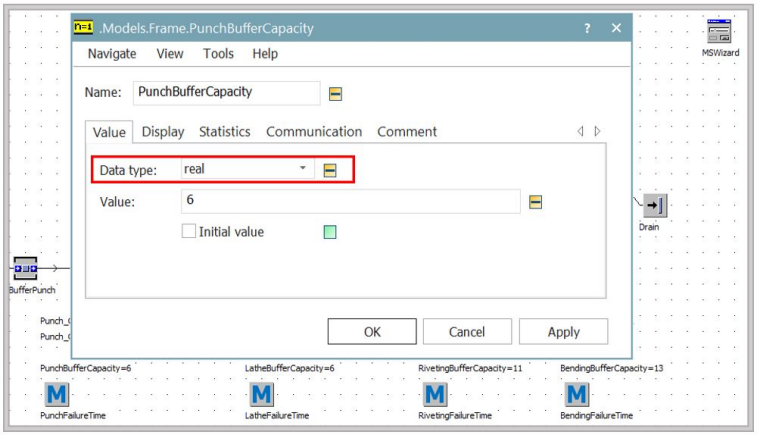


Figure B.3: DVs and OFs data types in the frame.

- It is important to also ensure that the *Initial value* check boxes for both DVs and OFs are **unchecked**.

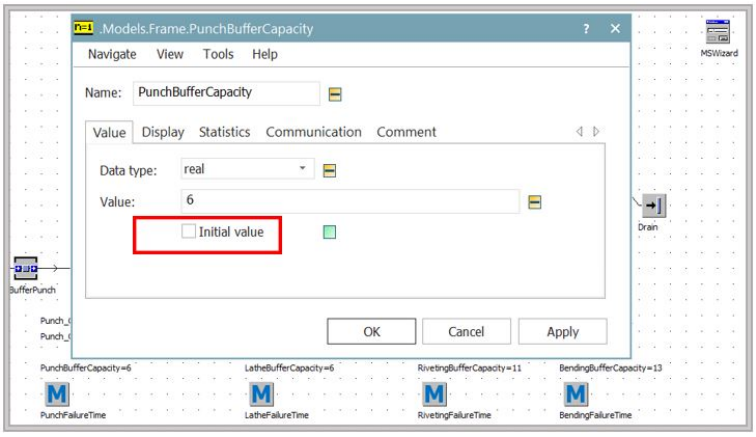


Figure B.4: The Initial value check boxes for all DVs and OFs Variable objects must be unchecked.

- If the model has an *Init method* in it, it is important to ensure, in cases where the method contains the DVs, that the DVs are not assigned values in this method.

This is in effect the same as enabling initial values for the DVs via the Variable object, as discussed before. The problem with having initial values for DVs is that they overwrite the values that MOOSolver proposes as candidate solutions to be estimated via simulation during the optimisation procedure; hence making the SO process invalid. This must be avoided.

Appendix C

MSWizard: a walk-through on how to use the MOOSolver user-interface for Tecnomatix Plant Simulation

MSWizard is the MOOSolver graphical user-interface for Tecnomatix Plant Simulation. In the previous appendix, directives on how to build a MOOSolver-ready model were provided assuming that the user had a valid model already. Here, it is assumed that the model is both valid and MOOSolver-ready. Next are a number of steps that should help the user utilise the MSWizard effectively.

C.1 Step One: Placing the MSWizard in the frame

- The user must ensure that the MSWizard is in the same frame as their model (Figure C.1).

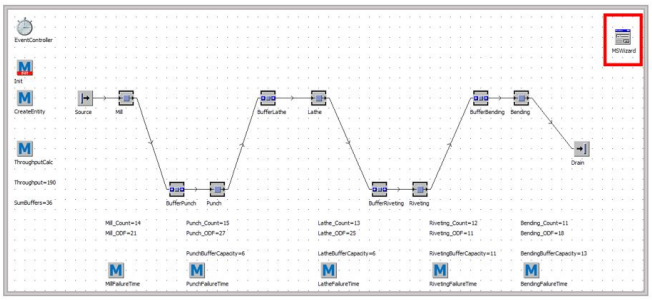


Figure C.1: MSWizard in frame.

- To do this, the user must go to the *Home* tab and click on the *Manage Class Library* icon (Figure C.2).

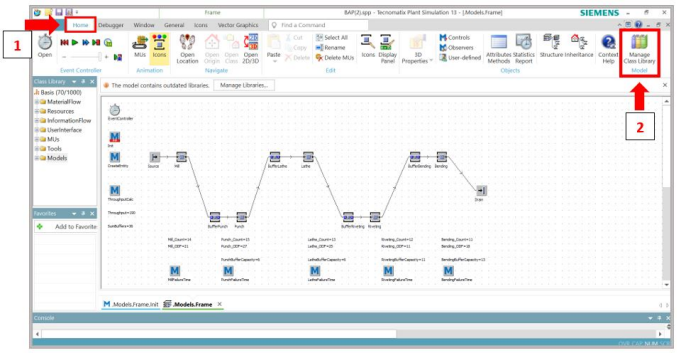


Figure C.2: Opening the Manage Class library icon on the Home tab in TPS.

- On the window that opens, the user must go to the *Libraries* tab and click on the check box to the left of MSWizard. Click Apply then OK (Figure C.3).

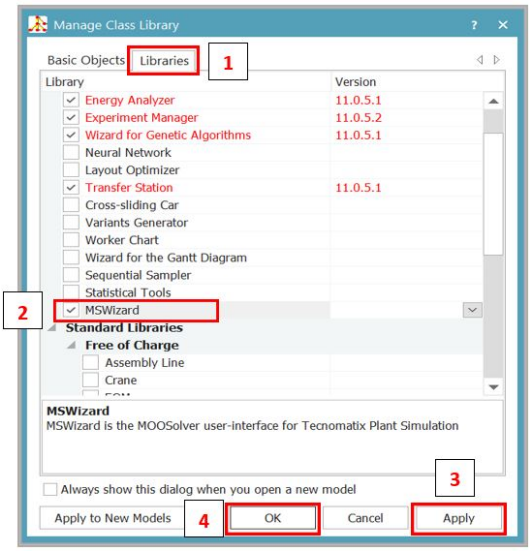


Figure C.3: Activating the MSWizard in TPS.

- On the *Class Library* pane, the MSWizard icon should now be visible. The user can now drag the MSWizard object and drop it in the same frame as their model (Figure C.4).

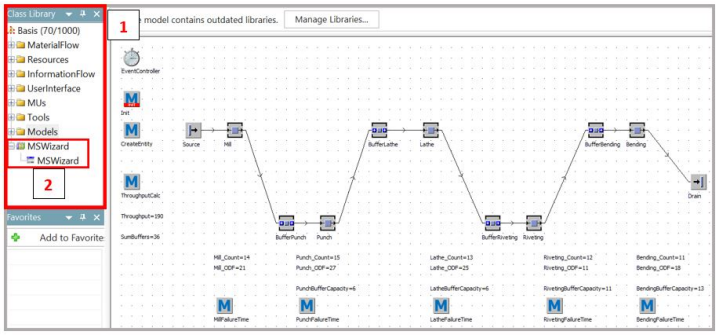


Figure C.4: The MSWizard in the Class Library pane, ready to be dragged and dropped.

C.2 Step Two: Defining the MOSO problem to MOOSolver

In this step, the user defines their MOSO problem to MOOSolver via the MSWizard. With the MSWizard now in the frame, the user must right click the object and choose the *Show Dialog* option. The MSWizard GUI should now be opened. On the *Define* tab of the wizard, the user must do the following:

- First, specify the number of DVs that their model contains (Figure C.5).

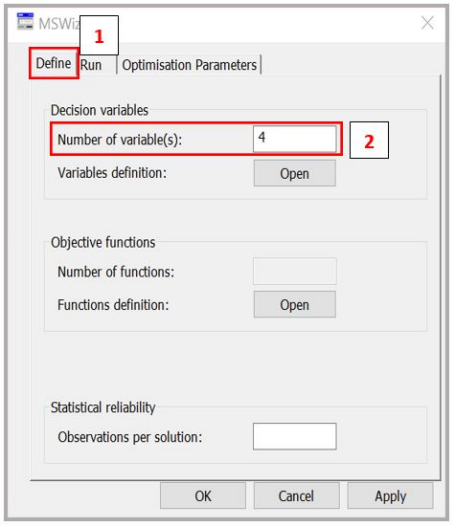


Figure C.5: Specify the number of DVs in the model.

- Then, click on the *Open* button next to *Variables definition* in the *Decision variables* group box, to open the DVs definition table. When the table opens,

before the user is able to enter the DVs parameters, they must *Deactivate* the *Inherit Contents* option on the *List* tab, by clicking on the Inherit Contents icon as shown in Figure C.6. Once deactivated, it should no longer have the blue shade around it and the user should now be able to enter the DVs parameters into the table.

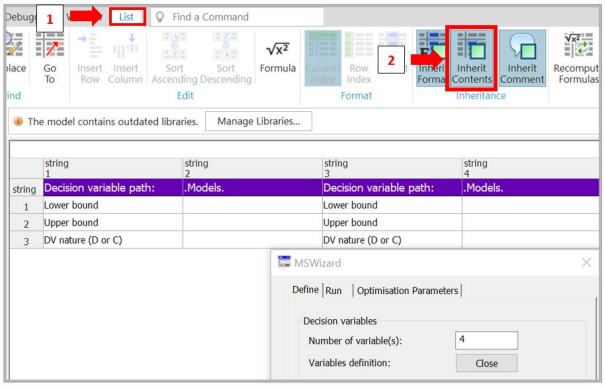


Figure C.6: Deactivating the Inherit Contents icon.

- Next, the user must enter the DVs location in the appropriate cells in the table (Figure C.7), up to 10 DVs can currently be entered. The number of DVs entered must be consistent with the number of DVs specified previously (Figure C.5). DVs locations can be entered in two ways: by manually typing them out in the cells, in which case care must be taken in following the right format *i.e.* **.Models.NameOfTheFrame.NameOfTheDV**; or by dragging and dropping them from the frame to the cells.

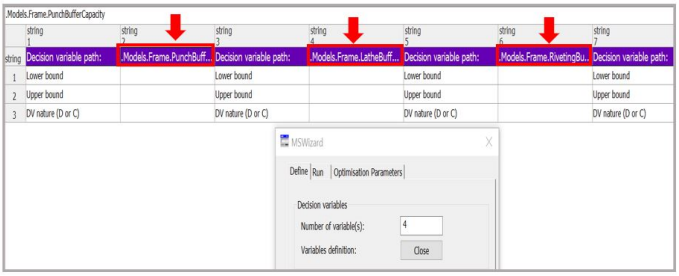


Figure C.7: Entering the DVs' locations or paths into MSWizard.

- Then, the user must specify the DVs' boundaries as well as each DV's respective nature (Figure C.8). Click Apply and close the table.

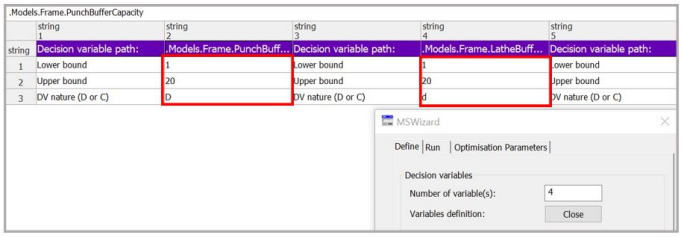


Figure C.8: Entering the DVs boundaries and natures into MSWizard.

- Similarly for OFs, the user must open the OFs definition table, deactivate the inherited contents, enter the OFs' locations (Figure C.9, Step 2) and enter the OFs' respective optimisation directions (Figure C.9, Step 3). If solving a small-scale problem, then the IZ values for each OF must be specified as well (Figure C.9, Step 4), otherwise click Apply and close the table.

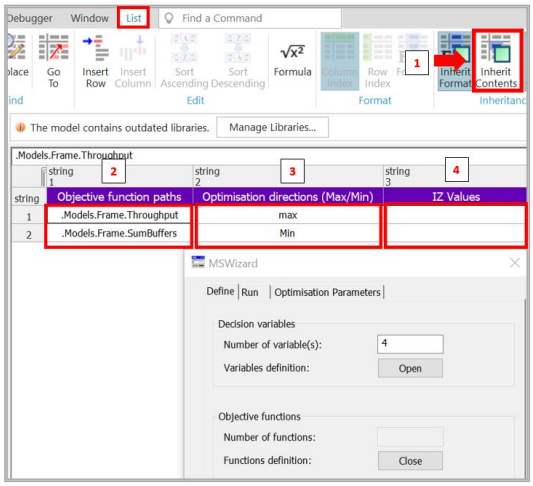


Figure C.9: Entering the OFs parameters into MSWizard.

- Finally, the user must specify the desired number of observations per solution which the metaheuristic will utilise during the optimisation procedure (Figure C.10). Click Apply and go to the *Optimisation Parameters* tab.

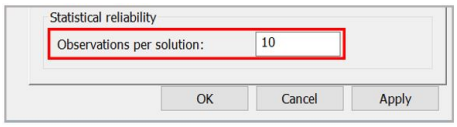


Figure C.10: Entering an arbitrary number of observations per solution for the metaheuristic.

C.3 Step Three: Running the MOOSolver suite

Now that the MOSO problem has been defined, it is time to solve it. The steps are as follows:

- The user must first specify the optimisation parameters of the algorithm they desire to run, in the *Optimisation Parameters* tab (Figure C.11).

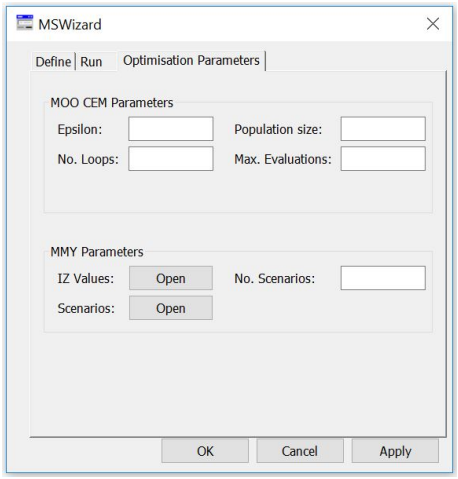


Figure C.11: The MSWizard Optimisation Parameters tab.

C.3.1 Running the MOO CEM

- For the MOO CEM, the user must ensure that they provide appropriate values for the metaheuristic's parameters (Figure C.12). It is advised to run the metaheuristic a number of times while adjusting the parameters each time. This way, the results obtained by each set of parameters can be compared (see Section A.2).
- **Before running** the metaheuristic, the user **must save** the model. This is important because MOOSolver opens an instance of the model of interest as a

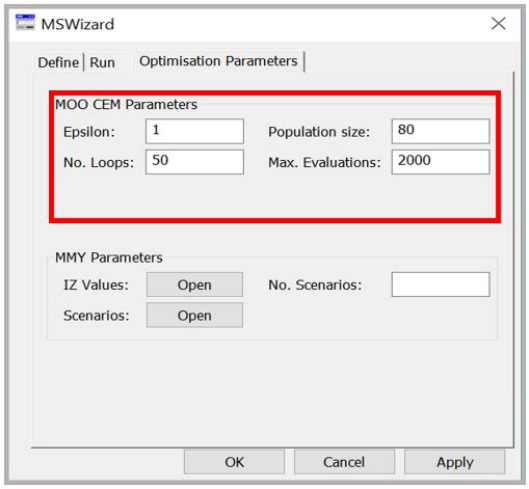


Figure C.12: Specifying the optimisation parameters for the MOO CEM.

background process during the SO process. To ensure that the model to be opened has the latest updates or changes made since the last save, **it is recommended to always save before running**.

- To run the metaheuristic, the user must press the *Start* button in the *Simulation-Optimisation* group box in the *Run* tab (Figure C.13). The message shown in Figure C.14 is then displayed, to which the user must press OK.

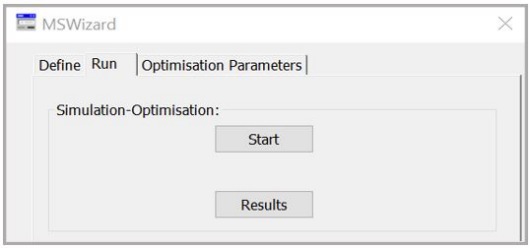


Figure C.13: Starting the MOO CEM.

- When the execution is complete, the message shown in Figure C.15 is displayed, to which the user must press OK. A table containing the approximate Pareto solutions also appears on the screen when the execution is complete (Figure C.16). The table has the format illustrated in Table 5.1. Once closed the table can always be accessed again, by pressing on the *Results* button (Figure C.13).

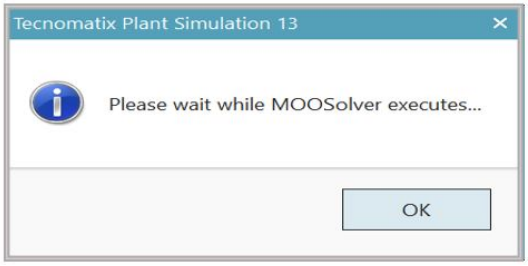


Figure C.14: Prompt message by MOOSolver before execution.

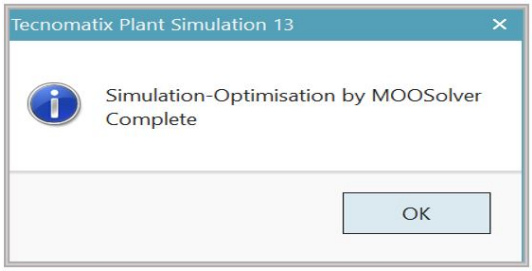


Figure C.15: Prompt message by MOOSolver when the MOO CEM run is complete.

	real 1	real 2	real 3	real 4	real 5	real 6
string	.Models.Frame.PunchBufferCap...	.Models.Frame.LatheBufferCap...	.Models.Frame.RivetingBufferC...	.Models.Frame.BendingBufferC...	.Models.Frame.Throughput	.Models.Frame.SumBuffers
55	17.0000	11.0000	10.0000	3.0000	204.6000	41.0000
56	10.0000	16.0000	10.0000	6.0000	205.6000	42.0000
57	10.0000	16.0000	10.0000	6.0000	205.6000	42.0000
58	14.0000	13.0000	11.0000	5.0000	205.8000	43.0000
59	12.0000	11.0000	16.0000	4.0000	206.0000	43.0000
60	15.0000	17.0000	8.0000	5.0000	206.2000	45.0000
61	17.0000	11.0000	13.0000	4.0000	206.4000	45.0000
62	14.0000	13.0000	12.0000	6.0000	206.6000	45.0000
63	14.0000	13.0000	12.0000	6.0000	206.6000	45.0000
64	18.0000	9.0000	9.0000	10.0000	207.4000	46.0000
65	19.0000	13.0000	8.0000	7.0000	208.0000	47.0000
66	18.0000	12.0000	7.0000	12.0000	208.8000	49.0000
67	19.0000	15.0000	9.0000	8.0000	210.0000	51.0000
68	18.0000	14.0000	20.0000	12.0000	210.2000	64.0000

Figure C.16: The MOO CEM results table.

C.3.2 Running the $\mathcal{MMY}$

- For the $\mathcal{MMY}$, the user must ensure that they provide the R&S procedure's parameters (Figure C.17). Particularly, the user must first specify the number of scenarios to be compared (Figure C.17, Step 1); then they must define the scenarios (*i.e.* specify the DVs' values for each scenario) in the $\mathcal{MMY}$ Scenarios

table (Figure C.18, Step 2); finally, if not done already when defining the OFs, the user must specify their desired IZ values for each OF in the OFs definition table (Figures C.17, Step 3; and C.9, Step 4).

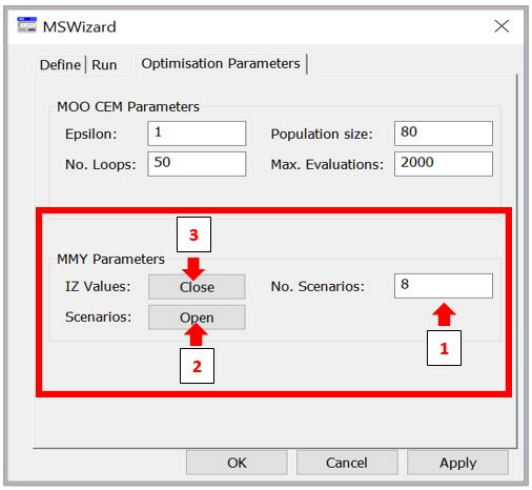


Figure C.17: Specifying the optimisation parameters for the $\mathcal{MMY}$ procedure.

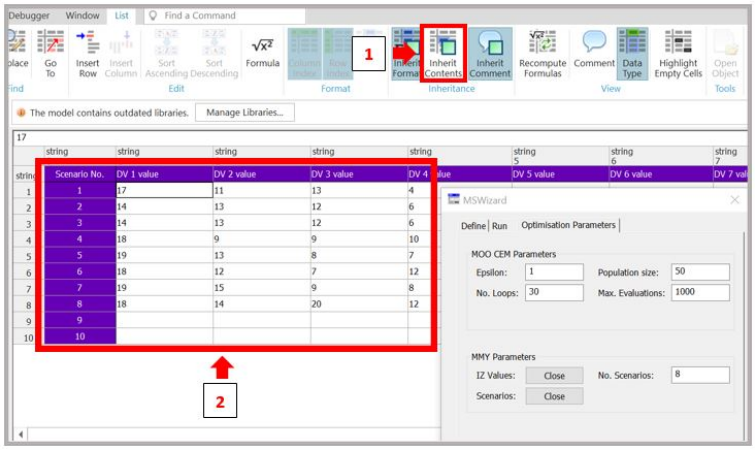


Figure C.18: Defining scenarios for the $\mathcal{MMY}$ procedure.

- In the case where the user is utilising solutions from the metaheuristic's results as scenarios for the R&S procedure, the user can simply *copy* their preferred solutions (the DVs values) from the MOO CEM results table (Figure C.19) and *paste* them in the Scenarios table.

- **Before running** the R&S procedure, the user **must save** the model. This is

	real 1	real 2	real 3	real 4	real 5	real 6
string	.Models.Frame.PunchBufferCap...	.Models.Frame.LatheBufferCap...	.Models.Frame.RivetingBufferC...	.Models.Frame.BendingBufferC...	.Models.Frame.Throughput	.Models.Frame.SumBuffers
55	17.0000	11.0000	10.0000	3.0000	204.6000	41.0000
56	10.0000	16.0000	10.0000	6.0000	205.6000	42.0000
57	10.0000	16.0000	10.0000	6.0000	205.6000	42.0000
58	14.0000	13.0000	11.0000	5.0000	205.8000	43.0000
59	12.0000	11.0000	16.0000	4.0000	206.0000	43.0000
60	15.0000	17.0000	8.0000	5.0000	206.2000	45.0000
61	17.0000	11.0000	13.0000	4.0000	206.4000	45.0000
62	14.0000	13.0000	12.0000	6.0000	206.6000	45.0000
63	14.0000	13.0000	12.0000	6.0000	206.6000	45.0000
64	18.0000	9.0000	9.0000	10.0000	207.4000	46.0000
65	19.0000	13.0000	8.0000	7.0000	208.0000	47.0000
66	18.0000	12.0000	7.0000	12.0000	208.8000	49.0000
67	19.0000	15.0000	9.0000	8.0000	210.0000	51.0000
68	18.0000	14.0000	20.0000	12.0000	210.2000	64.0000

Figure C.19: Selecting scenarios for the $\mathcal{MMY}$ procedure from the MOO CEM results table.

important because MOOSolver opens an instance of the model of interest as a background process during the SO process. To ensure that the model to be opened has the latest updates or changes made since the last save, **it is recommended to always save before running**.

- To run the R&S procedure, the user must press the *Start* button in the *Ranking and Selection* group box in the *Run* tab (Figure C.20). The message shown in Figure C.14 is then displayed, to which the user must press OK.

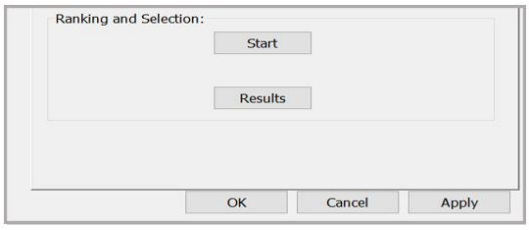


Figure C.20: Starting the $\mathcal{MMY}$ procedure.

- When the execution is complete, the message shown in Figure C.21 is displayed, to which the user must press OK. A table containing the R&S results also appears on the screen when the execution is complete (Figure C.22). The table has the format illustrated in Table 5.2. Once closed, the table can always be accessed again by pressing on the *Results* button (Figure C.20).

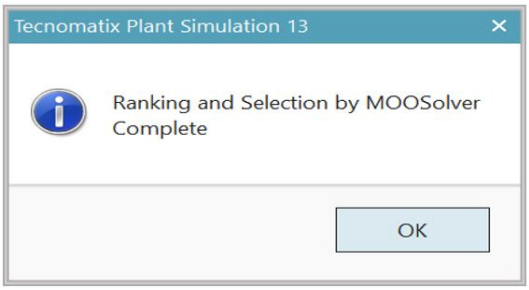


Figure C.21: Prompt message by MOOSolver when the $\mathcal{MMY}$ run is complete.

	real 1	real 2	real 3	real 4	real 5	real 6
string	.Models.Frame.PunchBufferCap...	.Models.Frame.LatheBufferCap...	.Models.Frame.RivetingBufferC...	.Models.Frame.BendingBufferC...	.Models.Frame.Throughput	.Models.Frame.SumBuffers
1	17.0000	11.0000	13.0000	4.0000	197.5088	45.0000
2	14.0000	13.0000	12.0000	6.0000	197.5658	45.0000
3	14.0000	13.0000	12.0000	6.0000	197.5658	45.0000
4	18.0000	9.0000	9.0000	10.0000	197.9912	46.0000
5	18.0000	12.0000	7.0000	12.0000	198.0727	49.0000
6	18.0000	14.0000	20.0000	12.0000	198.1720	64.0000
7	19.0000	13.0000	8.0000	7.0000	198.2763	47.0000
8	19.0000	15.0000	9.0000	8.0000	199.1590	51.0000
9						
10						

(a) Part one.

real 7	real 8	real 9	real 10	real 11	real 12
Objective 1 Variance	Objective 2 Variance	Dominated by	Observations ran	RnS Status	Scenario No.
117.5109	0.0000	0.0000	228.0000	0.0000	1.0000
139.4009	0.0000	0.0000	228.0000	0.0000	2.0000
139.4009	0.0000	0.0000	228.0000	0.0000	3.0000
122.9515	0.0000	0.0000	228.0000	0.0000	4.0000
134.1225	0.0000	1.0000	220.0000	0.0000	6.0000
149.5664	0.0000	2.0000	157.0000	0.0000	8.0000
119.3550	0.0000	0.0000	228.0000	0.0000	5.0000
146.7225	0.0000	0.0000	239.0000	0.0000	7.0000

(b) Part two.

Figure C.22: The $\mathcal{MMY}$ results table.

C.4 Troubleshooting

In Appendix B, directives were given that must be followed in order to successfully make use of the MOOSolver suite. Moreover, Sections C.2 and C.3 provide detailed steps to be followed when making use of the MSWizard. Following these instructions and carefully defining a model's parameters into the wizard should ensure that technical errors are avoided, and that the MOOSolver execution is done smoothly.

Nevertheless, we are humans and involuntary lack of attention may lead to mistakes

such as forgetting to ensure that one or two directives from Appendix B are followed, typos, inconsistencies in the parameters definition process *etc.*

In this section, three typical errors that can occur are presented as well as the reasons why they may occur and what the user should do when they occur.

C.4.1 Error type one: Severe run time error in C-Interface

The following mistakes may cause this error to occur:

- An inconsistency in the parameters definition process *e.g.* entering the number of DVs as, for example, 4 and yet defining only three. Or, similarly, entering a number of scenarios that is inconsistent with those defined.
- A typo or typos, specifically in the DVs definition process.

When MOOSolver is executed in the presence of these mistakes, TPS error messages such as the one in Figure C.23, may be displayed. The user must ensure that no such messages are returned by TPS upon execution of the suite. If presented with such messages, however, the user must do the following:

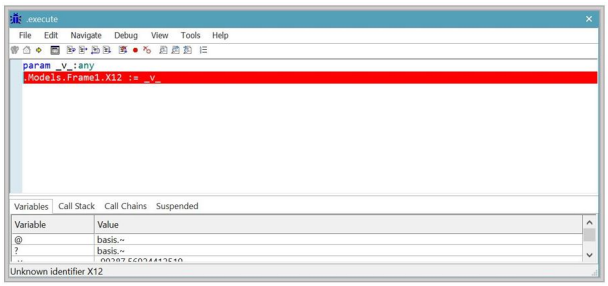
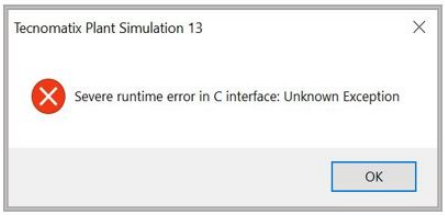
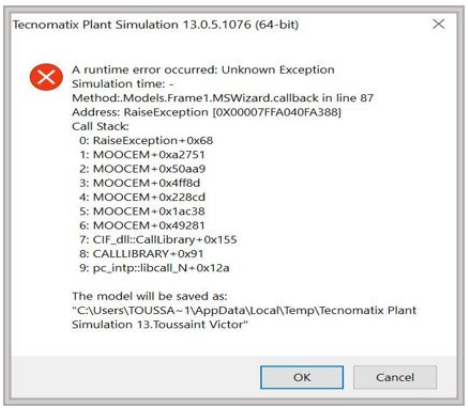


Figure C.23: A typical TPS error message that may be caused by a typo.

1. Close the error message.
2. The following error message *i.e.* Figure C.24a may then be displayed, followed by another one *i.e.* Figure C.24b. They must both be closed.
3. Then, as a result of this error, the TPS application closes automatically while, simultaneously, saving a backup for the model.



(a) First message.



(b) Second message.

Figure C.24: Possible error messages caused by the severe run time error in C-Interface error type.

4. Before opening TPS again, the user must first ensure that they manually close the TPS application that was opened as a background process when they executed MOOSolver. (This is because TPS background processes opened by MOOSolver, close automatically **only** after a MOOSolver execution has been successful; which is not the case here.) To do so, they must open the *Task Manager* by simultaneously pressing: Ctrl + Alt + Delete on the keyboard. In the Task Manager, find the TPS application in question (it should be among other background processes being run by the computer), and close it. Some computers clearly distinguish between main and background processes while others do not. In cases where the computer does not make the distinction, the user must be careful in selecting the right TPS application to close if there are other TPS applications opened.

5. When opening TPS anew, the message in Figure C.25 should be presented to the user. If the user is not sure whether they saved their model before execution, they must choose *Yes*. The saved backup model should then open. This version

of the model still has the errors that caused the application and the model to crash. The user must, therefore, check for potential mistakes before attempting another execution of the optimisation suite. Moreover, the user must note that the backup model has a new, generic name. They may want to *save* the model *as*, and give it back its original name. On the other hand, the user may also choose *No*, in which case they can open their model themselves. This model may or may not have the error that caused the model to crash depending on when it was last saved before execution. But since it is recommended to always save before execution, chances are that the mistakes are still present; and so the user must still check for potential mistakes before attempting another execution of the optimisation suite.

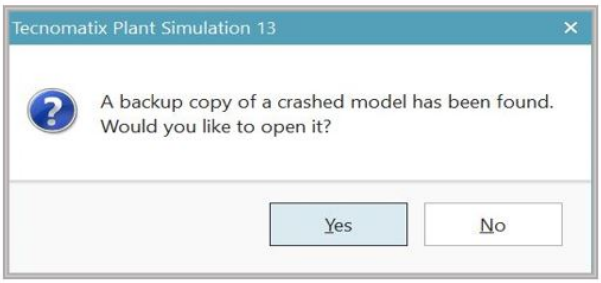


Figure C.25: Possible TPS message after the application crashed.

C.4.2 Error type two: Error in external C function

As far as the author has tested the optimisation suite, only one kind of mistake seem to be the cause of this error: typos in variables definition. When MOOSolver is executed in the presence of such mistakes, TPS error messages such as the one in Figure C.23, may again be displayed. The user must ensure that no such messages are returned by TPS upon execution of the optimisation suite. If presented with such messages, however, the user must do the following:

1. Close the error message.
2. The following error message may then be displayed *i.e.* Figure C.26. It must be closed.

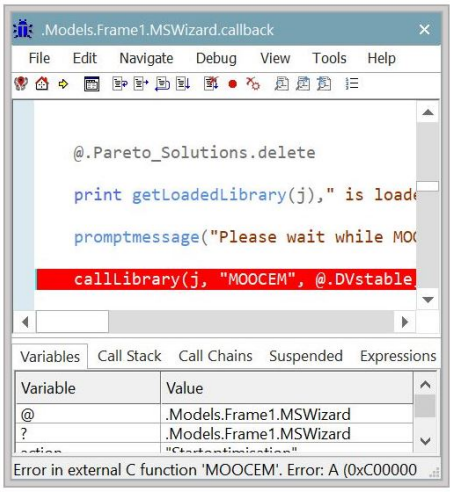


Figure C.26: A possible error message caused by the error in external C function error type.

3. Unlike in the previous error type case, this error does not, typically, cause TPS to crash. Once all error messages are closed, the user may proceed with fixing the mistakes that caused the error. However, a TPS background process may have been opened and be active. The user must, therefore, check the Task Manager first (see Section C.4.1) and possibly close it.

C.4.3 Error type three: Infinite loop

The last error type is the possibility for the SO process to be caught in an infinite loop. A typical error message that may indicate this is illustrated in Figure C.27. Although this error message does not always mean infinite loop, it would normally disappear after a short while if there is no infinite loop. In the case of an infinite loop, it remains on the screen. The following mistakes may cause this error type:

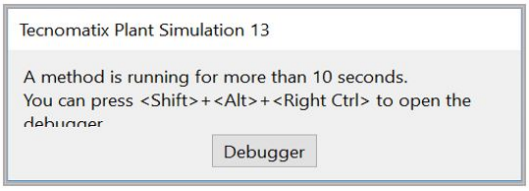


Figure C.27: The error message signifying a possible infinite loop error type.

- Using initial values for the DVs.

- Using data types other than *real* for DVs and OFs.
- Typos.

When this error type occurs, the user must do the following:

1. Go to the Task Manager (see Section C.4.1) and close both the main and the background TPS processes.
2. Open TPS anew and open the model. There is no backup models saved in this case.
3. Fix the mistakes and execute the optimisation suite again.

C.5 Final recommendation

In general, multi-objective SO problems take longer to run than their single-objective counterpart. It is hence recommended to the user to be a little conservative in their optimisation parameters selection when solving their problems for the first time using MOOSolver. Doing this will serve as a way of giving them an idea of how long they should expect to wait after the MOO suite is executed with their preferred parameters.