DEVELOPMENT OF A DISCRETE-EVENT, STOCHASTIC MULTI-OBJECTIVE METAHEURISTIC SIMULATION OPTIMISATION SUITE FOR A COMMERCIAL SOFTWARE PACKAGE

T. Bamporiki[1]* & J. Bekker[1]

ARTICLE INFO

Article details
Presented at the 29th annual conference of the Southern African Institute for Industrial Engineering (SAIIE), held from 24-26 October 2018 in Stellenbosch, South Africa

Available online 9 Nov 2018

Contact details
* Corresponding author
17664276@sun.ac.za

Author affiliations
1 Department of Industrial Engineering, Stellenbosch University, South Africa

DOI
http://dx.doi.org/10.7166/29-3-2045

ABSTRACT

In this paper, the authors present the development of an optimisation suite and its implementation. This paper is part of an ongoing project that aims at developing a hybrid metaheuristic optimisation suite to solve multi-objective simulation optimisation problems. The suite is developed as a third-party library to solve discrete-event, stochastic simulation optimisation (SO) problems with multi-objectives, and is integrated with Tecnomatix Plant Simulation, a simulation software package developed by Siemens. Two real-life multi-objective SO problems are selected to test the suite's performance, and the test results are discussed.

OPSOMMING

In hierdie artikel bespreek die outeurs die ontwikkeling en implementering van 'n optimering-suite. Die studie vorm deel van 'n lopende projek wat die ontwikkeling van 'n hibriede optimering-suite vir multi-doelwitoptimering nastreef. Die suite word ontwikkel as 'n derdeparty-biblioteek om diskrete gebeurtenis, stogastiese simulasieprobleme met multi-doelwitte op te los, en dit dan met Tecnomatix Plant Simulation van Siemens te integreer. Twee realistiese multi-doelwitsimulasie-optimerings-probleme is gekies om die werkverrigting van die suite te evalueer, en die resultate word bespreek.

1 INTRODUCTION

Industry 4.0, or the fourth industrial revolution, is marked by emerging technology breakthroughs in a number of fields, including artificial intelligence (AI) [1]. Colloquially, the term 'artificial intelligence' is applied when a machine mimics 'cognitive' functions that humans associate with other human minds, such as 'problem-solving'. Many tools are used in AI, including versions of search and mathematical optimisation [2], [3]. The work presented in this paper relies on techniques from the field of mathematical optimisation.

Many problems that industrial engineers must solve require that multiple objectives be simultaneously evaluated while searching for optimal or near-optimal solutions. These problems are sometimes referred to as 'multi-objective optimisation' (MOO) problems. Handling multiple objectives simultaneously makes MOO problems more difficult to solve and more computationally demanding than their single objective counterparts. Moreover, the reality of uncertainty in real-world systems further complicates solving these problems in real life.

Researchers and practitioners often combine techniques from the fields of mathematical optimisation (metaheuristics in particular) and computer simulation (discrete-event simulation in particular) in an attempt to solve MOO problems in uncertain or stochastic systems. It is from this combination of techniques that the field of simulation optimisation (SO) emerges.

As in mathematical optimisation and computer simulation, SO techniques can be used to create decision support system (DSS) tools to assist managerial decisions in complex problems such as those described here.

We present in this work the development of an SO DSS tool that can be integrated into commercial simulation software to solve MOO problems in stochastic systems. We recognise that powerful commercial packages like OptQuest exist, but we desire to develop a package including the novel procedures for ranking and selection from Yoon [4] in future.

The paper is organised as follows: after this introductory section, the second section provides a brief literature review. In the third section, the architectural design of the DSS tool is illustrated. The fourth section discusses the development, implementation, and validation of the tool, while the fifth section discusses its practical testing. The sixth section concludes this work.

2 LITERATURE REVIEW

This paper reports on the progress of an ongoing project. The first paper [5] related to the project was written in 2017, and the present paper is its direct follow-up. In the earlier paper, we proposed a solution architecture to improve the existing simulation optimisation technique of a simulation software package in the multi-objective optimisation context. The software in question is Tecnomatix Plant Simulation (TPS), a commercial, discrete-event simulation software package developed by Siemens. The paper provided a comprehensive literature study on which the content of the proposed solution was based. We refer the reader to the paper for details on the literature aspect of the present work. The focus of the present paper will be on the development, implementation, and validation of the proposed solution. A brief literature review is nonetheless provided in this section to support subsequent sections.

2.1 Simulation optimisation

The term ‘simulation optimisation’ (SO) is an umbrella term for techniques used to optimise stochastic simulation problems [6]. These are optimisation problems that are subject to stochastic behaviours, as described in section 1. We refer to them simply as ‘SO problems’ in this work.

Because of the stochastic behaviours, solving SO problems consists of seeking estimates of the best solutions through a number of n simulation replications. We discuss this further in the next subsection.

In their work, Fu *et al.* [7] distinguished between two kinds of approaches to solving SO problems: one in which a constraint set (possibly unbounded and uncountable) is provided over which an algorithm seeks improved solutions; and one in which 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 solution. According to Fu *et al.* [7], the focus in the first approach is on the searching mechanism, whereas in the second approach, statistical considerations are paramount.

The work presented in this paper uses the former approach in the multi-objective optimisation (MOO) context.

2.2 2.2 Multi-objective simulation optimisation

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

$$\begin{gathered}\text{Minimise } (\mathrm{E}[\mathrm{f}_1(\mathbf{x},\xi)], \mathrm{E}[\mathrm{f}_2(\mathbf{x},\xi)], \dots, \mathrm{E}[\mathrm{f}_\mathrm{k}(\mathbf{x},\xi)])^\mathrm{T} \\ \text{Subject to} \\ \mathbf{x} \in \mathrm{X}; \end{gathered} \quad [1]$$

where the expression $\mathrm{f_i}(\mathbf{x},\xi),\ \mathrm{i} = 1,\dots,\mathrm{k}$ represents varying values that objective i can assume when solution $\mathbf{x}$ is selected in the presence of element ξ, which is responsible for the stochastic behaviour of the system. $\mathrm{E}[\mathrm{f_i}(\mathbf{x},\xi)]$ is the expected value of objective i. Because it is difficult to obtain the true value of $\mathrm{E}[\mathrm{f_i}(\mathbf{x},\xi)]$ due to ξ, we seek rather for an estimate that can be obtained with some confidence when a number of n simulation replications (or observations) are made.

Consider the notation $\mathrm{f_{ij}}(\mathbf{x},\xi)$, where $\mathrm{j} = 1,\ldots,n$ represents the $\mathrm{j^{th}}$ observations made for objective i; then

$$\widehat{\mathrm{E}}[\mathrm{f_i}(\mathbf{x},\xi)] = \left(\frac{1}{n}\right)\sum_{\mathrm{j}=1}^{\mathrm{n}} \mathrm{f_{ij}}(\mathbf{x},\xi) \qquad [2]$$

is an estimated value for objective i.

We are interested in solving MOSO problems, as described in this section, using the decision support system (DSS) tool whose development and implementation is discussed in section 4. In section 5, we present two instances of real-world MOSO problems that we solve using the DSS tool. We shall only consider bi-objective MOSO problems for the purpose of this work.

In the next section, the architectural design for the proposed DSS tool is presented. This is a more detailed version of its predecessor in the earlier article [5].

3 ARCHITECTURAL DESIGN

We proposed in the earlier article [5] an improved SO process for TPS. This was illustrated using a flow diagram that showed how this could be achieved following a literature study and an analysis of TPS existing SO process and its limitations. We present here an updated version of the proposed solution in the earlier article [5]. The concept is still the same. In this work, however, we show more details.

The flow diagram in Figure 1 illustrates a high-level architectural design of the concept, and shows the inter-process communication between the DSS tool and TPS. The tool is a suite, an external or third-party library, that was named *multi-objective optimisation solver* (*MOOSolver*).

The solid lines in the diagram represent actions controlled by TPS, while the dashed lines represent those controlled by MOOSolver.

We now show how MOOSolver was developed, implemented, and validated.

4 DEVELOPMENT, IMPLEMENTATION, AND VALIDATION

In this section, the development, implementation, and validation of the MOOSolver suite are described. In the first part of the section, we discuss the MOOSolver suite itself. This is followed by discussions on the interfaces that were used to integrate the suite with TPS. We first discuss the C-Interface. A subsection on the limitation of the C-Interface then follows in order to put the next section into context, which is the COM-Interface. In the last section, we describe the MOOSolver user-interface for TPS.

4.1 MOOSolver: A dynamic-link library solver for MOSO problems

We needed MOOSolver to be accessible from TPS as a suite containing the necessary functions that would control the SO process of a user-defined multi-objective SO problem, or model, to be more precise. We also wanted to make it a stand-alone, callable module that would link with TPS at run time. This can be seen in the updated architecture of the improved MOO SO process for TPS that was first presented in the earlier article [5] (Figure 1). It was then decided to implement the suite as a dynamic-link library (DLL).

A DLL is a module or a file 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 run time 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++ networks** [8], 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. So a program written in a highly efficient language is important if these problems are to be solved effectively.

In the next subsection, we briefly discuss the metaheuristic used by MOOSolver, the multi-objective optimisation algorithm using the cross-entropy method (MOO CEM) [9].

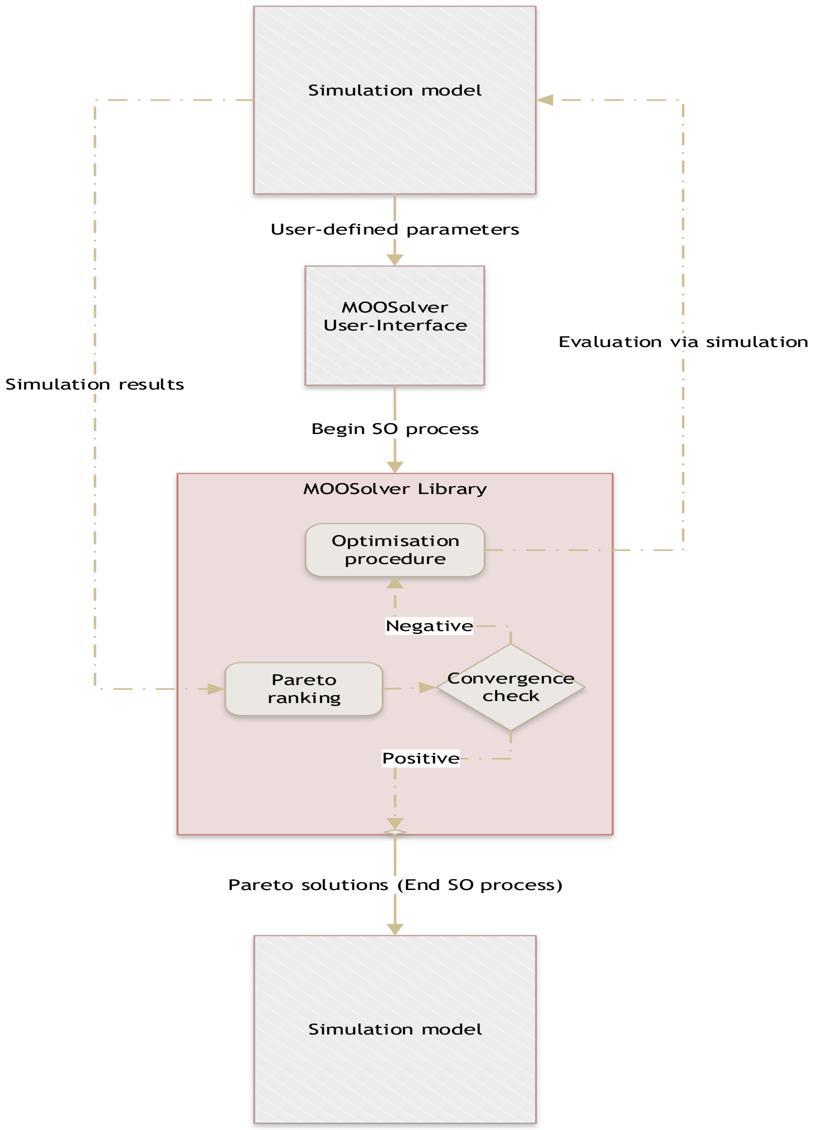


Figure 1: Architectural design of the improved MOO SO process for TPS using MOOSolver

4.1.1 The multi-objective optimisation algorithm using the cross-entropy method (MOO CEM)

At the time of writing this article, one metaheuristic has been successfully implemented as part of the MOOSolver suite, which is the MOO CEM developed by Bekker [9], [10].

The MOO CEM is a population-based metaheuristic that uses the statistical principle of importance-sampling in its search phase and the Pareto approach in solving MOO problems. Its ability to converge to good results relatively fast makes it an excellent metaheuristic for computationally demanding problems such as MOSO problems. Additionally, part of the proposed solution description in the earlier article [5] was the inclusion in the suite of metaheuristics that used the Pareto approach. The MOO CEM was thus a good candidate metaheuristic for MOOSolver. The metaheuristic was implemented as a set of functions, both external and internal.

The integration of the MOOSolver with Tecnomatix Plant Simulation using the C and COM interfaces will now be described, as well as how these interfacing procedures were validated.

4.2 The C-Interface

According to the Tecnomatix Plant Simulation software help guide, the features and functionalities of the software can be considerably extended 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.

TPS C-Interface makes it possible to load an external DLL and call the functions in the DLL from the simulation software. It also makes it possible to manipulate TPS objects from within the DLL. A two-way interaction can thus exist between the two platforms. This opportunity was exploited in the manner illustrated by the diagram in Figure 3a, which shows how MOOSolver and TPS were designed to interact via the C-Interface. Positioned on the boundary between the two platforms in the diagram are the interfacing functions that were used. The functions' colours are indicative of the platform in which the interfacing functions are actually implemented. *DVs* and *OFs* stand for decision variables and objective functions respectively. *Others* represent additional parameters such as the nature of the DVs, the number of DVs, the statistical reliability, etc. (These will be discussed further in Section 4.4.)

4.2.1 Validation

To validate the integration of the suite with TPS thus far, MOO test problems were modelled in TPS. (Please refer to Bekker and Aldrich [10] for details of MOO test problems.) In this way, successful test results would validate not only the metaheuristic but also the data exchange process between TPS and MOOSolver. Selected test results are shown in Figure 2: they compare the results returned by MOOSolver on the left with the results returned by a MATLAB implementation of the metaheuristic, together with the known solution sets to the test problems, on the right [10].

These results validate the C-interfacing procedure implementation. Of course, there was no need to execute simulation runs for these test problems, as they are deterministic. Evaluation of solutions was done by executing a method that contained the objective function equations as illustrated on the activity diagram.

4.2.2 The limitation of the C-Interface

Having reached this point, the next step was to test MOOSolver with actual MOSO problems. It is here that a TPS limitation occurred that was not anticipated when designing the solution architecture.

The goal of integrating the two platforms is so that simulation runs can be executed from the DLL as part of the SO process. This means that simulation runs need to be executed in an iterative manner to support the optimisation procedure in progress (please refer to Figure 1). Unfortunately, it was discovered that a TPS application cannot execute simulation runs while other methods are being executed within the same application. To be more precise: while the method calling MOOSolver in TPS is still executing, it is not possible to simultaneously execute simulation runs from the suite. The simulation runs are postponed, and are executed only after all other executing methods have been completed (including the method calling the suite). This limitation makes it impossible to have a valid SO process using our architecture.

The obstacle was overcome, however, by using a COM-Interface within the existing C-Interface. TPS supports COM-interfacing. In summary, an instance of the TPS application of interest can be opened through a COM-Interface, and simulation runs can be executed via COM. This is possible because the created instance exists as an independent module. It is unaware of the existing C-Interface link between the original TPS application and MOOSolver.

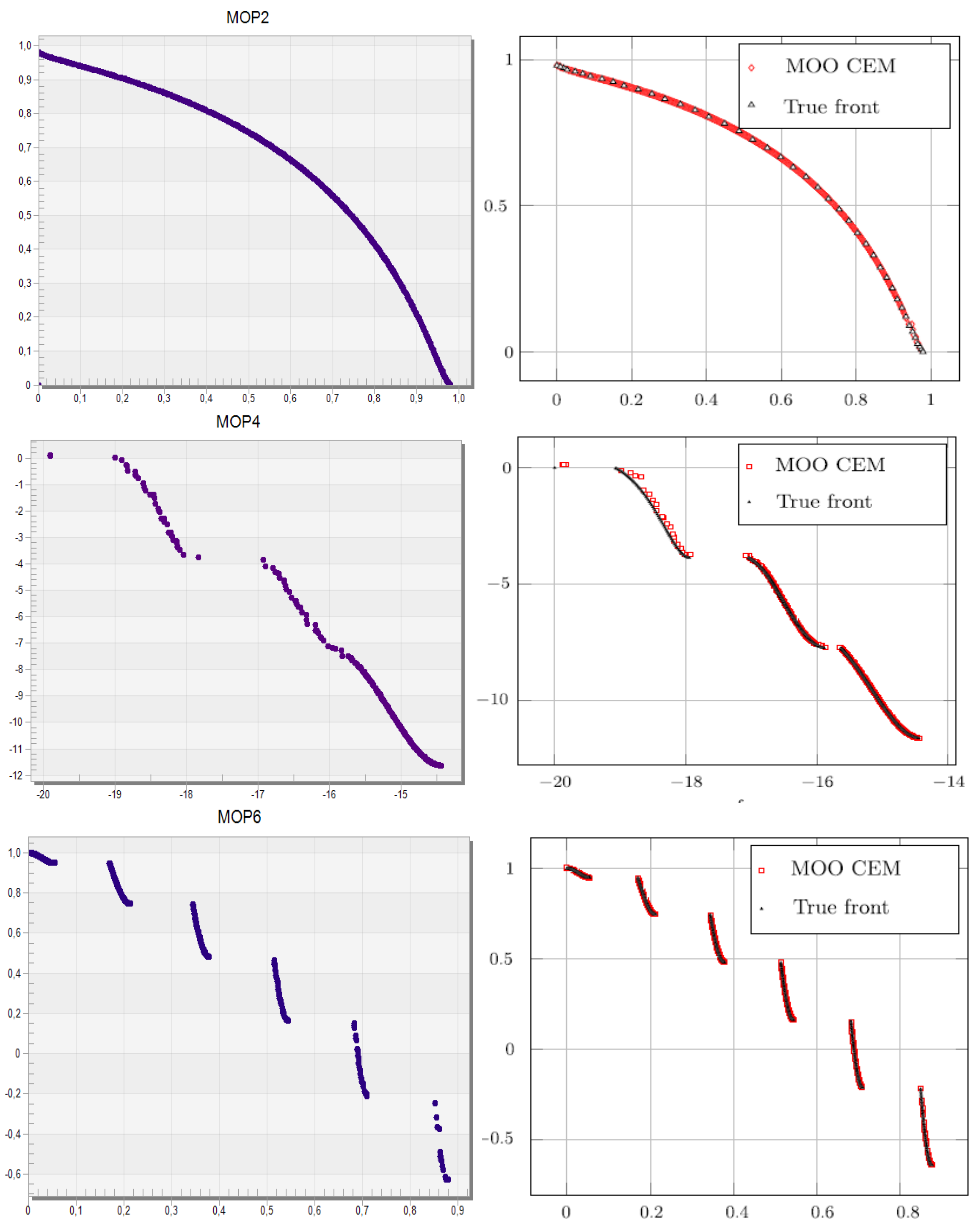


Figure 2: Comparison of selected MOO test problems results obtained by MOOSolver (on the left) and results obtained in MATLAB together with the known results (on the right)

4.3 The COM-Interface

Wikipedia contributors [11] and Microsoft [12] define the component object model (COM) technology as a binary-interface standard for software components. It is a language-neutral way of implementing objects that can be used in environments different from the one in which they were created, even across machine boundaries. It is used to enable inter-process communication object creation in a large range of programming languages.

The COM-Interface was used to overcome the obstacle faced by the C-Interface. Unlike in the case of the C-Interface, the COM-Interface only offers a one-way connection between the two platforms. In other words, it is possible to manipulate TPS from the suite (the calling program) but not the other way round. Using the COM-Interface, MOOSolver can execute simulation runs by opening TPS and loading an instance of the model of interest. This model being opened through COM is totally independent of its original version being executed through the C-Interface. This makes it possible to have both interfacing procedures work in unison. In other words, the solutions found by the optimisation procedure are evaluated via the COM-Interface. The evaluation results can then be accessed once the simulation runs have been completed. The COM-Interface activity diagram in Figure 3b illustrates how this is achieved.

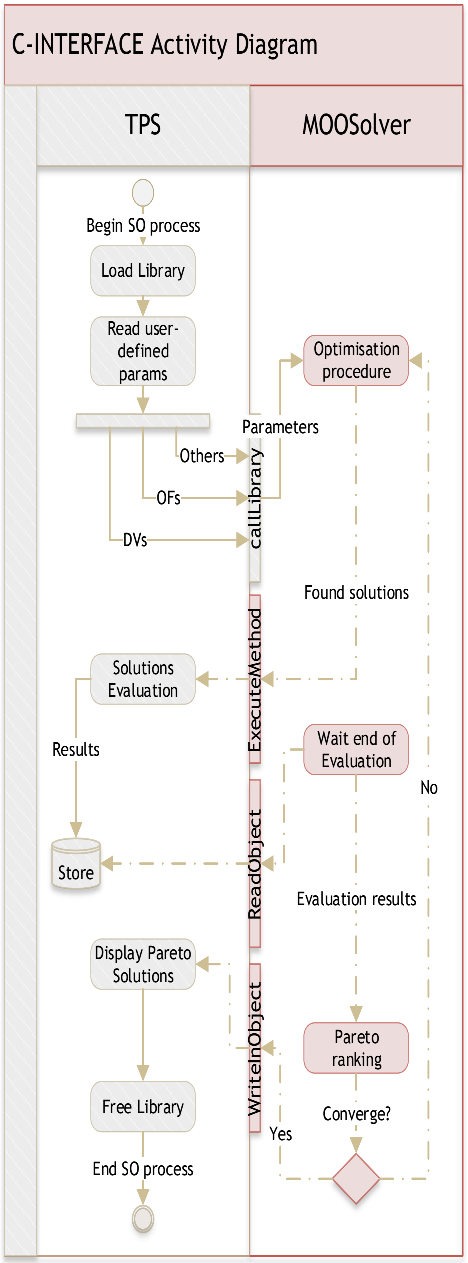
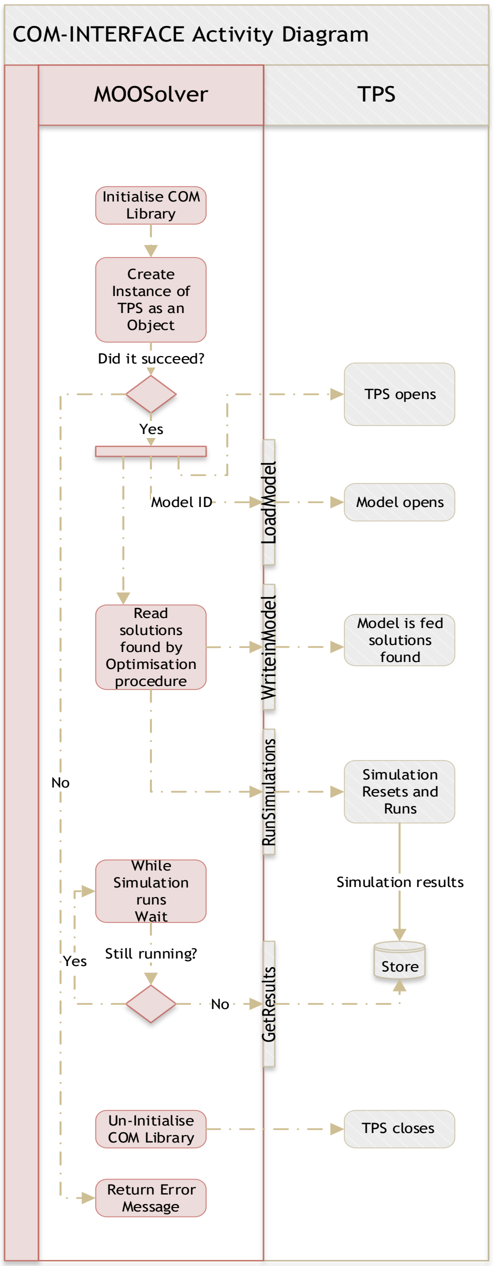


Figure 3a: The C-Interface activity diagram



COM-INTERFACE Activity Diagram
MOOSolver
TPS
Initialise COM Library
Create Instance of TPS as an Object
Did it succeed?
TPS opens
Yes
Model ID
LoadModel
Model opens
Read solutions found by Optimisation procedure
WriteInModel
Model is fed solutions found
No
RunSimulations
Simulation Resets and Runs
Simulation results
While Simulation runs Wait
Yes
Still running?
Store
No
GetResults
Un-Initialise COM Library
TPS closes
Return Error Message

Figure 3b: The COM-Interface activity diagram

Figure 4 shows the part of the C interfacing procedure that is replaced by the COM interfacing procedure in order to make the MOO SO process possible for TPS.

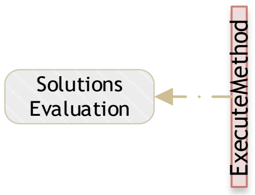


Figure 4: The COM interfacing procedure replaces this part of the C-interfacing procedure

It is important to note that all the activities described in Figure 3 are hidden from the user. The user initiates the entire process via a user-interface (section 4.4), and waits for the final results.

4.3.1 Validation

To validate the COM-Interface is to validate MOOSolver itself as a solver of MOSO problems. This will be done in Section 5. We first provide a functional description of the MOOSolver user-interface.

4.4 The MOOSolver user-interface for TPS

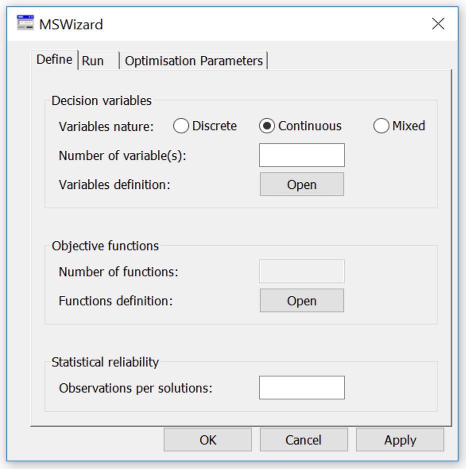
To allow the user to interact with the optimisation suite, a graphical user-interface (GUI) was developed and implemented in TPS. The GUI was inspired by the TPS GAWizard GUI and was named *MSWizard*.

MSWizard currently has three main tabs: *Define*, *Run*, and *Optimisation Parameters*. Screenshots of the wizard can be seen in Figures 5, 6, and 7. The paragraphs below describe the functions of each tab.

In the first tab, 'Define', the user defines the simulation model parameters. The tab has three sections called *group boxes*. In the group box 'Decision variables', the user first specifies the nature of the decision variables (DVs). This information is used by MOOSolver to determine the appropriate metaheuristic to be used during the SO process. We currently use the MOO CEM for all different natures DVs can assume. The user then specifies the number of DVs present in the model and their respective paths so that MOOSolver knows where to find them (see Figure 6a). In the second group box, the user defines the objective functions (OFs) following a similar approach to that in the first group box (see Figures 5 and 6b). The final group box is for the number of observations that the user wants MOOSolver to use for each solution that will be evaluated.

The second tab is the 'Run' tab, where the user starts the SO process when all the parameters have been entered as desired.

Finally, the 'Optimisation Parameters' tab allows the user to specify parameter values for the metaheuristic that will be used during the SO process.

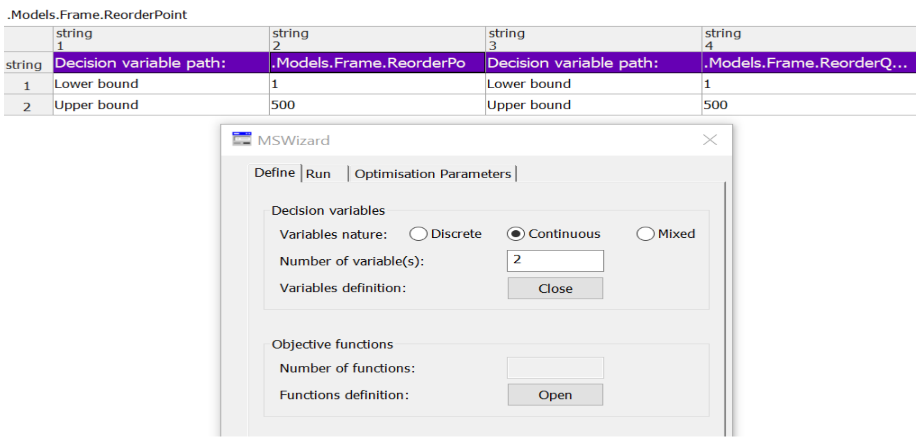


Decision variables
Variables nature: ○ Discrete ● Continuous ○ Mixed
Number of variable(s):
Variables definition:

Objective functions
Number of functions:
Functions definition:

Statistical reliability
Observations per solutions:

Figure 5: MSWizard 'Define' tab



.Models.Frame.ReorderPoint

	string 1	string 2	string 3	string 4
string	Decision variable path:	.Models.Frame.ReorderPo	Decision variable path:	.Models.Frame.ReorderQ...
1	Lower bound	1	Lower bound	1
2	Upper bound	500	Upper bound	500

MSWizard

Define | Run | Optimisation Parameters

Decision variables

Variables nature: Discrete [✓ Continuous] Mixed

Number of variable(s): 2

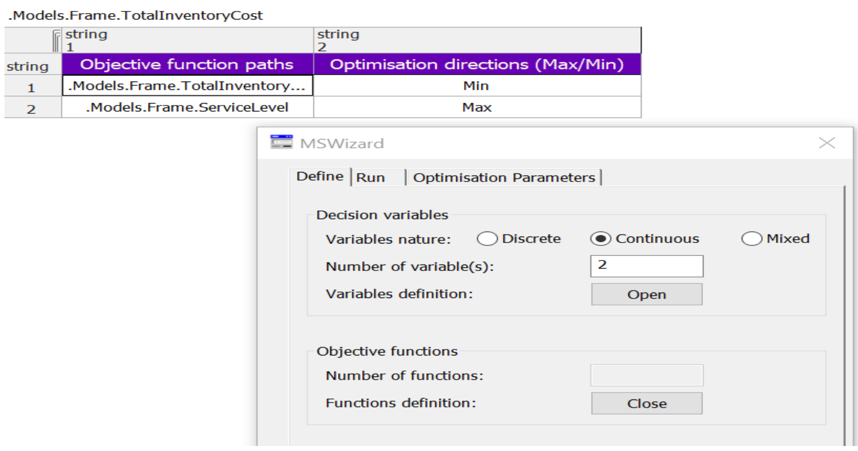
Variables definition: Close

Objective functions

Number of functions:

Functions definition: Open

Figure 6a: DVs definition table



.Models.Frame.TotalInventoryCost

	string 1	string 2
string	Objective function paths	Optimisation directions (Max/Min)
1	.Models.Frame.TotalInventory...	Min
2	.Models.Frame.ServiceLevel	Max

MSWizard

Define | Run | Optimisation Parameters

Decision variables

Variables nature: Discrete [✓ Continuous] Mixed

Number of variable(s): 2

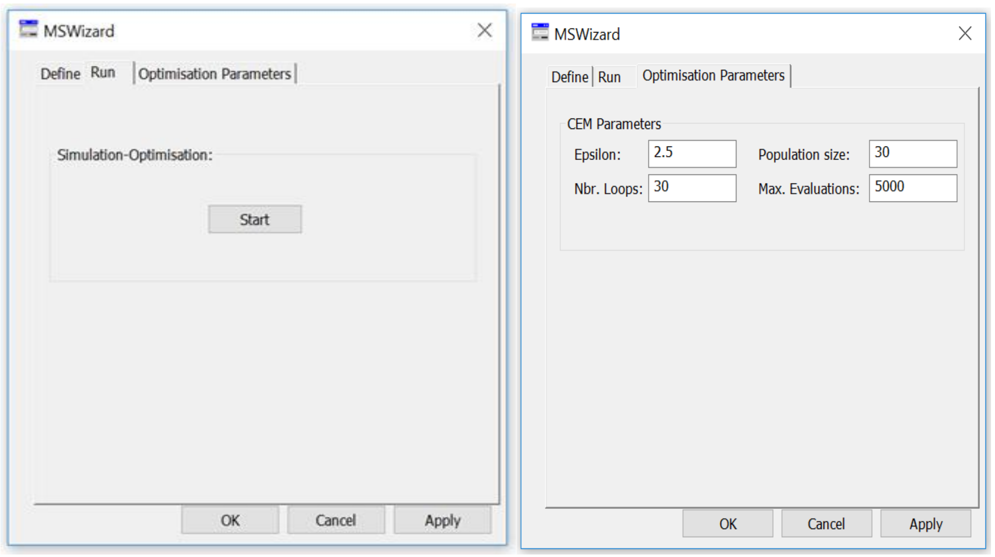
Variables definition: Open

Objective functions

Number of functions:

Functions definition: Close

Figure 6b: OFs definition table



MSWizard

Define | Run | Optimisation Parameters

Simulation-Optimisation:

Start

OK | Cancel | Apply

MSWizard

Define | Run | Optimisation Parameters

CEM Parameters

Epsilon: 2.5 Population size: 30

Nbr. Loops: 30 Max. Evaluations: 5000

OK | Cancel | Apply

Figure 7: MSWizard ‘Run’ and ‘Optimisation Parameters’ tabs

Next, testing of the suite with practical MOSO problems is presented.

5 PRACTICAL TESTING

Multi-objective optimisation is an important aspect of industrial engineering practice. We consider in this article two multi-objective optimisation problems in stochastic systems, and we use them for the practical validation of the proposed simulation optimisation DSS tool. The first problem is the (s, S) inventory problem, while the second is the buffer allocation problem (BAP). Both are well known; their details can be found in Bekker [9] and Bekker and Aldrich [10]. Next, we present instances of these problems.

5.1 The (s, S) inventory problem

Consider a system in which a single, discrete commodity is sold to customers who arrive according to a Poisson process with the 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 $\text{weibull}\ (1{,}8)$, and all demands are processed according to an exponential distribution with a mean of $18\ \text{min}$. 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 (according to a triangular distribution $(14, 12, 20)$ in hours), during which customers still demand the commodity.
The following notation applies:

I_t: Inventory level at time t when customer c arrives.
S_L: Service level.
D_c: Number of units demanded by customer c.
N_c: Total number of customers arriving in period $[0, T]$
$\overline{I_c}$: Total inventory cost during period $[0, T]$.

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. On the other hand, carrying inventory incurs a cost. Holding cost is taken as ZAR10/unit/unit time, and the administration fee of a reorder is taken as ZAR100.

The service level is given by

$$S_L = \frac{\sum_{c=1}^{N_c} D_c - \sum_{c=1}^{N_c} |S_c|}{\sum_{c=1}^{N_c} D_c} \cdot 100\% \ . \qquad [3]$$

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 is 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 $\overline{I_c}$ over period $[0, T]$ and the service level S_L.

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

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

The manager of this process will want to service all customers while carrying as little inventory as possible. Note that the objectives are in conflict, and that 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 set at 5 (5 was considered high enough for the purpose of this test). The simulation model was treated as a terminating system.

The CEM parameters were selected as follows: Maximum Evaluations was made $1\,800$, Epsilon was made 2.5, the Number of Loops was made 10, and the Population size was made 30.

The run time of the SO process took about $09{:}40$ min, and the approximate Pareto front in Figure 8 was obtained.

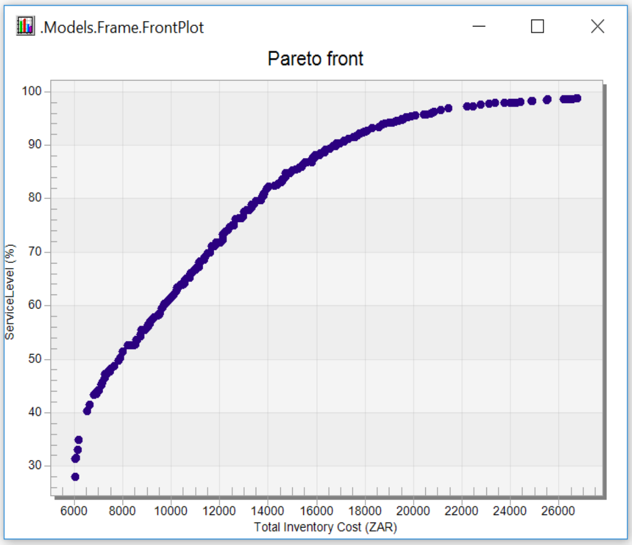


Figure 8: Approximate Pareto front obtained by MOOSolver for the (s, S) inventory problem

To confirm that the metaheuristic converged to the right front for the given problem, we ran MOOSolver further with different values of the Maximum Evaluations parameter. All results are summarised in Table 1.

Table 1: A comparison of the results of the (s, S) inventory problem using different Maximum Evaluations parameters

	Max. Evaluations	No. Elements in Front	Run time
1	$1\,800$	272	$09{:}40$ min
2	$2\,000$	272	$09{:}58$ min
3	$5\,000$	271	$09{:}53$ min

The fronts obtained in tests 2 and 3 looked identical to the one in Figure 8. Because the times and the number of elements in the fronts were similar, we concluded that convergence did indeed occur, and that the front obtained in Figure 8 could be used to support management in making good decisions.

5.2 The buffer allocation problem

Consider a production line consisting of $m = 5$ machines arranged in series. Jobs are processed by all machines in sequential order. The processing times at all machines have, generally, a fixed exponential distribution with the rate $\mu_i, i = 1, 2, \dots, m$. The machines are assumed to be unreliable, and the machine failures are operation-dependent failures (ODF) with Poisson distributions having parameters λ_i. The repair times are exponential with parameters β_i. Table 2 summarises the information about all machines.

The machines are separated by $m-1$ storage areas or niches in which jobs (i.e., *work-in-progress* (WIP)) can be stored (Figure 9). 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.

Table 2: Machines information for the BAP

	m_1	m_2	m_3	m_4	m_5
Processing times (μ)	60 min	55 min	50 min	46 min	43 min
ODFs (λ)	20	20	20	20	20
Repair times (β)	120 min	120 min	120 min	120 min	120 min

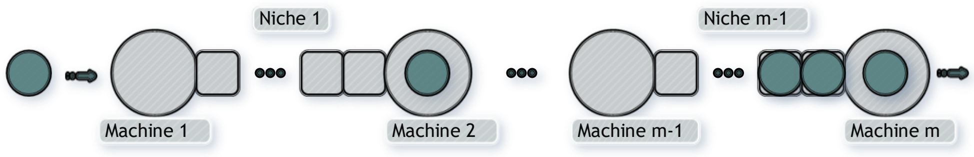


Figure 9: A production line with $\mathbf{m}$ machines and $\mathbf{m-1}$ niches

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

The decision variables are the elements of vector $\mathbf{x} = (x_1, \dots, x_{m-1})$, where x_i is the number of buffer spaces at niche i and is arbitrarily limited as $0 \le x_i \le 20$. The objective functions in this problem are the throughput $T(\mathbf{x})$ (to be maximised) and WIP (to be minimised), denoted by $W(\mathbf{x})$. The system is subject to the stochastic element ξ, caused by the machines' failures and by repair and processing times. The MOSO problem can thus be formulated as

$$\begin{gathered}\text{Minimise } E[W(\mathbf{x},\xi)],\\ \text{Maximise } E[T(\mathbf{x},\xi)]\\ \text{Subject to}\\ 0 \le x_i \le 20.\end{gathered}$$

The problem was modelled in TPS and solved with MOOSolver. The optimisation parameters were taken as follows: Maximum Evaluations was made 5 000; the number of Loops was set at 100; Epsilon was made 1; and the Population size was made 100.

The number of observations per solution was selected as 15 (considered high enough for testing purposes) and observations were made over a period of 10 days. The system was treated as a terminating system.

The approximate Pareto front in Figure 10 was obtained.

To confirm that the metaheuristic converged to the right front for the given problem, we ran MOOSolver further with different values of the Maximum Evaluations parameter. The fronts obtained were then compared with the original front on a single graph. This was done in MATLAB; the result obtained is shown in Figure 11.

Figure 11 shows that the metaheuristic does indeed converge for this problem as the front for 5 000 and 10 000 Maximum Evaluations are almost identical. The results obtained by MOOSolver can thus support management in making good decisions.

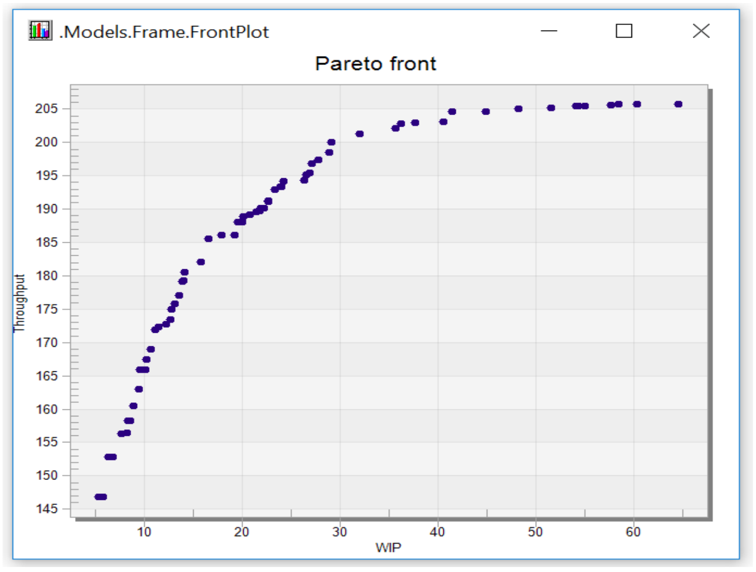


Figure 10: Pareto front obtained by MOOSolver for the BAP problem

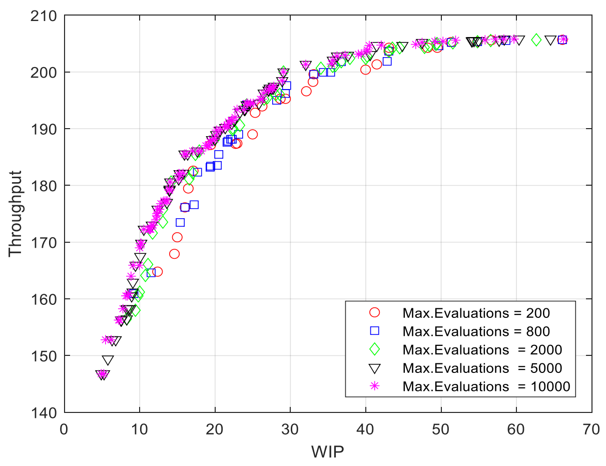


Figure 11: Pareto fronts comparison for various Maximum Evaluations values to test the metaheuristic convergence

6 CONCLUSION

In this paper, the authors reported on the progress made in an ongoing project to add multi-objective optimisation capability to Tecnomatics Plant Simulation. The solution appearing in this work was first proposed in an earlier paper [5] as a theoretical concept. In this paper, the authors discussed its development, implementation, and validation. The solution, which is a decision support tool for multi-objective problems subject to stochastic behaviours, was implemented as a dynamic-link library (DLL) and validated using known test problems and two practical real-world problems. All validation tests showed that the implemented solution behaves and works as expected. Future work related to this project includes the addition of more metaheuristics in the DLL, and testing implemented (but not reported on here) ranking and selection methods.

REFERENCES

[1] **Schwab, K.** 2016. The fourth industrial revolution: What it means, how to respond. [Online]. Available: https://www.weforum.org/agenda/2016/01/the-fourth-industrial-revolution-what-it-means-and-how-to-respond/. [Accessed 18 June 2018].

[2] **Wikipedia contributors.** 2018. Artificial iIntelligence. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_intelligence. [Accessed 18 June 2018].

[3] **Russell, S. and Norvig, P**. 2003. Solving problems by searching, Artificial iIntelligence: A modern approach, 2nd eEdition., Pearson Education, pp 59-93.

[4] **Yoon, M**. 2017. New multi-objective ranking and selection procedures for discrete stochastic simulation problems. PhD Thesis, Stellenbosch University. http://hdl.handle.net/10019.1/103298.

[5] **Bamporiki, T. and Bekker, J.** 2017. Solving stochastic multi-objective optimisation problems with simulation. SAIIE28 Proceedings, pp 1-13.

[6] **Amaran**, **S.**, **Sahinidis, N.V., Sharda, B., & Bury, S.J.** 2014. Simulation optimization: a review of algorithms. Quarterly Journal of Operations Research, p. 12:301–333.

[7] **Fu**, **Michael C.**, Andradóttir, S., Carson, J.S., Glover, F., Harrell, C.R., Ho, Y., Kelly, J.P., & Robinson, S.M. 2000. Integrating simulation and optimisation: research and practice. *Proceedings of the 2000 Winter Simulation Conference*.

[8] **The C++ resources networks**., 2016. [Online]. Available: http://www.cplusplus.com. [Accessed May 2018].

[9] **Bekker, J**. 2012. Applying the cross-entropy method in multi-objective optimisation of dynamic stochastic systems, PhD. Thesis, Stellenbosch University. http://hdl.handle.net/10019.1/71717.

[10] **Bekker, J. and Aldrich, C.** 2011. The cross-entropy method in multi-objective optimisation: An assessment, European Journal of Operational Research, 211(1), pp 112–121.

[11] **Wikipedia contributors.** 2018. Component object model. [Online]. Available: https://en.wikipedia.org/wiki/Component_Object_Model. [Accessed May 2018].

[12] **Microsoft,** 2010. What Is a COM iInterface? [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ff485850(v=vs.85).aspx. [Accessed May 2018].