Master Degree Project

UNIVERSITY OF SKÖVDE

SIMULATION-BASED OPTIMIZATION OF PRODUCT PRIORITIZATION IN A MANUFACTURING FLOWSHOP AT GKN AEROSPACE

Master Degree Project in Industrial Informatics
30 ECTS
Spring term 2013

Patrik Gustavsson

Supervisor: Anna Syberfeldt
Examiner: Amos Ng

Abstract

GKN Aerospace in Trollhättan manufactures different components for aircraft engines and aero derivative gas turbines. A new workshop has recently been installed that is highly automated and includes operations such as laser welding, x-ray and burring. Their production flow is mostly based on first in first out rules except for the x-ray stations where the workers can select which part they want to begin with. Currently, the workers select the parts based on experience which sometimes is not the most optimal solution. Therefore, they want to improve their selection process in order to reduce delays in their production which is why a priority system is required. Simulation based optimization will be used in order to find near optimal priority lists.

Experts at GKN Aerospace have created a discrete event simulation model over their workshop using SIMUL8. In order to find near optimal priority lists for the workshop a new optimization program needs to be created that is compatible with SIMUL8. The program will need to fulfill some requirements to both work as an optimization program and also to be used by the workers at GKN Aerospace.

Since multi objective evolutionary algorithms have the advantage of exploring the objective space, this type of method is utilized. The objective is to minimize the total delay in the system which makes this a single-objective optimization problem but since the simulation model contains stochastic behavior the result will differ between each run and therefore several replications is needed. With several replications more outputs can be read, in this case mean value and standard deviation will make this a multi objective optimization problem.

The question that is investigated throughout this thesis is: Can a multi-objective evolutionary algorithm efficiently find a robust solution with low delays by considering both mean value and standard deviation as objectives?

The aim of this project is to create a priority system that uses an optimization algorithm which together with the simulation model can optimize the production at GKN Aerospace. The purpose of the optimization algorithm is to find robust solutions with low delays in order to improve the production at GKN Aerospace. With the priority system GKN Aerospace may improve their process by following priority lists which has been optimized and therefore reduce waste in form of delays. With the aim of reducing both standard deviation and mean value, a robust solution may be found which means that a small change in the production will not disturb the outcome of the workshop.

During this project a priority system was created that considers both the developer and the workers at GKN Aerospace. The priority system is web based where the workers uses a web browser to access the priority list while the developer uses an experiment system in order to improve the underlying optimization algorithm. With the experiment system the non-dominated sorting genetic algorithm II was implemented in order to solve the real case problem.

Contents

1 Introduction 1
1.1 Background 1
1.2 Problem definition 2
1.3 Research methodologies 3
1.4 Aim and objectives 4
1.5 Limitations 4
1.6 Report structure 4
2 Theory 5
2.1 Prioritization 5
2.2 Simulation 6
2.3 Optimization 7
2.4 Robustness in prioritization 9
2.5 Evolutionary Algorithms 9
2.5.1 Encoding 10
2.5.2 Initialization 10
2.5.3 Selection for mating pool 11
2.5.4 Crossover operator 11
2.5.5 Mutation operator 12
2.6 Simulation-based optimization 12
2.7 Development of a simulation-based optimization system 13
2.7.1 Requirements 13
2.7.2 System architecture 14
3 Methodology 15
3.1 Create the experiment system 15
3.2 Select algorithm 16
3.3 Implement/improve the algorithm 17
3.4 Run experiments 17
3.5 Algorithm approved? 18
3.6 Create the priority user interface 18
3.7 User acceptance test 19
4 Case study 20
4.1 Workshop 20
4.2 Simulation model 21
4.3 Priority system 22

5 Experiment and priority system 23
5.1 Conceptualization 23
5.1.1 General Optimization Framework 24
5.1.2 Software design 26
5.2 System architecture 27
5.2.1 Experiment system 27
5.2.2 Priority system 28
5.3 Implementation of SIMUL8 problem 29
6 Optimization algorithm 30
6.1 Literature review 30
6.2 Benchmark problems 32
6.3 Implementation of NSGA-II 33
6.3.1 Operators for real input problems 33
6.3.2 Operators for combinatorial input problems 34
7 Experiments and results 36
7.1 ZDT benchmarking 36
7.2 TSP benchmarking 37
7.3 Real case experiments 40
8 Analysis and discussion 43
9 Conclusion and future work 45

Figures

Figure 1 priority problem in queues. 2
Figure 2 Priority list to select correct product from a queue. 5
Figure 3 SIMUL8 program. 7
Figure 4 Single-objective problem results. 8
Figure 5 Multi-objective problem results. 8
Figure 6 Example of a sample versus a robust optimal solution for a function f(x). 9
Figure 7 Illustration of a general evolutionary algorithm. 10
Figure 8 Encodings from left to right, binary, permutation and value. 10
Figure 9 Binary tournament selection. 11
Figure 10 Uniform crossover, transfers part of the encoding based on mixing ratio. 11
Figure 11 Bit-inversion mutation. 12
Figure 12 Process of simulation-based optimization 12
Figure 13 Work approach for this project. 15
Figure 14 Turbine frame structure. 20
Figure 15 Model of the workshop. 21
Figure 16 Concept of the priority system. 22
Figure 17 UML model of ParameterList and Parameter. 24
Figure 18 General Optimization Framework. 26
Figure 19 Overview of experimental framework. 26
Figure 20 Full overview of the priority system for both developer and user. 27
Figure 21 Picture of the experiment system. 28
Figure 22 Web interface for use of priority list. 29
Figure 23 Non-dominated sorting. 30
Figure 24 Crowding distance calculation. 30
Figure 25 NSGA-II process. 31
Figure 26 TSP similar to real case. 33
Figure 27 Single point binary crossover. 34
Figure 28 Order crossover for combinatorial inputs. 35
Figure 29 Mutation for permutation encoded inputs. 35
Figure 30 Greedy crossover, constructs the offspring based on the TSP problem. 35
Figure 31 Results for NSGA-II on ZDT1 problem. 36
Figure 32 NSGA-II on TSP as black box, with std. dev. (left), without std. dev. (right). 38
Figure 33 NSGA-II on TSP as white box, with std. dev. (left), without std. dev. (right). 38
Figure 34 NSGA-II on TSP treated as black box (left) and treated as white box (right). 39
Figure 35 Hill-climbing (left) versus NSGA-II (right) on deterministic single-objective TSP. 40
Figure 36 Results from NSGA-II without std. dev. on SIMUL8 problem. 41
Figure 37 Results from random search on SIMUL8 problem. 41
Figure 38 Results from NSGA-II without std. dev. on deterministic SIMUL8 problem. 42
Figure 39 Search space on mean value and std. dev. 44

Tables

Table 1 Specification of ZDT problems. 32
Table 2 NSGA-II settings for ZDT benchmark problems. 36
Table 3 NSGA-II settings for TSP benchmark problem. 37
Table 4 Hill-climbing settings for TSP benchmark problem. 39
Table 5 NSGA-II settings for real case. 40

Equations

Equation 1 Preference based calculated fitness value. 8
Equation 2 Standard deviation based on samples. 9
Equation 3 Calculation of TSP distances. 33
Equation 4 Polynomial mutation 34

1 Introduction

This chapter introduces the reader to the problem and why it is important to solve it. The chapter covers concepts of solving this kind of problem by introducing evolutionary algorithms, simulation and simulation-based optimization. Then this chapter covers more detailed problem definition, research methodologies, aim and objectives of this thesis, limitations of the project and finally an explanation of the structure of the report.

1.1 Background

GKN Aerospace in Trollhättan manufactures different components for aircraft engines and aero derivative gas turbines in both commercial and military markets. A new workshop has recently been installed that is highly automated and includes operations such as laser welding, x-ray and burring. Their production flow is mostly based on first in first out rules except for the x-ray stations where the workers can select which part they want to begin with. Currently, the workers select the parts based on experience which most often is not the best solution. This is because the workers only comprehends their own work area and tries to make the best solution for themselves and the problem is that the best solution for each work area are seldom the best solution for the complete workflow. Therefore, the company wants to improve their selection process in order to reduce delays in their production which is why a priority system is required.

Experts at GKN Aerospace have created a discrete-event simulation model over their workshop using SIMUL8. The model contains all operations (laser welding, x-ray, etc.), workers, fixtures and buffer zones to make the model as realistic as possible. The model starts with predefined products specified by a resource enterprise system at GKN Aerospace in order to set the model to the same state as the reality. These products specified in the model need priority numbers which makes up the priority list of the workshop. The outputs from the model are delays for the different products.

To solve the problem of selecting parts to the x-ray operation the priority system needs to work with the simulation model by optimizing the priority list. This is called simulation-based optimization when an optimization program works together with a simulation model. The optimization program should focus on minimizing the delays of the products but since the simulation model is stochastic the delays will vary between each run. Therefore, the optimization program should find robust solutions i.e. solutions where the delays have small variation between each run. This way small change in the production e.g. late product, machine breakdown, etc. will not affect the outcome significantly.

The idea is to find solutions that have low delays and low standard deviation (robust solutions) which means that the single-objective of finding solutions of low delays can be interpreted as two-objectives including the standard deviation. Since two objectives exist when solving the problem, this kind of problem can be categorized as a multi-objective optimization problem. Evolutionary algorithms are effective algorithms to solve multi-objective optimization problems. This is because of their population based natural evolution that they can explore large search spaces with less computational time (Carson & Maria, 1997).

By interpreting the single-objective problem as a multi-objective problem a question is formed: *Can a multi-objective evolutionary algorithm efficiently find a robust solution with low delays by considering both mean value and standard deviation as objectives?*

1.2 Problem definition

In order to optimize the production for this workshop, an optimization program needs to be created that solves optimization problems and are compatible with SIMUL8. To communicate with SIMUL8 the optimization program needs to:

- Change inputs for the, products that are currently in the production, priority list, workers at each work area and production rate;
- Start the simulation with specified simulation time, full simulation speed and without animation; and
- Receive delay outputs to measure the performance of the workshop with the specified priorities.

The optimization system should produce a priority list for the workers to simplify the selection process at the x-ray stations. Since it is a priority list and the real-world case contains stochastic behavior the products may enter the operations in different orders when running several simulations. Illustrated in figure 1 there is a queue of products and a process, the products are identified by the color and the priority of the product is in the number in the square. In the first run the first event is that a new product arrives to the queue that has a higher priority than the rest and therefore goes to the first location of the queue. Second event the operation is available and the first product in the queue enters the operation, in this case the yellow product. In the second run the first event is that the operation is available and therefore the first product in the queue enters the operation which in this case is the orange product. In the second event the yellow product arrives and since the priority is higher than the rest of the queue this product goes to the first location of the queue.

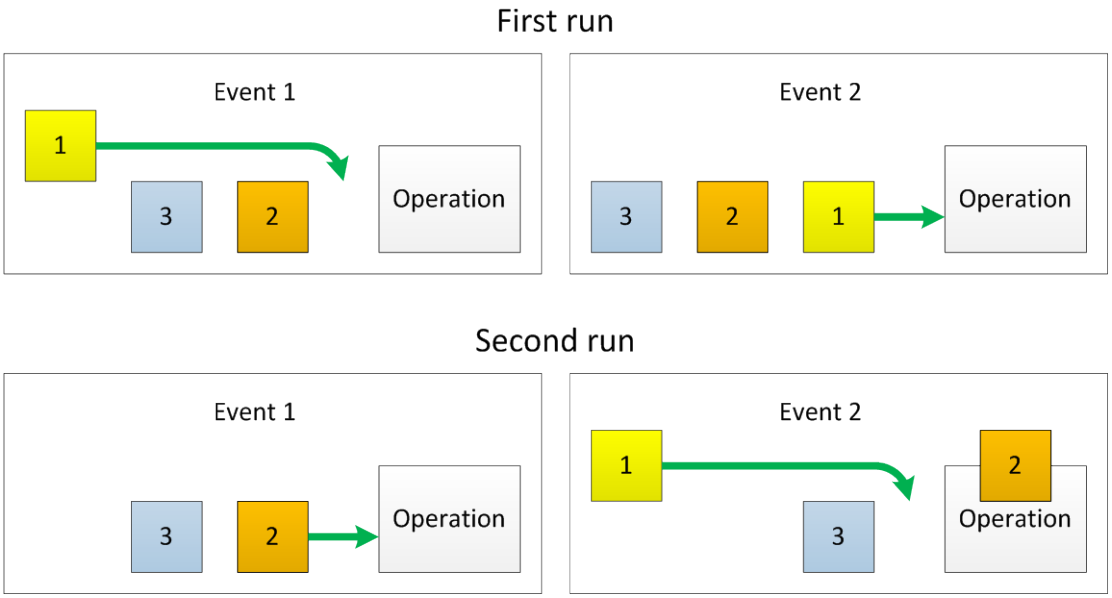


Figure 1 priority problem in queues.

To avoid that this problem occurs repetitively and to make sure that it does not affect the outcome significantly between each run, robust solutions needs to be generated. Robust solutions can only be found when running several replications so that the variation between each run can be measured. Solutions with the lowest standard deviation are the most robust solutions.

A priority list problem has NP-hard complexity since the number of combination follows the permutation rule i.e. complexity of size faculty. The number of combinations increases exponentially when increasing the size of the priority list. Therefore, an optimization method needs to be developed in order to find near-optimal solutions without explicitly testing all combinations. This optimization method should be a multi-objective evolutionary algorithm in order to answer the question formed in chapter 1.1. To develop the method a system needs to be created that can simplify the process of implementing algorithms and make algorithm specific settings.

1.3 Research methodologies

A research methodology is a combination of process, methods, and tools that are used when conducting research. For certain research fields, one methodology is not sufficient to conduct a research, e.g. engineering and information system research. In those cases a multimethodological approach is necessary, where each methodology complement one another. When the research involves creating a system in order to run experiments the research contains a combination of research methodologies with the central methodology system development (Nunamaker, et al., 1991).

In this project, three main research methodologies are used: literature review, system development and empirical study.

- The literature review is used to find information regarding the research fields throughout the project by searching through books, articles, conference proceedings, etc. The information that is collected will help to gain knowledge within the research fields, to select an appropriate algorithm and to implement the priority system in a structured approach.
- The system development methodology is used for the implementation of the priority system, to create the system in a structured approach. The general system development methodology is explained by (Nunamaker, et al., 1991) with five steps: Concept design, constructing the architecture of the system, prototyping, product development and technology transfer. Concept design is the customization of known technology or theoretic advances into potential systems. Constructing the architecture of the system defines the structure of the system, all the necessary relations between objects, functionalities, etc. in order to fulfill the objectives of the system. Prototyping is used for testing new ideas and theories in order to gain knowledge for proving feasibility. If the prototyping step is successful then the product development step is used to build the system. Finally the technology transfer is the acceptance test to an organization which confirms that the different theories and concepts during the development is successfully implemented into the system.
- The empirical study is a methodology where several experiments are run in order to observe the outcome and compare the different methods. This methodology will help to analyze the algorithm and answer the question formed in chapter 1.1.

These three methodologies are structured within a work approach to define how the research is conducted and how this priority system is created. The methodology is explained in further detail in chapter 3.

1.4 Aim and objectives

The aim of this project is to create a priority system that uses an optimization algorithm which together with the simulation model can optimize the production at GKN Aerospace. The purpose of the optimization algorithm is to find robust solutions with low delays in order to improve the production at GKN Aerospace and remove waste in form of delays. To successfully achieve this aim there are several objectives that needs to be completed:

1. Create an experiment system with graphical user interface for easier development of algorithms and easier analysis of results.
2. Identify and compare different optimization algorithms then select the algorithm that is most suited for this case.
3. Implement the selected optimization algorithm in the experiment framework and adjust it to the case study.
4. Run experiments and analyze the results.
5. Does the algorithm produce sufficiently good results? If not, improve the algorithm by changing it with regards to the analysis then continue from the fourth objective.
6. Create the priority system which should be compliant with the experiment system.
7. Present and deliver the priority system to GKN Aerospace.

After the project is done the following question should be answered: Can a multi-objective evolutionary algorithm efficiently find a robust solution with low delays by considering both mean value and standard deviation as objectives?

1.5 Limitations

Verification and validation of the simulation model are not included in this project. This is assumed to be finished by experts at GKN Aerospace since they have most knowledge about their own workshop.

Only one evolutionary algorithm is implemented in this project to fully evaluate how the algorithm performs and how the standard deviation as an objective affects the outcome. The algorithm will only treat the real case problem as a black box. This is because finding the characteristics of the simulation model would require an extensive analysis of both the workshop and the model.

1.6 Report structure

This thesis is structured as follows. Chapter 2 focuses on terminology and theory that concerns this project which includes prioritization, simulation, optimization etc. Chapter 3 presents the work approach used in this project and some research methodologies that it covers. In chapter 4 the case study is described in detail to allow the reader to fully understand this case. Chapter 5 has a detailed description of the created systems with both conceptual models and system architectures. In chapter 6 the literature review and implementation of the algorithm is explained. Chapter 7 covers the experiments done in this project and also the results gained from these experiments. Then in chapter 8 the systems and the results are analyzed, why the results became a certain way and the quality of the systems created. Last chapter in this thesis concludes the report by presenting what was discovered, what was created and future work.

2 Theory

This chapter exists to help the reader to understand important terminology in this report and understand different research fields that were involved in this project. The structure of the theory is based on the research fields, prioritization, simulation, optimization and simulation-based optimization. Finally in the last chapter the theory is summarized and connected to the report by explaining why the different theories are relevant to this thesis.

2.1 Prioritization

Scheduling is one of the most important and also one of the most difficult functions in a production system or service organization. This is because scheduling requires interaction with a lot of other functions in a system due to the extent of information that is required when scheduling. Modern companies often have a large information system that includes many different functions such as customer sales, supplier purchase, worker management etc. which can be connected with a scheduling system. These kinds of system are called enterprise resource planning systems and often have a central server computer which several terminals work with. There are several software companies that develops these kind of systems e.g. SAP, J. D. Edwards, PeopleSoft (Pinedo, 2012). At GKN Aerospace they use an enterprise resource planning system developed by SAP.

Prioritization is one type of scheduling where the jobs are decided by a priority list. An example of prioritization is illustrated in figure 2 where the highest priority (lowest value) goes first into the operation. In stochastic scheduling Pinedo (2012) classifies four different scheduling policies.

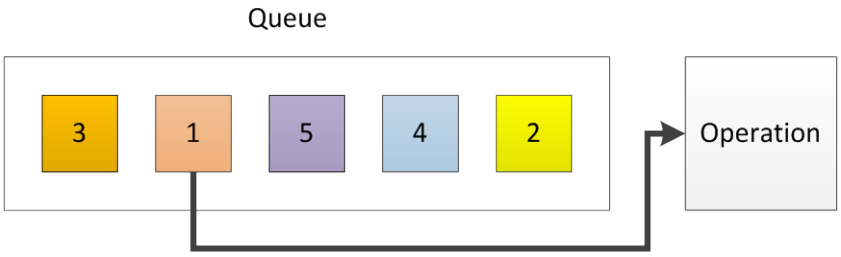


Figure 2 Priority list to select correct product from a queue.

Non-preemptive Static List Policy, in this policy the worker order a job at time zero according to a priority list. This priority list stays the same during the process and every time an operation has finished a new job is ordered according to the priority list. This class of non-preemptive static list policies is also referred as the class of permutation policies.

Preemptive Static List Policy, using this policy the workers orders a job at time zero according to a priority list. This priority list contains jobs with nonzero release date i.e. depending on the current time different jobs may be selected. The priority list does not change during the process and every time an operation has finished a new job is ordered according to the priority list.

Non-preemptive Dynamic Policy, every time an operation is freed the worker can select which job to order. The workers decision at that time may depend on all the information available, e.g. current time, current jobs in other operations, amount of jobs active etc. However, once the job has started the worker cannot interrupt it.

Preemptive Dynamic Policy, in this policy the workers may decide which jobs that should be processed at any point in time. This policy gives the workers the largest amount of freedom. This way it is clear that this policy leads to the best planning if enough information exists to make a reliable decision.

2.2 Simulation

Simulation is the practice of creating a model to imitate a real world system. This model usually has a set of assumptions that simplifies the real world system but is complex enough to achieve the same behavior. These assumptions are expressed as mathematical, logical and symbolic relationship between the entities/objects in the model e.g. people, machines, vehicles, etc. When a model has been developed, assuming having correct behavior, the model can answer "what if" questions about the real world system (Banks, et al., 2010).

There are two kinds of simulations, discrete-event simulation and continuous simulation. Continuous simulation uses a system time clock with a specified time interval; each interval the model is updated to analyze all objects. Since continuous simulation always updates the complete model to analyze each object it is suited for collision detection, deformation analysis etc. Discrete-event simulation on the other hand uses the time-advance mechanism which basically is a system time clock that moves forward according to events in the simulation. These events are only important events that affect the outcome of the simulation, e.g. machine breakdown, product arrives to machine/store, operator arrives to a machine, machine has finished its process, etc. Since discrete-event simulation only includes the important events in a system this method is less time consuming than the continuous simulation approach. Therefore, it is widely used in industries to model a process and experiment with different settings, layout changes, administration changes etc. (Law, 2007).

There are several simulation programs that provide simulation practitioners with a system to create a model with built in objects. Some of these are Plant Simulation, SIMUL8, Quest, FACTS, etc. In GKN Aerospace the discrete-event simulation software SIMUL8 was used to build the model over their newly installed workshop.

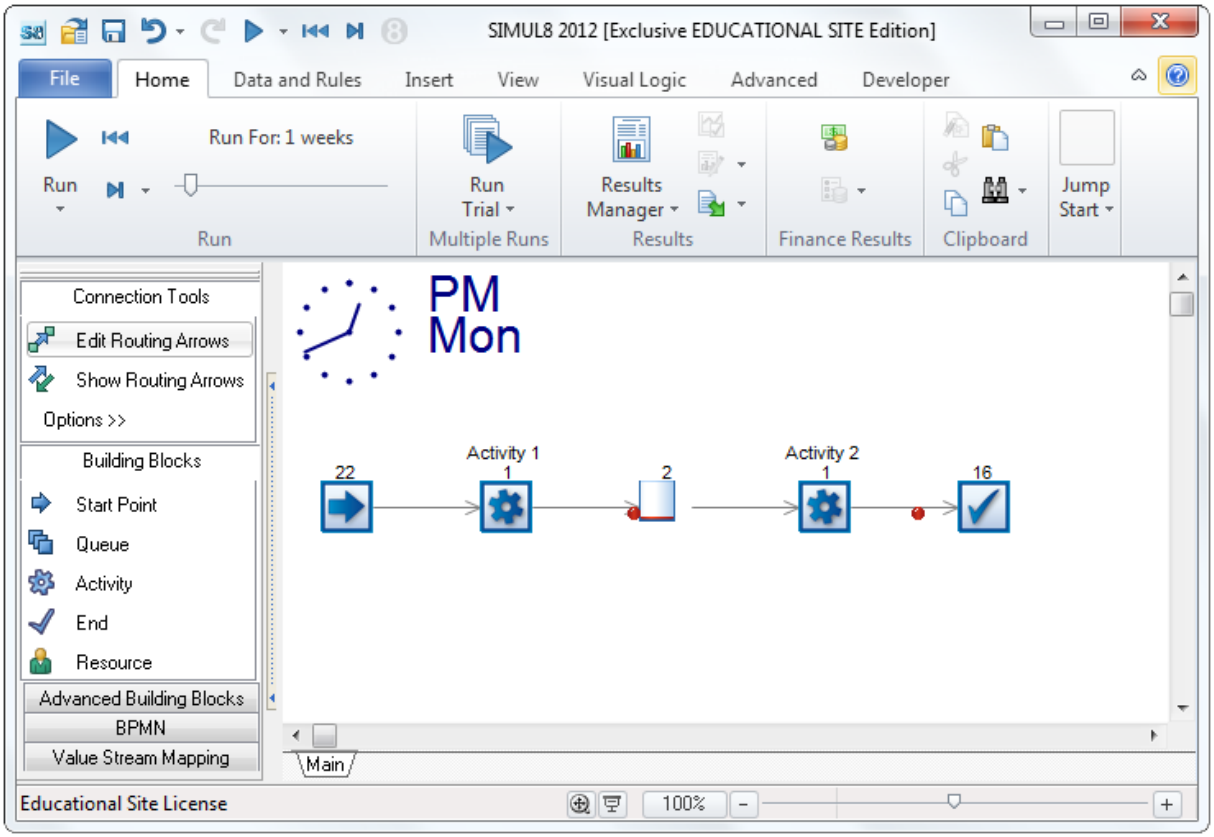


Figure 3 SIMUL8 program.

When constructing a model SIMUL8 has a "what you see is what you get" developing interface shown in figure 3. This includes a set of basic objects to the left. These basic objects/building blocks are then put into the workspace, to the right, where complex models can be created. The workspace also contains a simulation clock which keeps track of time during the simulation. The control panel is on the top where more settings and control possibilities exists. SIMUL8 also includes a built in programming language called Visual Logic which helps building detailed logic into the simulation model (SIMUL8, 2013).

2.3 Optimization

Optimization is the process of generating feasible solutions that has extreme values with regards to one or more objectives (Deb, 2001). Objectives in this case are a measurement to determine how good a solution is and can be measurement of costs, delays, volume, or other factors dependent on the case. When only one objective is used in an optimization it is called single-objective optimization, this kind of problem are often solved by classical search algorithms such as hill-climbing or simulated annealing (Russell & Norvig, 2010). However, in most real-world examples there exists more than one objective that is conflicting. These optimizations are called multi-objective optimization (Deb, 2001).

The main difference between single-objective optimization problems and multi-objective optimization problems, apart from number of objectives, is the number of optimal solutions generated. The single-objective optimization problem can most often be solved by finding one solution that is better than all the other solutions, illustrated in figure 4. The multi-objective optimization problem contains more than one criteria of measurement which most often means that there exists no one solution that is better in all of the objectives, if the

objectives are conflicting. In these types of problems there are on the other hand several solutions that are optimal, based on non-domination (Deb, 2001).

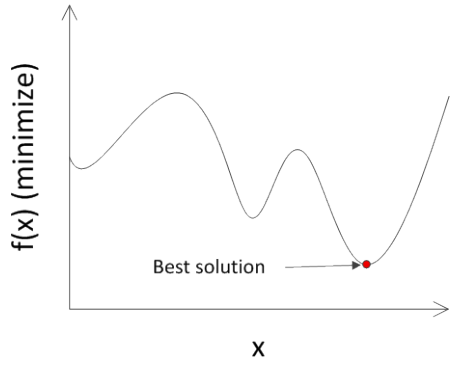


Figure 4 Single-objective problem results.

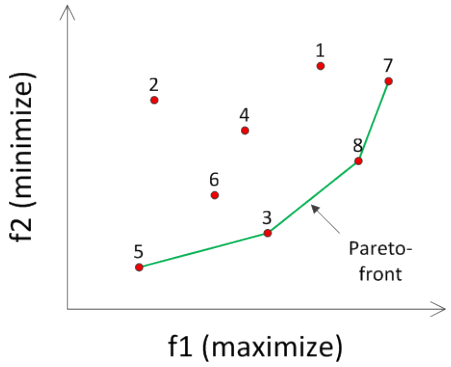


Figure 5 Multi-objective problem results.

Optimal solution is based on non-domination where no other solution in the current collection is better in all objectives. Dominance follows these rules, Solution A dominates solution B if: solution A is no worse than solution B in all of the objectives and solution A is strictly better than solution B in at least one of the objectives. The set of solutions that are non-dominated are called Pareto-front, i.e. no other solution in the collection dominates any of these solutions. In figure 5 a typical result from a multi-objective optimization problem is presented where the objectives are to maximize f1 and minimize f2. Example of domination in this case is solution 4 dominates solution 2 but not solution 1. The Pareto-front in this case consists of solutions 3, 5, 7 and 8.

A problem that arises with a collection of optimal solutions is which solution will be selected. There are two approaches to solve this, the preference based approach and the ideal approach according to (Deb, 2001). The preference based approach uses a weight constant to transform the multi-objective to a single objective, this way a single-objective optimization algorithm can be utilized for solving the problem. In this approach only one solution is generated which solves the selection problem. The transformation is calculated according to equation 1 where f_1 to f_n are all the objectives and w_1 to w_n are the weighted constant for each objective.

$$F = f_1 * w_1 + f_2 * w_2 + \cdots + f_n * w_n$$

Equation 1 Preference based calculated fitness value.

It is hard to define weight constants since different objectives are hard or even impossible to compare, e.g. finding the best solution to produce where first objective is to maximize throughput per hour and second solution to minimize products in the production flow.

The ideal approach works with all the objectives during the optimization process, the solutions are then generated without any weight. This way a collection of solutions are generated and as explained in the example of figure 5 there exists a set of optimal solutions (Deb, 2001). To select a solution from the Pareto-front a decision maker needs to be involved and decide what trade-offs is needed. Similar to the preference based approach a weight is needed to find one solution but instead of generating only one solution, that may have sorted out several other good solutions, all the optimal solutions are presented which is weighted by the decision maker in the end.

2.4 Robustness in prioritization

The primary focus of optimization is usually to find the one optimal solution or the set of optimal solutions which dominates all other solutions. However, in practice the problems are seldom deterministic which means that the solutions may have stochastic behavior. Therefore, a theoretically global optimal solution might be sensitive to variable perturbation and may result in a different set of objective values. In these cases such solutions are misleading and therefore it is important to find robust solutions which is not sensitive to variable perturbation (Deb & Gupta, 2006).

For a single-objective optimization problem a robust solution is defined as a solution which is insensitive to the perturbation in the decision variables (Deb & Gupta, 2006). In the case of a priority list the perturbation in decision variables can be defined as the difference in the arrivals of new objects with priorities. Depending on when the objects arrives to a queue then different objects would be selected as explained in chapter 1.2. To measure the robustness of prioritization it is imperative that several replications is run. This is because the effects of the perturbation of the variables can only be measured when the different arrivals of objects are simulated.

The example in figure 6 illustrates a single-objective optimization problem, to minimize $f(x)$, which is sensitive to variable perturbation in certain areas. The figure shows both the average objective values and the objective values from one sample. The solution x_1 is the global optimal solution, but due to the variance of the objective values between different replications this would seldom be recommended in practice. However, solution x_2 is insensitive to variable perturbation, i.e. has low variance, and is therefore a robust optimal solution.

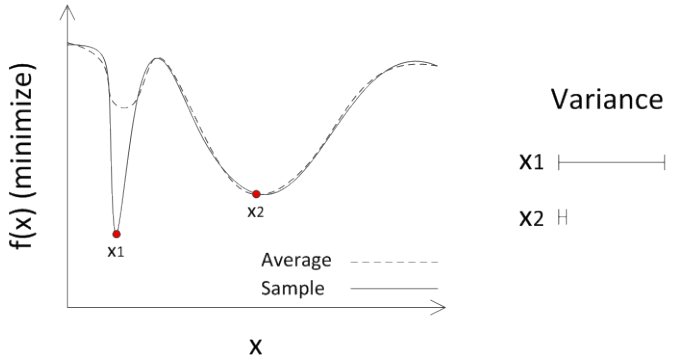


Figure 6 Example of a sample versus a robust optimal solution for a function f(x).

$$s_N = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Equation 2 Standard deviation based on samples.

Standard deviation can be calculated in order to measure the robustness of the solutions. The standard deviation is calculated according to equation 2 where s_N stands for standard deviation with N replications, x_i is the i:th replication objective value and $\bar{x}$ is the mean objective value of all the replications.

2.5 Evolutionary Algorithms

In classical search methods a point-by-point approach is used, where one solution is changed iteratively to hopefully find a better solution and then continue on that path. One problem is that only one solution is generated, this makes it both harder to search the objective space

i.e. the algorithm gets easily stuck on local optima (Halim & Seck, 2011) and impossible to find all optimal solutions since only one solution is generated.

To solve problems of getting stuck on local optima and generate more solutions, evolutionary algorithms are often used. These algorithms are population based and use the process of natural evolution (Zitzler, et al., 2003). The general process of evolutionary algorithm is illustrated in figure 7.

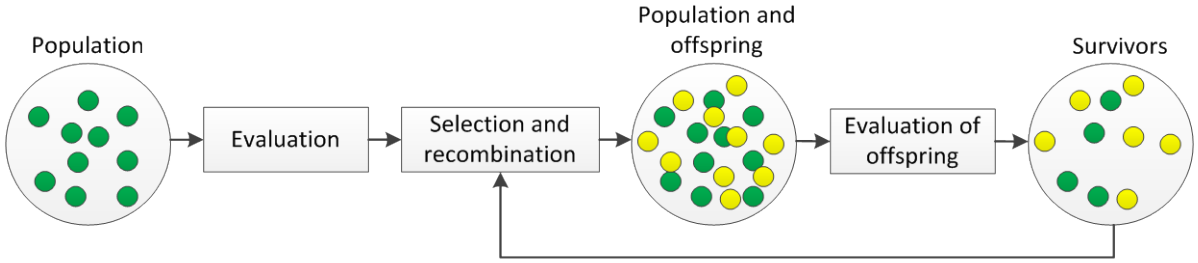


Figure 7 Illustration of a general evolutionary algorithm.

The general evolutionary algorithm begins by initiating a start population and evaluating the solutions. Then starts the main loop by producing offspring, evaluating the offspring and finally select the survivors that will be the next loops population. This continues until a termination condition is true e.g. max amount of evaluations, time constraint, objective achieved, etc. To produce offspring, the evolutionary algorithms go through two steps: selection for mating pool and recombination. Recombination is the process of reproducing one or more solutions to generate offspring. This is typically made in two steps for evolutionary algorithms, especially genetic algorithms, by crossover and mutation.

2.5.1 Encoding

A genetic representation of the solution is necessary when using an evolutionary algorithm. The genetic representation is used to encode appearance, behavior and physical qualities of each solution. In order for the evolutionary algorithm to work it is needed to design a good genetic representation that is both expressive and evolvable.

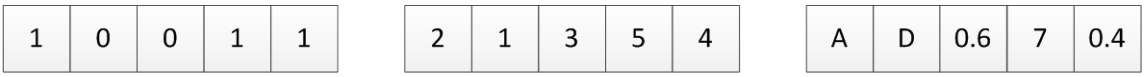


Figure 8 Encodings from left to right, binary, permutation and value.

There are several different encodings that can be used depending on what type of problem that needs to be solved, see figure 8. Binary encoding is strings of bits (0 or 1); permutation encoding is a set of values rearranged in a particular order (mostly integers in a sequence) and value encoding is a collection of variables that can be of different types, integers, real values, characters, etc. The priority list uses the permutation encoding where the encoding consists of the number sequence $1 \rightarrow n$ where n stands for the size of the priority list.

2.5.2 Initialization

The initialization of an optimization is important since that determines the outcome of the optimization. A good initialization helps to speed up the search performance and therefore increase the quality of the results. In many applications the initialization step is random since that often covers a lot of area in the search space. However, if the initialization step would use another algorithm such as factorial design the search performance would increase.

Sometimes some solutions have been generated in an earlier optimization or there are prior knowledge about the problem which can help finding initial solutions. The optimization can then start the search from previous solutions.

2.5.3 Selection for mating pool

Selection for mating pool usually requires two steps, fitness assignment and sampling. Fitness assignment, generated in the evaluation step, is used as a comparator where the fitness value for each solution represents the quality used to determine which solution is dominating the other. The fitness value is assigned based on the objective space and a scalar to define the quality of each solution. Sampling is the process of selecting a mating pool by randomly select solutions to build a new population based on the fitness values (Zitzler, et al., 2003).

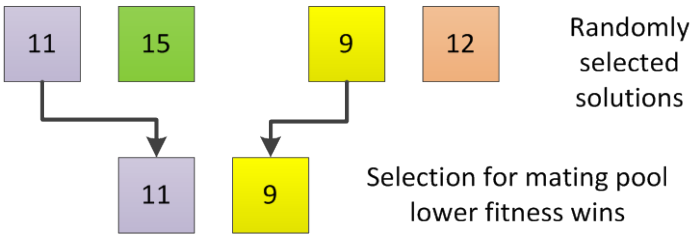


Figure 9 Binary tournament selection.

A commonly used sampling method is binary tournament selection as illustrated in figure 9, where two individuals are randomly picked and the one with the best quality gets selected into the mating pool. In this example the lower value inside the square is the better quality, this continues until the mating pool is filled (Zitzler, et al., 2003).

2.5.4 Crossover operator

The crossover operator uses two or more solutions that act as parents these parents are then combined to produce one or more children. The crossover is dependent on the encoding since the permutation types have strict rules, with unique objects in a specific order, which the binary encoding does not. To produce good quality offspring it is necessary to recombine the parents in a way so that good properties from each parent are inherited. Therefore, the type of crossover is important and there are a lot of different crossovers with different characteristics (Deb, 2001).

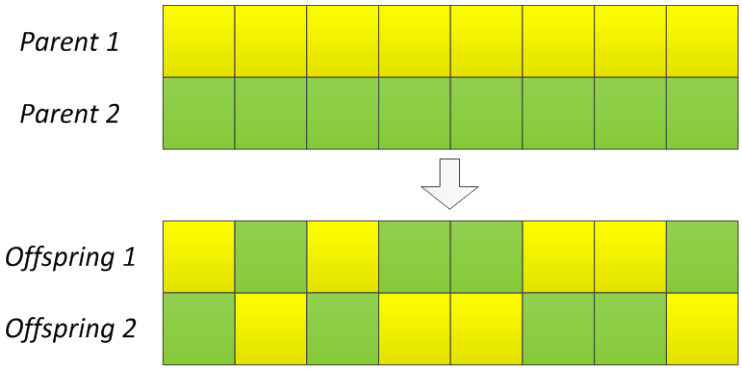


Figure 10 Uniform crossover, transfers part of the encoding based on mixing ratio.

In figure 10 is an example of how a crossover could operate, in this case it is a uniform crossover. The uniform crossover is a simple operator using two parents that transfers parts of the string from each parent based on a mixing ratio. If the mixing ratio is 0.5 then the offspring inherits half of the string from each parent.

2.5.5 Mutation operator

To explore the search space even more the mutation operator exists. This operator changes the solution in a particular way to get solutions that deviates from its parents, that way if the population gets stuck on local optima the algorithm gets a chance to search further. This operator is normally applied to only a few percentages of all the offspring since the offspring should inherit as much as possible from their parents. There are a lot of different mutation operators which each have different characteristics (Deb, 2001).

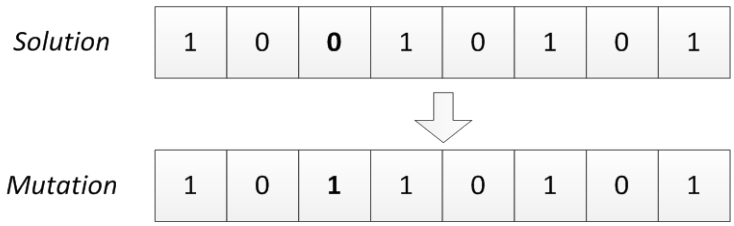


Figure 11 Bit-inversion mutation.

In figure 11 is an example of how a mutation could operate. This mutation is a simple mutation that inverse one bit in a string of bits. The value represented by this string of bits is changed from 149 to 181.

2.6 Simulation-based optimization

Since simulation is a tool to answer "what if" question the use of simulation itself does not provide practitioners with the "best" solutions, only the "best" solutions of the already tried experiments (Law & McComas, 2000). Simulation-based optimization is used to overcome this challenge and achieving the "best" solutions i.e. near optimal solutions.

Simulation-based optimization is the process of finding input combinations in order to optimize some key performances e.g. maximize throughput or minimize work in process. The optimization function works as the search method to explore the objective space to find the optimal performance of the simulation model. The simulation-based optimization method allows an accurate representation of a real system containing complex structure with stochastic behavior, unlike the use of a mathematical model. The optimization also allows the possibility to find near optimal solutions without manually trying to explore the different alternatives. In a system, a lot of input combinations can exist and evaluating all combinations will in some cases be very time consuming e.g. 10^6 combinations * 1 second evaluation time will add up to 11.5 days. Optimization algorithms are therefore used to find a near optimal input combination without explicitly evaluating all possible input combinations (Law, 2007).

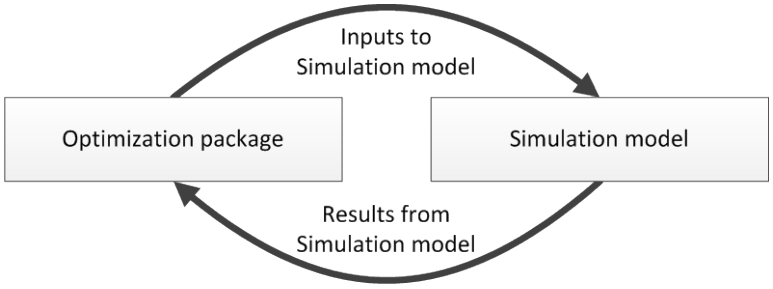


Figure 12 Process of simulation-based optimization

To find near optimal solutions in a discrete-event simulation model an optimization package is used. The optimization package contains an optimization algorithm, e.g. evolutionary strategies, simulated annealing, genetic algorithm. The process of SBO is illustrated in figure 12, where the optimization package sends inputs to the simulation model, the simulation model runs with the inputs and return results back to the optimization package. This process will then run until a termination condition is fulfilled e.g. max number of evaluations or end time.

2.7 Development of a simulation-based optimization system

Even though the simulation-based optimization is a prominent method of finding near optimal solutions on complex and stochastic models, there is a knowledge gap in this research field. Currently, there are no formal and detailed explanations of how simulation and optimization should be integrated. Therefore, the knowledge gap might hinder the effective use of simulation-based optimization. Halim and Seck (2011) presented a SImulation-based Multi-objective Evolutionary OptimizatioN (SIMEON) framework that stresses generality, efficiency, high-dimensionality and usability features. During Halim and Secks research they successfully implemented a simulation-based optimization program that includes a simulation application made in Java. They used the jMetal library for their optimization programming which is a framework for developing multi-objective optimization metaheuristics made for Java (Durillo, et al., 2006).

Halim and Seck successfully identified the integration of simulation and optimization techniques. They successfully implemented a simulation-based optimization method that considers generality, efficianct, high-dimensionality and usability features. They presented the definitions and the structure of how to integrate optimization and simulation techniques in a framework. Following sub chapters present the requirements that were considered during their project and the architecture that was used for their simulation-based optimization software.

2.7.1 Requirements

Depending on what functionalities are required in a system it is important to establish the features from the beginning to avoid unnecessary work. The conceptualization in the SIMEON approach focuses on four main features formulated by (Fu, 2002), generality, efficiency, high-dimensionality and usability (Halim & Seck, 2011).

Generality is the feature that makes sure a system can be used for different kind of applications. This feature is normally most valued according to (Fu, 2002) since it enables the user to apply different optimization methods to different simulation problems. Since the SIMEON framework includes different problem domains the optimizer structure is independent of the problem.

Efficiency, this feature addresses the system performance with regards to computational time. High efficiency in this case is how close the results from the optimization converge to the true Pareto-front with less computational time. This feature is affected by the generality feature and a consequence of treating the simulation model as a black box, which is the common way in simulation-based optimization, is that the information about the problem structure is disregarded (Fu 2002). This problem can be reduced by extending the SIMEON framework with algorithms that make use of the problem structure.

High-Dimensionality is a feature that enables the optimization system to use different kind of inputs for the optimization. Inputs can be continuous, discrete, combinatorial, etc. and this feature is what distinguishes the SIMEON framework from other simulation-based optimization frameworks.

Usability, this feature considers the users of the system that is developed. It is therefore important to adopt the system to whether the users are experts or non-experts in the area.

2.7.2 System architecture

The architecture of the optimization program follows the three layers proposed by (Halim & Seck, 2011) which is coherent with a standard decision support system suggested by (Burstein & Holsapple, 2008).

The presentation layer: This layer provides the user interface which serves as the communication between the other layers.

The problem processing layer: This layer serves as the problem solving/processing layer with which the optimization engine, the communication with SIMUL8 and the communication with the knowledge center will run in.

The knowledge/data layer: This layer provides the storage of knowledge/data/information. With this layer all data from the optimization, settings, etc. that is used by the optimization program are stored. Example of storage types can be Microsoft Excel, databases (Access, MySQL, PostgreSQL, SQLLite, etc.), coma separated values etc.

3 Methodology

This chapter explains the work approach to develop the complete priority system. The approach is based on solving the objectives mentioned in chapter 1.4 in order to successfully achieve the aim of this project. There are seven steps in this work approach illustrated in figure 13, the numbers above each step refers to the more detailed explanation in the following sub chapters.

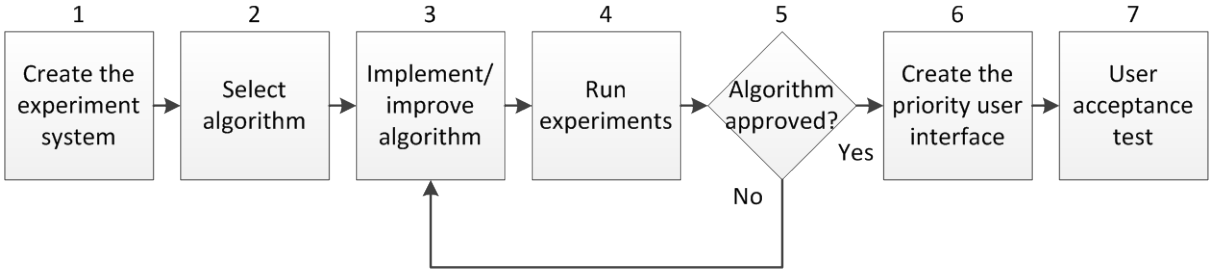


Figure 13 Work approach for this project.

Each step in this work approach, illustrated in figure 13, connects to the actual work done in this thesis. Step 1 and 6 is connected to chapter 5 which is the implementation of the priority system. Step 2 and 3 are connected to chapter 6 how the algorithm was selected and implemented. Step 4 is connected to chapter 7 where all the experiments are listed. Step 5 is connected to chapter 8 where the algorithm is analyzed. Step 7 is not included in this project.

3.1 Create the experiment system

The first step is to create the experiment system. This system sets up the building blocks which are needed in order to easily implement an optimization algorithm and helps with analyzing the results from the algorithm. The system contains as proposed by (Burstein & Holsapple, 2008) three layers which makes up a standard decision support system: the presentation layer, the problem processing layer and the knowledge/data layer, explained in chapter 2.7.2. Building this system facilitates the process of developing the algorithm, since the algorithm needs to be adjusted for this case. It simplifies the experimentation step, to easier change settings in the algorithm and to easier analyze the changes.

The development of the experiment system follows a system development methodology suggested by (Halim & Seck, 2011) the SIMEON approach. Halim and Seck presented a simulation-based optimization implementation that stresses generality, efficiency, high-dimensionality and usability features explained in chapter 2.7.1. This approach is selected since through a literature review it is shown that the SIMEON framework successfully integrated simulation with optimization and has similar requirements as this case. The development of this framework is threefold: searching for information, building conceptual model and implementation.

The search for information is mainly used to investigate how to implement the program, structure of the program, programming language, etc. Internet search engines, programming forums and books are used to find information regarding programming language and examples of how to solve certain problems. Scientific search engines is, however, used to find reliable programming structure, system architecture, etc. that has been checked and verified by other people.

Building a conceptual model it is possible to design and document the software before it has been programmed. This is helpful in several ways, the model can be used as a map to easier implement the different components; the complete program can be evaluated in an early stage of the project to avoid rework and bugs or possible design flaws can be found already in this phase which would otherwise be dreadful during or after the implementation. The conceptual model always considers the four features explained in chapter 2.7.2 and make sure the program design follows these guidelines:

Generality is of importance since the algorithm needs to work with both the simulation model and with benchmark problems. The benchmark problems are needed to both verify that the algorithm works correct and to experiment on benchmark problems that are similar to the case study to save time (optimizing the simulation model is time-consuming).

Efficiency, this feature is not important for the experiment framework but is important for the final product. Since the experiment framework has the purpose of developing and improving the optimization algorithm the efficiency of the final product will be dependent on the experiments made using this framework.

High-Dimensionality is in this case restricted to only two variable types, list of continuous values and list of unique discrete values.

Usability, the users of this experiment framework will only be experts in the area which means that there is a need for extensive settings possibilities. When delivering the final product then the users will be non-experts in the areas of simulation and optimization which means that a simple user interface is needed.

It is important to make the experimental framework modular so that it can adapt to the final product when delivering the system to GKN Aerospace. Therefore, the three layers are needed to encapsulate the presentation layer from the processing layer and knowledge/data layer in order to adapt the graphical user interface for GKN Aerospace in the final product. The other two layers are still the same since the algorithm and storage system does not need any changes.

Once the conceptual model is created the implementation of the program is fairly simple. Each component that is created needs to follow the conceptual model and if additional functionality is needed the conceptual model should also be updated to make sure that the additional functionality works together with the design. When implementing, the use of some kind of debug tool is necessary to verify that the implemented components are correct. Each component should be verified in an early stage since trying to figure out a problem when the complete system is implemented might be difficult.

3.2 Select algorithm

The second step in this project is to select an appropriate algorithm that works well with this case study. A prominent method to do this is literature review which involves searching for information through books, articles, journals, conference proceedings etc. Literature review is a proven method of how to investigate scientific fields which is why this method is used to find, evaluate and select the algorithm. The approach of the literature review is to search information using scientific search engines such as IEEExplore to find articles. Books are used in some cases but journals and conference proceedings come first. Books are mainly used for finding general information about each research field.

The selection process is in three steps. First step is to gather information about simulation, optimization and priorities, which is the main research fields involved in this case, to find the behavior of the different fields and also how these fields can interact with each other to find optimal solutions. Secondly find different optimization algorithms that might be suitable with this case, only select algorithms that have both well documented pseudo-code for the implementation and well documented experiments with other algorithms for the selection process. Finally compare these algorithms and select the most suited algorithm for this case.

The algorithm is selected based on:

- How well it performs with regards to other algorithms in, preferably, similar cases otherwise how well they perform in general. This is measured against how fast (number of evaluations needed), how close it converges towards the true Pareto-optimal front and also the spread of solutions.
- How the algorithm can adapt to a scheduling problem (in this case priority list as a permutation problem). This is compared by looking at the generality and high-dimensionality of the different algorithms. If an algorithm cannot be applied to different kind of problems then it is immediately disregarded. Algorithms that has been successfully used in scheduling problems before are ranked higher.

3.3 Implement/improve the algorithm

This step covers the development of the algorithm which includes both the implementation of the algorithm and the improvement of the algorithm if it was not approved in step five according to figure 13. The development of the algorithm is made in two steps. First the algorithm is programmed using conceptual models and pseudo-code from the authors of the specific algorithm. Secondly the benchmark problems are programmed which are used for verifying the algorithm.

During the first programming phase the structure of the algorithm should follow the building blocks/interfaces made by the experiment framework. To make sure that the algorithm works with the experiment framework during the development it is necessary to use some kind of debug tool. This helps to test the algorithm step by step since it may be difficult to solve a problem after the algorithm has been completely developed. The programming is based on current knowledge and is complemented by searching for information through internet and books.

The programming phase to implement benchmark problems follows the same implementation method. The first benchmarking is to verify the implemented algorithm and make sure it works as it should. The benchmark problems should have documented cases with the selected algorithm so that the implemented algorithm can be compared with others implementation. The next step is to develop a benchmark problems that corresponds with the case study (priority problem with permutation inputs and stochastic behavior) to find out how the implemented algorithm works with this kind of problem.

3.4 Run experiments

When the algorithm is implemented it is time to run experiments and analyze the results. This will be done using empirical studies in three steps, by running experiments with the selected algorithm on selected problem output values is measured to analyze how the algorithm performs. First experiments on general benchmark problems to analyze the

implementation of the algorithm i.e. verification that the algorithm works; secondly experiments on benchmark problem similar to the real case, verification that the algorithm is applicable on the case study; finally running experiments on the real case, the implemented algorithm is compared with a random search algorithm to verify that the implemented algorithm is not random in finding solutions.

In the two last experiments, benchmark on similar case and on the real case, several replications are needed to find both low delay output and robust solutions. Robustness of the outputs requires at least two replications to be calculated, as described in chapter 2.4, preferably more to gain a better confidence. This is a trade-off on confidence and computational effort. More replications lead to better confidence but increase the time to receive results.

3.5 Algorithm approved?

This step is the validation of the algorithm, do the algorithm produce good results that are relevant to this case study, if not then go to step three and change/improve the algorithm. The validation step will focus on three main measurement, producing solutions that have low delays, robustness of the solutions (the variation between each run) and the computational effort. There will be a trade-off between these factors since more computational effort is needed to find lower delays and robust solutions. The approval of the algorithm will be discussed with both involved persons at GKN Aerospace that have been part of the project and the supervisor to find a satisfying trade-off.

3.6 Create the priority user interface

When the algorithm produces satisfying results the final priority system can be created. The only difference between the experiment system and the final priority system is the user interface. This is because the underlying processing and database/knowledge layer will be the same. This step focus on how the workers will use this priority system and therefore most weight will be on the usability feature, explained in chapter 2.7.1, when implementing. When developing the user interface the implementation should consider that the users of this priority system are non-experts in the simulation or optimization area. The priority system should be useful but still easy to use otherwise the workers will not approve the system.

The user interface is strictly dependent on how the presentation layer is encapsulated when the experiment system was created. The implementation of the user interface will follow the building blocks of the experiment system. The development of the user interface will be similar to the creation of the experiment system following the system development methodology SIMEON. Three steps are used within this approach, searching for information, building conceptual model and implementation. The search for information is mainly used to investigate how to implement the user interface, structure, programming language, etc. This is done by using books for information regarding programming language and the internet for examples of how to solve certain problems. To successfully implement the user interface a conceptual model should be created before the implementation to find design flaws or other bugs in an early stage. The implementation will follow the structure of the conceptual model and some kind of debug tool will be used in order to test the interface.

3.7 User acceptance test

Finally when the complete priority system is created it is time to deploy this system to GKN Aerospace which means that a user acceptance test is needed. This test will help the deployment of the system by introducing it to several different stakeholders (workers, production engineers, managers etc.). The acceptance test will be in two main steps, the first step is to get feedback from a few selected persons at GKN Aerospace and change the program (mainly focus on the graphical user interface), and the second step is to present the system for all the different stakeholders in this project.

The first step involves people that are immediately connected with this system, production engineers that help developing and the workers using the system. These people are both experts in the simulation/optimization area and also non-experts that only uses the priority system in their production. To start the test a typical scenario needs to be generated by production engineers at GKN Aerospace to make the test realistic. The test-subjects works with the system against this scenario and logs their experience, afterwards they give feedback about their experience through an interview. The feedback is used to improve the system (mainly the graphical user interface). This way the workers and production engineers get involved with the system and are part of the development which makes it easier to introduce the system.

The second step is to introduce the system to all the other stakeholders. This is the final presentation of the system and the most important acceptance test; if this presentation appeals the decision makers then the system will be implemented. Therefore, it is important to have other stakeholders supporting the system which makes the first step more important to involve the workers and production engineers in the final development.

4 Case study

This chapter aims to give more detailed information about the real world case. Information of the workshop, details about the simulation model and a description about the priority system.

4.1 Workshop

The newly installed workshop at GKN Aerospace produces turbine frame structures which demands high quality controls. The workshop consists of washing machines, x-ray stations, liquid penetrant testing, automatic laser and plasma welding machines, manual welding stations, CNC machines and manual burring machines. There is also a maintenance area in this workshop but GKN Aerospace chose to not include this in the project. In the workshop the same stations are grouped together, i.e. x-ray stations are grouped together and the burring machines are grouped together. All products are moved manually within the area using pallets and only one product at a time is moved. In figure 14 is a typical frame structure that is produced in this workshop.

Figure 14 Turbine frame structure.

The quality control consists of the penetrant testing and x-ray stations. The liquid penetrant testing is a quality check to locate possible surface-breaking defects in the product. This is done by covering the product in a liquid, the liquid penetrates all surface-breaking defects and if they exists the defects will lit up showing the inspector where they exists. The penetrant test is semi-automatic where several products are automatically transported through the penetrant station and in the end of the penetrant is the final inspection. The x-ray station locates defects inside the material. The x-ray station uses the same functionality as going to the dentist when they x-ray the teeth, with the detector and the reflector. The detector which sends then receives the x-ray waves and the reflector which is put behind the object to reflect the x-ray waves sent by the detector. Difference is that the detector and reflector are positioned using robots which mean no manual labor is needed between the photos, but changing products is manual. The film used is analog which are manually inspected to make sure there are no defects.

The manufacturing stations consist of CNC machines, burring machines and welding machines, these machines are responsible for forming the product. The welding machines

are used for welding the wings, see figure 14. Before arriving at the welding machines there are spot welding robots that pre welds the frame. The CNC-machines are responsible for manufacturing the assembly areas. In the machine there can be several products which are automatically manufactured one at a time, but the assembly and disassembly on the fixtures are manual. The burring machines are used for furbishing the assembly areas of the product, these machines require manual labor.

In this workshop the production flow follows mostly first in first out rules except for the x-ray stations where the products are manually selected. The workers select the next part to the x-ray stations based on experience which seldom is the most optimal selection. This is because the workers try to optimize their own work area which does not consider the complete work flow.

4.2 Simulation model

The simulation model was created using the discrete-event simulation software SIMUL8, figure 15. In this model all stations/machines are included except for the maintenance area since GKN Aerospace chose to disregard this. The machines have setup times, operation times and breakdowns settings to represent the real machines. Some of these times are product dependent since the process may be different for each product. The model is initialized with a number of products and this product information can be imported using the resource enterprise software from SAP. This resource enterprise software is used throughout GKN Aerospace in Trollhättan. By importing the product information the model can be initialized with the same state as the real workshop. To set the operation/setup times and to import product information and other settings the model uses the built in programming language visual logic.

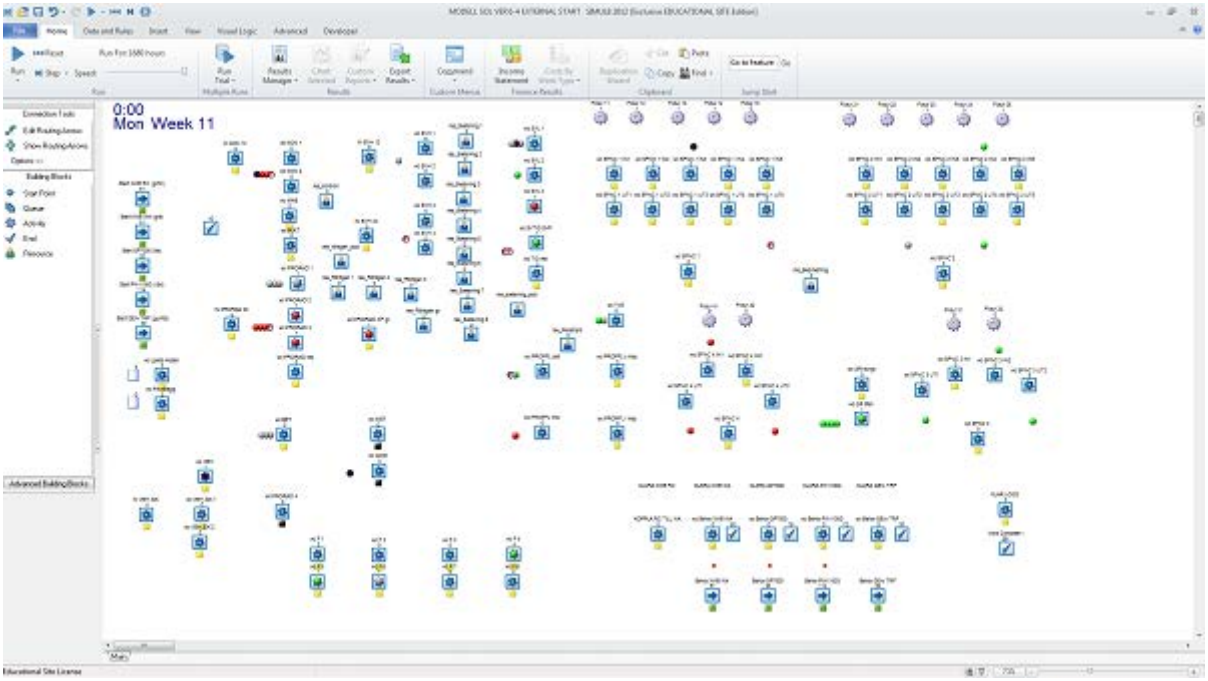


Figure 15 Model of the workshop.

The inputs to the model can be categorized as decision inputs and environment inputs. The decision inputs specify the priority settings for each product and these inputs should be generated by the priority system. In the model there are four x-ray stations which have one

priority list each, this means that the complexity of the problem is, size faculty raised by the power of four $(n!)^4$. The environment inputs specify the current state of the model, what products are in the production flow, how many workers are situated at each work area and what the production rate is for each product type. These settings are specified by the user and the resource enterprise system used at GKN Aerospace. The resource enterprise system specifies the products in the production flow while the user specifies the workers and production rate. The model reads all the settings on reset, i.e. when the reset function is executed a visual logic function is called that sets all settings.

The outputs measured from the model are delays for the different product types and the total delay. These outputs will be used in the optimization program as fitness values where lower delay means better fitness. To ensure the stability of the solution the outputs also needs to be robust i.e. low standard deviation.

4.3 Priority system

The priority system for GKN Aerospace needs a simple-to-use graphical user interface since the users will not be experts in either simulation or optimization. To easily deploy the system to GKN Aerospace the system should be easy to access. Best approach to easy access is the use of internet which is why a web server will be the access point for GKN Aerospace, see figure 16. A web system separates the graphical user interface from the priority system which makes it easier to keep the optimization expertise within the university. This way GKN Aerospace can use the priority system and the university can continue the development of the priority system.

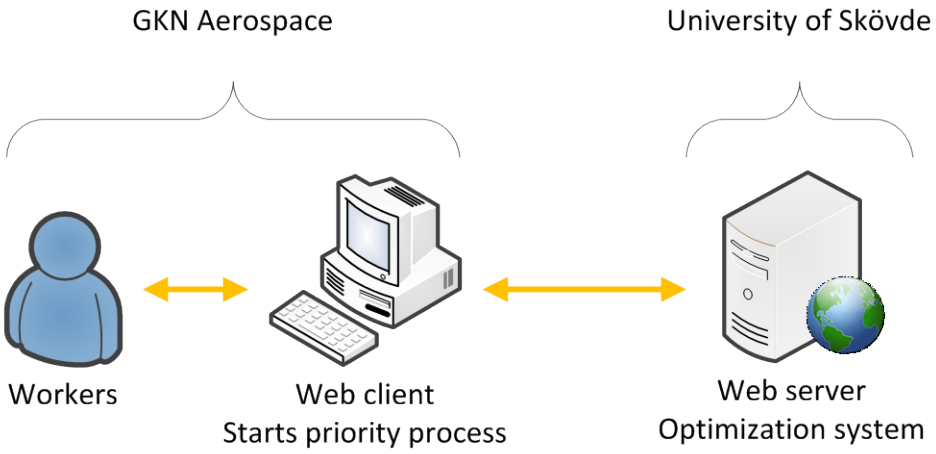


Figure 16 Concept of the priority system.

Illustrated in figure 16 is the concept of the priority system which will be the final product after this project. The priority system should work as a tool to simplify the selection process for the x-ray stations. The concept is to start a priority process at the start of the shift, the server will run the optimization and return the best priority list to the client, after that the workers can use this priority list the entire shift. To produce this priority list the optimization system should use the simulation model created in SIMUL8 to evaluate the delays of the model.

5 Experiment and priority system

In order to facilitate the experimental phase of this project a system needs to be created that can easily change settings and present the results. This is the experimental system which is the base for the entire priority system i.e. this system will be the same as the priority system except for the graphical user interface. Therefore, it is important to separate the three different layers explained in chapter 2.7.2. The experimental system also needs an architecture where different algorithms and different problem types (e.g. benchmark, SIMUL8) can be implemented without reprogramming parts of the system. To simplify the analysis process the results needs to be presented in a graph where different objectives can be measured against one another.

After the experiment system is implemented the only part left in the priority system to implement is the user interface. This user interface will mainly focus on the usability feature to be simple and easy to work with. The users will be non-experts in the simulation and optimization area. Therefore, the user interface should only focus on how they can start an optimization (without any simulation/optimization specific settings) and how they can use the priority list.

C# is the selected programming language in this project due to two main reasons: C# is an object oriented language which makes it easier to implement different algorithms, operators, problems, etc. compared to a procedural or declarative language; C# is designed for the common language infrastructure that is the core of the .NET framework (Hejlsberg, et al., 2010) and this is necessary since the communication to SIMUL8 is possible through the built-in .NET libraries.

Following chapters describes how the priority and experimental system is implemented. First the basic design of the systems i.e. the building blocks on which algorithms, operators and problems are implemented. Second the overview of the complete experimental and priority system. Finally a description of how the simulation problem is implemented.

5.1 Conceptualization

The purpose of the experimental system is to facilitate the development of new algorithms and the evaluation of these algorithms. This is achieved if, algorithms, operators and problems can easily be implemented, settings can easily be changed and the results can be visually presented using graphs or other representative diagrams. When the experimental system is created the priority system (or user interface) should be easy to implement. All this is possible if the system has a good architecture which is why the conceptual phase is important.

Unified Modeling Language (UML) is used as the tool to design the experimental system. UML is a tool used by software engineers to design the software in projects. UML is a visual modeling language which is used to specify, visualize, construct, and document the components of a system. It is used to understand, design, browse, maintain and control information about a system to structure the development (Rumbaugh, et al., 1999). Using UML in the conceptualization phase it is possible to design and document the software before it has been programmed. It is also helpful when implementing new algorithms where the UML model can be used to browse the different components when developing new algorithms. By using UML the system can be evaluated in an early stage to avoid rework

when actually developing. UML helps in finding bugs and possible design flaws in the architecture which might be dreadful when found during or after the implementation.

The system is designed in three layers, graphical user interface layer, processing layer and storage/knowledge layer. First the design of the processing layer will be explained since this is the core of the system and then the complete software design is explained.

5.1.1 General Optimization Framework

The processing layer manages the actual optimization and makes up the core of the system. This core is structured as a framework and is called General Optimization Framework (GOF). The purpose of GOF is to create building blocks for implementation of different problems, operators and algorithms. GOF also focuses on how to evaluate the solutions, this is important since distributed simulation-based optimization may be required. This is because optimizing simulation models are often time consuming and could take hours/days to execute, however distributed optimization would reduce that time.

There are other frameworks made for simplifying the process of implementing meta-heuristics such as jMetal and PISA. jMetal is a framework for developing multi-objective optimization metaheuristics in java (Durillo, et al., 2006) and PISA is a text based interface framework that allows to separate the algorithm-specific part of an optimizer from the application-specific part (Bleuler, et al., 2002). These frameworks do not fulfill all the requirements that are needed in this case but they are used as an information source to successfully implement the GOF.

The key component for easier access of the settings in the algorithms or operators is the ParameterList, illustrated in figure 17. This is a list of parameters containing key, type, value and description. Using this list the overhead system can easily access the parameters for the operator/algorithm dynamically. The .NET framework provides the System.Type class which is a Meta class that is used to describe other classes/types. With System.Type all classes/types can be instantiated dynamically. This is essentially helpful when the developer wants to dynamically select which operators should be used in the algorithm. Using this list, parameters such as max evaluations and type of operator can be accessed.

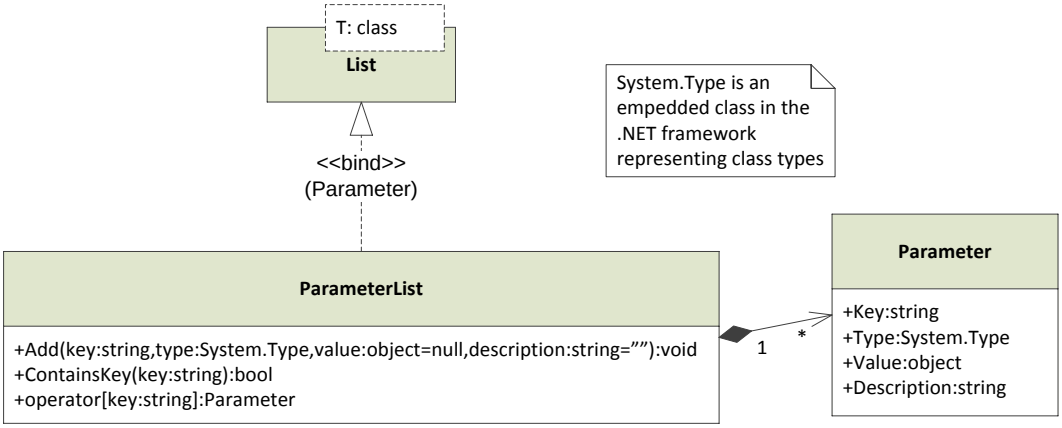


Figure 17 UML model of ParameterList and Parameter.

Each class in the GOF has a specific task to perform and by assigning these specific tasks it is simpler to implement new algorithms/operators into the framework. Illustrated in figure 18 is the conceptual design of the GOF, there are eight main components in this design:

Algorithm, the component that is responsible for generating new solutions based on an algorithm (multi-objective evolutionary algorithm, hill climbing, random search, etc.). This is where the processing of finding optimal solutions takes place.

Operator, component that executes a specific task needed for the algorithm. Explained in chapter 0 an evolutionary algorithm contains three types of operators Selection, Mutation and Crossover. Initialization is also included as an operator type in case a different algorithm in addition to the random initializer should generate the start population.

ProblemManager is the evaluator component which executes the evaluation of each solution. There are two approaches to evaluate, direct (used for benchmark problems) and distributed (used for SIMUL8 problem). The direct approach evaluates the solutions one after the other. The distributed approach may evaluate the solutions by distributing the processing to parallel processes, utilizing several cores of the processor, or through network distribution, utilizing several computers to do the task, or even both. This case the LocalProblemManager uses a direct approach while the NetworkProblemManager uses the network distributed approach.

Problem, this is where the actual problem is implemented. The component has two responsibilities, providing with the solution specific to the problem and evaluating the solution. This design is necessary since only the problem knows what type of inputs and quantity of inputs is necessary e.g. SIMUL8 uses a priority list (permutation input type).

Solution is the component where the inputs, outputs and objectives reside. This is what the algorithm generates in order to find the best combination of inputs for the problem. The component contains additional variables which are specific to solve certain algorithms. When implementing new algorithms that may need some additional information from the solutions then the variables are added here.

Input, this is the input which is abstract to allow different type of inputs i.e. if more input types are necessary they can be implemented. For the SIMUL8 problem the permutation input is necessary.

Output is the output measure from the problem. The output can only accept doubles (which is a real value in C#) since they are the only type of values that can be used for optimization. A solution can be evaluated several times and that is why the output receives samples which generate both mean value and standard deviation.

Objective, in addition to the outputs the objectives exists to define both how the outputs should be optimized and also what value of the outputs is the optimization measure. An output can be optimized either by maximizing or minimizing the output measure which can be both mean value and standard deviation.

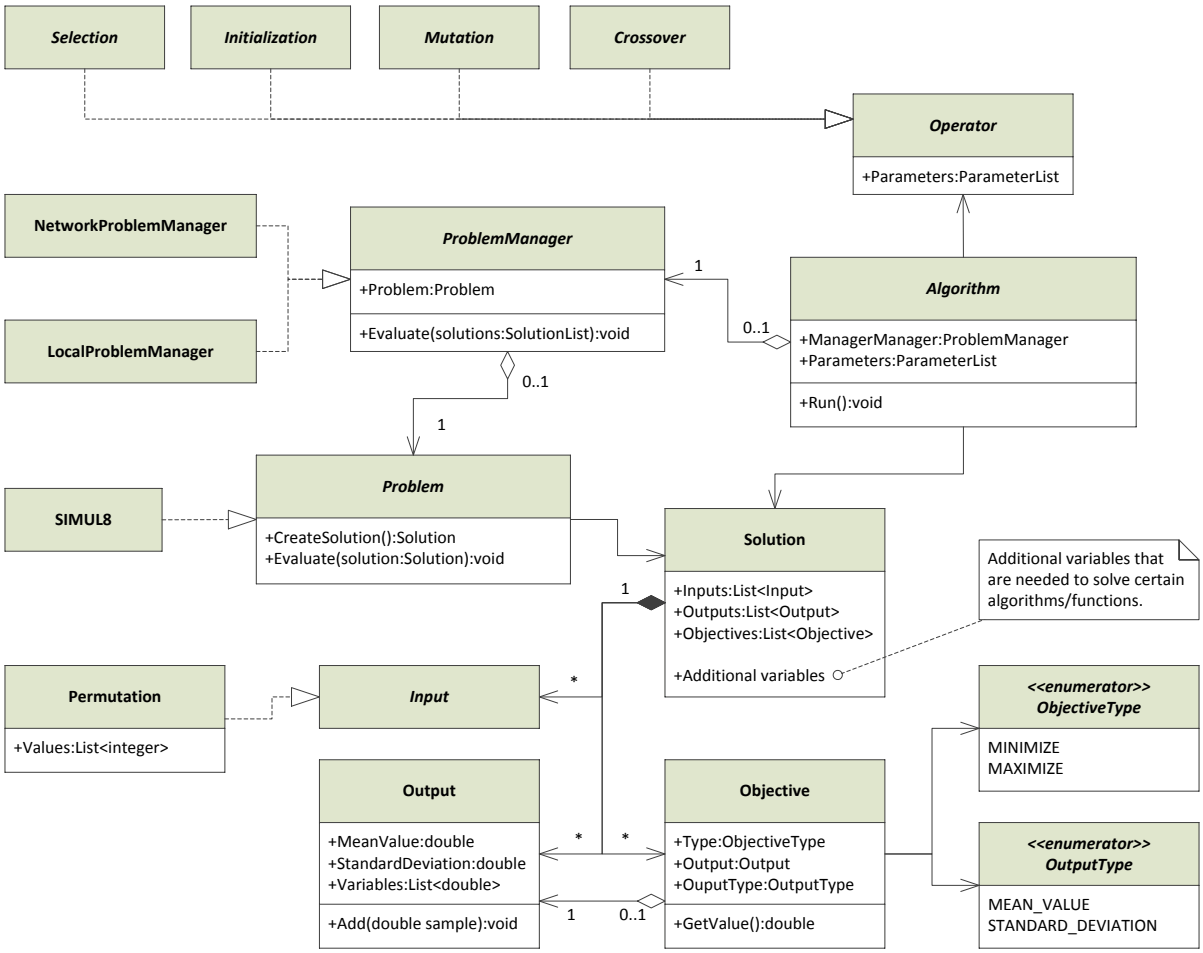


Figure 18 General Optimization Framework.

5.1.2 Software design

The full design of the software including both the experiment and the priority system focuses on the three layers, see figure 19. By separating the presentation layer from the processing and database/knowledge layer it is possible to make the software modular to easily deploy the system for GKN Aerospace. The design considers using the GOF as the core of the processing layer to run both, simulations using SIMUL8 and benchmark problems. The SIMUL8 package imports/exports information from/to a database server, the information in this case will be settings and results.

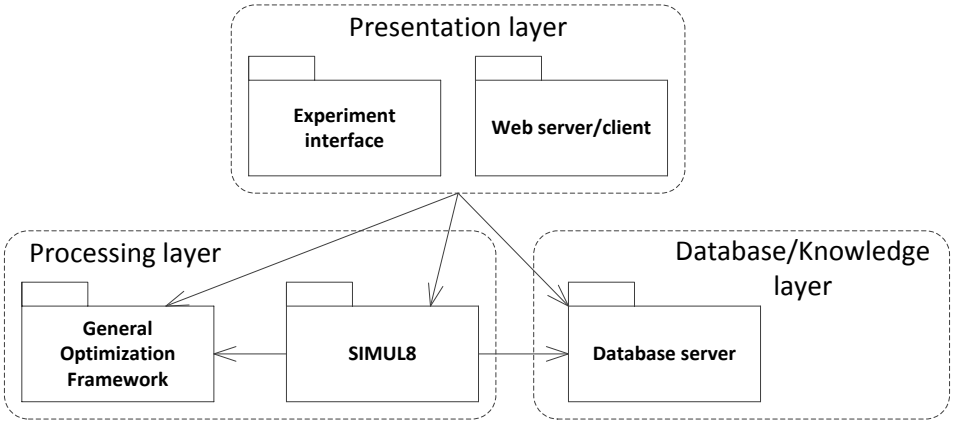


Figure 19 Overview of experimental framework.

5.2 System architecture

The architecture of the web server system follows the basic concept mentioned in chapter 4.3, but has additional functionalities since the architecture needs to consider both the developing and the user system. From the user perspective the web system is used (brown arrows), see figure 20, and from the developer perspective they are connected directly to the optimization system. The web server is used for two purposes, providing the client/user with the priority list and starting the optimization process.

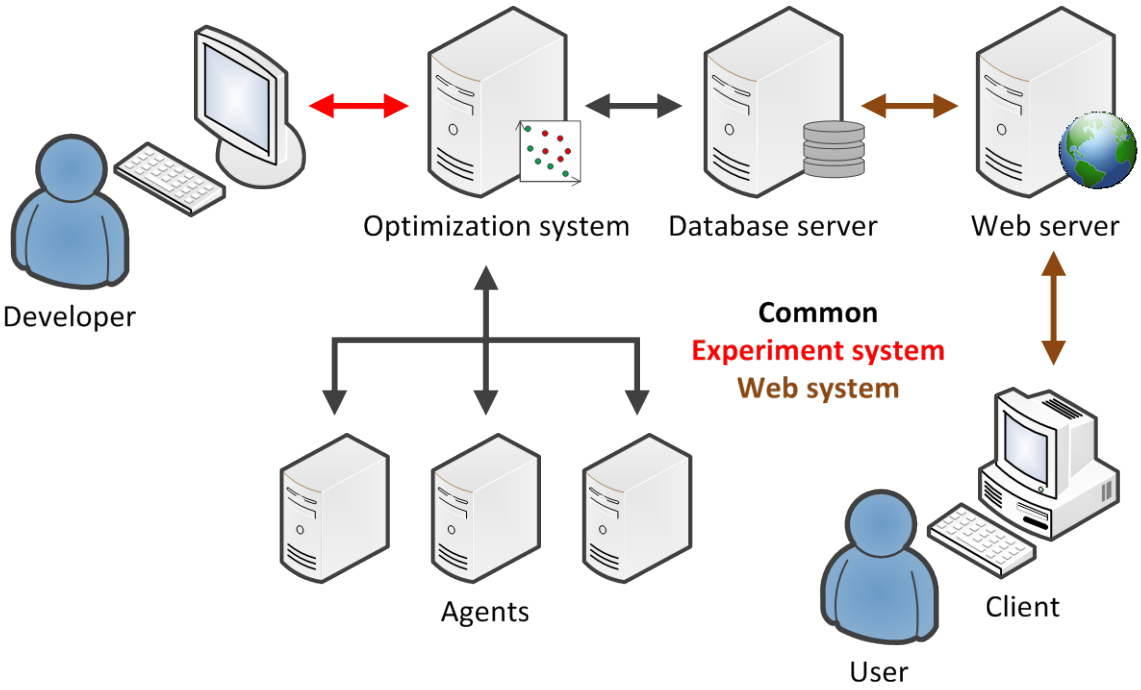


Figure 20 Full overview of the priority system for both developer and user.

Apache was selected as the web server with PHP as server programming language and PostgreSQL was selected as the database server. Getting communication to work directly from PHP to a C# application is difficult and therefore the database server is used as a middleman. The optimization system uses polling to get information whether to start a new optimization and also retrieve the settings from the database server. This way the web server is not dependent on the optimization system and vice versa the optimization system is not dependent on the web server.

Distributed optimization is used when optimizing the SIMUL8 model. Distributed optimization is based on a server distributing the computational expensive execution on several computers, also referred as agents. To implement a distributed optimization system the priority system needs a server/client based communication between the optimization server and the agents. The communication to each agent is based on the agent being the listener and the server sending and receiving information.

5.2.1 Experiment system

The experiment system lies within the optimization system which is used to evaluate/implement different algorithms and settings to maximize the performance of the optimization. In figure 21 the experiment system is shown which has three main tabs, the problem settings, the optimization settings and the results/optimization control. These three tabs fulfills the requirements, explained in chapter 2.7.1, generality (to optimize with

different algorithms against different problems), efficiency (since the purpose of the experiment system to make the algorithm efficient), high-dimensionality (different type of inputs are possible) and the usability (easy to experiment for the developer). For more information see appendix F.

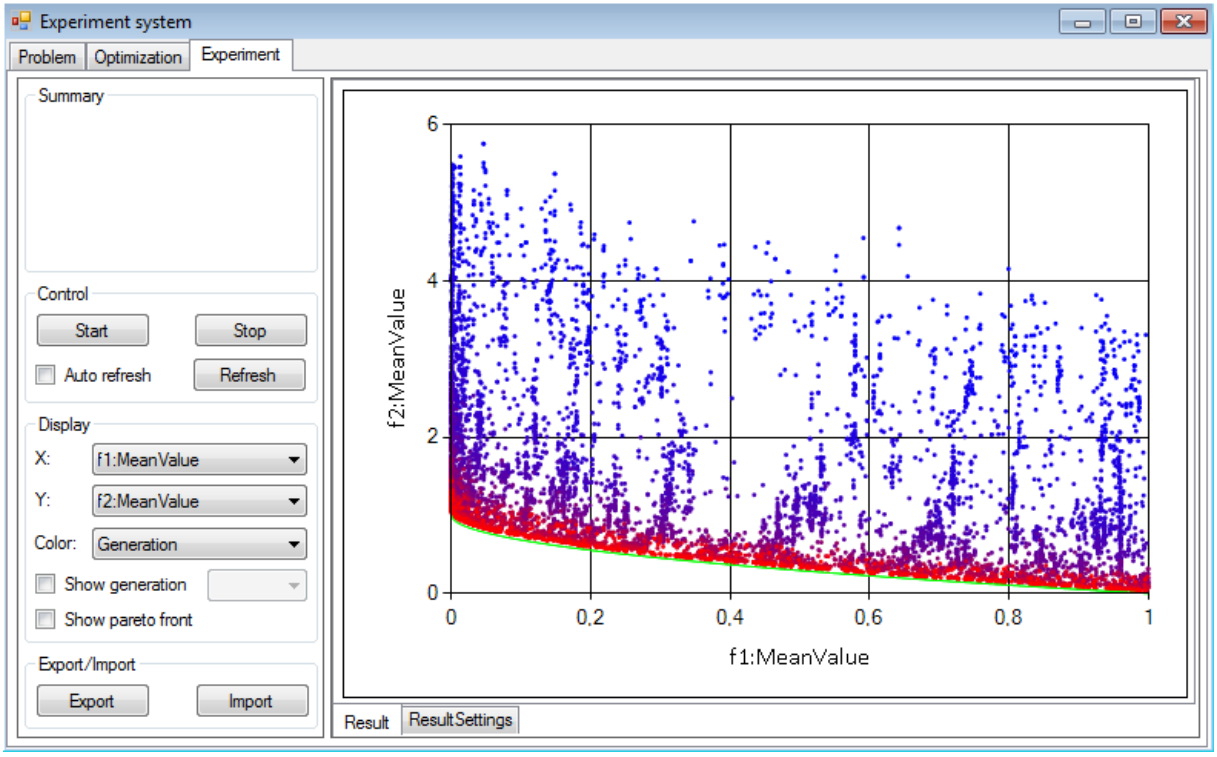


Figure 21 Picture of the experiment system.

A two dimensional graph is used to display the results from the optimization. To extend the information gained from the graph the results also have another dimension with colors, where blue color represents lower value and red color represents higher value. This simplifies the analysis to both be able to compare two different objectives against each other in the x and y axis but also see for example approximately which generation the results belongs to, as in figure 21. All outputs, generation and identification of the results can be selected in each of the x, y and color axis.

5.2.2 Priority system

The user interface is web based with the server language PHP and client language HTML5. GKN Aerospace is located in Sweden and therefore the used language in the web page is Swedish. JavaScript is utilized throughout the user interface since the priority management is dynamic where products are moved from the priority list to each x-ray queue. Figure 22 shows a picture of the web page where the priority list is used. The framed text boxes are for illustrating different areas in the interface i.e. not included in the web page. To the left of the web page is the navigation field which includes four buttons, three buttons to change interface and the bottom button is to log out. The product list is in the left column where the products can be imported into the queue for each x-ray station. There is an input on top of the product list where the products can be filtered in order to find the incoming product. Following is a description of how to work with this user interface:

- When a product arrives to a queue for one of the x-ray stations the worker pushes the corresponding x-ray button. The product is then automatically inserted to the x-ray queue in the correct position based on the priority list.
- When the current product is done the workers will remove this product from both the real x-ray station and the virtual x-ray queue in the interface. Then the next product is selected from the x-ray queue in the interface to be started in the station.

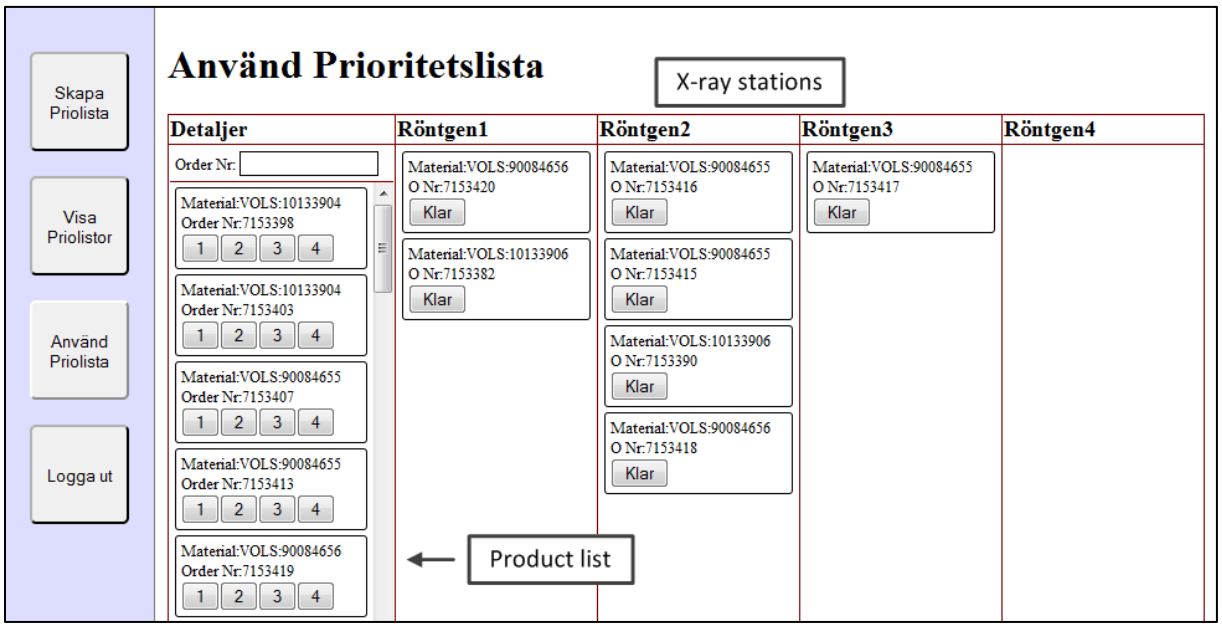


Figure 22 Web interface for use of priority list.

For information about the complete web user interface see appendix G.

5.3 Implementation of SIMUL8 problem

To establish the communication with SIMUL8 the .NET framework is used. The steps required to run a simulation are, open the correct model, setting the environment variables, setting the decision variables, reset the model (reads in the settings), and start the simulation. Open the model and set the environment variables belongs to the initialization phase which is only needed one time for each optimization. Setting the decision variables needs to be done for the evaluation phase. Reset and start of the model needs to be done every replication. The computational time is reduced by separating these phases, initialization, evaluation and replication.

During the implementation of this experiment system it was noted that to start the simulation with the built in function using the .NET library was very slow (approximately 2.5 seconds depending on processor). But running a visual logic with the use of .NET library takes only a couple of milliseconds. Therefore, to start the simulation a visual logic "ExternalStart" is implemented that starts the simulation from within. This speeds up each simulation run from approximately 3.5 seconds to 1 second.

6 Optimization algorithm

To optimize the simulation model an algorithm needs to be implemented. This algorithm needs to perform well on problems which are treated as black box problems i.e. not utilizing the problems characteristics. This is because analyzing the characteristics of the problem is difficult and not included in this project.

Following chapters describes which algorithm was selected and how that algorithm is implemented. First a literature review regarding which evolutionary algorithm that was selected and how that algorithm works. Then a description of which benchmark problems that are implemented both for verification and experimentation on similar case. Last a description of the implemented algorithm and its operators.

6.1 Literature review

Multi Objective Evolutionary Algorithms (MOEAs) are popular methods to solve Multi Objective Optimization Problems (MOOPs) due to their capability to explore large search spaces with less computational time (Carson & Maria, 1997). Over the past decades several MOEAs have been suggested such as Non-dominated Sorting Genetic Algorithm II (NSGA-II), Strength Pareto Evolutionary Algorithm II (SPEA-II), Pareto Envelope based Selection Algorithm II (PESA-II) (Deb, et al., 2008) (Zitzler, et al., 2001) (Corne, et al., 2001).

NSGA-II which is one of the state-of-the-art methods to solve MOOP (Deb, et al., 2008) is widely used in simulation-based optimization. When compared with SPEA-II and PESA-II in benchmark problems it converges to the true Pareto-front equally fast or faster than the two mentioned algorithms (Zitzler, et al., 2001). SPEA-II shows some tendencies to be better when using higher dimensional objective space (Zitzler, et al., 2001), but since in this case a maximum of two dimensional objective space is used NSGA-II were selected in first hand.

Extended methods of NSGA-II exists such as the reference point based NSGA-II which helps the search towards a specific area (Deb & Sundar, 2006). But in this project the aim is to reduce the total delay and most weight is on that goal. However, the additional objective of finding robust solutions might improve the search for the NSGA-II so that it finds better solutions than a single-objective optimization algorithm.

NSGA-II is an elitist genetic algorithm that sorts the solutions based on non-dominated fronts. Illustrated in figure 23 different fronts of a population are visualized.

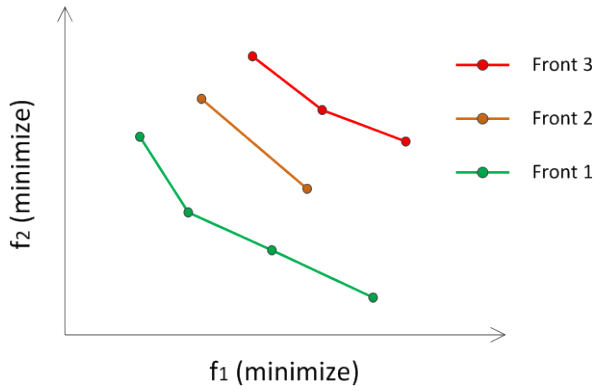


Figure 23 Non-dominated sorting.

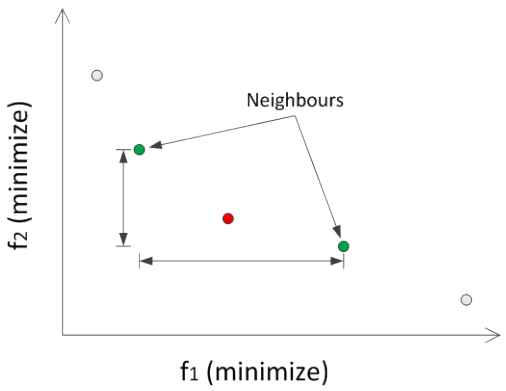


Figure 24 Crowding distance calculation.

It is important to keep the spread of solutions in the population so that the algorithm does not stop in local optima, e.g. like a hill-climbing algorithm. NSGA-II preserves the spread of solutions by calculating the crowding distance of the solutions. In each front the crowding distance is calculated by adding the normalized distance in each objective of the two closest neighbors, illustrated in figure 24. The crowding distance is assigned using the algorithm defined by (Deb, et al., 2008).

$Crowding\ distance\ assignment(F)$	Assignment to front F
$l \coloneqq Size(F);$	Size of front
$Zero\ distance(F);$	Assign crowding distance 0 to front F
$for\ each\ objective\ m$	Loop through all objectives
$F \coloneqq sort(F, m);$	Sort the front based on objective m
$F(1)_{distance} \coloneqq F(l)_{distance} \coloneqq \infty;$	The edges of the front is indefinite
$for\ i \coloneqq 2\ to\ l-1$	For all the other calculate distance
$F(i)_{distance} \coloneqq F(i)_{distance} + (F(i+1)_m - F(i-1)_m)/\left(f_m^{max} - f_m^{min}\right)$	

The main loop of NSGA-II, see appendix A for pseudo code, starts by initializing a population $P(0)$, this can be done by randomizing a set of solutions. The population is then sorted based on non-domination and each solution is assigned a rank based on which front it belongs to, where lower rank is better. Then a tournament will take place in order to select the parent population $P'(0)$, since elitism is used in this algorithm lower rank wins the tournament. From the parent population a crossover and mutation will produce the offspring population $Q(0)$.

After the initiating step the procedure will change according to figure 25. First the new offspring population $Q(t)$ will be evaluated, then the new population $P(t)$ and the child population $Q(t)$ will be added to $R(t)$ for sorting based on non-domination. Since the population now is $2N$, where N is the initiate population size, a selection needs to take place in order to reduce the population to size N again. This is done by first creating a new empty population $P(t+1)$, then assigning crowding distance to the fronts that is added to $P(t+1)$, finally adding solutions to the population, based on better fronts and larger crowding distance, until it reaches population size N.

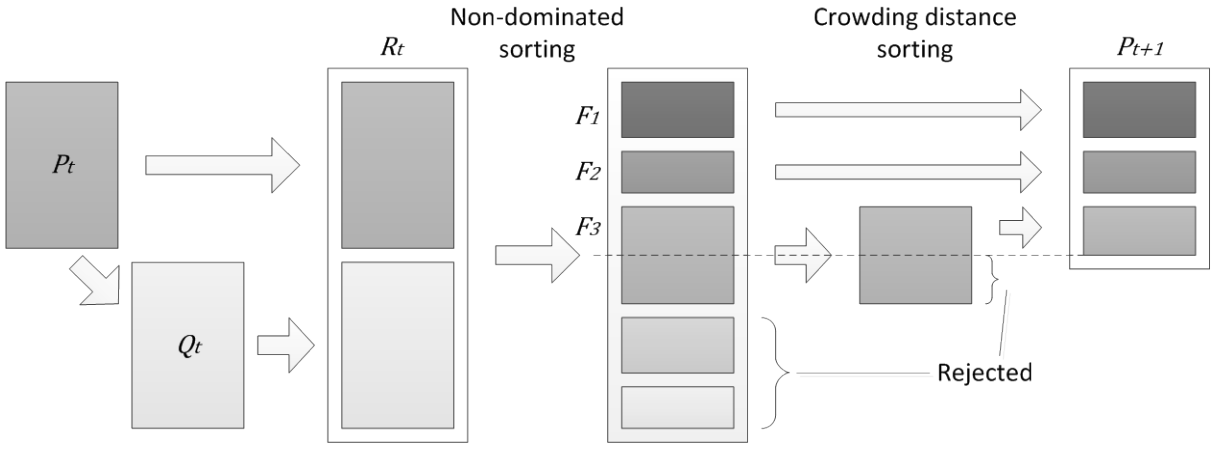


Figure 25 NSGA-II process.

Last step is to produce the offspring by generating the parent population $P'(t+1)$ by tournament where first the lower rank wins and second the larger crowding distance wins. Then produce the offspring population $Q(t+1)$ by crossover and mutation. Finally, the

procedure will start over by incrementing the generation counter t until a termination condition is fulfilled (Deb, et al., 2008).

6.2 Benchmark problems

The ZDT benchmark problems that are suggested in (Zitzler, et al., 2000) are used during this project since they have known Pareto-fronts. Each of these benchmark problems tests the evolutionary optimization algorithms with different approaches. This way the algorithms are tested on what kind of problems they are suited for. In this case the benchmark problems are used as a verification process. The result from running the algorithm on these problems is compared to results from documented experiments to make sure the algorithm is successfully implemented.

Since the benchmark problem uses equations to evaluate each solution a direct approach is used to achieve the results i.e. no need for distributed optimization since that will only require more computational time. The implemented benchmark problems are ZDT1 to ZDT3. For these problems the input values are real values in the range [0, 1] and 30 inputs are used during the experiments as proposed in (Zitzler, et al., 2000). The specifications of the implemented ZDT problems are listed in table 1.

Table 1 Specification of ZDT problems.

Problem	Objective functions	Optimal solutions	Equation type
ZDT1	$f_1(x) = x_1$ $f_2(x) = g(x)\left[1 - \sqrt{x_1/g(x)}\right]$ $g(x) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$x_1 \in [0,1]$ $x_i = 0,$ $i = 2, \dots, n$	Convex
ZDT2	$f_1(x) = x_1$ $f_2(x) = g(x)\left[1 - \left(x_1/g(x)\right)^2\right]$ $g(x) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$x_1 \in [0,1]$ $x_i = 0,$ $i = 2, \dots, n$	Non-convex
ZDT3	$f_1(x) = x_1$ $f_2(x) = g(x)\left[1 - \sqrt{x_1/g(x)} + \frac{x_1}{g(x)}\sin(10\pi x_1)\right]$ $g(x) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$x_1 \in [0,1]$ $x_i = 0,$ $i = 2, \dots, n$	Convex, disconnected

To make a benchmark problem that has the similar characteristics as the SIMUL8 model it needs to have the same input type. Therefore, a Travelling Salesman Problem (TSP) is used as a benchmark problem which has a list of integers which is unique, similar to the priority list. The TSP is a well-known problem which is based on a tour where a number of cities are to be visited once and finish at the first city visited creating a loop (Grefenstette, et al., 1985). If all cities have a unique number in a sequence starting from one then the tour list is identical to a priority list. Since the variation in the SIMUL8 model may be different depending on the priority numbers i.e. products skipping ahead, the TSP should act the same. Each city in this TSP has x, y coordinates and also the constant a according to figure 26. The x, y coordinates is used to calculate the distance between each city and the constant c is used to vary the distance between each run.

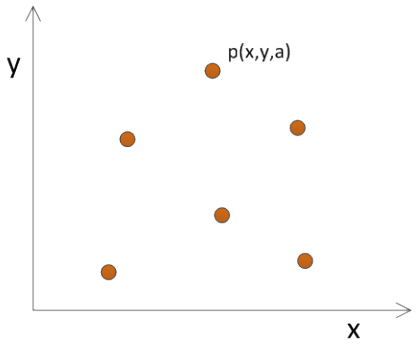


Figure 26 TSP similar to real case.

$$u \in [-0.5,0.5], c \in [0,1]$$
$$dc = c_2 - c_1, dx = x_2 - x_1, dy = x_2 - x_1$$
$$distance = \sqrt{dx^2 + dy^2}$$
$$stoch.distance = distance * (1 + dc * u)$$

Equation 3 Calculation of TSP distances.

The distances between the cities in the tour are calculated according to equation 3 using Pythagoras theorem. To apply the stochastic behavior for each distance in the tour they are multiplied with one plus the c difference multiplied with a random number u. The maximum possible stochastic behavior in this case range within 50% to 150% of the distance. The stochastic behavior will in this case be different depending on the order of which each city is visited. Less difference in the constant c between two cities means less stochastic behavior. With this setup the TSP have the same type of list to optimize and also different stochastic behavior depending on the order of cities visited.

6.3 Implementation of NSGA-II

The selected algorithm to implement is NSGA-II. The algorithm is implemented according to figure 25 using pseudo code described by (Deb, et al., 2008). In this algorithm there are four operators needed, crossover, mutation, selection and initialization. The selection process is the only operator independent of the problem and solution type since the selection is based on rank and crowding distance. But crossover, mutation and initialization needs implementation on both combinatorial input problems and real input problems.

6.3.1 Operators for real input problems

The initialization for the real input is a simple random real number in range [0, 1] for each value. In this case only random initialization is needed because the real encoded inputs are only used for benchmark problems.

For NSGA-II (Deb, et al., 2008) proposed using simulated binary crossover and polynomial mutation for real encoded values. Simulated binary crossover uses the convention of the single point binary crossover. But since the values are real encoded the binary crossover needs to be simulated in order to achieve the same characteristics. Illustrated in figure 27 is an example of the single point binary crossover, the values from the parents have a mid-value of (54+93)/2=73.5 and the values from the offspring also have a mid-value of (61+86)/2=73.5. The behavior of the crossover is that the mid-value of the parents respectively offspring stays the same but the offspring values gets either contracted or expanded (Deb & Agrawal, 1995).

Figure 27 Single point binary crossover.

$$u \in [0,1]$$

$$\bar{\delta} = \begin{cases} (2u)^{\frac{1}{n+1}} - 1, & if\ u < 0.5 \\ 1 - [2(1-u)]^{\frac{1}{n+1}}, & if\ u \geq 0.5 \end{cases}$$

$$c = p + \bar{\delta}\Delta_{max}$$

Equation 4 Polynomial mutation.

For the real encoded values (Deb, et al., 2008) proposed using the polynomial mutation for the NSGA-II algorithm. This mutation utilizes the polynomial distributed probability to mutate the solution using equation 4. Here the variable u is a random value [0, 1] and n is the distribution index to control the distribution curve. The variable c is the mutated value, p is the original value, Δ_{max} is the maximum possible change and $\bar{\delta}$ the polynomial distributed value [-1, 1] based on the random value u (Deb & Goyal, 1996).

6.3.2 Operators for combinatorial input problems

The inputs used for the real case benchmark problem and the SIMUL8 model are permutation list encoded, where the values are numbers in a sequence $1 \rightarrow n$ where n is the size of the permutation list. The initialization operator generates the sequence of numbers in the permutation list and randomly shuffles these values to gain a random order of the values.

An order crossover was used for both the TSP benchmark problem and SIMUL8 model illustrated in figure 28. The crossover starts by copying a range of the string from P1 to O1, then the rest of the numbers will be copied from P2 to O1 in the order that they appear and only numbers that have not already been copied. The same process applies to O2 but with P2 as the first parent and P1 as the second. This crossover treats the problem as a black box.

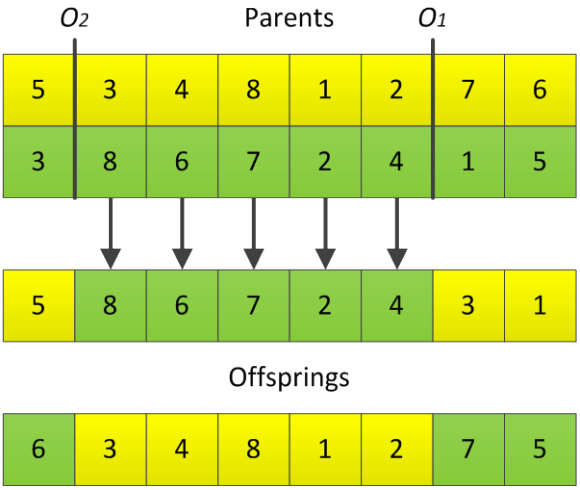


Figure 28 Order crossover for combinatorial inputs.

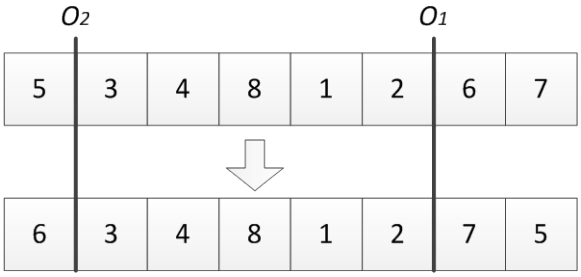


Figure 29 Mutation for permutation encoded inputs.

The mutation randomly swaps numbers in a range shown in figure 29. The algorithm starts by randomly generating two offsets; it does not matter in which order the offsets ends up since the edges can be included in the swap, see figure 29. Then every input in the range between the first and the second offset will be swapped randomly to mutate the solution.

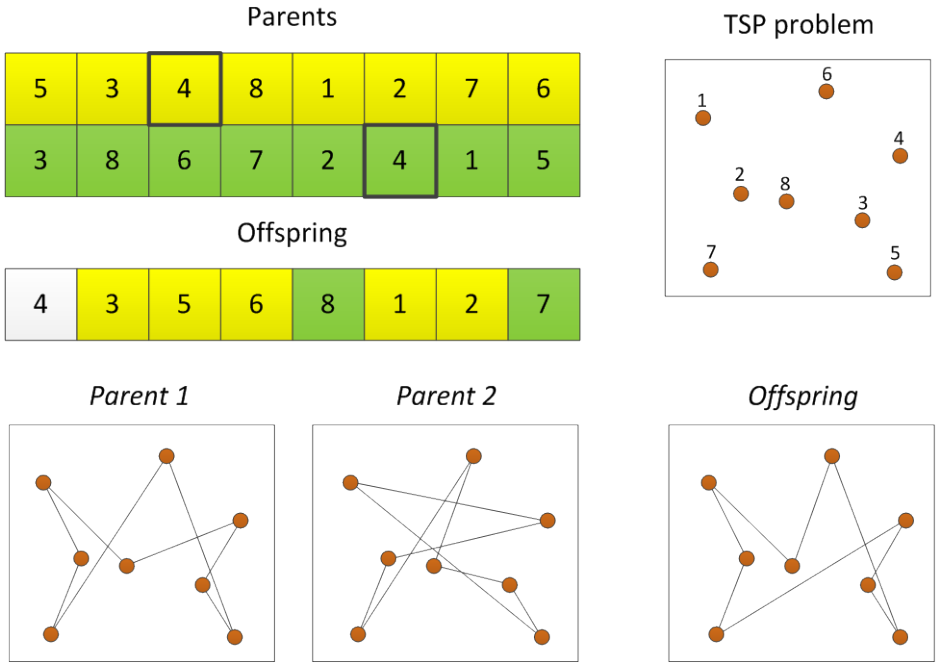


Figure 30 Greedy crossover, constructs the offspring based on the TSP problem.

The greedy crossover is implemented to analyze the difference of treating the problem as a black box versus white box, see figure 30. This crossover utilizes the knowledge about input/output correlation for the TSP problem. The greedy crossover selects a random city from the TSP problem as the first city in the route. Then the loop starts by selecting the closest neighbor from its parents. In the example the neighbors for city 4 are city 3, 8, 2 and 1, where city 3 is the closest. Then the loop continues and selects the closest neighbor for the new city from its parents which has not yet been visited (Grefenstette, et al., 1985).

7 Experiments and results

The experiments are made in three steps as described in the methodology, chapter 3.4. The first benchmarking experiments are used to verify the implemented NSGA-II algorithm. The second benchmarking is used to test whether the algorithm can produce good results on problems similar to the real case. The last experiments are the real case optimization.

7.1 ZDT benchmarking

To verify NSGA-II and see whether it finds near optimal solutions the algorithm is used on ZDT benchmark tests. ZDT1 to ZDT3 are used in these experiments which have 30 real inputs [0, 1] and tests the algorithm in different kind of problems, more explanation in chapter 6.2. All experiments are run with the same settings according to table 2. The results are presented using a two-dimensional graph with a color dimension representing a third value. These benchmark problems have two objectives to minimize both f1 and f2, these are presented on the x and y axis. Which generation the result belongs to is presented with colors, blue meaning early generations and red meaning later generations.

Table 2 NSGA-II settings for ZDT benchmark problems.

Population size	100
Max evaluations	10000
Initialization	Random
Selection for mating pool	Binary tournament
Crossover	90% Simulated binary crossover
Mutation	10% Polynomial mutation

The results from all the experiments show that the algorithm finds better solutions on later generations. The green line represents the true Pareto optimal front, calculated according to the specifications of ZDT problems, see table 1. The results show that the algorithm converges towards the true Pareto front by finding close values and also the spread of values on the implemented ZDT problems.

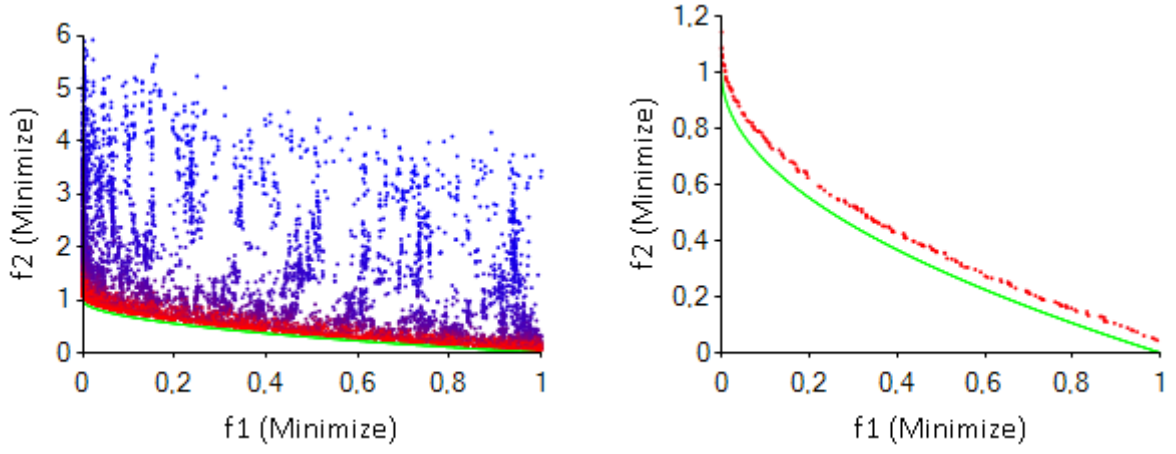


Figure 31 Results for NSGA-II on ZDT1 problem.

The results are similar with all the implemented ZDT problems and figure 31 shows the results from running the NSGA-II algorithm on ZDT1. The colors represent which generation (early generations are blue and late generations are red) the specific solution belongs to. There is a clear tendency that the algorithm quickly searches towards the true Pareto optimal front, see left graph, where the concentration of red solutions are gathered near the Pareto optimal line. The right graph shows the Pareto-front generated from this experiment which indicates that the algorithm not only produces solutions near the true Pareto-front but also with solutions with the spread throughout the Pareto-optimal line. For all results see appendix B.

7.2 TSP benchmarking

The TSP benchmark problem is used as the case adjusted benchmark problem. This benchmark represents the real case problem and therefore the implemented TSP contains stochastic behavior. Since this generates two outputs (mean value and standard deviation) there can be two objectives, finding low tardiness and robust solutions. A question was formed in chapter 1.1, that better solutions can be found when using standard deviation as an additional objective, by utilizing the traits of NSGA-II as a multi-objective optimization method. To evaluate this question experiments with both objectives and with only one objective have to be executed.

The real case optimization treats the problem as a black box since it is difficult to gain knowledge about the characteristics of the simulation model. Therefore, it is important to evaluate the impact of treating the benchmark problem as a black box versus a white box. This knowledge may be important for the analysis of the real case optimization depending on the results.

The TSP settings are 100 randomly placed cities with randomly selected c constants, see equation 3. The randomly placed cities has better representation since a circle, triangle or square have short distances between each city which leads to one solution that is far better than the others. In total there are four necessary experiments, using the NSGA-II algorithm, by combining two objectives versus one objective with treating the benchmark problem as a white box versus a black box. The settings used during the experimentation are listed in table 3.

Table 3 NSGA-II settings for TSP benchmark problem.

Population size	100
Max evaluations	10000
Replications	10
Initialization	Random
Selection for mating pool	Binary tournament
Crossover	Black box -> 90% Order crossover White box -> 90% Greedy crossover
Mutation	10% Shuffle mutation

The results are presented by displaying the results in a graph where the x-axis show the mean value of the distance, the y-axis show the standard deviation of the distance and the color axis show the generation (blue early generations and red late generations). The results from the first two experiments, treating the TSP as a black box, are displayed in figure 32. The results show that the search performance improves when disregarding the additional objective standard deviation for this problem. The left graph, with standard deviation, shows that the algorithm finds better solutions but also retains bad solutions throughout the search. The right graph, without standard deviation, shows that the algorithm finds better solutions and disregards the bad solutions to continue the optimization.

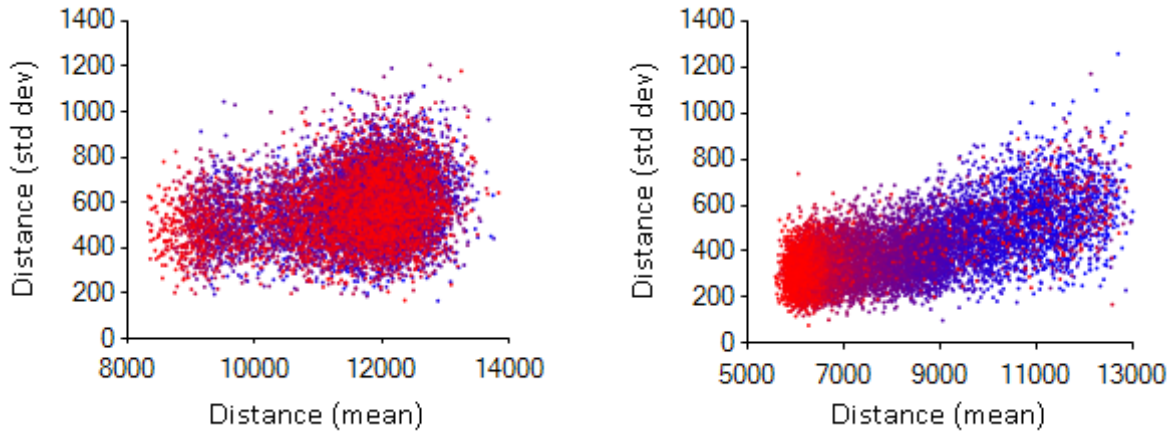


Figure 32 NSGA-II on TSP as black box, with std. dev. (left), without std. dev. (right).

Figure 33 shows the results from treating the TSP as a white box, when utilizing the problem characteristics. The difference between using standard deviation and not using it is similar in this case as well. The results show that without the additional objective the search performance is improved and disregards bad solutions in the search. With standard deviation the search is conservative.

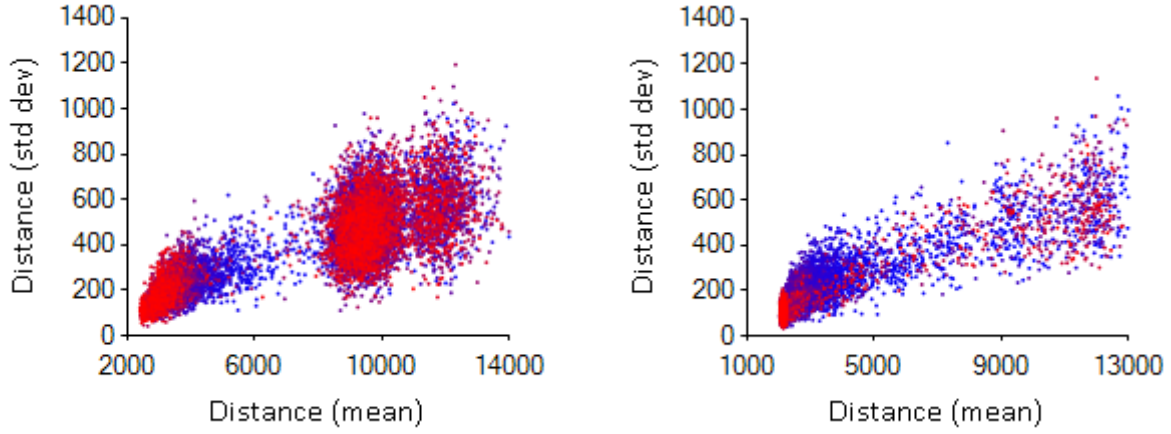


Figure 33 NSGA-II on TSP as white box, with std. dev. (left), without std. dev. (right).

The difference from treating the problem as a black box versus a white box is large. Comparing figure 32 with figure 33, it shows a significant improvement in both mean value and standard deviation where the advantage lies within treating it as a white box.

The city tours are illustrated in figure 34 to show the impact the choice of method has. This is illustrated by visualizing the TSP problem with the best generated city tour based on

lowest mean value. The thickness of the lines show the stochastic behavior of the result, more thick lines means more stochastic behavior. Here the results are clear that using a white box crossover is significantly better. This, however, is obvious since optimizing a problem when knowing the correlation between inputs and the output helps the process of constructing new children. In figure 35 the left graph is from treating the TSP as a black box, the generated route is not very optimized. The right graph on the other hand is quite optimized, but the human eye can still spot some changes which would improve the route.

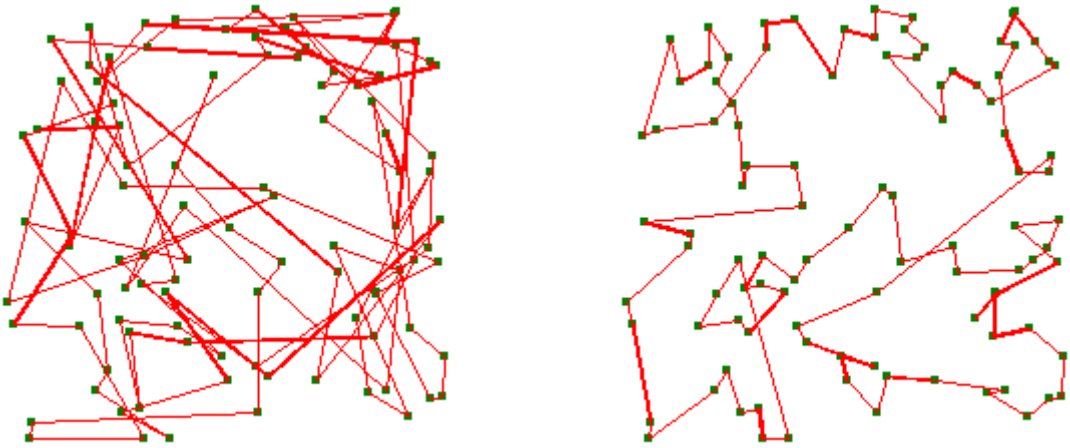


Figure 34 NSGA-II on TSP treated as black box (left) and treated as white box (right).

For more detailed results see appendix C.

Since the NSGA-II algorithm is a multi-objective evolutionary algorithm it may not be suited for single-objective problems which is the case without the standard deviation. Therefore, the NSGA-II algorithm is compared against a hill-climbing algorithm as well to analyze the performance when using only one objective. The TSP problem is treated as a black box in this case. Since no standard deviation is used the experiments are made on a deterministic TSP problem. The settings used for NSGA-II are the same as previous but with one replication. The settings for the hill-climbing algorithm are specified in table 4.

Table 4 Hill-climbing settings for TSP benchmark problem.

Max evaluations	10000
Mutation	100% Shuffle mutation

The results are presented by using a graph where the x-axis show the id of the solutions, the y-axis show the distance and the color axis show the generation (blue early generations and red late generations).

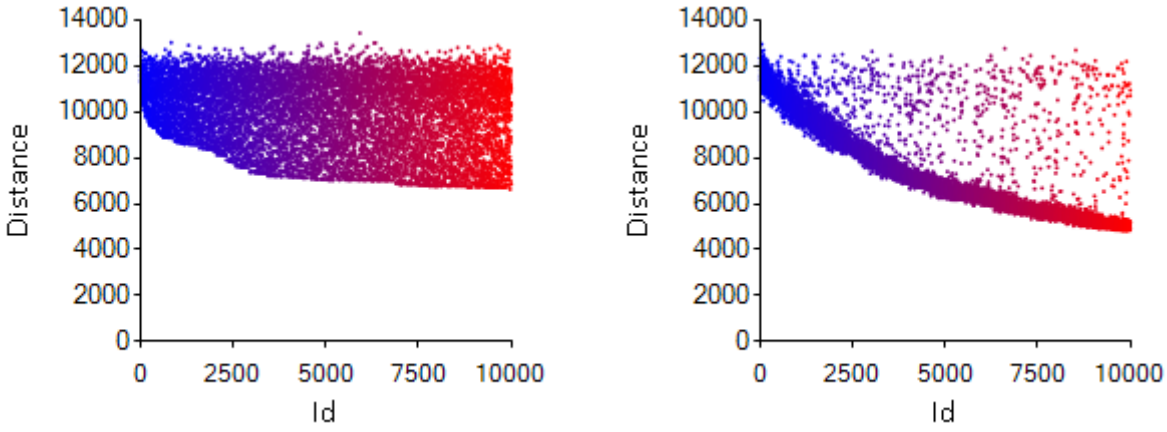


Figure 35 Hill-climbing (left) versus NSGA-II (right) on deterministic single-objective TSP.

Figure 35 displays the results from comparing hill-climbing versus NSGA-II on a single-objective TSP. The results show that the NSGA-II finds better solutions in the long run but in the first 1000 solutions the hill-climbing method has a small advantage. For more detailed results see appendix D.

7.3 Real case experiments

The real case experiments involve optimization on the SIMUL8 model. The environment inputs, explained in chapter 4.2, are the settings which represents a scenario. Experts at GKN Aerospace have created a typical scenario for this experiment. There are 59 products that are included in this scenario which makes the complexity of this problem $(59!)^4$. The results from the TSP benchmarking showed that the best algorithm and settings to use was NSGA-II without standard deviation. This setup was used for the first experiment and the settings are according to table 5.

Table 5 NSGA-II settings for real case.

Population size	100
Max evaluations	10000
Replications	10
Initialization	Random
Selection for mating pool	Binary tournament
Crossover	90% Order crossover
Mutation	10% Shuffle mutation

The result is displayed in figure 36 where the x-axis show the mean value of the total delay, the y-axis show the standard deviation of the total delay and the color axis show the generation (red being later generations). The result shows no trends that the algorithm improves the solutions in the long run, which means that the optimization is not working. Therefore, additional experiments were made with other algorithms and settings to see if they could make any improvements. This problem was run with, NSGA-II using standard deviation as an additional objective, hill-climbing algorithm and a random search algorithm,

see figure 37. All these experiments were run with 10000 evaluations and they gave approximately the same results as figure 36, for all results see appendix E. This means that in this case the algorithm does not perform better than a random search algorithm.

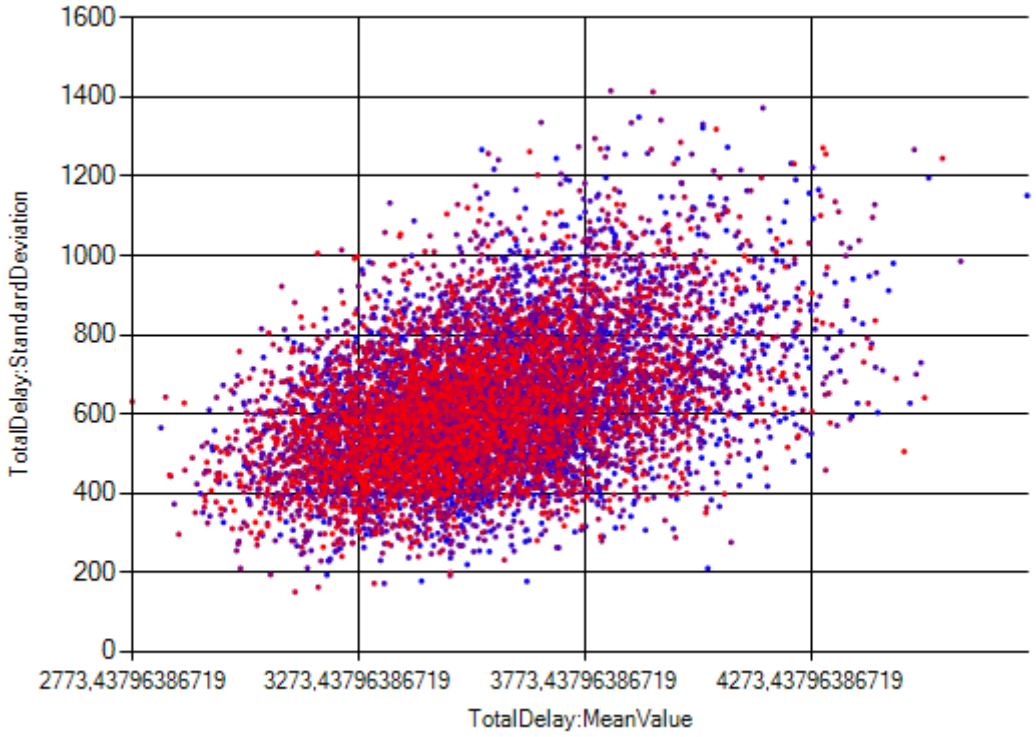


Figure 36 Results from NSGA-II without std. dev. on SIMUL8 problem.

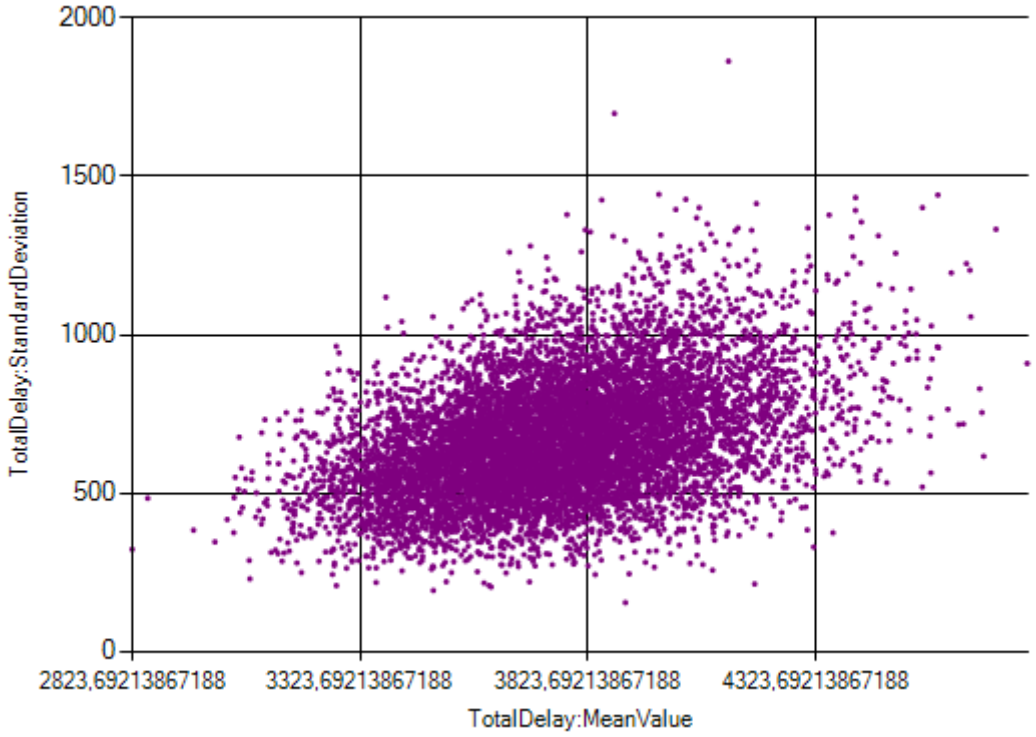


Figure 37 Results from random search on SIMUL8 problem.

Since the algorithms did not optimize the problem, the SIMUL8 model was changed into a deterministic model to verify that the previous model was not too random in its behavior. This experiment uses NSGA-II without standard deviation as an additional objective. The settings for the algorithm follow the specification according to table 5 but with only one replication, since the model is deterministic.

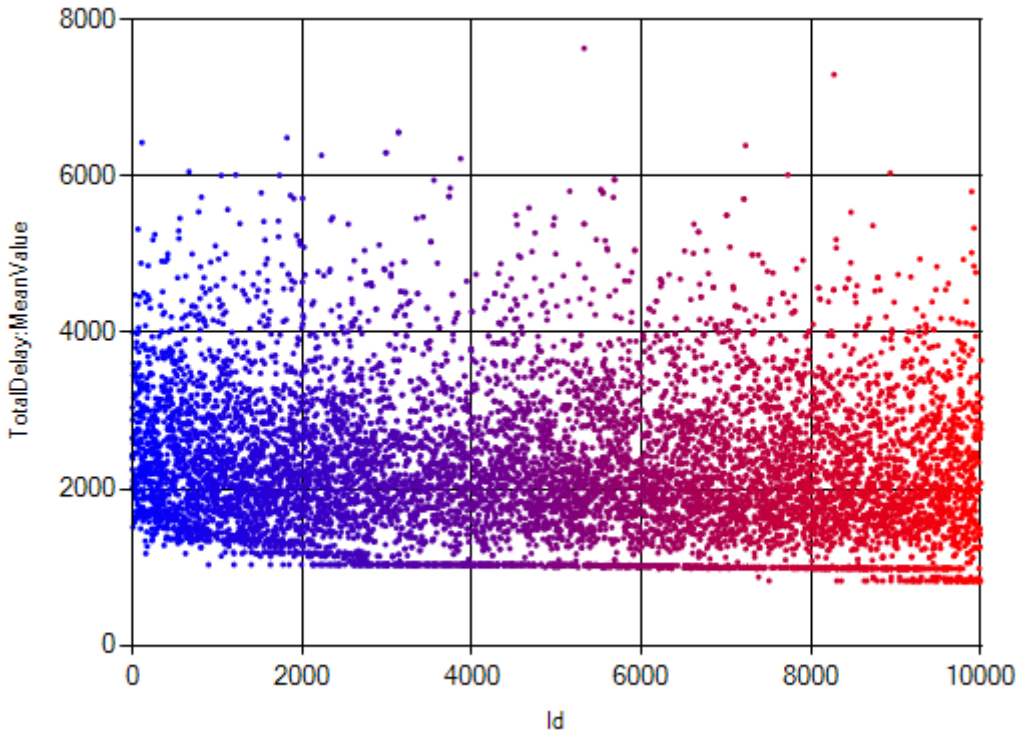


Figure 38 Results from NSGA-II without std. dev. on deterministic SIMUL8 problem.

Figure 38 shows the results from this experiment where the total delay is shown in the x-axis and the identification of each solution on the y-axis. This graph shows that the algorithm now optimizes the problem but the search performance is slow. Another experiment was also made to see whether the algorithm would improve the search performance by decreasing the settings to only one x-ray station. This means that the complexity is reduced from $(59!)^4$ to $59!$. However, the results were similar to figure 38, for all results see appendix E.

8 Analysis and discussion

Benchmarking was used to verify that the NSGA-II algorithm was correctly implemented. The selected benchmark problems were ZDT1, ZDT2 and ZDT3. The graphs from these experiments, figure 31, showed that the algorithm produced solutions that converged towards the true Pareto-optimal front. This concludes together with documented cases, such as (Zitzler, et al., 2000), that the algorithm is correctly implemented.

The experiments with the TSP benchmark problem were executed in order to find settings for the algorithm to facilitate the experiments made on the real case. The TSP benchmarking was also used to analyze the impact of treating the problem as black box versus white box. When treating the problem as a black box the NSGA-II algorithm found it more difficult to optimize the problem compared to treating it as a white box, see figure 34. This, however, is obvious since the crossover can make use of the correlation between inputs and outputs to recombine the parents into offspring. But utilizing the possibility of treating the real case problem as a white box would certainly be a significant improvement.

The experiments on the NSGA-II algorithm were made using both with standard deviation and without standard deviation. The question if using standard deviation as an additional objective improves the search performance showed negative response. In all the experiments it was better to exclude standard deviation as objective since using it made the algorithm more conservative, see figure 32 and figure 33. The conclusion is that for this type of problem the answer is that using standard deviation as an additional objective slows down the search performance.

Since the answer to the question was negative, the NSGA-II algorithm is used for single-objective problems. NSGA-II is a multi-objective evolutionary algorithm and using it for a single-objective problem would be considered wrong. However, to analyze if this still can be used the NSGA-II algorithm was compared to the hill-climbing algorithm on a deterministic TSP problem, see figure 35. The results from that experiment showed that the NSGA-II algorithm has better search performance than hill-climbing in the long run and therefore the NSGA-II algorithm was the main algorithm used.

The results from the real case experiments gave no indication that the NSGA-II algorithm optimized the problem, see figure 36. This was compared with random search and hill-climbing but with no difference. However, when changing the problem into a deterministic model the results showed that the NSGA-II optimized the result but the search performance remained low. Why the problem in this case is difficult to optimize can be of several reasons:

- The real case optimizations are run with priority size of 59 and since there are four lists this problem has a complexity of $(59!)^4$ which is a very large problem. This problem might be too difficult to solve within a reasonable time limit.
- The recombination might be insufficient for the real case. The good properties from the parents might not be inherited with the order crossover. Therefore, finding a more suitable crossover is important. If the problem could be treated as a white box the offspring could be recombined using the knowledge of input/output correlation.
- The SIMUL8 model may be too random in its behavior which makes the optimization more difficult. This is proven to a certain extent since the NSGA-II algorithm can optimize the deterministic model.

The formed question in chapter 1.1, if adding standard deviation as an objective would improve the search performance, was proven false in this case. However, there might be other cases where the answer may be different. Imagine a two-dimensional graph which axes are equidistant then the search space would be approximately according to figure 39. This is because the standard deviation should not (if disregarding skewness) be larger than its mean value since the value cannot go below zero. The figure shows that when lowering the mean value the standard deviation is also lowered. This means that the two objectives are not conflicting which is desirable in multi-objective evolutionary algorithms (Deb, 2001). So in an optimization problem where the objective is to minimize a value, then the additional objective standard deviation does not improve the search performance. If, however, the objective were to maximize a value, such as improve the throughput per hour in a manufacturing process, then the additional objective to minimize the standard deviation might improve the search performance since then the objectives would be conflicting.

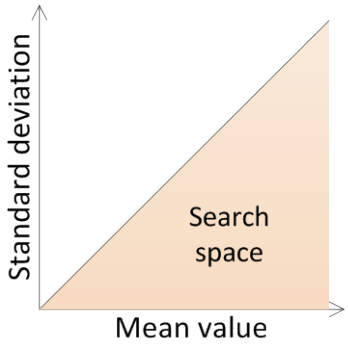


Figure 39 Search space on mean value and std. dev.

During the development of the algorithms the implementation of each algorithm/operator into the experiment system was simple. The GOF designated each algorithm/operator with a specific task which made it easy to implement new classes. Therefore, this system can be of use for future work when trying new algorithms to improve the search performance.

9 Conclusion and future work

A priority system has been developed during the course of this project in order to improve the selection process to the x-ray stations in the work shop. This priority system uses simulation-based optimization in order to find near-optimal priority lists. The architecture of the priority system considers two different kinds of users, the workers and the developers. The workers use this system in order to simplify the selection of the next product to the x-ray station while the developer can improve the algorithm. This priority system uses a web server which separates the optimization system from the user interface. This way the University of Skövde can continue the development of algorithms while GKN Aerospace uses the priority list.

An experiment system was created to facilitate the implementation of algorithms and to easier make settings when running experiments. Most of the time was spent creating this experiment system but once it was created the algorithms and operators were easy to implement. This experiment system made it possible to easily make settings for the algorithm and to analyze the results from the algorithm. The architecture of the experiment system was made up by three layers, the presentation, the processing and the knowledge/database layer. By separating these three layers the priority user interface was implemented without any need for rework.

The implemented algorithm was the state-of-the-art method NSGA-II which is a multi-objective evolutionary algorithm. The implemented NSGA-II algorithm displayed good performance on benchmark problems. Even when only one objective was used the NSGA-II algorithm preceded the single objective optimization algorithm, hill-climbing. However, the NSGA-II algorithm did not manage to optimize the real case problem. Possible reasons for this could be:

- The real case optimization has a very difficult complexity of $(59!)^4$ possible solutions which might be nearly impossible to solve.
- The recombination might be insufficient for the real case.
- The SIMUL8 model may be too random in its behavior. This was proven to a certain extent where the algorithm managed to optimize a deterministic model.

The complexity of the simulation problem might be too difficult to solve when treating it as a black box. Therefore, future work should focus on either reduce the complexity of the problem or to try and develop an algorithm that utilizes the characteristics of the problem. However, the SIMUL8 model was proven to a certain extent that the model was too random in its behavior which means that future work could also try and make the simulation model less stochastic.

A question was formed during this project that the search performance would improve by adding an additional objective of minimizing the standard deviation, for this otherwise single-objective optimization problem. The question was proven false for this type of problem when the objective is to minimize both the mean value and standard deviation. But this question is not answered when the problem is to maximize the mean value. Therefore, to investigate this question further, future work should focus on experimenting on problems where the mean value is to be maximized and the standard deviation is to be minimized.

List of references

Banks, J., Carson II, J. S., Nelson, B. L. & Nicol, D. M., 2010. *Discrete - Event System Simulation..* 5th red. New Jersey: Pearson education, inc..

Bleuler, S., Laumanns, M., Thiele, L. & Zitzler, E., 2002. *PISA - A Platform and Programming Language Independent Interface for Search Algorithms,* Zürich: Institut für Technische Informatik und Kommunikationsnetze.

Burstein, F. & Holsapple, C. W., 2008. *Handbook on Decision Support Systems 1.* Berlin: Springer.

Carson, Y. & Maria, A., 1997. *SIMULATION OPTIMIZATION: METHODS AND APPLICATIONS.* New York, Winter Simulation Conference, pp. 118-126.

Corne, D. W., Jerram, N. R., Knowles, J. D. & Oates, M. J., 2001. *PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization.* San Francisco, s.n.

Deb, K., 2001. *Multi-Objective Optimization using Evolutionary Algorithms.* Chichester: John Wiley & Sons.

Deb, K. & Agrawal, R. B., 1995. Simulated Binary Crossover for Continous Search Space. *Complex Systems,* Volume 9, pp. 115-148.

Deb, K. & Goyal, M., 1996. A Combined Genetic Adaptive Search (GeneAS) for Engineering Design. *Computer Science and Informatics,* 26(4), pp. 30-45.

Deb, K. & Gupta, H., 2006. Introducing Robustness in Multi-Objective Optimization. *Evolutionary Computation,* 14(4), pp. 463-494.

Deb, K., Pratap, A., Agarwal, S. & Meyarivan, T., 2008. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation,* 6(2), pp. 182-197.

Deb, K. & Sundar, J., 2006. *Reference Point Based Multi-Objective Optimization Using Evolutionary Algorithms.* New York, ACM.

Durillo, J. J. et al., 2006. *jMetal: a Java Framework forDeveloping Multi-Objective Optimization Metaheuristics,* University of Málaga: Departamento de Lenguajes y Ciencias de, la Computacion.

Fu, M. C., 2002. Optimization for Simulation: Theory vs. Practice. *INFORMS Journal on COMPUTING,* 14(3), pp. 192-215.

Grefenstette, J., Gopal, R., Rosmatia, B. & Gucht, D. V., 1985. *Genetic Algorithms for the Traveling Salesman Problem.* New Jersey, Lawrence Erlbaum, pp. 160-168.

Halim, R. A. & Seck, M. D., 2011. *THE SIMULATION-BASED MULTI-OBJECTIVE EVOLUTIONARY OPTIMIZATION (SIMEON) FRAMEWORK.* s.l., Winter Simulation Conference, pp. 2839-2851.

Hejlsberg, A., Torgersen, M., Wiltamuth, S. & Golde, P., 2010. *The C# Programming Language.* 4 ed. s.l.:Microsoft Corporation.

Law, A., 2007. *Simulation Modeling & Analysis.* 4 red. New York: McGraw-Hill.

Law, A. M. & McComas, M. G., 2000. *SIMULATION-BASED OPTIMIZATION.* s.l., Winter Simulation Conference, pp. 46-49.

Nunamaker, J. F., Chen, M. & Purdin, T. D., 1991. Systems Development in Information Systems Research. *Journal of Management Information Systems,* 7(3), pp. 89-106.

Pinedo, M. L., 2012. *Scheduling: Theory, Algorithms, and Systems.* 4 ed. New York: Springer.

Rumbaugh, J., Jacobson, I. & booch, G., 1999. *The Unified Modeling Language Reference Manual.* s.l.:Addison Wesley Longman, Inc.

Russell, S. J. & Norvig, P., 2010. *Artificial Intelligence A Modern Approach.* third ed. New Jersey: Pearson Education Inc..

SIMUL8, 2013. *SIMUL8.* [Online] Available at: www.simul8.com [Accessed 11 04 2013].

Zitzler, E., Deb, K. & Thiele, L., 2000. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation,* 8(2), pp. 173-195.

Zitzler, E., Laumanns, M. & Bleuler, S., 2003. *A Toturial on Evolutionary Multiobjective Optimization,* Zurich: Swiss Federal Institute of Technology.

Zitzler, E., Laumanns, M. & Thiele, L., 2001. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm,* Zurich: Swiss Federal Institute of Technology.

Appendix

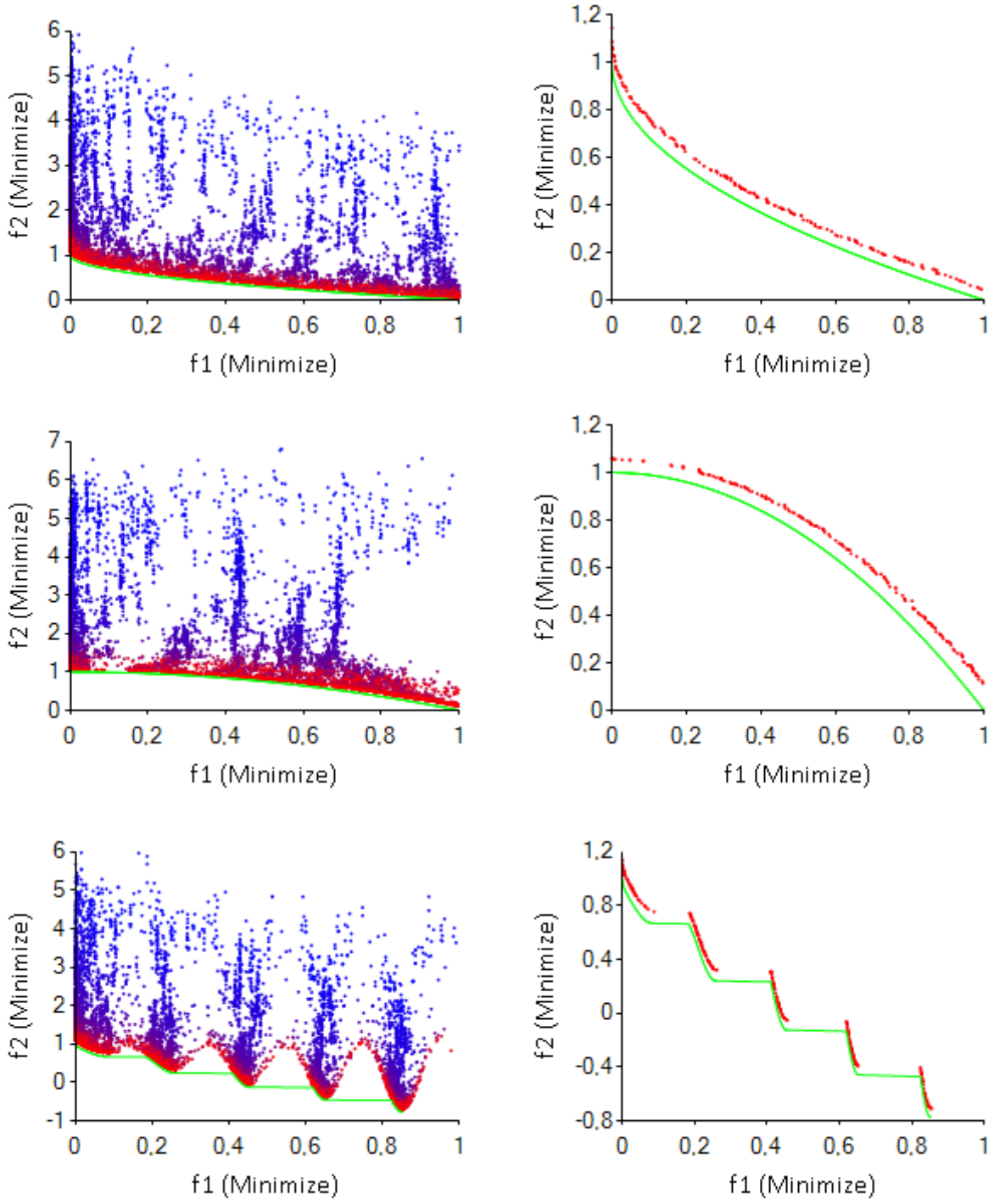
A. Pseudo code NSGA-II 49
B. Results from optimization on ZDT1 to ZDT3 with NSGA-II algorithm 50
C. Results from optimization on TSP problem with NSGA-II 51
D. Comparison of hill-climbing and NSGA-II on single objective TSP 55
E. All SIMUL8 results 56
F. Experiment program 59
G. Priority system web interface 62
H. Complete UML model of GOF 65
I. Sample codes 66

A. Pseudo code NSGA-II

$begin$
$t \coloneqq 0;$ Generation counter
$Initialize\ P(t);$ $P(t)$ Population of generation t
$Evaluate\ P(t);$
$NonDominatedSort(P(t));$ Used for assigning rank to solutions
$P'(t) \coloneqq\ TournamentByRank(P(t));$ $P'(t)$ Population of selected solutions
$Q(t) \coloneqq\ CrossoverAndMutation(P'(t));$ $Q(t)$ New child population
$while\ not\ terminate$ $terminate$ is a stop condition
  $Evaluate\ Q(t);$ e.g. max generations, user stop etc.
  $R(t) = P(t) \cup Q(t);$ Make $R(t)$ a union of $P(t)$ and $Q(t)$;
  $F = NonDominatedSort(R(t));$ Sort $Fronts$ with current population
  $P(t+1) = \emptyset;$ and the last generations population
  $i \coloneqq 1;$ i front counter, N start population size
  $until\ Size\big(P(t+1)\big) \geq N$ Loop until $P(t+1)$ is filled
    $AssignCrowdingDistance(F_i);$ Assign crowding dist. to current front
    $If\ Size\big(P(t+1)\big) + Size(F_i) > N$ If F_i doesn't fit into $P(t+1)$
      $SortCrowdingDistance(F_i);$ Sort based on largest crowding distance
      $P(t+1) \coloneqq P(t+1) \cup F_i[1: N - (Size(P(t+1)))];$ Add the first solutions in F_i to $P(t+1)$ until it's filled
    $else$
      $P(t+1) \coloneqq P(t+1) \cup F_i;$ Add entire front to $P(t+1)$
    $i \coloneqq i + 1;$
  $P'(t+1) \coloneqq\ TournamentByRankAndCrowdingDistance(P(t+1));$
  $Q(t+1) \coloneqq CrossoverAndMutation(P'(t+1));$
  $t \coloneqq t + 1;$
end

B. Results from optimization on ZDT1 to ZDT3 with NSGA-II algorithm

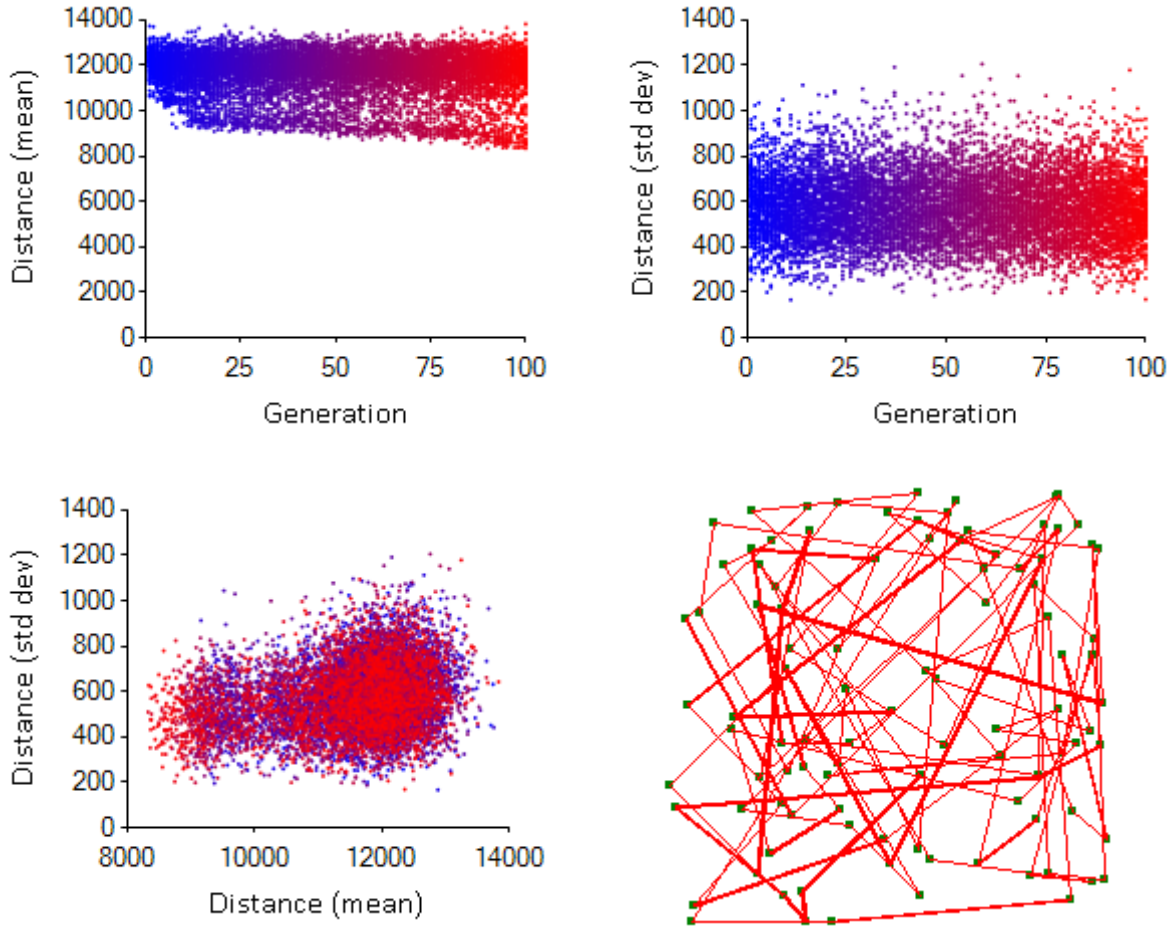
The results for all the ZDT experiments are included in this appendix. All experiments were run with 10000 evaluations, 100 population size, 90% crossover probability (simulated binary crossover) and 10% mutation probability (polynomial mutation). The figures show both all results and the Pareto-optimal results, the first row is ZDT1, second row ZDT2 and third row ZDT3. The results show that the implemented NSGA-II algorithm clearly converges towards the true Pareto-front.



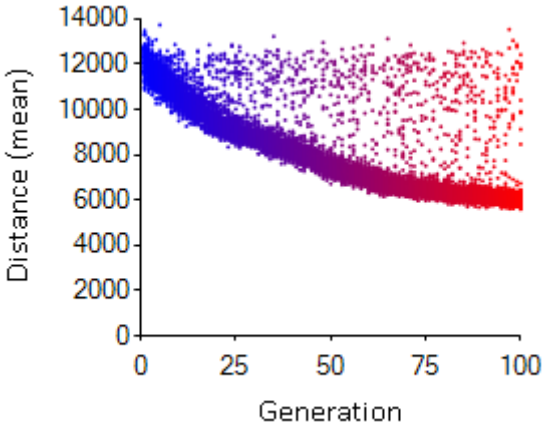
C. Results from optimization on TSP problem with NSGA-II

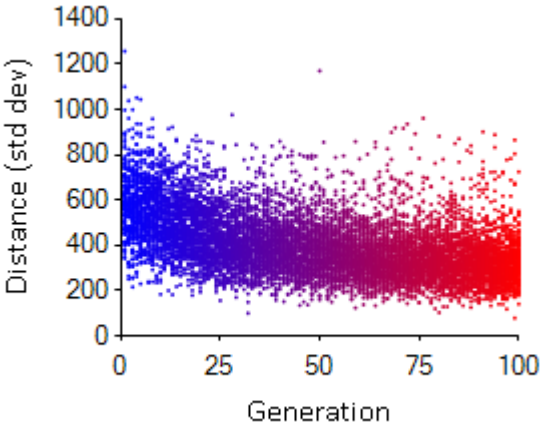
The TSP problem was run with four main configurations with NSGA-II, treating the problem as black box versus white box and using one objective versus two objectives (mean and standard deviation). For all these experiments the TSP problem has the same configuration with 100 random placed cities, with stochastic behavior and 10 replications. All results are presented with four graphs to easier see the tendencies of minimizing mean value/standard deviation and to see the best result visualized as the city tour. The city tour illustrates the result by the route it selected and by the thickness of the lines which represents the random behavior of the solution. Thicker line represents more random behavior. All experiments are run with 100 population size, 10000 evaluations, 90% crossover probability (black box order crossover and white box greedy crossover) and 10% mutation probability (shuffle mutation).

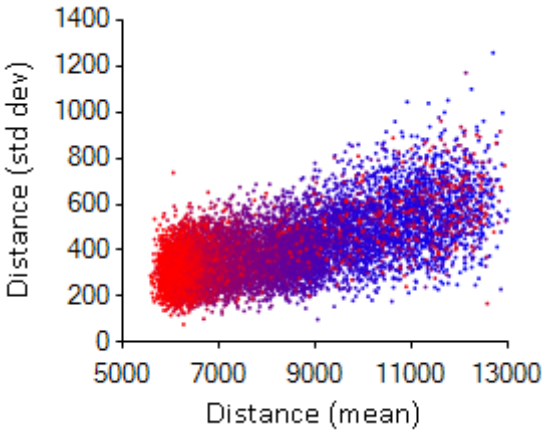
The first results are from treating the problem as a black box with standard deviation as an additional objective. The figures show that the algorithm optimizes the solutions but the search performance is slow. The upper left graph shows that the algorithm optimizes the solutions with regard to mean value but the standard deviation does not improve as shown in the upper right graph. Looking at the city tour the route of which the cities are visited is not very optimized.

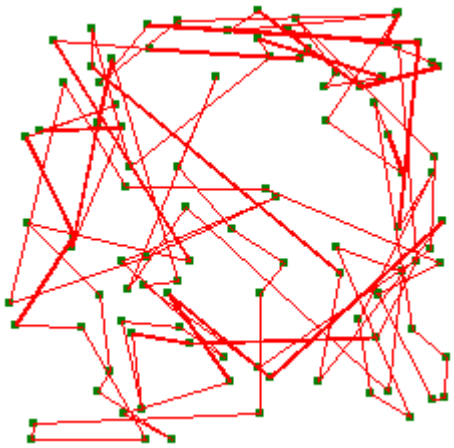


The second results are from treating the problem as a black box without the additional objective. The figures show that the algorithm optimizes the solutions and compared to the previous results the search performance is a lot faster. The upper graphs show that the algorithm optimizes the solutions with both regard to mean value and the standard deviation. Looking at the city tour the route of which the cities are visited is not very optimized but is improved compared to the previous result. This indicates that when treating the problem as a black box the additional objective slows down the search performance.

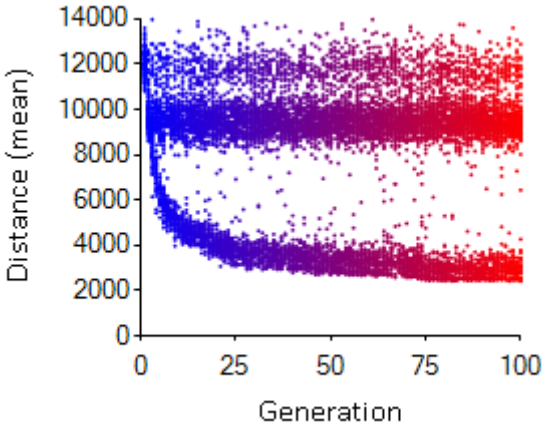


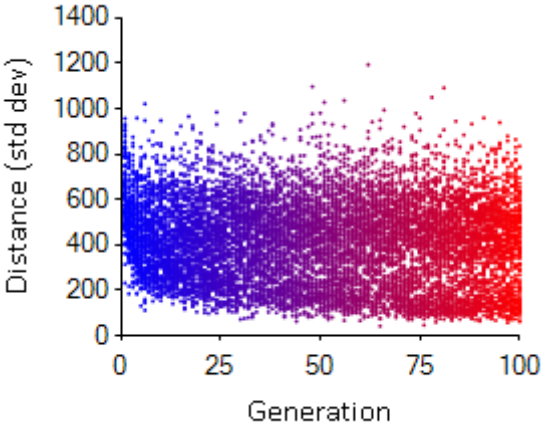


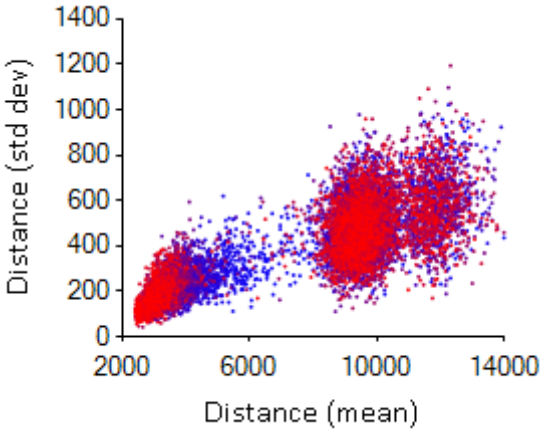


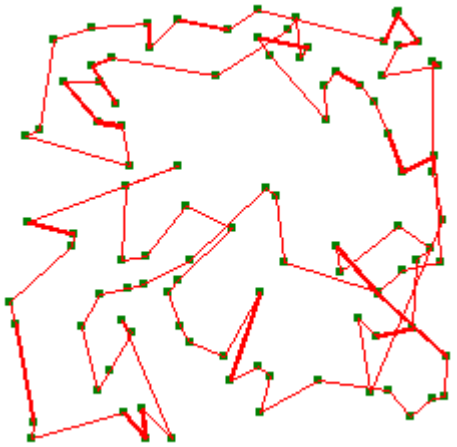


The third results are from treating the problem as a white box with standard deviation as an additional objective. The figures show that the algorithm optimizes the solutions with good search performance in the beginning but flats out after 25. The upper graphs show that the algorithm optimizes the solutions with both regard to mean value and the standard deviation. Looking at the city tour the route of which the cities are visited is much more optimized compared to the previous results. This is due to the fact that treating it as a white box the algorithm can consider the characteristics of the problem which speeds up the search performance.

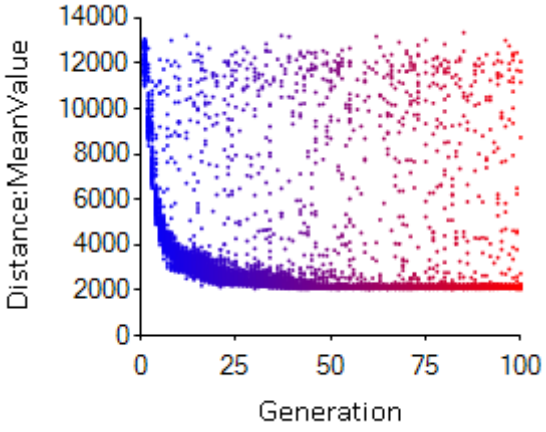


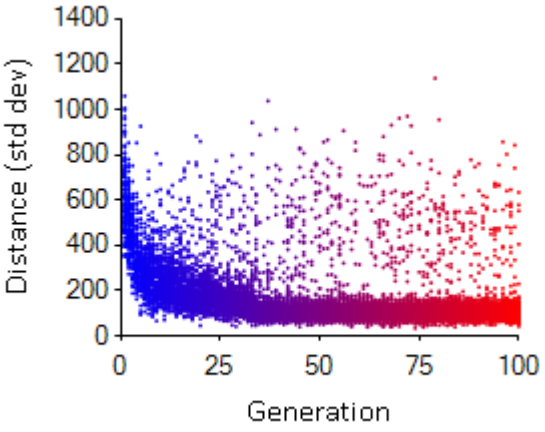


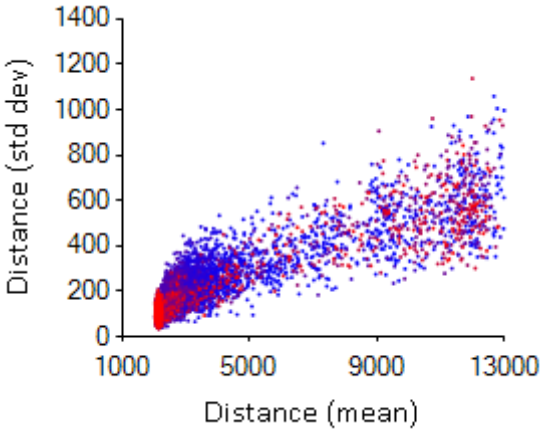


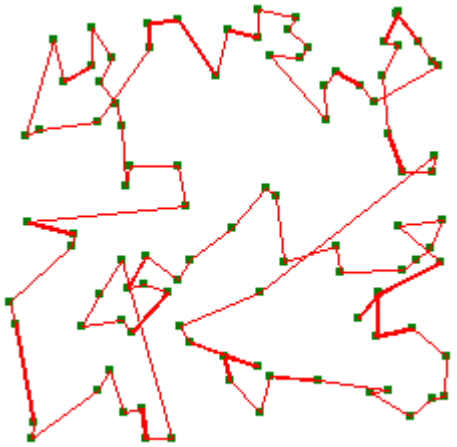


The final results are from treating the problem as a white box with only mean value as an objective. The figures show that the algorithm optimizes the solutions with good search performance and flats out after 25 generations. The upper graphs show that the algorithm optimizes the solutions with its peak at 25 generations, after that the solutions barely improves with both regard to mean value and the standard deviation. Looking at the city tour the route of which the cities are visited is even more optimized compared to the previous results. This indicates that when treating the problem as a white box the additional objective slows down the search performance.





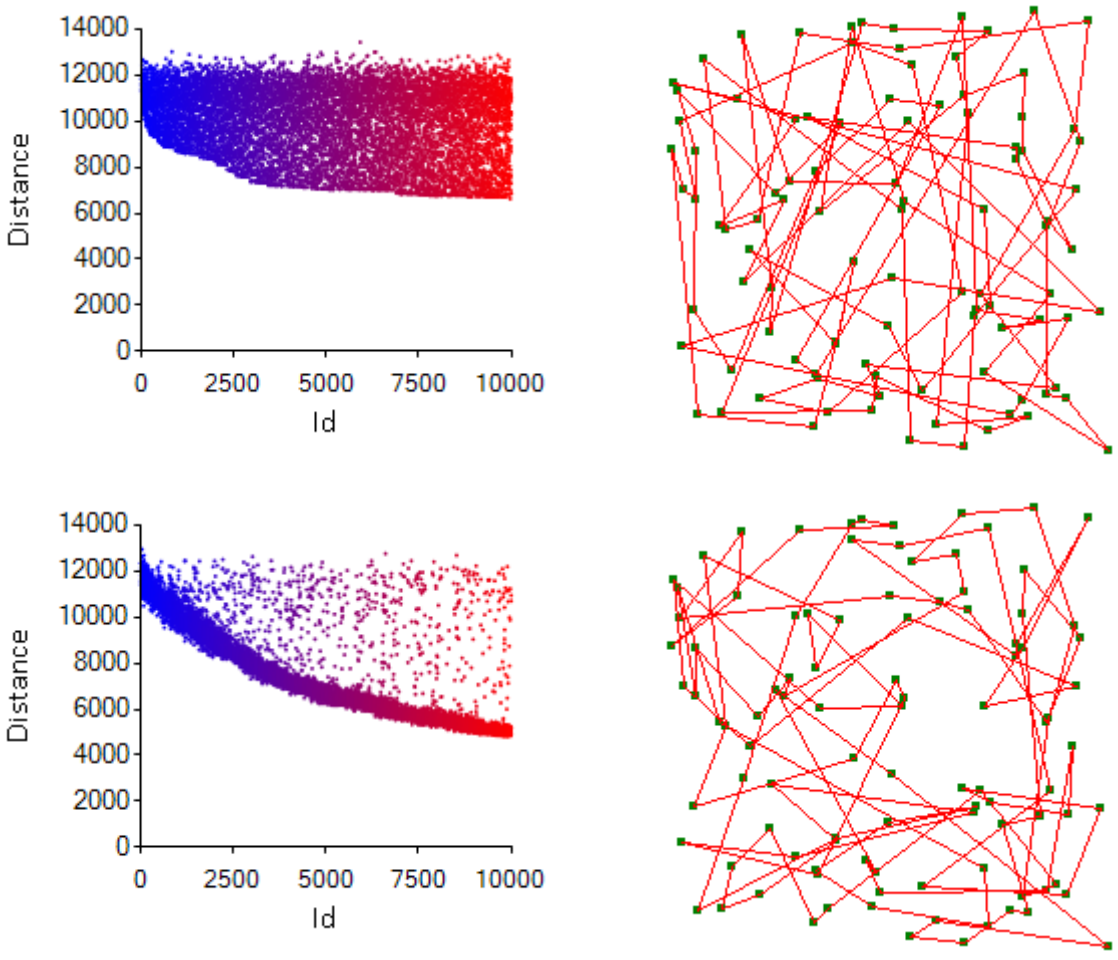




D. Comparison of hill-climbing and NSGA-II on single objective TSP

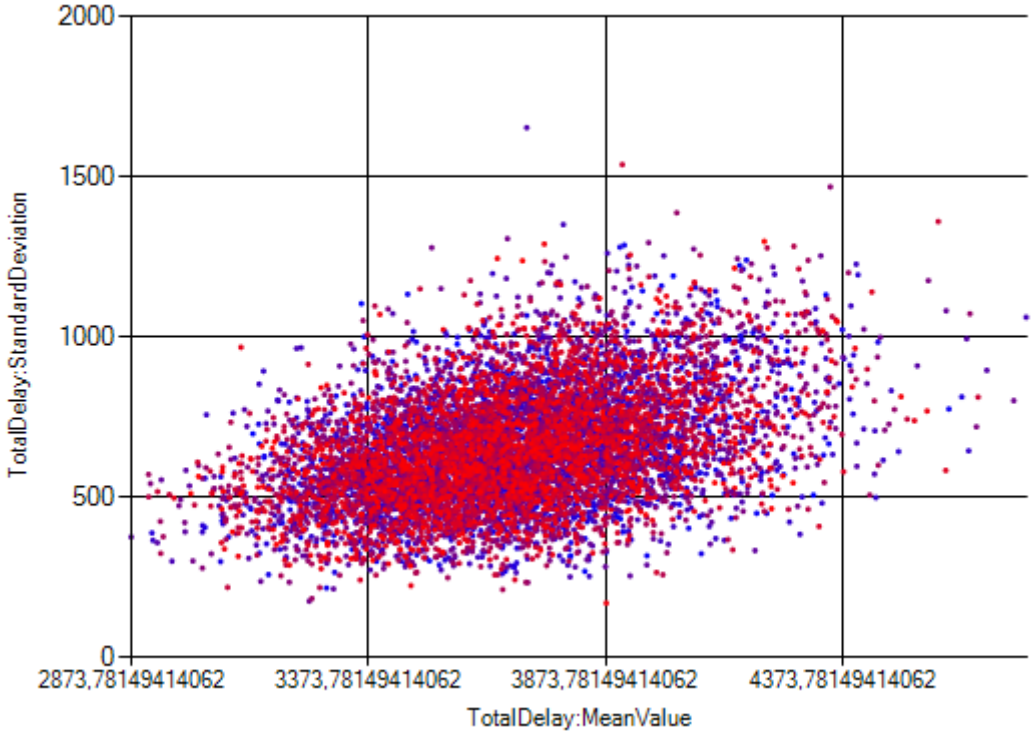
To evaluate the performance of NSGA-II on single objective optimization problems it is compared to a hill-climbing algorithm. For all these experiments the TSP problem has the same configuration with 100 random placed cities, with no stochastic behavior and one replication. All results are presented with two graphs to compare the search performance on finding low distances and also to compare the city tours where the route is illustrated. All experiments are run with 10000 evaluations. The NSGA-II has the settings 100 population size, 90% crossover probability using order crossover and 10% mutation probability using the shuffle mutation. The Hill-climbing algorithm has one population size and 100% mutation probability using the shuffle mutation.

First row is the results from the hill-climbing algorithm and the second row is the results from the NSGA-II algorithm. Looking at these two results the NSGA-II algorithm performs slightly better than the hill-climbing algorithm. The results from the NSGA-II algorithm show tendencies that it can continue the optimization while the hill-climbing flats out the search.

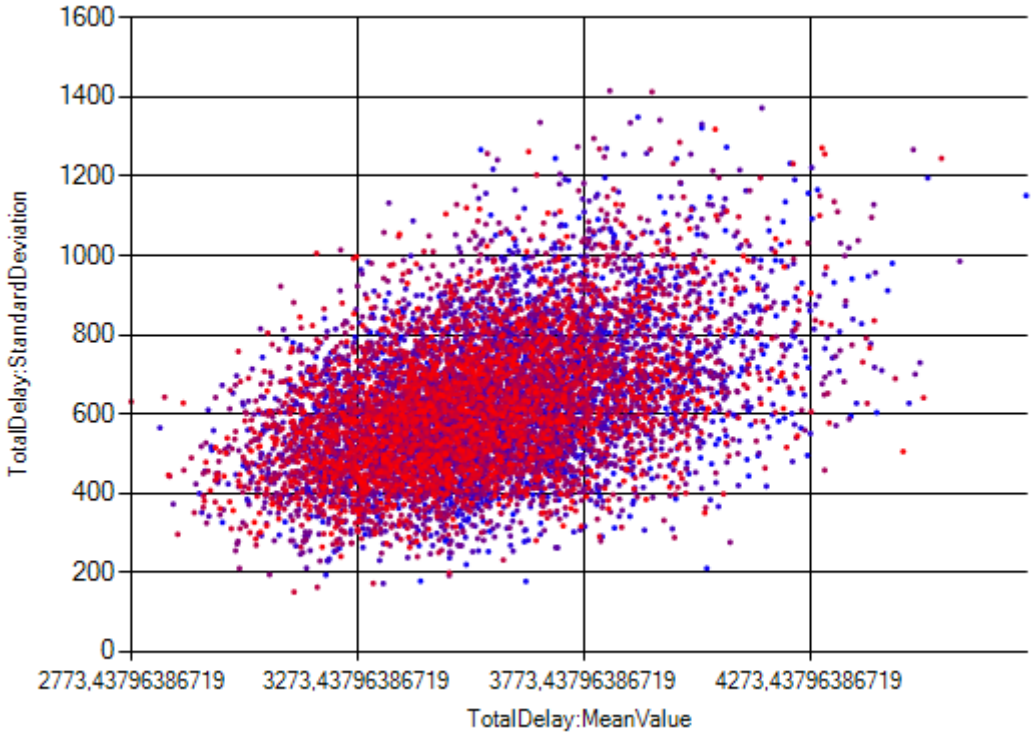


E. All SIMUL8 results

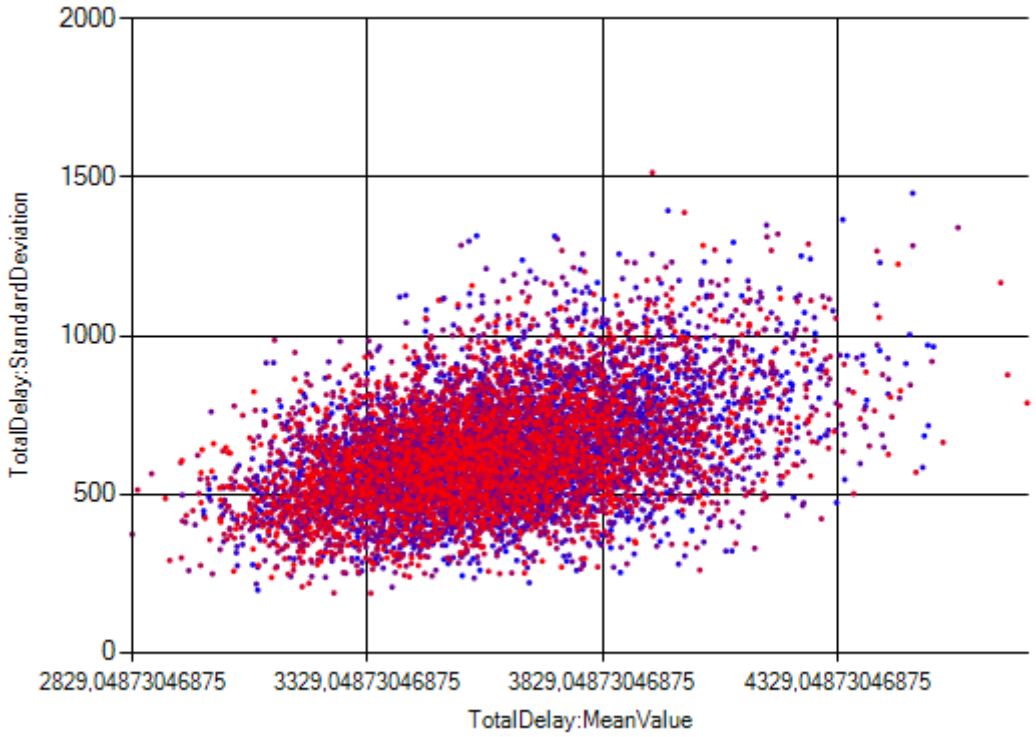
All results from experimentation with SIMUL8 model are in this appendix. All experiments run with 10000 evaluations. First result is for NSGA-II algorithm with standard deviation, 100 population size, 90% crossover probability (order), 10% mutation probability (shuffle).



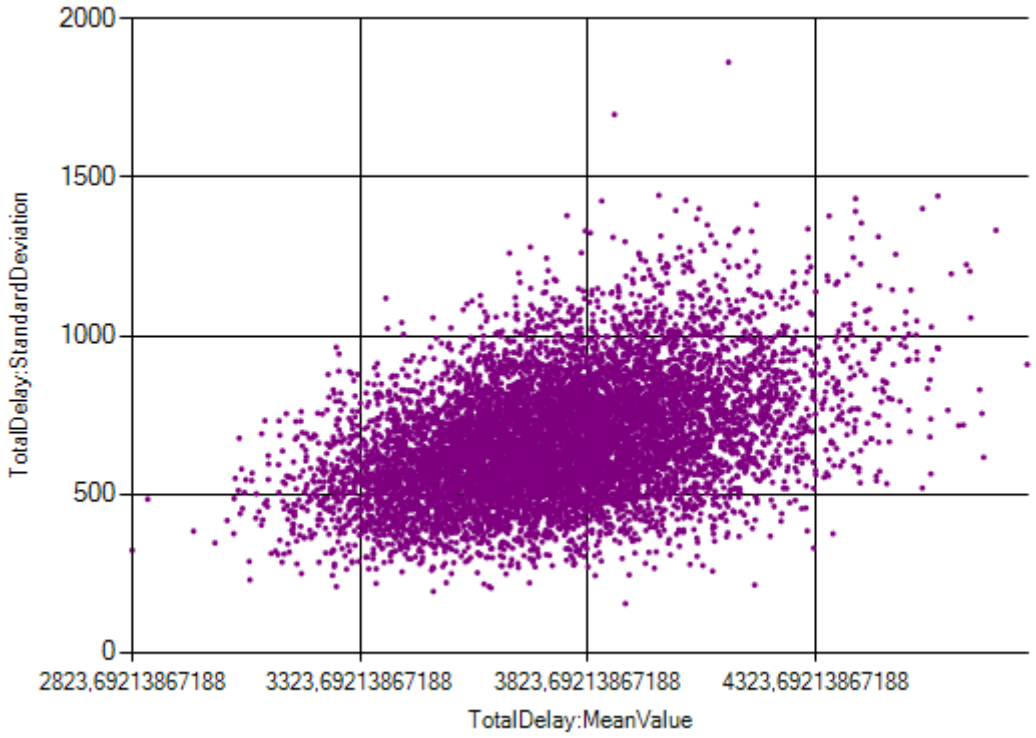
This result is for NSGA-II without standard deviation; all other settings are the same.



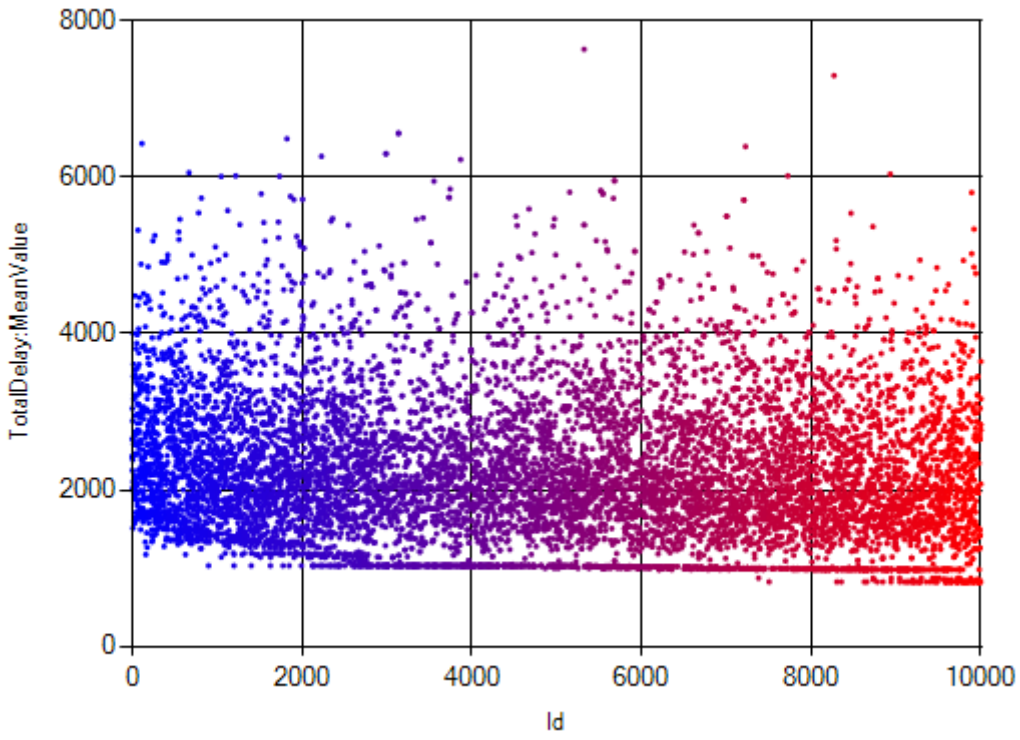
Hill climbing results with shuffle mutation, this result is similar to the NSGA-II results.



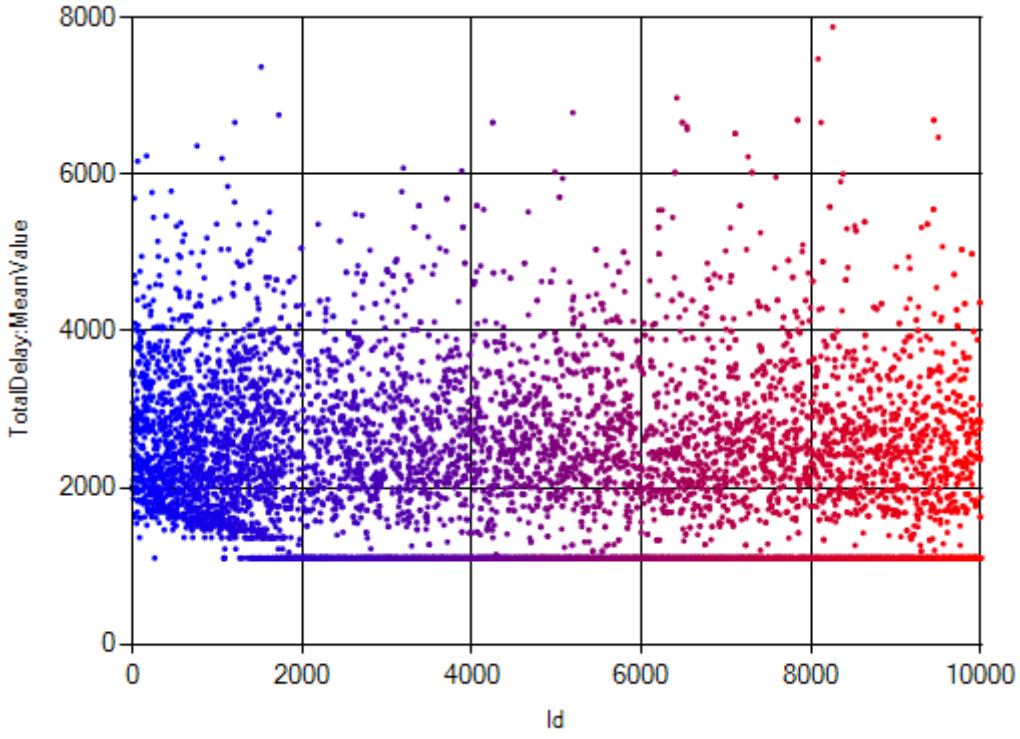
Results from a random search with 10000 random generated solutions. Also similar to the other results meaning that the model might be too stochastic.



This result is from NSGA-II algorithm on deterministic SIMUL8 model, NSGA-II settings without standard deviation but otherwise the same as before. With the deterministic model the algorithm searches towards better solutions but still very slow.



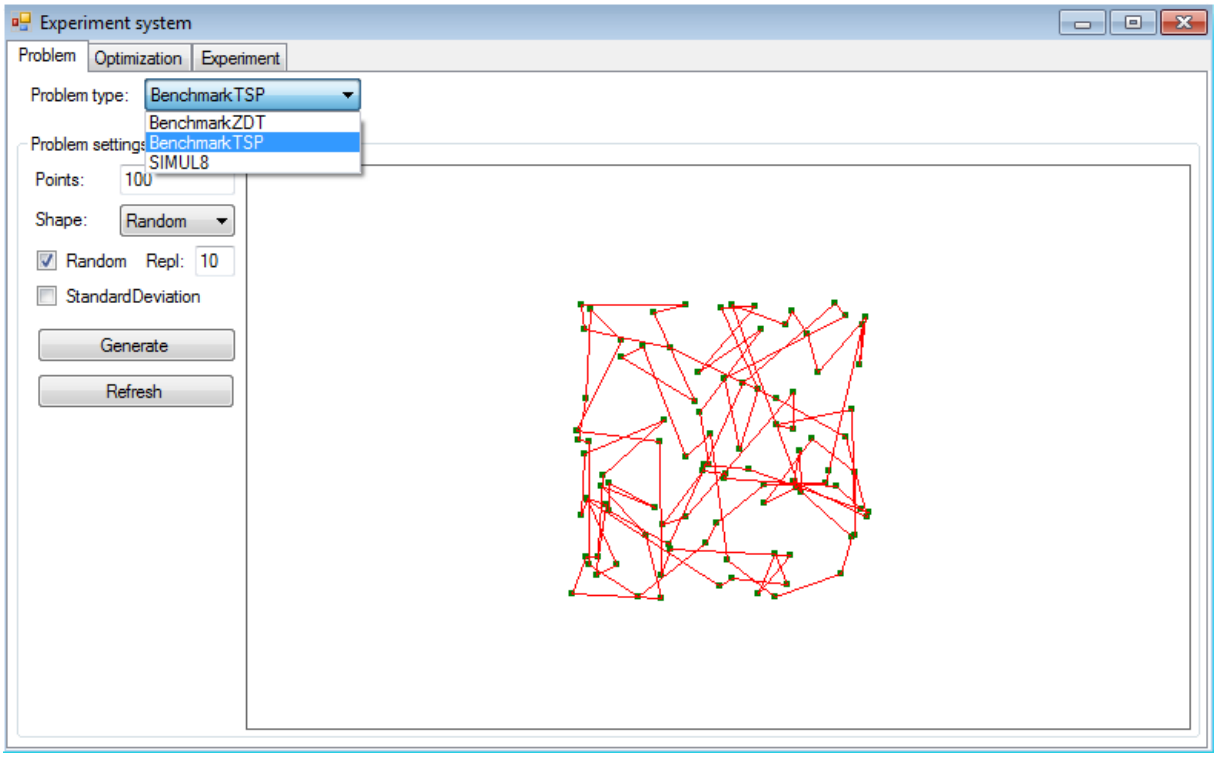
Results from NSGA-II on deterministic SIMUL8 model with the same settings as the previous. This time the experiment is made with 59 settings instead of 236 which the previous had.



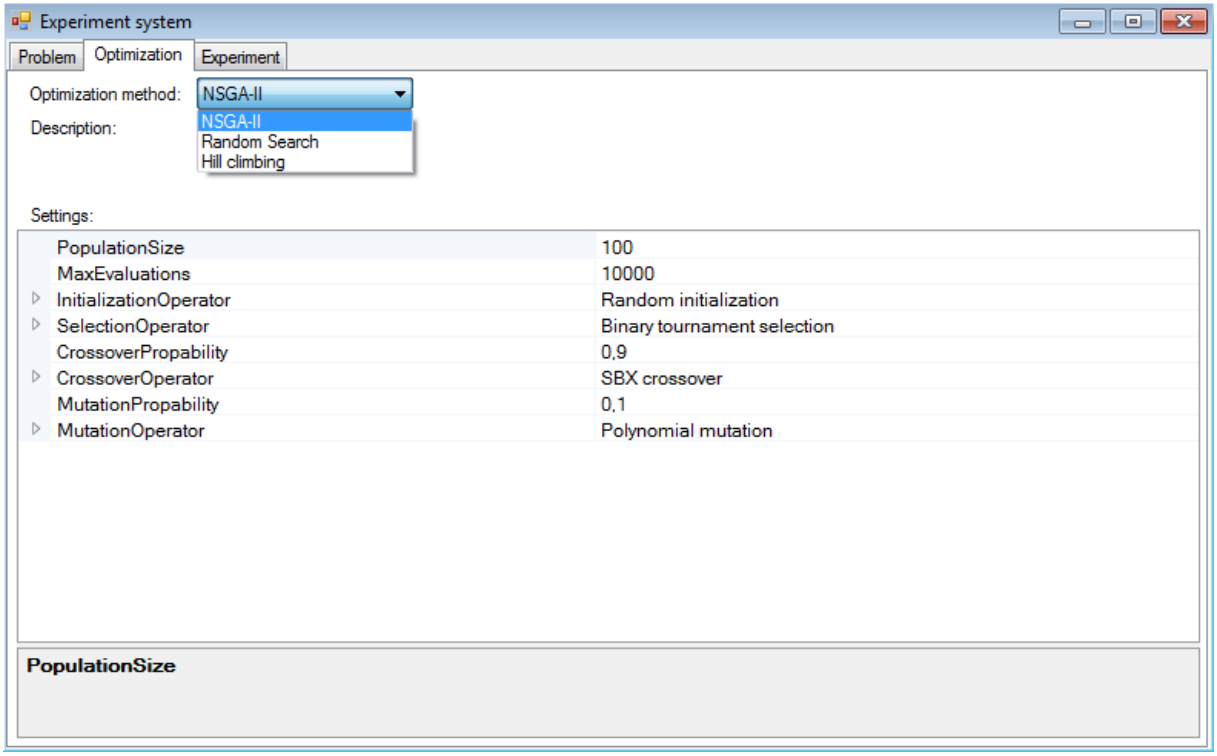
F. Experiment program

The experiment program is explained in this appendix. The experiment program has three main tabs, problem, optimization and experiments. These three tabs will be explained using pictures and text.

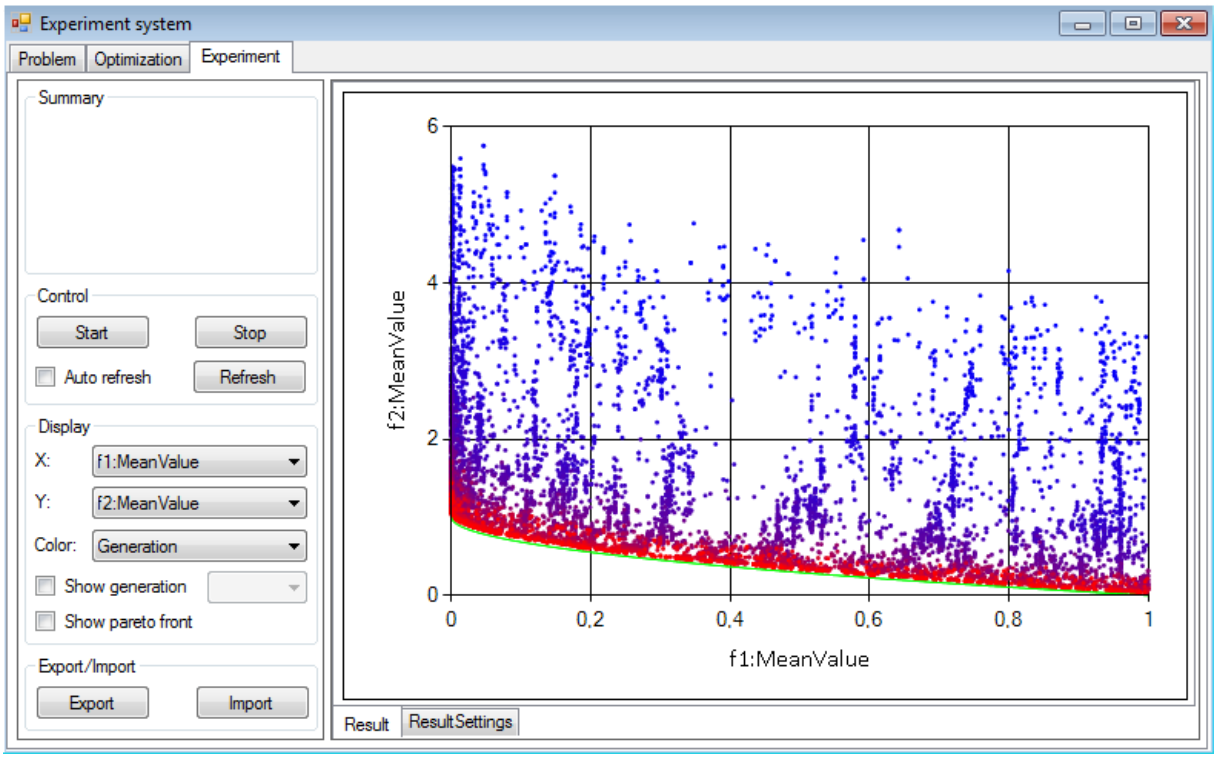
In the problem tab, shown in the first picture, the type of problem can be selected. There are three different types of problems implemented, ZDT benchmark tests, TSP benchmark test and the SIMUL8 problem. Here the TSP problem is visible showing an example of 100 randomly generated cities with a printed tour. In the TSP problem settings can be made if it should be stochastic, if we want to use standard deviation, what type of generated shape and how many cities to be generated. When selecting the ZDT problem the developer can select which problem to use ZDT1 to ZDT3. In the SIMUL8 problem settings can be made for which agents to be used, replications, use of standard deviation as objective, product settings etc.



In the optimization tab the developer can select what algorithm to use and also settings for the algorithm. There are three implemented algorithms, NSGA-II, random search and hill-climbing. In the NSGA-II algorithm settings can be made for what operators to use, population size etc. This is possible using the ParameterList class which contains all settings for every algorithm/operator. If the problem is changed and the encoding becomes different then the settings are changed appropriately i.e. if an operator only manages real encoded inputs then it is not listed here when using permutation encoded values.



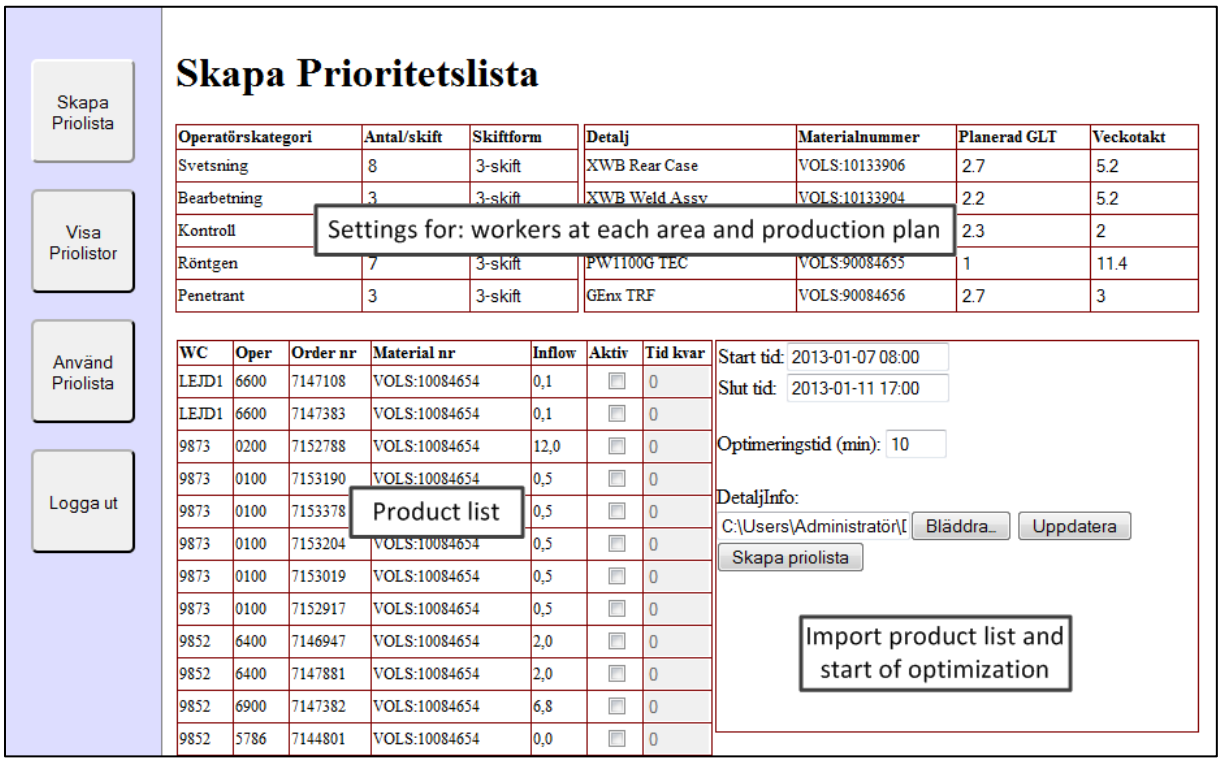
In the experiment tab the developer starts the experiments with settings made in the problem tab and optimization tab. Here the results can be analyzed with a two-axis graph and also a color-axis. To evaluate how the algorithm searches towards better results both the color-axis can be used but also the checkbox “Show generation” can be used to browse the generations. The results for the ZDT problem show the true Pareto-front, green line, and the generated Pareto-front can be shown. The results can also be exported and imported.



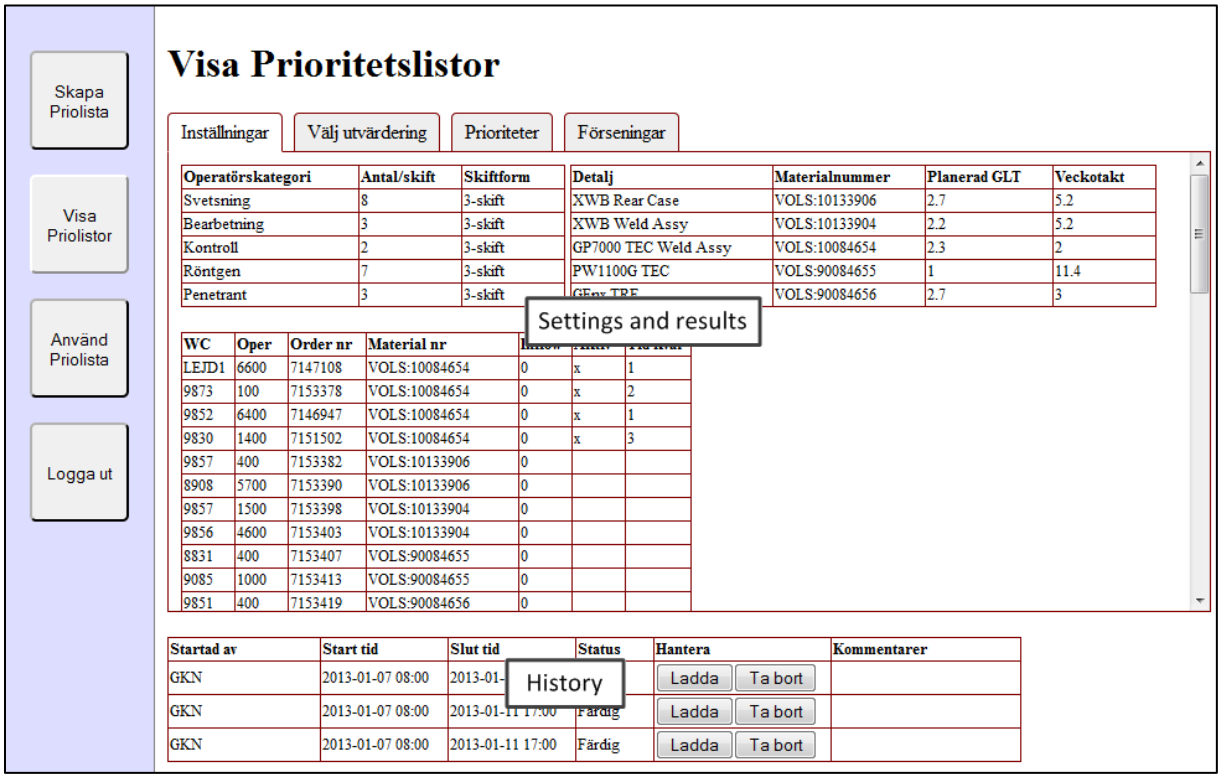
G. Priority system web interface

This appendix contains all pictures of the user interface that is intended for workshop at GKN Aerospace. The user interface is web based and uses HTML5. JavaScript is utilized throughout the user interface since the priority management will be dynamic based on the generated priority list. Since GKN Aerospace is located in Sweden the user interface uses Swedish language. The framed text boxes are for illustrating different areas in the interface i.e. not included in the web page. To the left of the web page is the navigation field which includes four buttons, three buttons to change interface and the bottom one is to log out.

In the first picture the creation interface is displayed. In this page the worker will fill in all data for the current state of the workshop. In the top there are settings for workers at each work area and production plan. To the left there are settings for product list which is imported from the resource enterprise system developed by SAP. To the right the product list can be imported selecting a file and here the optimization is started.

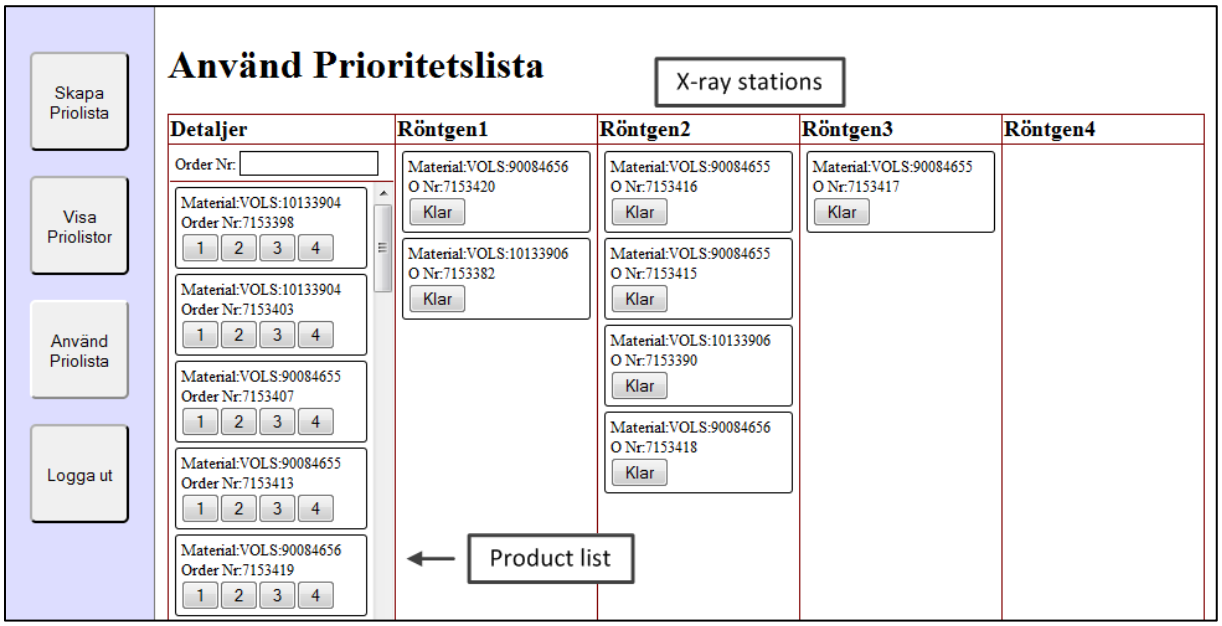


The next pictures is the history/result interface, here the optimization can be reviewed if the workers see fit. However, the best result is automatically selected for the priority user interface. If for example the worker needs to select a previous optimization then this interface can be used. In the top the optimization results are shown, here the evaluations can be reviewed where the result from each evaluation can be analyzed. Old optimizations can be loaded at the bottom of the page where the history can be browsed.



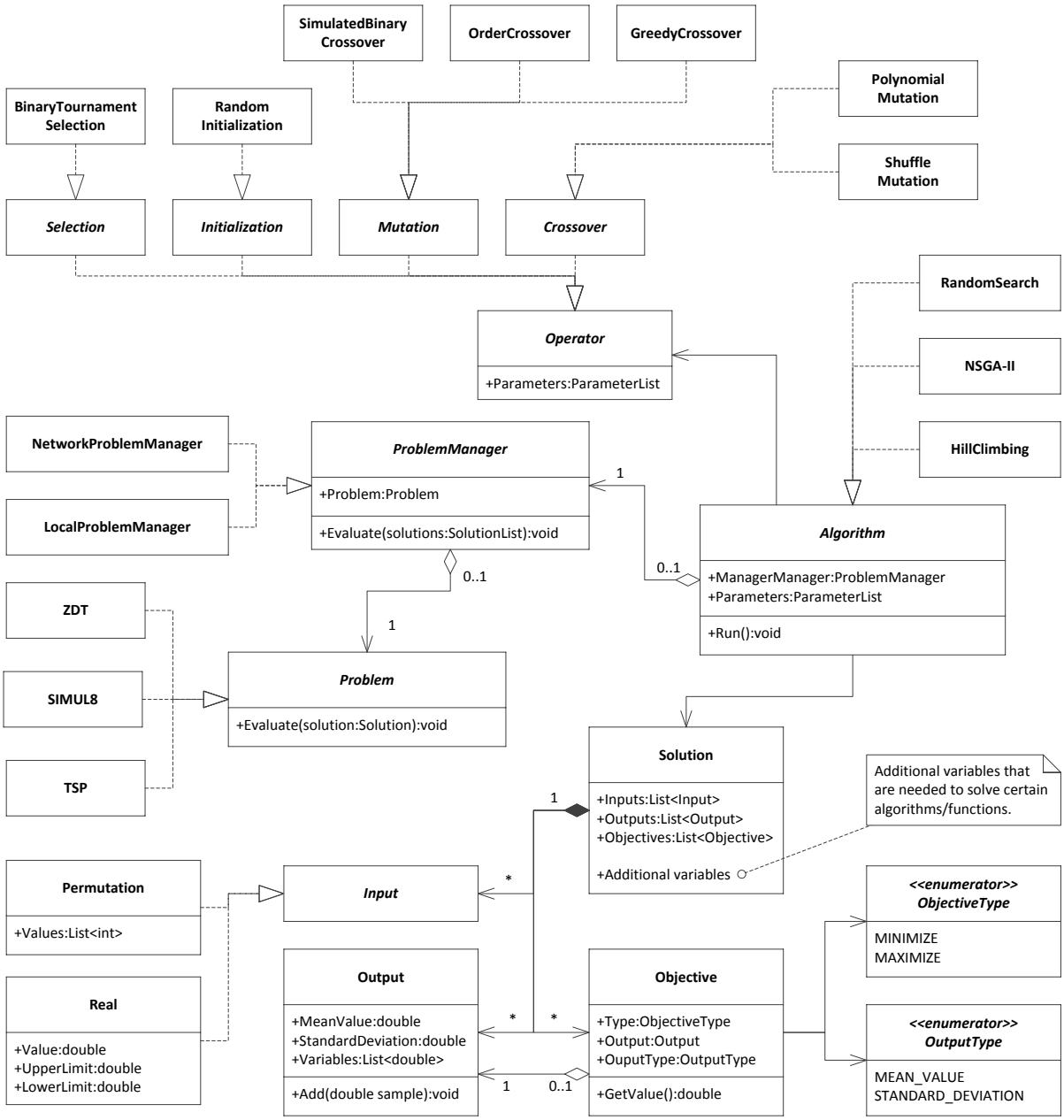
The final page is the interface for using the priority list. Here the product list is to the left where the products can be imported into the queue for each x-ray station. There is an input on top of the product list where the products can be filtered in order to find the incoming product. Following is a description of how to work with this user interface:

- When a product arrives to a queue for one of the x-ray stations the worker pushes the buttons for the corresponding x-ray station. The product is then automatically inserted to the x-ray queue in the correct position based on the priority list.
- When the current product is done the workers will remove this product from both the x-ray station and the x-ray queue in the interface. Then the next product is selected from the x-ray queue in the interface to be started in the station.



H. Complete UML model of GOF

In the figure is the complete UML model of the GOF with all the operators, inputs and problems.



I. Sample codes

This appendix details two classes to display samples of the programming done during this project. First the implementation of the NSGA-II algorithm is detailed. Here is an example of how the ParameterList is initialized by the algorithm.

```
/// <summary>
/// Represents the non-dominated sorting genetic algorithm II class.
/// This algorithm is a state-of-the-art method to solve multi-objective optimization problems.
/// It is a multi-objective evolutionary algorithm that with genetic evolution produces offspring.
/// The survival of the population is based on non-domination and crowding distance.
/// </summary>
public class NSGAII : Algorithm
{
  /// <summary>
  /// Initializes a new instance of GOF.Algorithm.NSGAII class with default parameters:
  /// population size, max evaluations, initialization operator, selection operator,
  /// crossover propability, crossover operator, mutation propability and mutation operator.
  /// </summary>
  public NSGAII()
  {
    // Set default parameters
    ParameterList.Add("PopulationSize", typeof(int), 100);
    ParameterList.Add("MaxEvaluations", typeof(int), 10000);

    ParameterList.Add("InitializationOperator", typeof(Initialization));
    ParameterList.Add("SelectionOperator", typeof(Selection));

    ParameterList.Add("CrossoverPropability", typeof(double), 0.9);
    ParameterList.Add("CrossoverOperator", typeof(Crossover));

    ParameterList.Add("MutationPropability", typeof(double), 0.1);
    ParameterList.Add("MutationOperator", typeof(Mutation));
  }

  /// <summary>
  /// Starts the NSGA-II algorithm.
  /// </summary>
  public override void Run()
  {
    //Get parameters
    int populationSize = (int)ParameterList["PopulationSize"].Value;
    int maxEvaluations = (int)ParameterList["MaxEvaluations"].Value;
    double crossoverPropability = (double)ParameterList["CrossoverPropability"].Value;
    double mutationPropability = (double)ParameterList["MutationPropability"].Value;

    //Get operators
    Initialization initialization = (Initialization)ParameterList["InitializationOperator"].Value;
    Selection selection = (Selection)ParameterList["SelectionOperator"].Value;
    Crossover crossover = (Crossover)ParameterList["CrossoverOperator"].Value;
    Mutation mutation = (Mutation)ParameterList["MutationOperator"].Value;

    //Prepare operators, this is done to reduce time when running the algorithm
    initialization.Prepare();
    selection.Prepare();
    crossover.Prepare();
    mutation.Prepare();
```

```
//Create local variables
SolutionList population = initialization.Execute(ProblemManager.Problem, populationSize);
SolutionList offspring = null;
Random random = new Random();
int evaluation = 0;
int generation = 1;
int id = 1;

//Set generation and id for each solution in this population
foreach (Solution solution in population)
{
  solution.Id = id++;
  solution.Generation = generation;
}

// Start the NSGA-II algorithm
ProblemManager.Evaluate(population);
NonDominatedSort.Execute(population); //Only to set ranks
evaluation += populationSize;

// Start the main loop
while (evaluation < maxEvaluations)
{
  //Increase the generation counter
  generation++;

  // Assign crowding distance
  SolutionPair parents = new SolutionPair();
  SolutionPair children = null;
  offspring = new SolutionList(populationSize);
  for (int i = 0; i < populationSize / 2; i++)
  {
    //If max evaluation then stop
    if (evaluation + offspring.Count >= maxEvaluations)
      break;

    //Get the mating parents
    parents.First = selection.Execute(population);
    parents.Second = selection.Execute(population);

    //Do crossover
    if (random.NextDouble() < crossoverPropability)
      children = crossover.Execute(parents);
    else
      children = new SolutionPair(parents.First.Clone(), parents.Second.Clone());

    //Do mutation
    if (random.NextDouble() < mutationPropability)
    {
      mutation.Execute(children.First);
      mutation.Execute(children.Second);
    }

    children.First.ParentIds = parents.First.Id + ":" + parents.Second.Id;
    children.Second.ParentIds = parents.First.Id + ":" + parents.Second.Id;
```

```
            //Add children to the offspring population
            offspring.Add(children.First);
            offspring.Add(children.Second);
          }

          //Set generation
          foreach (Solution solution in offspring)
          {
            solution.Id = id++;
            solution.Generation = generation;
          }

          //Evaluate the offspring
          ProblemManager.Evaluate(offspring);
          evaluation += offspring.Count;

          //Merge
          SolutionList union = new SolutionList(population);
          union.AddRange(offspring);

          //Clear population
          population.Clear();

          //Add fronts to the population until it's filled
          FrontList fronts = NonDominatedSort.Execute(union);
          foreach (SolutionList front in fronts)
          {
            CrowdingDistanceAssignment.Execute(front);
            if (population.Count + front.Count > populationSize)
            {
              // Add the best ranked solutions to the parents population until max population size been
              // reached
              front.Sort(new CrowdingDistanceComparator());
              foreach (Solution solution in front)
              {
                // Add solution to parent
                population.Add(solution);

                //Stop loop when population size is reached
                if (population.Count == populationSize)
                  break;
              }
            }
            else
            {
              // Add full range to the population
              population.AddRange(front);
            }

            // Stop loop when population size is reached
            if (population.Count == populationSize)
              break;
          }
        }
      }
    }
```

Here the algorithm non-dominated sort is detailed. This algorithm sorts a population into a list of fronts where each front is a non-dominated population.

```
/// <summary>
/// Represents a static class which function is to make a non dominated sort of a SolutionList.
/// The sort algorithm returns a FrontList where the solutions in each front are non-dominated.
/// </summary>
static public class NonDominatedSort
{
  /// <summary>
  /// Executes the non dominated sort of the solutions.
  /// </summary>
  /// <param name="solutions">The solutions to sort.</param>
  /// <returns>The sorted solutions in a FrontList</returns>
  static public FrontList Execute(SolutionList solutions)
  {
    // Create the front list
    FrontList fronts = new FrontList();
    SolutionComparator dominance = new DominanceComparator();

    // Clear the solution statistics
    foreach (Solution solution in solutions)
    {
      solution.DominationCounter = 0;
      solution.DominatedList.Clear();
    }

    // First begin the loop for setting domination counter and dominated list
    for (int x = 0; x < solutions.Count; x++)
    {
      Solution a = solutions[x];
      for (int y = x + 1; y < solutions.Count; y++)
      {
        Solution b = solutions[y];
        int domination = dominance.Compare(a, b);

        // If a dominates b then add to b to the list of solutions dominated by a
        // Else if b dominates a then increment the dominated counter
        if (domination == -1) //Solution a dominates solution b
        {
          a.DominatedList.Add(b);
          b.DominationCounter++;
        }
        else if (domination == 1) //Solution b dominates solution a
        {
          b.DominatedList.Add(a);
          a.DominationCounter++;
        }
      }

      // If a isn't dominated by any other then add to the first front
      if (a.DominationCounter == 0)
      {
        // Set rank
        a.Rank = 1;

        // Add to front
        if (fronts.Count == 0)
          fronts.Add(new SolutionList());
```

```
        fronts[0].Add(a);
      }
    }

    // Then loop through each front to detect the next front
    int i = 0;
    while (fronts.Count - 1 == i)
    {
      SolutionList front = fronts[i];
      foreach (Solution a in front)
      {
        foreach (Solution b in a.DominatedList)
        {
          // Decrement domination counter and see if b belongs to the next front
          if ((--b.DominationCounter) == 0)
          {
            // Set rank
            b.Rank = i + 2;

            // Add to front
            if (fronts.Count == i + 1)
              fronts.Add(new SolutionList());

            fronts.Last().Add(b);
          }
        }
      }
      i++;
    }

    // Finally return the fronts
    return fronts;
  }
}
```