Improving Steel Stockyard Planning by Coupling Optimization with Stochastic Simulation

Atakan SELAMOGLU

Master Thesis

Presented in partial fulfillment of the requirements for the double degree:
"Advanced Master in Naval Architecture" conferred by University of Liege
"Master of Sciences in Applied Mechanics, specialization in Hydrodynamics, Energetics and Propulsion" conferred by Ecole Centrale de Nantes

Developed at West Pomeranian University of Technology, Szczecin and Federal University of Rio de Janeiro, Rio de Janeiro in the framework of the

"EMSHIP"
Erasmus Mundus Master Course in
"Integrated Advanced Ship Design"

Ref. 159652-1-2009-1-BE-ERA MUNDUS-EMMC

Supervisor: Dr. Remigiusz Iwankowicz, West Pomeranian University of Technology, Szczecin, Poland

Prof. PhD Jean-David Caprace, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil

Reviewer: Prof. Dr.-Ing. Robert Bronsart, Universität Rostock, Rostock, Germany

Szczecin, February 2016

CONTENT

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

This master thesis was prepared within the European master course EMSHIP Integrated Advanced Ship Design. The research part was conducted at Federal University of Rio de Janeiro (UFRJ) in the Simulation Laboratory of Ship Building Processes (LABSEN). The thesis was written in West Pomerian University of Technology (ZUT), Szczecin, Poland.

I would like to thank Professor Jean-David Caprace, my supervisor in UFRJ, for his guidance and support throughout my project. He shared his experience, knowledge and vision with me and he encouraged me during the whole internship.

I also would like to tell my appreciations to Professor Remigiusz Iwańkowicz, my thesis supervisor in ZUT. He shared his knowledge with me and supported me during the writing of the thesis. He gave precious ideas and supplied great guidance to my work.

I want to address my special thanks to Gabriel Premoli Monteiro, who helped me with his knowledge and time during the research in LABSEN. He shared his computer skills with me. I also would like to thank whole my colleagues in LABSEN for their helps.

I would like to offer my appreciations to my friend Doğukan Melih Görmüş and my other friends, colleagues and professors in EMSHIP as well for their supports throughout whole program.

Finally I want thank my family who always supported me with my education and works. I owe everything I accomplish to them.

.

DECLARATION OF AUTHORSHIP

I declare that this thesis and the work presented in it are my own and have been generated by me as the result of my own original research.

Where I have consulted the published work of others, this is always clearly attributed.

Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.

I have acknowledged all main sources of help.

Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

This thesis contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma.

I cede copyright of the thesis in favour of the University of West Pomerian University of Technology (ZUT), Szczecin, Poland.

Date: 15.01.2016 Signature

ABSTRACT

Ship building involves complicated production processes in a highly competitive environment. Therefore improving and obtaining more efficient production facilities is getting more and more important. This fact results in the increase of usage of simulation and optimization tools in the industry. However, coupling these two fields of applications is still not common. This paper proposes a new application by coupling two commercial tools present in the market, QUEST by Delmia for Discrete Event Simulation (DES) and MODEFRONTIER for optimization in a case of improving a steel stockyard in a shipyard. A multi-objective optimization is carried out by taking three design variables and aiming to optimize the objective functions. Number of steel plate piles in stockyard, steel plate capacity of each pile and frequency of steel plate arrival to shipyard are selected as design variables. The objectives are minimizing the area used for stocking and minimizing the Work In Progress (WIP). It is suggested that shipyards with different steel processing capacities would require different sizes of stockyards and different frequencies of incoming steel plates. Therefore a layout plan should be made based on running optimization tasks. The findings provide that production cost can be reduced by carrying out proper planning. The fundamental knowledge of coupling optimization and stochastic simulation tools may result in significant reduction of production costs by minimizing storing area and work in progress. The new solution is also valid for other fields of ship building such as block erection or steel processing and may be modified for further applications.

1. INTRODUCTION

1.1. Context

The work presented covers the optimization of a steel stockyard of Atlantico Sul shipyard that is located in Ipojuca, in the state of Pernambuco, Northeast of Brazil. The study focused the coupling of a Discrete Event Simulation (DES) and an optimization software, which are respectively QUEST and MODEFRONTIER. The aim is to obtain a more efficient process and offer a solution for a real industrial problem.

In such cases, where an optimization task is carried out related with production processes, the main goal is generally reducing the cost. There are several factors that contribute to this purpose such as decreasing the time required, reducing the area that is used or using fewer resources (cranes, conveyors, etc.). In this thesis, a multi-objective optimization will be presented with the aim of minimizing the area used for storage of steel plates and minimizing the Work In Progress (WIP). The objectives and related parameters will be justified in the next sections.

For better understanding the importance of the subject and understanding the reasons of selection of the parameters and defining the objectives, economics of shipbuilding and global shipbuilding market will be described initially. Lean manufacturing in shipbuilding, use of DES and optimization approaches and tools in shipbuilding industry will also be discussed in the section “State of the art”.

1.2. Economics of Ship Building

A tanker, bulk carrier or other types of ships which can be classified under the title of “merchant ship” is world’s largest product that is produced in a factory. It requires tons of materials with the vast majority of steel. Besides steel, several other items such as cables, pipes or furniture are also covered in the production of a merchant ship. Therefore, shipbuilding process is a complex process, which includes several disciplines and thousands of operations. Shipyards are plants that consist of many different production facilities in order to fulfil the required actions.

Stopford (1997) states materials covers more than the half of shipbuilding cost. There are several other contributing costs such as engine and labour. Figure 1 shows the costs of building a merchant ship briefly.

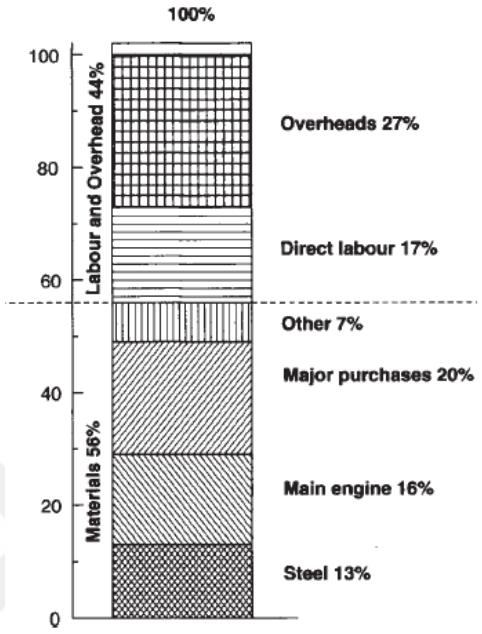


Figure 1: Cost distribution of building a merchant ship, Available from Stopford (1997)

As it can be seen from the Figure 1, among all the materials steel covers the majority of cost distribution.

For better understanding the economics of shipbuilding, the stages of production process should be briefly described. Stopford (1997) defines the process under nine stages. Figure 2 shows the stages.

1. Designing and estimating
2. The steel stockyard
3. Steel shot blast plant
4. Plane and stiffener preparation
5. Assembly
6. Pre-outfitting
7. Coating
8. Assembly on berth
9. Outfit and outfit quay

Figure 2: Stages of shipbuilding process

In the first stage, design and production planning are developed gradually starting from a general point of view to a detailed condition. Production drawings and parts lists are prepared. Required materials are identified and ordered. It should be noted that it is an essential stage and proper planning may dramatically increase the productivity of the shipyard.

Second stage covers the actions taking place in a steel stockyard. After the arrival of ordered steel plates, they are stored. As it is stated earlier, steel is a vast cost in shipbuilding and great amount of steel is stored and processed throughout the process. Therefore, it is crucial to properly plan the storage process to reduce costs.

Steel plates and sections are processed, shot blasted and primed in the third stage. This stage is followed by plate and stiffener preparation. Plates are cut to obtain to required size. They are prepared for welding. In the next stage, prepared steel plates, framing members are assembled into blocks.

The hull is fitted with several materials as well such as pipes, cables, machinery etc. These fittings are done during assembly stages. The key point is completing outfitting as much as possible in earlier stages of production. Coating is carried out in the late stage of production.

Parts that are already built are gathered in the dock, aligned and welded. Finally, the outfitting is completed.

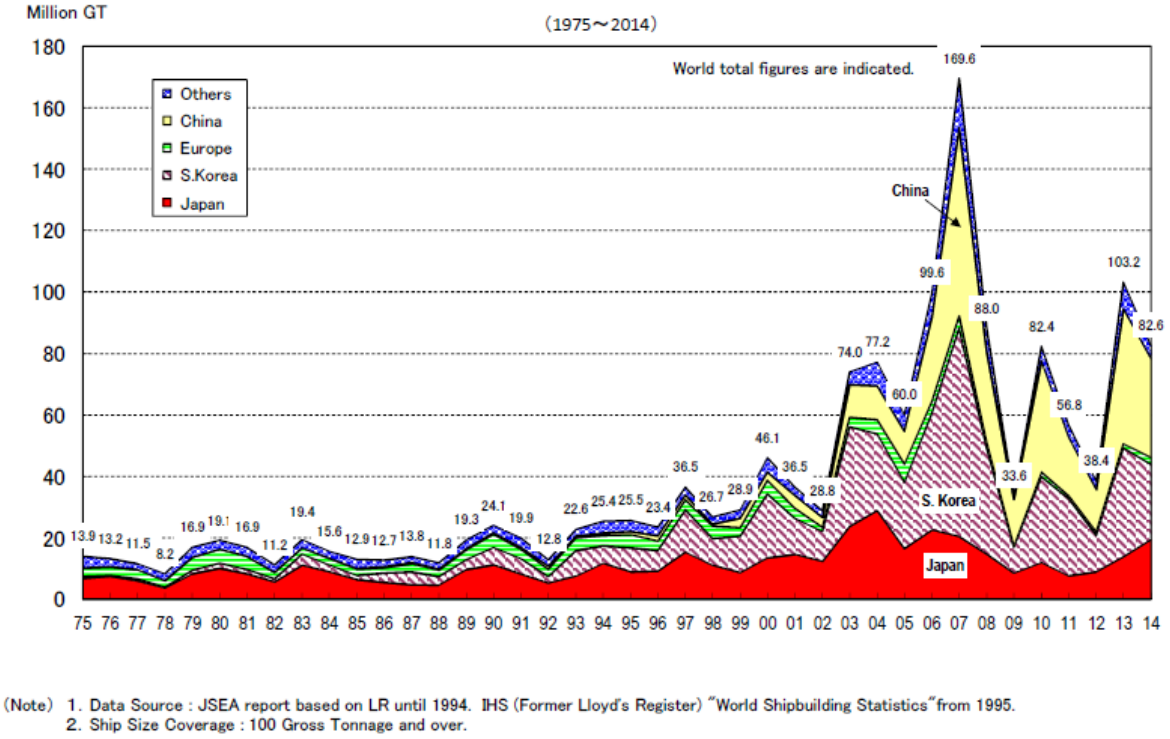
The stages that are briefly described above clearly show that shipbuilding is a sophisticated process. Every stage of production required unique attention and proper planning in order to accomplish desired goals in pre-defined dates.

Stopford (1997) also underlines the fact that competitiveness of a shipyard in the market regarding pricing depends on six key variables which are material supply, facilities, skilled labour, wages, labour productivity and exchange rates. Also in some cases subsidy may also affect the ship production. These factors determine the number of ships that are produced, their prices and the income of shipbuilder.

Despite the fact that facilities affect the production of ships, the performance and output of a shipyard is dramatically affected by the “productivity” of the shipyard. Since shipbuilding is a complex process proper management, planning and organization are key factors defining the efficiency and performance of the shipyard. The term “productivity” will be described in more details in the next section.

1.3. Global Shipbuilding Market

In recent years, despite the effect of global economic crisis, shipbuilding industry grew rapidly and competition became tougher. Figure 3 shows the new orders for ships in the world in million gross tonnages with respect to years between 1975 and 2014 (Shipbuilding Statistics, 2015). It can be seen that with newly participating and growing actors in industry such as China and South Korea, the competition is increasing.



(Note) 1. Data Source : JSEA report based on LR until 1994. IHS (Former Lloyd's Register) "World Shipbuilding Statistics" from 1995.
2. Ship Size Coverage : 100 Gross Tonnage and over.

Figure 3: World new orders of merchant ships with 100 Gross Tonnage and over between the years 1975 and 2014, Available from Shipbuilding Statistics, 2015

Before further discussion and evaluation of global market leaders, "productivity" should be briefly defined. Productivity is generally used as efficiency as well. Gebhart and Jarvis (2003) state that in shipbuilding point of view, productivity is generally defined as man-hours per ton for steel work or man-hours per compensated gross ton (CGT) for whole ship. In addition, competitiveness of a shipyard is often measured with total cost per CGT. CGT is a term used to define the amount of work that is required to build a ship. It is calculated by multiplying tonnage of ship with a certain coefficient which is dependent on type and size of the corresponding ship.

In present day, South Korea, Japan and China are competing for the biggest share from the global merchant ship building market. Korea offers modern facilities and low labour costs, while Japan is still leading in the field of productivity. Park et al (2006) states the fact that when Japan is set to be the baseline for productivity as 1.0, Korea is 0.7. It can be further described with delivery lead-time averages. In Japan, it takes 400,000 to 600,000 man-hours and 6-8 months while in Korea it takes 500,000 to 700,000 man-hours and 7-11 months to build a large, double-hull crude oil carrier.

Koenig et al (2003) also shares similar information with the addition of China and Western Europe into the comparison. Table 1 below shows evaluation of competitiveness factors such as productivity, labour cost and delivery time in Japan, South Korea, China and Western Europe.

Table 1: Competitiveness factors in countries

	Japan	**S. Korea**	**China**	**W. Europe**
Productivity	1.0	0.7	0.2	0.6
Labour cost	1.0	0.5	0.2	0.8 - 1.2
Delivery time	shortest	longer than Japan	much longer than Japan	slightly longer than Japan

From the statistics and information given above, it can be said that productivity is a key factor to be able to survive and compete in global shipbuilding market. Japan has more labour cost than its competitors, but still manages to lead the market with its high productivity profile. On the other hand, China rises with its low labour cost. Therefore, several approaches and activities are tried to be carried out in order to increase the efficiencies shipyard facilities.

It can be clearly seen that concept of lean manufacturing is becoming more important in shipbuilding industry in recent years. Lowering the costs of production and increasing the efficiency of production process is the key to hold a strong position in a highly competitive global market. With China rising with low labour costs, it is becoming crucial for other actors in industry to improve their productivity levels with using state-of-the-art facilities and proper planning and organization.

The concept of lean manufacturing and its applications in shipbuilding industry are described in the "State of the art" section in details. Use of simulation and optimization approaches will also be discussed under the same section with examples of applications done so far. It is proper to say that applications in these fields of knowledge are getting wider and wider in the industry with the help of developing technologies and software.

1.4. State of the art

1.4.1. Lean manufacturing in ship building

Before going through lean production in shipbuilding, the concept should be clarified. Lean manufacturing is also known as lean production or referred as "lean" in most cases. It can be briefly described as a systematic approach towards eliminating waste in a manufacturing process. It should be noted that the principles are firstly introduced by Toyota. Toyota developed a system, which is referred as Toyota Production System in order to become competitive in the market. It is clearly the reason behind the current success of Toyota and proves that applying lean approach results in significant improvements.

Waste, in the sense of lean manufacturing, is defined as any action or process that does not add any value to the product from the point of view of customer. The main idea is improving efficiency, productivity and profits by reducing wastes.

There are seven major wastes, which are also addressed as "Seven Deadly Wastes" in lean approach. The wastes and their brief explanations are given in Table 2 below.

Table 2: 7 major wastes in lean point of view

Waste	**Explanation**
Overproduction	Producing an item when it is not necessary
Waiting	Time spent by an item while waiting for the next value-adding production step
Transport	Unnecessary movements of materials and items
Motion	Unnecessary movement of people
Over processing	Processing more than customer requirements
Inventory	Excessive items or process that are not needed currently
Defects	Final goods that are useless or requiring rework

The goal of any producer should be reducing the major wastes that are stated above. Lean approaches and tools are used in several industries that are mainly based on mass production. However, the applications are also seen in shipbuilding industry as well. It can be said that it is getting more and more popular in recent years due to the severe competition in industry. Japan and South Korea made great leap and ensured their leading position in the global market. Several articles are published regarding the subject in the recent years that show the trend of industry. It is fair to suggest that lean manufacturing will be used much more in coming years.

Kolic et al (2013) published a scientific paper on the subject of lean manufacturing methodology for shipyards. It is stated that in recent years, parallel with improving manufacturing technologies, shipyards also made improvement in technologies. However, they could not succeed and reach a competitive level as some major shipyards like Ishiwajima-Harima Heavy Industries or Kawasaki in Japan. The reason behind this reality is lack of applying lean methodological changes in production processes. In most facilities, concepts such as group technology or design for production are neglected. It is suggested in the article that applying some lean tools will significantly reduce man-hours. In order to support the idea, a case study was carried out for a panel-block assembly process. The case study covers integration of one piece flow principle and levelled production. In the later stage of the study, Monte-Carlo methodology, which is also used by the DES methodology that is applied in our work, is used for man-hour estimation and reducing the risk in decision making.

Lang et al (2001) suggested that applying lean manufacturing might improve a shipyard and give a competitive advantage in the market. They conducted a case study in hypothetical shipyard. The work covers the estimation of current performance of the shipyard, study and search for lean application opportunities in shipbuilding. Authors state the fact that shipbuilding process is not like other industries due to the fact that a ship cannot be built entirely on an assembly line and ships are generally produced in fewer numbers. Still it is possible to identify the wastes and eliminate them. Some identified wastes are excessive inventory, excessive waiting time and excess motion of materials which are also the subject of our work. Reducing these wastes is believed to be significantly effective in increasing the efficiency of the production process. A new shipyard layout is suggested with a straight-line flow. Later the flow is evaluated in every possible point of view with their relationships with other steps of production. A new purchasing approach with the aim of achieving Just In Time

(JIT) principle is also implemented. The results prove that productivity is significantly improved with lean manufacturing developments.

Liker and Lamb (2002) also underline the success of Toyota Production System and suggests that it pioneered several approaches like just in time and one-piece flow. It can also be accepted as the basis of lean manufacturing. The purpose of the article is to present a guideline for lean applications in shipbuilding. The paper is really helpful for understanding the basics of lean manufacturing, the concept of waste and how it can be recognized. Toyota Production System is further discussed in details based on examples from shipbuilding process. Firstly, JIT is evaluated. Since it works best with a one-piece flow, it is an approach that is applicable mainly in mass production. However, it is stated that some leading shipyards in Japan managed to implement it in ship construction. Obtaining stability in shipyard processes is also discussed in the article. Lean approaches are offered as solution to increase lacking stability in a shipyard such standardizing the work or possessing an efficient workplace layout. Having a proper supply chain is also underlined as a key factor affecting the performance of the shipyard as having an excess amount of materials waiting to be processed or lacking the required amount of material directly result in problem in the workflow.

Phogat (2013) suggests that applicability of lean principles is limited due to the nature of ship production. However, it also states that, especially Japanese shipyards that managed to implement lean principles to shipbuilding improved their productivity dramatically, 150% in 30 years from 1965 to 1995. Also in case of applying lean manufacturing in shipbuilding, the productivity could improve around 45% and building time 85%. Information related with lean applications and trends in Norway and United States is also given. It can be clearly seen that lean manufacturing applications are getting more important and it is a trending field of work in recent years due to the fact that lean production is considered to be key to become competitive in the market.

Literature search in the field of lean manufacturing in shipbuilding industry showed that in the recent years lean applications are becoming widely used. In addition, the sources were helpful to identify the wastes that are needed to be improved in a shipyard. The design variables and objective functions are defined based on the knowledge and point of view that is gained through the articles.

1.4.2. Simulation in shipbuilding

Simulation is a common tool that is used for process planning in several industries. It enables us to define and evaluate various alternatives before actually setting up the system. The use of simulation is getting popular in shipbuilding industry as well in recent years. However, despite the fact that it is becoming widely used nowadays, it should be noted that currently the applications are limited especially compared to other industries such as automotive which is a pioneer industry in most of the technological advances.

The concept of Discrete Event Simulation (DES) will be described in details in the related section. Nevertheless, before further discussion of this document, it is necessary to give a brief description. In general sense, simulation is the imitation of a real-life process by developing a model. The key characteristics of the system and flow of operations are represented in the simulation. DES is a type of simulation in which the operation is modelled as discrete sequence of events. Each step occurs at an instant of time and causes a change in system. It is assumed that no change takes place between two consecutive events. Since a DES tool is used in our work, current applications in the industry are investigated regarding simulation and DES.

Cha and Roh (2010) states that recently simulation is becoming a more common tool in shipbuilding industry with the purpose of planning the processes. Shipyards develop in-house systems or use commercial tools for simulation. Both cases have some disadvantages. Developing an in-house tool requires time and effort while commercial tools lack adaptability to requirements of shipbuilding process. Authors propose a simulation framework by combining discrete event and discrete time simulation. A sample case is selected which is block erection process for applying the simulation framework and evaluating. In the article by Cha, Roh and Lee (2010) the case is described in a more detailed way.

Fernandez and Alonso (2015) underlines the fact that as a consequence of increasing use of CAD systems in the fields design and production virtual reality is becoming more applicable. Recent developments in hardware and software result in ability to process huge amount of data such as complex ship models. Therefore applications in shipbuilding increase like several industries. Different areas of usage of virtual reality are described in the article. Besides these information, a new virtual reality and design review tool, FVIEWER developed by SENER is described. The features and advantages of the software are explained within the context.

Despite the fact that simulation is not the main objective of our work, it is important to see the state-of-the-art of the shipbuilding industry. It can be said that simulation is a powerful tool for making proper process planning because of the fact that it saves time and reduces the mistakes that might be done while setting up a production layout.

1.4.3. Optimization in ship building

Optimization will be explained in details in the related section. However, in a brief way, it can be defined as finding the best alternative among possible alternatives based on some criteria in a systematic way. It is another concept that is important in shipbuilding as well. Throughout the design of a ship, there are several goals that are desired to be achieved such as reducing the resistance of the hull, reducing the weight or obtaining the best performance. Designers and engineers work on several alternatives and try to find the best possible outcome. Optimization is also crucial during the production process as well. Reducing the cost and minimizing the time of construction are some of the important key parameters that affect the competitiveness of a shipyard. It can be considered as a tool for achieving lean manufacturing goals that are already mentioned before. It can be said that optimization of shipbuilding process is a subject that is rising in the industry in recent years as well.

Park et al (2006) present a case in their article, an academy-industry collaboration project on improving productivity in steel stockyard management in South Korean Hyundai Heavy Industries (HHI) shipyard. HHI is stated to require productivity improvement due to increasing competition in the industry amongst Japan and China. Within the context of the work, ways of improving stockyard operations are researched, current situation of stockyard is evaluated and operational difficulties are assessed. A simulator is also developed due to the fact that currently decision making process is dependent on the experience of the manager.

The importance of steel stockyard is emphasized by Park et al (2006). It is explained that so far, stockyard operations are neglected. However, it has interaction with other departments and therefore directly affects the whole process and any problem occurring in stockyard directly causes consequences in other operations. After realization of the importance of steel stockyard management, Hyundai launched a project called Hyundai steel stockyard optimization project (HYSSOP) to search for approaches to improve. Initially the steel

stockyard process is described in the paper. Later two difficulties are addressed related with the operations which are long stay of steel plates and stock management method. The reasons for these difficulties are mainly great variety of steel plate delivery times and slight construction schedule changes which directly affect the stock time of plates. In HHI, the stockyard operations are controlled by an operations manager and the decision making process relies on experience mainly. HYSSOP developed a simulator to assist the process. The simulator takes into account current schedules, workloads, materials and schedule changes. The main aim of the shipyard is minimizing the long stay of materials. It is also stated that currently some Japanese shipyards reduced this duration to 3-4 days by supplying steel plates to the shipyard from stockyard in a JIT manner.

Caprace et al (2013) initially state the importance of space within a shipyard since space is generally considered as a resource and requires proper planning. It is stated that ship production is a highly complex process and includes several disciplines and operations that should be carried out simultaneously. Therefore, production planning is a key point. However planning is generally dependent on personal experience and ideas.

After presenting the current condition of industry in the related field, Caprace et al (2013) introduces a tool that may assist planners to use valuable space in an effective and efficient way. Space allocation is an important issue. It is a dynamic process and it is defined as a time consuming and difficult task to accomplish since space allocation for one block directly affects other blocks and other operations. Some changes or delays in schedule may also occur and quick response is required in such cases. Thus, a tool is necessary to assist planners to adjust the production to the current case. It is also asserted that shipbuilding has different characteristics and relatively more complex from the point of view of certain approaches such as three-dimensional bin packing problem (3D-BBP). It is time consuming to find optimum solutions with current approaches; therefore, a new tool is developed. An optimization task is carried out by defining design variables and objective functions. The problem is mathematically modelled and algorithm is developed. Later a case study is done.

In the article by Caprace et al (2013) the objective defined by the shipyard is maximizing the number of blocks produced in a given surface in a certain amount of time. This objective is referred as Space and Time Allocation (STA) in the article.

Article of Bair et al (2006), proposes an approach that integrates simulation and optimization. A shipbuilding workshop is simulated initially and then productivity is improved by using

optimization. For the optimization, genetic algorithm is used. As a concept that is also crucial in our work, it will be described in details in the relevant section. The article suggests promising results for further applications of coupling simulation and optimization tools. The objective stated in the article is minimizing the total production time. Despite the fact that it is not directly mentioned as a lean manufacturing approach, minimizing the wastes such as waiting plays a crucial part in optimization process while finding best solution for layout arrangement.

The articles mentioned are core of the work that is done. The current state-of-the-art of the shipbuilding industry shows that lean production, simulation and optimization concepts are becoming important and several applications are already present. It is also clear that these approaches directly yield improvements in production process and create great impact on productivity of shipyards.

1.5. Gap

Lean manufacturing is a relatively new subject for shipbuilding industry compared to others such as automotive. This situation is mainly resulted from the fact that shipbuilding is a relatively complex process that covers different disciplines. In addition, automation is not applicable for shipbuilding due to the fact that ships are not produced as mass production generally and construction may not be done in a sense of line production unlike a car.

Similar to lean manufacturing, simulation and optimization are also newly rising subjects in shipbuilding industry. The reasons behind this fact may be justified with the difficulties of applying these approaches. It can be stated that coupling these two concepts is rather difficult when the stochastic nature of the simulation is considered. It is not easy to obtain convergence in simulation. In addition to that, running an optimization task based on a stochastic simulation requires great amount of sophisticated work, computation capacity and time. Furthermore when the objective of the study is developing any layout, parametric simulations are required which enables to generate dynamically new layouts. Therefore, despite the advantages offered, it is clear that difficult nature of DES optimization is the main reason behind the gap in the industry.

However, it can be understood from the literature search and state-of-the-art of the industry that concept of lean manufacturing besides simulation and optimization raised the interest of shipbuilding industry as well and applications are increasing in the recent years. It is clear that simulation and optimization are powerful tools to improve productivity in a shipyard. These approaches are required to possess a sustainable and competitive position in the market.

Our work proposes a relatively new approach. It is seen that simulation and optimization are being used in the shipbuilding industry. There are several examples that are already mentioned in the previous section. However, simulation and optimization tools are not integrated for a shipbuilding process, except one example. Bair et al (2015) proposes a similar approach and yields promising results, which prove that coupling these two approaches results in more efficient and effective results.

The thesis also brings another type of application unlikely what is presented by Bair et al (2015). Within the context of the thesis, two commercial tools are used. By using already existing commercial software, great amount of time and effort is saved. As it is seen in several examples, generally in-house developed systems are used for required tasks. It is fair to say that developing new software to solve some unique problems is a necessity in some cases since commercial tools may not fulfil the requirements of shipbuilding processes. However, it results in consumption of long time and high costs. Therefore, using commercial software to model, simulate and optimize some tasks is a far more efficient solution.

To sum up, the thesis focuses on coupling two commercial software, QUEST for simulation and MODEFRONTIER for optimization, in order to solve real-time engineering problem and optimize a steel stockyard of a shipyard.

1.6. Objective

The main objective of the thesis is to offer a new approach of applying simulation and optimization concepts for improving the efficiency in a shipbuilding process. Despite the fact that both concepts are becoming popular in recent years, integration is still not prevalent.

Another goal of the thesis is to optimize a steel stockyard in a shipyard, which is based on a real layout. The shipyard that is the subject of the application is Atlantico Sul which is a

Brazilian company. The facilities were already modelled and based on what is already done, steel stockyard operations are simulated.

The simulation is coupled with an optimization tool in order to carry out further tasks. A multi-objective optimization is set up to minimize the area required and minimize the Work In Progress (WIP). Thus, the cost will be minimized.

Optimization steps are repeated with certain modifications in simulation as well in order to find optimum solutions for shipyards with different capacities of steel processing.

The thesis has the purpose to improve what has already been done in this field, present a new approach with correlated use of two commercial tools and hopefully be a basis for further works and developments.

2. METHODOLOGY

2.1. Steel Stockyard and Activities

Steel stockyards are generally ignored while planning the shipbuilding process. Most of the time managers or planners do not pay attention to operations taking place in a stockyard and mainly focus on other operations such as part fabrication or assembly with the aim of improving productivity. However, it should be stated that steel stockyard planning and performance directly affect other stages of production and activities of several departments of shipyard as purchasing, manufacturing or even design. A delay in feeding the latter stages of steel processing may cause severe problems eventually leading to loss of time and money. Whereas excess amount of steel waiting in a stockyard for further process is another problem and causes loss of productivity from lean point of view (WIP). Therefore, steel stockyard planning and management requires attention and proper planning just like any other process taking place in a shipyard.

Briefly, steel stockyard may be described as the place where all steel materials arrive initially. Later the steel plates are arranged, put into an order and stored until further steps of production based on a plan.Then they are transported for further steel fabrication processes.

The subject of the thesis is the steel stockyard of Atlantico Sul. Atlantico Sul is a shipyard which is placed in city of Ipojuca, in the state of Pernambuco, Northeast of Brazil. It has the capacity of processing 160.000 tons of steel per year. The shipyard is based on 1.62 million m^2 area and covers all necessary facilities to construct and repair ships and offshore platforms. Figure 4 shows the layout of the whole shipyard.

Figure 4: Atlantico Sul shipyard layout, Available from, www.estaleiroatlanticosul.com.br

As it is already stated, optimization is done in steel stockyard.

It can be seen from the Figure 4 that stockyard is open and has three sections which are identified as A, B and C from right to left. Sections A and B are slightly bigger than C. Sections A and B consist of 26 rows of lots each, while section C has 21 rows of lots. In each row, there are 3 lots. The term "lot" is used for the place to store steel plates as grouped based on certain parameters, which are dimensions, thickness and steel grade.

Each section has a crane. These cranes are used for moving steel plates arriving initially as a pile to the corresponding lot. Same cranes are also used for transporting plates that are placed in lots based on the requirement of steel processing stage where they will be shot blasted, primed, cut and shaped in later stages of production.

The arriving steel plates are determined based on the design of the ship. Production plan is prepared and based on that purchasing is done. During the purchasing, delivery lead-time should be taken into consideration. The steel plates are purchased from a variety of suppliers and in each case proper planning should be done. While making the purchase, several factors are taken into account such as amount of steel that is already stocked in the shipyard or

progress in production stage. The ordered steel plates arrive to the stockyard and stocked for latter stages. Finally, they are transported according to the production plan.

Several steel plates with different dimensions, thickness and grades are required for building a ship. Each section of ship requires different qualities and regulations are set by class societies as well. In our application, steel plates that are required for building a tanker is taken into account. The list of required steel plates for construction of a tanker is given in Appendix A.

2.2. Simulation

2.2.1. Discrete event simulation

Creating a simulation before actually trying several design alternatives enables to obtain results related with process planning in a shorter period. In addition, it assists planners to act quickly to sudden changes in system or any problem.

Discrete Event Simulation (DES) is an approach in the field of simulation. Fishman (2001) states that unlike a continuous process that evolves through time, events that are simulated are assumed to be taking place step by step. Each event occurs at an instant of time and causes a change in the system. It can be said that simulation jumps from one event to the next without any change of state in the system between two events.

DES is applicable in shipbuilding process like many other production processes. From the operations in a steel stockyard to assembly of blocks, each stage of production can be described as events taking place step by step without any change between stages occurring. There are some examples with DES applications in shipbuilding and it is clearly a valid approach to simulate shipbuilding with DES.

DES offers some important advantages as well. One of the most important aspects of DES is that it enables the use of stochastic modelling. Concept of stochastic simulation is also introduced in the title of the thesis. Thus, it should be explained in order to properly explain the advantage of using it.

A stochastic simulation is a type of simulation that takes into account the random change of variables with certain probability distribution. The simulation is repeated until sufficient data

is gathered. Finally, the distribution of the outputs is obtained and the most possible outcome can be seen and evaluated from the results. For all production processes, the factors such as human or equipment cause some uncertainty which result in a probability distribution of variables. Therefore, it is beneficial to run stochastic simulation in order to reach precise results.

Despite the fact that DES is used in the thesis, it is important to note that simulation is not stochastic. Due to the limited time of work, simulation is based on constant values in variables such as crane and conveyor speeds. However, the simulation is useful and open for further developments and turning it into a stochastic simulation might be the next future step to improve the work that is presented.

Besides being a stochastic simulation, DES has some other advantages. The results are useful to identify any bottlenecks in the process and it is easy to apply several approaches in order to obtain best results.

2.2.2. Software, basic elements and modelling

The software used for creating DES is QUEST that is developed by Dassault systems. It is a tool that enables 3D modelling of a production environment, simulation and analysing of the flow. It is flexible and thus it is applicable in shipbuilding process as well. It also offers visualization of the model, which is beneficial while modelling and analysing the results.

Another advantage of QUEST is that it is capable of exporting and importing data from other sources. It enables coupling with other tools. It is crucial in the thesis since main goal is to couple QUEST with an optimization tool to run the trials.

Before further describing how the simulation is made and introducing several assumptions and approximations, it is necessary to briefly inform how the model is prepared in QUEST. QUEST has a user-friendly interface to create and define every single detail of production process. It is also capable of running in batch code mode which saves time but requires more knowledge and experience.

In QUEST, parts are the entities that move through the system. In our case, parts are steel plates. To simulate the process, those parts should be created, carried and undergo processing.

Therefore, some basic elements are needed to be created as well. The elements are also shown in Figure 5 below.

Source is the element that is needed to create parts. It is shown with 1 in Figure 5. A source is defined for the arrival of new steel plates to the shipyard with a defined frequency.

Second step is to define buffers, which are lots, where the parts are stored and wait for the next move. Buffers are represented with 2 in figure. The steel plates initially arriving randomly to shipyard are stored as piles in lots before they are moved for further processing based on the request from the processing unit. Therefore, buffers are created based on the model under three sections. Also an additional initial buffer is defined to hold the steel plates created in the source and transfer the held steel plates once a week to the main buffers in each section.

In our case, we are not considering any process related with steel plates. Only the storage and transportation from initial arrival to the machine is simulated. Therefore, instead of a machine, a sink is defined in the model. Sink is an element in software, where the parts leave the system which can be also stated as “destroyed”. Sink has no output. It is placed in the building shown with 3 in Figure 5.

Cranes are needed to move plates from their initial position, source, to storage units, buffers, and from buffers to conveyors. Therefore, they are created in QUEST as well. In our case, cranes have hoists. Cranes are not represented in the simulation. However the hoists are seen and numbered with 4 in the figure.

Finally, three conveyors are created to move plates to the sink. Cranes transfer plates from buffers to conveyors. To accomplish this move, decision points are defined on conveyors as well. Each section has its own conveyor and it can be seen from the figure numbered as 5.

2.2.3. Simulation

The process taking place in the steel stockyard is modelled in QUEST. The model obtained is presented below in Figure 5. Further explanations regarding how the events take place and the sequence are given below as well. Also some approximations and assumptions that are made while simulating the system are also explained.

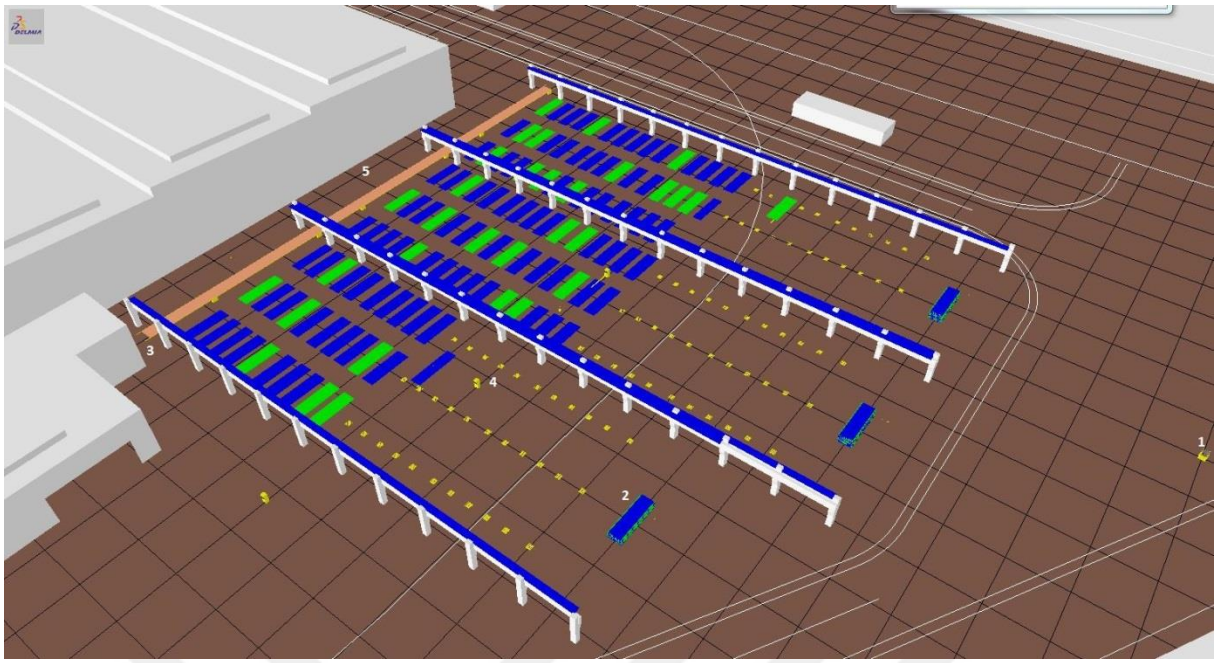


Figure 5: Steel stockyard model in QUEST

Process begins with the creation of steel plates in the source. Created plates arrive to an initial buffer where they are transferred into the main buffers in each section equally. Each week, the plates that are waiting in the initial buffer are transferred to the active sections. This process is modelled in this way due to the fact that number of arriving plates to the shipyard in a week is selected as a design variable. The selection of the design variable and how it is implemented to MODEFRONTIER will be explained in the next sections. However, it should be noted that the frequency of arrival of steel plates is assumed to be continuous and constant. An initial buffer is created in the system. Thus plates are not transferred continuously to the lots. Instead, they are stored initially in the main buffers in each section. Every week the stored plates are moved to lots. This way, the reality is implemented in simulation modelling the arrival of a steel plate pile each week with the defined fraction of plates within. This is an assumption since in reality, steel plates are purchased from a variety of suppliers and the arrival interval is not constant.

In order to solve the problem that rises due to the fact that one week of waiting time is required to have distributed steel plates in the sections, an initial stock is defined in the main buffer with the number of steel plates that will be gathered within a week. This pile has the same fraction of steel plate types and is randomly sorted as well.

As it is stated the number of steel plates in a pile arriving to the stockyard is defined as a design variable, therefore it changes throughout the optimization process to obtain best outcome. It will be explained in the related section in details.

The distribution of steel plates with respect to the dimensions, thickness and grade is always the same. It should be noted that steel plate information is taken based on real data of required steel for building a tanker. However, the number of different steel plate types is narrowed down to 10 from a value of almost 150. Therefore, most frequent 10 types are taken into consideration while defining fractions and distribution of each one. Ones that have more than 1% as number of plates in overall distribution are taken for further progress. Different types of steel plates regarding their dimensions, thickness and grade which are presented in the table below.

Table 3: Thickness, dimension and weight values of steel plates that are used in simulation

Plate dimensions (mm)			**Quantity**	**Weight (tonnes)**	**Weight percentage**	**Number of plate percentage**
Thickness	**Width**	**Length**				
Steel grade A						
11,0	2440	9200	224	434,198	9,037	12,037
12,0	2200	9200	223	425,174	8,849	11,983
12,0	2440	9200	163	344,680	7,174	8,759
12,5	2440	9200	190	418,515	8,711	10,210
13,0	2750	9200	120	309,823	6,448	6,448
14,0	2440	9200	206	514,729	10,713	11,069
15,0	2750	9200	144	428,987	8,929	7,738
Steel grade AH36						
15,0	2750	9200	170	512,401	10,665	9,135
16,0	2750	9200	158	502,074	10,450	8,490
17,5	2750	9200	263	914,080	19,025	14,132
			1861	4804,661	100,000	100,000

This is also another assumption made in simulation. In real life, it is impossible to group only steel plates with same dimension, thickness and grade due to the fact that as stated almost 150 different combinations of these values exist in steel plate list which is given in Appendix A. However, piling/unpiling operation is not integrated to the model yet. Piling/unpiling operation enables the cranes to reach to the steel plate that is stored not on the top of the pile. It is a function that replaces plates to another empty lot and reaches to the requested one. In case the plate requested is beneath several other plates, crane relocates plates on top to another empty lot and feeds the conveyor with the requested plate. As it is stated, this

operation is not implemented yet. This fact is believed to cause problems in the working of simulation. Therefore, in order to prevent any crashes in the system, number of plate types are reduced to 10 and same type of plates are stored in the same lot. Thus, any crash in the simulation is prevented.

After one week is completed in the simulation, the pile formed in the initial buffer is separated into three sections that can be seen from figure 5 as well. It should be noted that, in some iterations that will be conducted with MODEFRONTIER, some sections will be completely deactivated. Thus, it will be possible to see whether more efficient results can be obtained when one or two of the sections are not used for storage at all. In these cases, pile of arriving steel plates will be divided into only active section or sections.

Consequently, cranes in each section begin to place steel plates in storage lots based on their types. There is one crane in each section working to place plates from the pile to the lots and from lots to the conveyor. As mentioned before plates will be grouped based on their type which is in our case 10 different types. Each plate will be taken and put by the cranes to the respective lot. The maximum number of steel plates in a lot is also set as a design variable since it directly affects the storage area and Work In Progress (WIP).

Meanwhile, steel plates are requested for further progress and same cranes place requested steel plates to the conveyor for transportation. The request is continuous and distribution of requested plates is same with the distribution of arriving plates, which is the required plates for building a tanker. It should be noted that the request is continuous unlike the arrival of new piles. Another assumption made is that request is random. In a real case, the request is defined based on the production plan and it is known that which dimension, grade and thickness of steel plate are required for the next step of production. Furthermore, the purchase and arrival of new plates are planned according to this plan. However, in order to simplify the case, it is assumed that the request is constant and fraction of requested steel plates is the same with the fraction of steel plates arriving to the stockyard.

For better explaining the process, push and pull systems should be explained clearly. In QUEST, the flow of materials is defined with two options, either as pull or push. In our simulation, both are used. As it can be seen, arrival of steel plates to the shipyard is modelled as push in the system. Plates are created and transferred to initial buffers with push. Also they are placed in the lots that they are stored before further processing with push as well. Plates keep on arriving and being stored not depending on any request from the sink. On the other

hand, request of plates for further processing, which is modelled as request of sink in the simulation, is a process defined as pull in the system. The request is defined with a certain frequency.

The process can be briefly summarized and shown with the Figure 6.

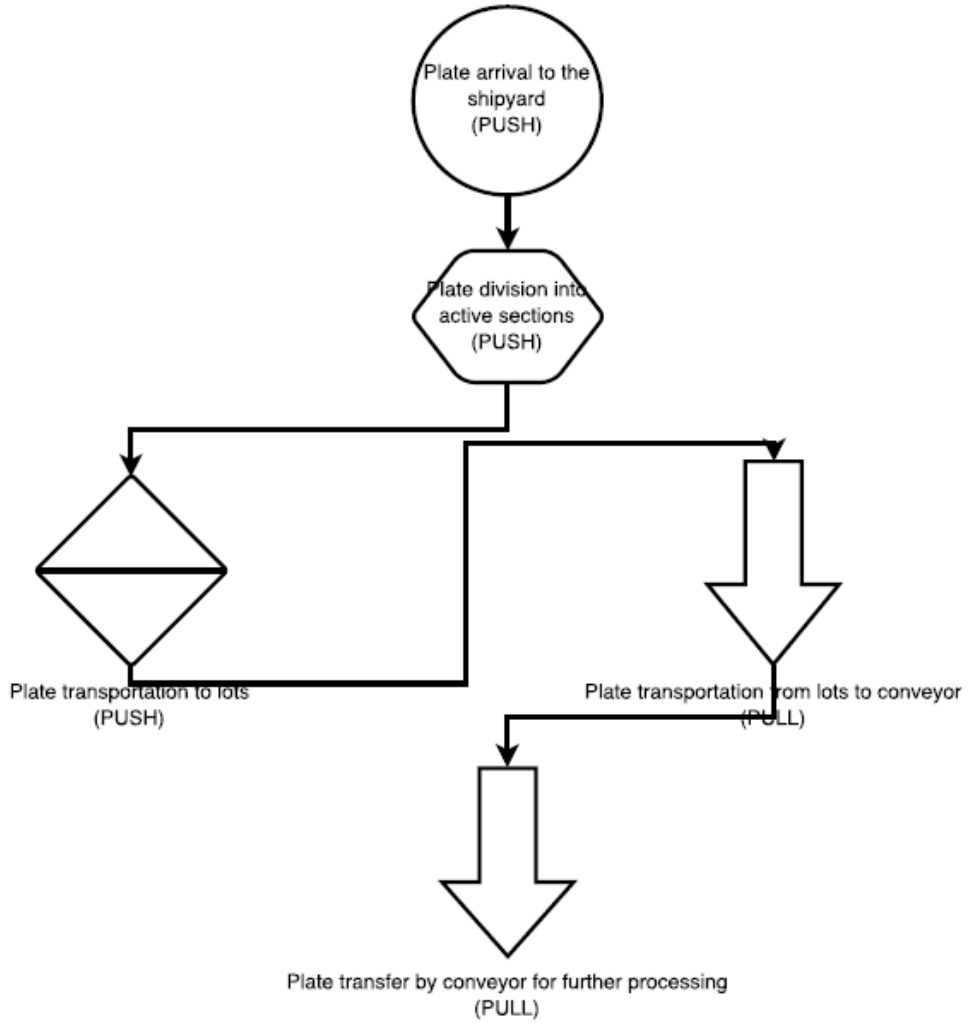


Figure 6: Stockyard operations flowchart

The simulation is set to be 1 month. This may also be stated as an assumption. Normally, it would be better to have a simulation for a longer period. However, due to the fact that longer simulation will result in longer computation time and we have limited time to perform several runs, the duration is set to be relatively short. In addition, it is clear that without having piling and unpiling function in the simulation, the simulation would crash in a long time interval definitely. Therefore, despite the fact that having a 1 month simulation time affects the outcomes of the optimization the time is set to be 1 month. In case of having a stochastic

simulation, this assumption would result in more severe problems due to the warm-up time of the simulation having more effect on the outcomes. Finally, it is assumed that 1 month of simulation time is efficient and would be enough to evaluate WIP and storage area. It can be further noted that, for future development of the work, having a simulation time of at least 1 year will give clearly better and more valid results.

As it is mentioned, it should be also reminded that there is no stochastic simulation taking place and all the values are constant due to the limited time to prepare simulation and carry out optimization. This fact affects the results dramatically since in reality it is not possible to assure that all activities occur without any change throughout the whole process. In order to have a simulation to identify and investigate entire process, it is crucial to have stochastic simulation with distributions. In this stage of work, the main objective is to have a working coupling of simulation and optimization. Thus it is assumed that the results that are obtained are relatively acceptable and give valid idea related with the subject. However, it is an important point to improve for further work which will be discussed later.

The process taking place described above is modelled in QUEST and then linked with MODEFRONTIER for optimization. The simulation is not run every time with graphics interface to save time. Instead, it is run in batch mode offered by software which enables user to obtain results faster. The codes that are essential and used in correlation with MODEFRONTIER will be described in “Coupling and Set Up” section and presented as appendices.

2.3. Optimization

2.3.1. Multi-objective optimization and genetic algorithm

As it is briefly explained before, optimization in mathematics is selection of best alternative based on pre-defined criteria among possible alternatives with a systematic approach. In real life engineering problems, in most cases there are more than one objective that is needed to be optimized. Those objectives are contradicting in most cases as well. Such applications, in which there is more than one objective defined, are called multi-objective optimization.

As it is already stated, contradicting objectives cause the result that each objective cannot be evaluated separately. Therefore, a different approach is required and all objectives are evaluated together.

In their paper, Konak, Coit and Smith (2006) state that there are two general approaches for multi-objective optimization. The first one is turning multi-objective task into single-objective with the help of methods such as weighted sum or utility theory. However, in application, it is difficult to define the importance and weights of each objective. Therefore, second approach is more common. In this approach, a set of Pareto solutions are obtained. Having several solutions help decision makers to see possible alternatives and make better calls by making trade-offs among possibilities. Figure 7 taken from the progress report by Justesen (2009) shows the multi-objective optimization process.

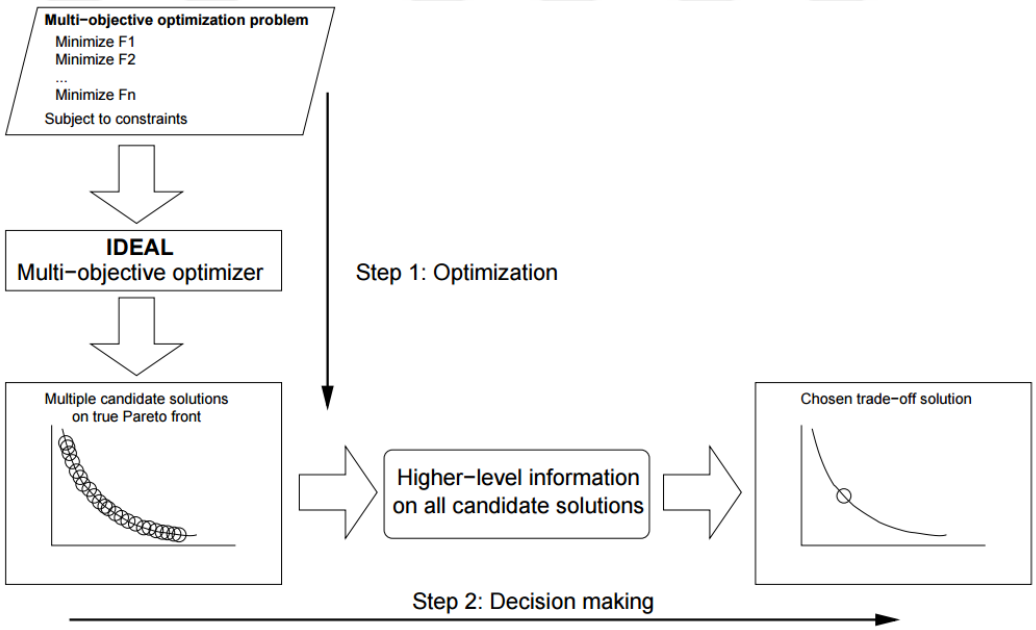


Figure 7: Multi-objective optimization process, Available from Justesen, 2009

MODEFRONTIER tool is used for running optimization tasks. The information regarding software will be presented in the next section. However, it should be noted that it enables user to create optimization set-up without deep investigation of mathematical formulation of optimization. Still it is important to understand the logic of multi-objective optimization and the concept of genetic algorithm since Multi-Objective Genetic Algorithm (MOGA) is used for optimization within the context of thesis.

As it is understood from the title of the subject, multi-objective optimization deals with several functions that are needed to be optimized, which means minimized or maximized. Justesen (2009) briefly explains the mathematical description of the multi-objective optimization.

Let x be the solution that the functions are going to be optimized. In this case, x is a vector of decision variables which can be shown as in Equation 1

$$x = (x_1, x_2, \dots, x_n)^T \tag{1}$$

where xi is bounded by lower and upper bounds which are xiL and xiU. Boundaries define the decision variable space (D). Objective functions can be denoted as fm(x) define a space (M) and creates a link from D to objective space (Z). The link is presented in Figure 8.

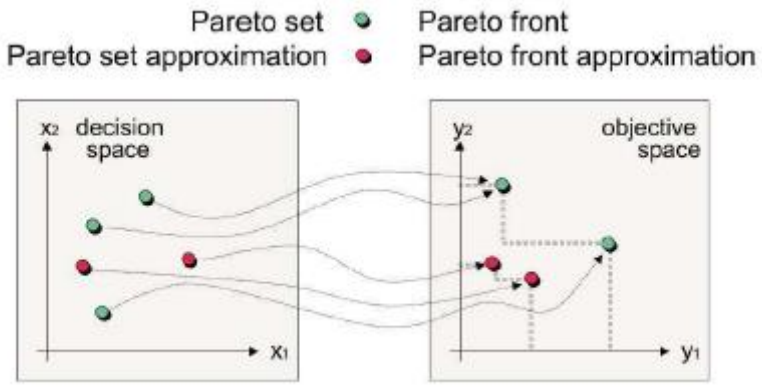


Figure 8: Decision and objective spaces, Available from Justesen, 2009

There may be several constraints. Solutions satisfying the constraints are addressed as feasible while others are unfeasible. The equations given below simply introduce the mathematical representation of a multi-objective optimization.

$$f_m(x), m = 1,2, \dots, M \tag{2}$$

$$g_j(x) \geq 0\,, j = 1,2, \dots, J \tag{3}$$

$$h_k(x) = 0\,, k = 1,2, \dots, K \tag{4}$$

$$x_i^L \leq x_i \leq x_i^U\,, i = 1,2, \dots, N \tag{5}$$

In the equations given above, M in Equation 2 corresponds to objective functions whereas J and K, shown in equations 4 and 5 respectively are constraints and N is space of decision variables.

As it seen in Figure 8, we are interested in all feasible solutions in a Pareto-optimal set instead of domination of a certain function.

Using genetic algorithm (GA), which is also known as evolutionary algorithm is an approach applicable for multi-objective optimization. MODEFRONTIER offers simple use of the approach by its user-friendly interface and simple running. Still, it is significant to understand the concept of GA to understand how the procedure occurs through the software.

GAs are algorithms that are based on up the main idea of Darwin's theory of evolution which is simply fittest one survives. Therefore, in application, solutions are referred as individuals among a population. Fitness of these individuals is evaluated based on how good is the solution they offer to the problem.

Before explaining the details of working principle of MOGA, a brief history of GA will be presented as well. Melanie (1996) introduces the history of evolutionary computation in his book. It is stated that in 1950s and 1960s several scientists studied this concept separately and considered the idea that evolution might be used an optimization tool for real life engineering issues. The core idea was evolving a population of possible solutions for a given problem with an approach inspired by evolutionary theory and its operators such as natural selection.

After contribution of several scientists to the subject throughout years, in 1960s, GAs were invented by John Holland and developed by him and his colleagues at University of Michigan. His objective was not to design GA to solve problems but to study adaptation phenomenon in nature and to search for ways to adapt it to computer systems. His works were a great leap, had a huge impact on improvement of these applications and became a key point for developing further. Therefore, it is fair to say that Holland set the foundation of works based in GAs and the terminology he used is still valid and used. As it was already shared in "State of the art" section, nowadays GAs are used for optimization tasks in several fields including shipbuilding as well.

Since MOGA is used in our work, it should be described in details. To start with, the working principle of MOGA is briefly summarized in Figure 9 below.

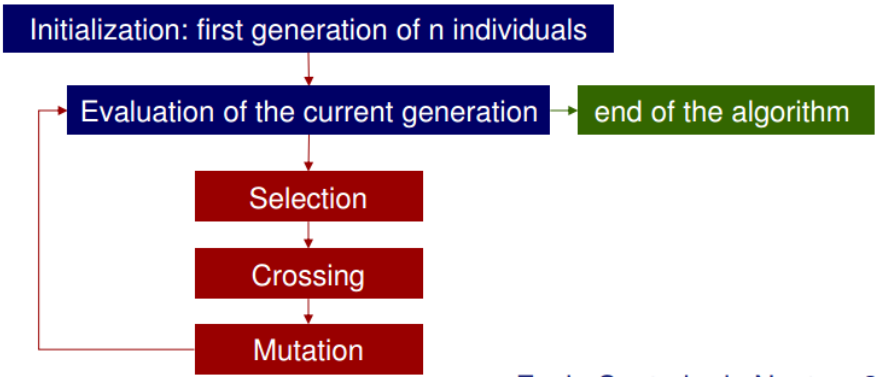


Figure 9: Working principle of MOGA

MODEFRONTIER has MOGA-II present in the software as a possible algorithm. It has four important operators, which are mutation, selection, elitism and crossover. Algorithm use each of these operators based on predefined probabilities of occurrence. Operators are needed to be explained briefly, in order to better understand the working principle of algorithm.

Crossing is the combination of two individuals to create new individuals. Crossover occurs with a predefined probability in random points. Figure 10 below may be used to better represent the case and some different approaches while performing crossing.

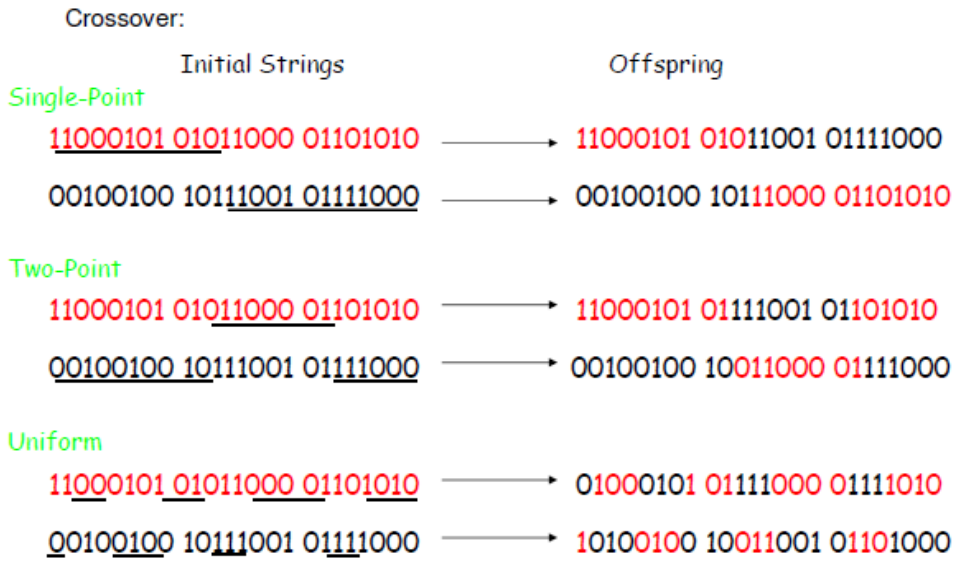


Figure 10: Crossing

Mutation is the random modification of an individual. It also occurs with a predefined probability. As it can be seen from Figure 9, it is generally applied after crossing. Figure 11 shows a simple example regarding mutation with a mutation probability of 0.05.

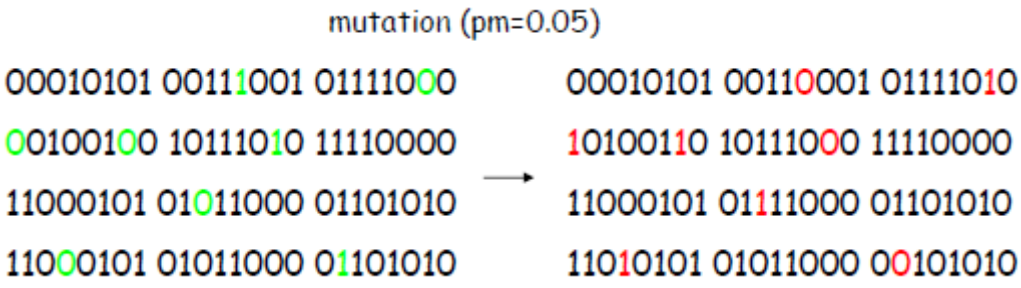


Figure 11: Mutation

Whereas selection defines the frequency of a parameter, remain constant without any alteration throughout the process. Elitism operator maintains good results and ensures that each new result is as good as the previous one or better than it.

According to Justesen (2009) GA is a stochastic metaheuristic based on probabilistic operators. Therefore, unlike deterministic operators, each run may result in different solutions. It also uses a set of solutions which results in combination of several solutions. GA is a useful approach in multi-objective optimization. It is applicable in high dimensional objective space cases which means that not linked objectives can be evaluated with a trade-off among possibilities.

2.3.2. Software

MODEFRONTIER is a tool developed by Esteceo and can be briefly described as a platform that enables integration of optimization, automation of design processes and analytic decision making providing coupling with other engineering tools for several various disciplines.

It has a simple interface, which enables users to create optimization loops to solve the problems. Software includes several single- and multi-objective optimization approaches. The loop that is set-up for the thesis will be presented in the later sections. A quick review of the software will be given briefly in this section.

As it is already stated, software has a rather simple graphic interface that enables entering all related information and coupling with other required software. Graphics interface enables users to create set-ups as a logical and simple workflow. Simple modules are used to define input variables, decide algorithm or add constraints in case they are required. Figure 12 below

shows a simple application in MODEFRONTIER with basic modules. It should be noted that it is an example and it is not the workflow used within the work.

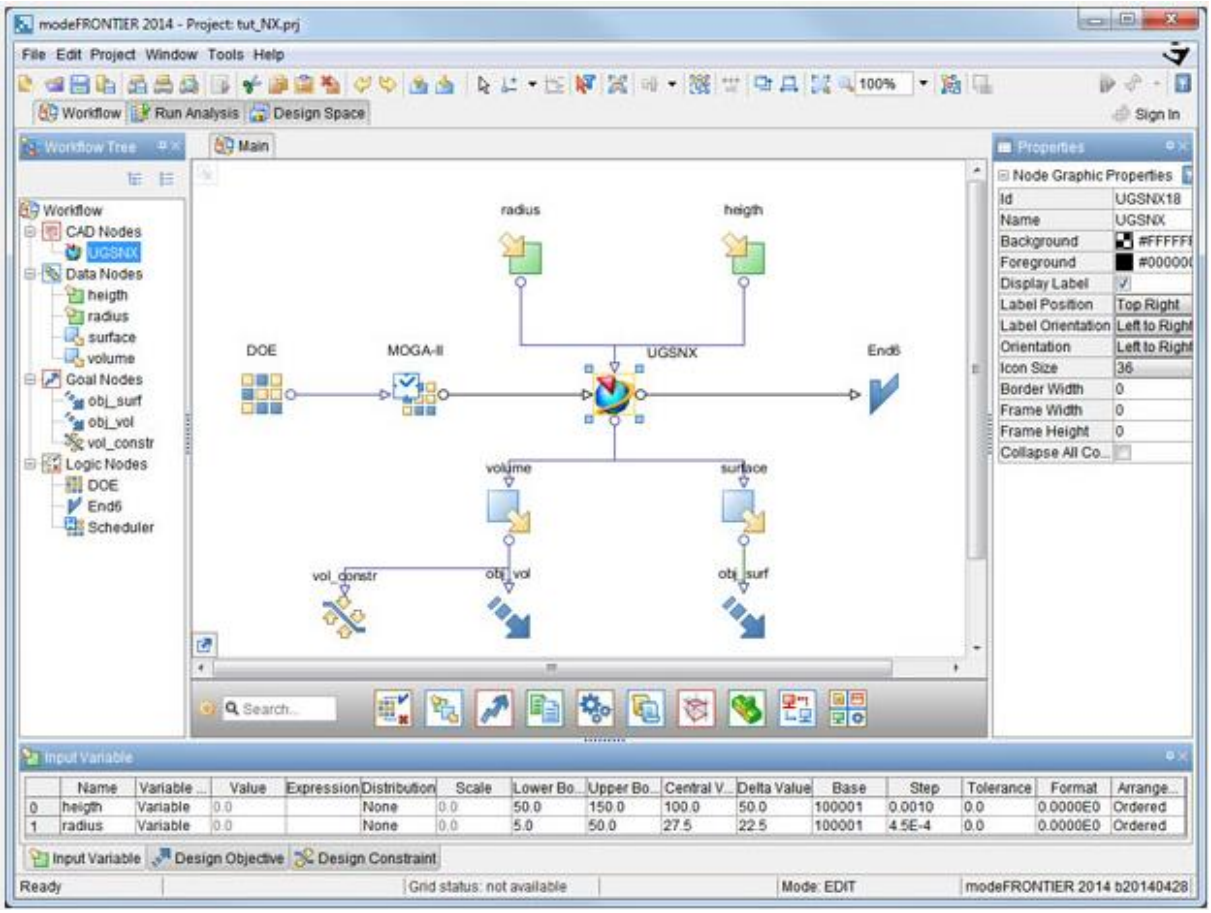


Figure 12: MODEFRONTIER view, Available from http://www.deskeng.com/de/esteco-unveils-modefrontier-2014/

Use of each module and how they are set up will be described in details in the section "Coupling and Set-Up". A brief introduction is sufficient in this stage.

Design variables are defined by using the module "Input Variable" and files that contain information are linked to the software by using "Input Files".

Design of experiments (DOE) module is used to decide initial variables that will be used in the process. There are several options in this case. Throughout the optimization process, new combinations of design variables will be formed by software based on the algorithm used.

Scheduler module enables to select the algorithm that will be used. Within the context of the thesis, MOGA-II is selected. DOS Batch script is the key point of the software and can be identified as the black box that processes inputs into outputs. "Output File" and "Output

Variable" modules are used to obtain results. Whether minimizing or maximizing is the objective of the process is defined by "Design Objective". Any required constraint is defined by "Constrain" module as well.

As it is stated, mentioned modules will be explained in details about how they are defined and connected later. It should also be noted that MODEFRONTIER offers tools to see results clear with graphs and a Pareto diagram may be obtained.

2.3.3. Optimization model

Before further explaining how the optimization set-up is prepared and coupled with QUEST, the optimization task that is subject of the thesis and how the related parameters are defined should be described. This section covers the definition of objective functions, design variables and constraints used in our application.

Design variables and objective functions are presented in Figure 13 with their limits and constraints stated below them.

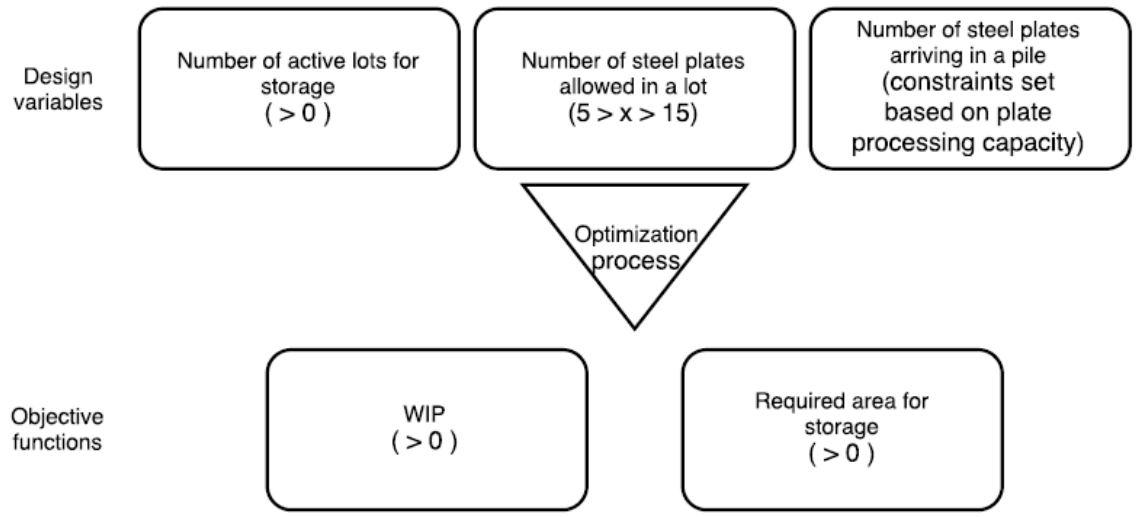


Figure 13: Design variables, objective functions and constraints

2.3.3.1. Objective functions

For better understanding the selection of objectives and design variables, it is thought to be better to start with objective functions. Later, design variables will be explained based on them.

As it is stated, a multi-objective optimization set-up is prepared for the task. Therefore, two objectives that are contradicting with each other are needed to be defined to have a valid and useful Pareto result in the end. The definition of objective functions is done based on the lean manufacturing principles and ideas derived from the literature search that presented.

The main aim of any optimization in a production facility is to increase productivity and thus reduce the cost. Therefore, it can be stated that main objective of two objective functions is reducing cost ultimately.

Two objectives that are defined in the thesis regarding the work are minimizing the Work-In-Progress (WIP) and minimizing the area of storage. The selection of these objectives is explained in this section. Their application and representation within the task will be stated in the section "Coupling and Set-Up".

WIP

To start with, work-in-progress is a crucial concept. Work-in-progress (WIP), or also known as work-in-process, is a term used to define partially finished items that are being processed or just waiting for the next processing stage in a queue or being stored in a buffer. It is taken into account in a company's balance sheet. Normally WIP excludes finished goods and raw materials since they are out of production process. The term WIP is used for steel plates that are grouped and stored in steel stockyard in our case. Despite the fact that the plates are currently in the state of raw material, they are considered to be in production level.

Most of the production facilities keep track of WIP throughout whole process. The aim is to minimize WIP in an optimal case. Just-in-time (JIT) is one of the most well-known production concepts regarding the minimization of WIP. Briefly, JIT is based on the idea that the next step of production should be taken into account at each step of production and defining the sequence of processes regarding this overall view. It aims to equalize demand from a production process with supply of required items in that stage of production. The main

purpose of JIT is producing perfect quality of exactly demanded goods in the demanded time without producing wastes.

Minimizing WIP is an essential task in production management. It creates the necessity of having storage space for the items. It also bounds capital that otherwise may be used for other investments. Long waiting time may also lead deterioration of goods that are stored and eventually results in loss of money and serious problems in production.

From lean manufacturing point of view, it is also possible to define wastes that are related with WIP. It is directly related with waiting and inventory which are 2 of "7 deadly wastes" that are already described as waiting of an item for the next stage of production and excessive amount of items that are not need in the present time are clear wastes.

In our case, minimizing WIP is selected as one of the objective functions. Park et al (2006) also states that long stay of steel plates in a stockyard is a major difficulty to fix. Steel plates arriving to the shipyard wait for processing and are stocked in steel stockyard which is a production stage that adds no value to the product. Also purchasing steel plates that will not be used in a short period is bounding great amount of capital. It also increases the risk that steel plates get deteriorate which will result in the necessity of further preparation of steel plates for production stages. Therefore minimizing WIP is selected as an objective function.

A simple approach is followed to estimate and calculate the WIP which is mainly based on the idea that WIP is integration of weight of the materials that are waiting for the further processing over the period of time they are stored. A code is implemented to make the calculation. The code is given as Appendix B.

The working principle of the code can be described as follows. First of all it should be noted that a "part history" file is obtained from QUEST showing all the activities of parts are shown such as when they are first created or when they reach and leave the lots.

The code written summons this file for further calculations. Later, simulation time is divided into 1000 instances which means that samples are gathered for those instances related with part history. For each instant the weight of steel plates stored in lots are found. Finally, they are integrated to deliver the result, which is WIP.

Required area for storage of steel plates

The other design objective is selected to be reducing the required area for storage of steel plates. As Caprace et al (2013) states, space within a shipyard is an important resource. Use of space unnecessarily for a process that is not value-adding to the final product may be considered as a waste of resource. It is also fair to state that using an excess area for storing steel plates is a loss of money. Reducing the area used for stocking plates will create an area that can be used for other activities in an already working shipyard. Furthermore, in case of initial planning and constructing a shipyard facility, less amount of space needed for storage will result in change and improvement of plans. Proper planning of spaces within a shipyard is an important task to accomplish. Minimizing the area of storage is directly connected to the proper use of resources. Therefore, it is selected as the other design objective.

While calculating the area required for storage of steel plates in the stockyard, an approximation is done. The approximation is that calculation of exact area in meter squares is not necessary to be evaluated and optimized in our current task. Therefore, instead of calculating the area, number of active lots is taken as an indicator of the area used for storage.

An excel file is created for activating or deactivating lots based on binary coding. The file is presented in Appendix C as well. In the excel file, the identities of lots are described with their sections, rows and columns, which in our application they are being activated as whole row as well. In the file, "1" stands for active while "0" stands for not active.

It can be said that these two objective functions are contradicting in real cases. Reducing the area of storage increases the time required to reach for the steel plate located below in a lot and thus increases WIP. However, since the piling and unpiling operation is not integrated to the simulation yet, the contradiction of results is not clear unlike reality.

The application of objective functions will be present in "Coupling and set-up section" in details.

2.3.3.2. Design variables

After deciding for the objective functions, design variables are evaluated among several possibilities. Finally, three design variables are chosen and selected. These three variables are

number of active lots that are used for storage, number of steel plates allowed in a lot, which can be also stated as lot capacity, and number of steel plates in the pile arriving each week to the shipyard. Defining conveyor or crane speeds as variables are also taught to be applied but later decided not to be used to make case simpler. Also for further works in the area, these speeds may be defined with a stochastic distribution in QUEST instead of constant values which will represent a more appropriate simulation.

Number of active lots

Number of active lots that are used for storage is set to be one of the design variables. It is directly related with the area and WIP. They are activated or deactivated in a logical sense. Lots are grouped as rows while they are being activated and deactivated. Also, in order to have a realistic case, they are being started to be activated from the closest row to the conveyor to the farthest one. In some cases, some sections are totally deactivated in order to see whether it is also feasible to work with less area and less cranes.

As it is already explained, activation of lots is controlled by a binary approach given as Appendix C. The lots are activated or deactivated as rows. Furthermore, in some cases, some sections are deactivated completely. This action is also controlled with the same Excel File which has the binary code logic. The creation of lots by rows is implemented with another code which given in Appendix D. Finally, number of active rows in a section is a variable defined in another code which is given in Appendix E. The connection of these variables with optimization software will be described and shown in section “Coupling and set-up” clearly.

Also it should be noted that number of active lots is assumed to be 1 design variable. However, in MODEFRONTIER they are modelled as 3 different variables for each section separately to investigate the case when some sections are completely not active.

Number of steel plates allowed in a lot

Number of steel plates allowed to be stored in a lot is another design variable. As number of active lots decrease, it is expected that number of plates in a lot would increase. In reality, it would have major impact on piling and unpiling time. However, it is not possible to see this fact in current application, since piling and unpiling function is not implemented in the simulation yet. Still, it has direct effect on area and WIP.

It is defined within the same code in simulation folder which is given as an appendix D.

Number of steel plates arriving in a pile

New pile of steel plates is assumed to arrive once a week. This frequency will be kept constant throughout each iteration and also is kept same for all cases. However, number of steel plates in arriving pile is selected as the last design variable. Thus, the result gives clear idea how to plan purchasing. Distribution of steel plate types among a pile is also kept constant in all cases.

The number of steel plates is also placed in a code for simulation. The code that is presented in Appendix E is used to alter this variable as well.

It might be seen from the code given that this variable is defined with source frequency. As it is already described, it is assumed in the simulation that a new pile arrives to the shipyard once a week with a certain number of steel plates in the pile. In order to model this operation, a main buffer is defined which holds all the created plates for a week and transfers them to active sections once week. Therefore, the frequency of creation of steel plates is altered to have different numbers of steel plates held in the main buffer. Despite the fact that number of plates in an arriving pile is set to be the design variable, in MODEFRONTIER set-up, frequency is the input variable.

2.3.3.3. Constraints

As it is presented in Figure 13, some constraints and limits are needed to be defined for design variables and objective functions in order to have realistic and feasible results. Also not defining certain constraints result in longer computation time.

Some of these limits are defined within selection of parameters, while some are defined with additional constraint nodes which will be shown clearly in next section when the set-up will be explained.

To start with, in all cases, number of active row of lots will be minimum zero. Therefore, it would be possible to see the results for conditions when some sections are not active. This might cause the result that when none of the sections are active, the optimization may yield the optimum results that WIP and area used for storage are both “0”. In order to prevent that constraints are defined for objective functions.

A limit is defined for number of allowed steel plates in a lot as well. Having too low number of steel plates in a lot is an unfeasible condition in reality. Contrary, having huge piles is also an unfeasible and practically dangerous and impossible. Besides, in a real case, pile and unpile operation occurs in the shipyard and having high number of steel plates in a lot will result in too much consumption of time during this process. Therefore, an assumption is made and this number is limited between 5 and 15 in all cases.

Another design variable is number of steel plates in the pile arriving each week. This value is also limited with lower and upper boundaries. However, this number is altered for different cases due to the fact that steel processing capacity of a shipyard is altered. Different numbers of steel plates are defined based on different capacities of processing. The fact that having too much steel plate arriving each week will cause increasing number of stored materials in stockyard and WIP will get higher due to this fact, upper limits are decided close to capacities. Also in MODEFRONTIER it will be represented with altering frequency, therefore the limits are defined as time steps, not as number of steel plates in the pile.

Constraints are defined for both of the objective functions as well. This due to the fact that having "0" value for WIP or required area for storage is an unfeasible solution which corresponds to the meaning that no process takes place in stockyard and it is not the case we try to analyse. Therefore the results are constrained to have values greater than "0".

2.3.4. Coupling and set-up

Both QUEST and MODEFRONTIER are capable of coupling with other software. In fact, MODEFRONTIER offers easy coupling with some modules for some software already such as ANSYS and SOLIDWORKS. However, a batch code is required to make it work with QUEST. Also it should be noted that MODEFRONTIER and QUEST use different file formats. Therefore, batch codes are used to convert formats from one form to another in order to supply connection and flow of data between them.

Before further explaining the process, it may be helpful to take a look at the loop created in MODEFRONTIER to run the optimization. Figure 14 in the next page shows the loop. It can be seen that MODEFRONTIER offers an easy to use and understand user interface. Each module will be explained in details about how they function and how they are set in our task.

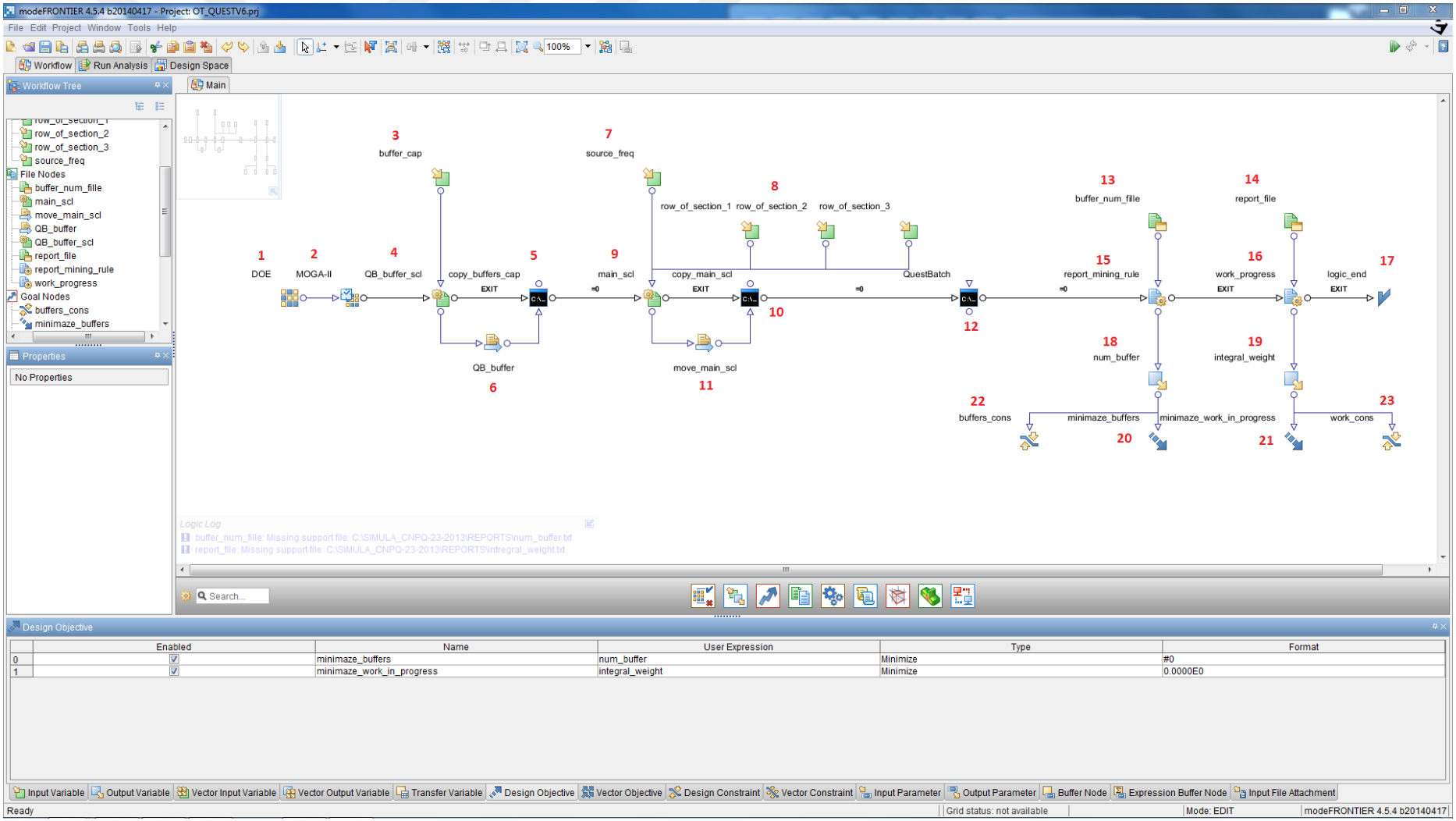


Figure 14: Optimization loop

A number is given for each of the modules for easier follow up of descriptions. Now they will be explained in details.

Module 1

As it is seen in the Figure 14, module 1 is called as “DOE”. It is a module to create a design space. DOE is the abbreviation of “Design of Experiments” which has the meaning that it covers every possible combination of design variables. In MODEFRONTIER, DOE module is used to create the initial design variable combinations for the optimization iterations.

Creating 10 random designs is sufficient in application. Other designs are created by the software based on MOGA-II algorithm. It can be seen that values for each variable are created within the defined limits.

Module 2

The module presented with MOGA-II is actually named as “scheduler” in the software. It enables users to select any optimization algorithm that is fitting their purpose and case. Figure 15 shows the Scheduler window.

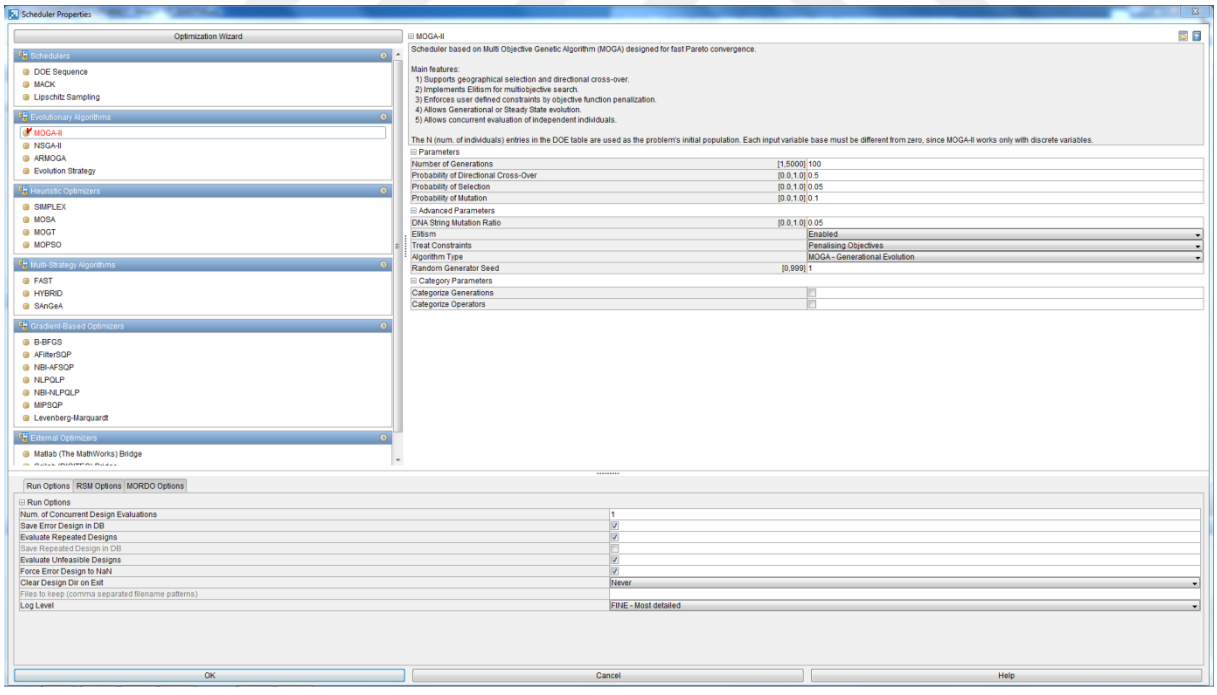


Figure 15: Scheduler

It can be seen from Figure 15 that several algorithms are offered by software. Since we already decided to use MOGA as an evolutionary algorithm, it is selected. Also it can be seen that there are brief explanations related with each algorithm on the top of the window which

may be helpful for selection as well. The probability values for directional cross-over, selection and mutation may be entered in this window. Also whether or not elitism will occur in process may be selected. Since the default values presented by the software are used commonly and widely, these values are kept constant as well. The number of iterations is selected as well with the "Number of Generation" line in this window. This means that the loop will be repeated for given number of times. However it is also possible to stop the run before reaching the decided iteration number if any convergence is observed in the results.

Module 3

Module 3 is an input variable. "Buffer_cap" refers to design variable which is stated as number of steel plates that a lot may store. The module is used to define the limits and steps of the defined input variable. In our case, the values are integers; therefore the step is defined as 1.0. Also lower and upper boundaries are selected as 5 and 15.

MODEFRONTIER gives the chance for defining a distribution for the interval of possible values. However in our application we do not define any distribution.

Module 4

Module 4 is used to define input template. It is used to define the selected input variables that are changed from the database. The file is connected and then the corresponding data is selected from the "Edit Input Template" section. It may also be noted that in each window, other connected data and processes are shown in the lower part of the window.

After selection of template file from the QUEST folder, the relevant data is selected from the input template file as shown in Figure 16 given below.

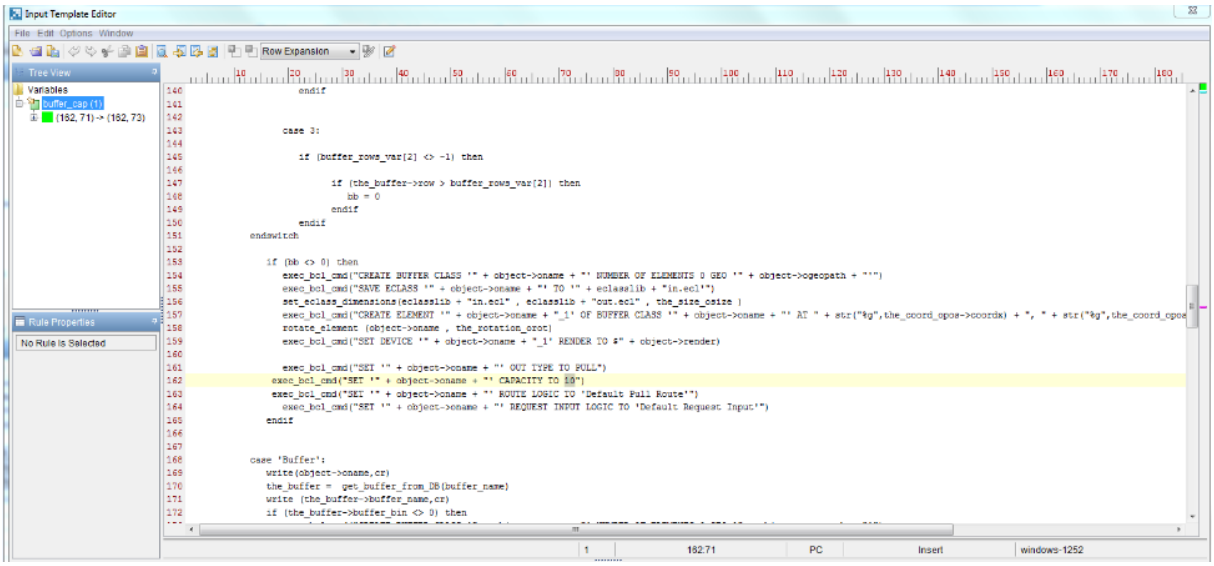


Figure 16: Input Template Editor – Number of steel plates allowed in a lot

As it is seen, the data that defines the number of steel plates is selected from the code that is placed in the database of QUEST which is already given in Appendix D.

Module 5

Module 5 is a DOS batch module. It is used to take the changed file and place it in the database of QUEST. It is one the trickiest parts of the coupling due to the fact that without this module, the changed input variable is not altered in the QUEST database. This results from the fact that QUEST and MODEFRONTIER use separate folders for creating and storing data files. Therefore the required data is parsed to the system folder used for simulation.

Module 6

This module is a transfer file. It is used to make MODEFRONTIER transfer the generated files to database folder of simulation. The file “QB_buffer”, which is given in Appendix D, controls the creation of lots, which is named as “buffer” in QUEST.

Module 7

This module represents another input variable. Number of steel plates is chosen as design variable. However as mentioned previously, source frequency is defined as the input variable in MODEFRONTIER set-up.

The same procedure is followed as module 3 for defining the source frequency as input variable.

It can be again stated that since all possible values are integer, the step is defined as 1.0. The lower and upper boundaries are defined. It should be noted that these values are time steps defined in QUEST. As briefly stated before, QUEST uses time steps for inputting frequency. It has the meaning that a new plate is created in the source in every time step entered which has the unit of seconds.

Module 8

All input variables defining active buffers in sections are numbered together as Module 8. They all have the same working principle with different connections in the corresponding data file. They are defined as separate input variables to enable the search for cases when any section is totally deactivated. It is possible to see the results when they are arranged and planned separately.

As it is described below, in case of the fact that no rows are active in a section, it corresponds to the meaning that that section is totally out of work and only other 2 or 1 section are currently being used for storage. By this approach, it is possible to observe results showing the conditions with 1 or 2 sections are totally deactivated.

Module 9

Module 9 works in a similar way with module 4 previously described for the input variable number of steel plates allowed in a lot. However in this case, all 4 input variables, which are source frequency and number of active rows in 3 sections, are connected with one module since they are all read from the same database folder. All variables are defined in the code called as “main_scl” which is given in the appendix E.

Module 10

It has the same duty with Module 5 and it is another DOS batch module. It is defined to place changed variables in QUEST database for simulation running.

Module 11

This module is a file transfer module like module 6 too. It is used to inform MODEFRONTIER that the generated data is going to be placed in the database of simulation. It is required to make the connection of two software. The data that is transferred is

“main_scl” code which is given as Appendix E. It contains the data related with source frequency and number of active rows which is already explained.

Module 12

“QuestBatch” is a module that connects QUEST and MODEFROTIER basically. The code that is used is given in the Appendix F. It is summoning QUEST to run with MODEFRONTIER. By this file, in each iteration design variables are taken from MODEFRONTIER and a simulation is conducted in QUEST. The results are gathered from QUEST and then processed in MODEFRONTIER. MODEFRONTIER takes the results and runs the optimization task. As it is a loop, the same flow of data is repeated for each of the iterations.

It is a DOS batch module. It can be stated that it is the black box that connects input variables to output variables. As is it is described, modules that are presented so far are related with input of the optimization task. This module calls QUEST and starts to runs the simulation. Next modules are related with the output of the task.

Module 13

It is a support file module. It is used to define the data that is going to be minimized or maximized. Therefore it can be stated that it is the module that is used to implement objective function in MODEFRONTIER.

It is taken from the report file of the simulation. This file is connected to gather the results of number of active buffers in each iteration. As described before this data is used to represent the area used for storage of steel plates in the stockyard.

Module 14

Module is the other support file for the second objective function, which is minimizing the WIP. It can be seen from the figure below that again a file is created within the report folder of QUEST.

The required data for calculating the WIP is gathered from that file. For the calculation of WIP, another code will be used which is already described and given in Appendix B.

Module 15

Similar to input template modules, this module is used to define the data that is going to be altered in the output files which are integrated to the MODEFRONTIER as support files.

Module 16

Similar to what is explained for module 15, this module is used for WIP output. The respective data is selected from the created report file. The connection supplies the change of data after each iteration in the selected data in the file.

Module 17

It is a logic end module which is used to define that the loop is completed. The same steps of optimization will be repeated for the rest of the task until the predefined number of iterations is reached or the run is manually stopped.

Modules 18 and 19

These modules are used to define the properties of output data which are the final results obtained from the simulation and processing of these data into the values we search for. Number of active lots and calculated WIP are defined with these modules. Figure 37 shows the properties window for number of active buffers.

Since number of buffers is an integer value, format is set to be that way. However WIP is selected to be a value with 3 decimals. This selection of properties is done due to the fact that WIP is calculated by integration and a more precise result may be required.

Modules 20 and 21

The objective function is formulated with these two modules. They are explained in the same section since both are done with similar approach. In our case both of our defined output variables are needed to be minimized since our objective functions are defined as minimizing the are required for storage of steel plates, which is corresponding to the objective defined in MODEFRONTIER as minimizing the number of active lots, and minimizing WIP.

Modules 22 and 23

As it is described before, some constraints are needed to be defined to reach valid and feasible results for objective functions. For both objectives, having “0” as number of active buffers

and WIP would result in most optimum solutions as “0” is the minimum solution for both. However in reality this result does not have any equivalent because of the fact this result means that no operations or flow of process takes place. Therefore constraints are defined for both objectives and they are set to be greater than “0”. It may be noted as well that, as a design variable, number of active lots in each section are selected as they can be equal to “0”. This approach is followed due to the fact that the results of trials for the cases when 1 or 2 of the sections are deactivated.

3. RESULTS

This section covers the results for three trials run with the optimization set-up presented in the previous section. Before showing the results for test cases, the parameters that are different in each case will be explained.

It is already stated that the model and simulation are done based on the steel stockyard of the Atlantico Sul Shipyard. Therefore, in the first case, the steel processing capacity of the shipyard is selected as the given value in the website of the shipyard, which is 160.000 tons per year.

The other two cases are defined to evaluate the cases for shipyards with different capacities. As 160.000 tons per year may be assumed as the capacity for a big shipyard, in other cases lower steel processing capacities are considered which are 100.000 and 40.000 tons per year. However, it should be noted that the layout of Atlantico Sul is used in all cases.

With other two cases under consideration, some parameters are needed to be altered in both QUEST and MODEFRONTIER which will be described below.

The frequency of steel plate request is a parameter defined in QUEST. It may also be reminded that it is a pull production system. In the first case, it was defined 160.000 tons per year which is the actual capacity of shipyard. Since other cases deal with lower capacities, this fact should be represented in simulation as well. As it is mentioned, QUEST uses time steps to define frequency. Therefore, a simple calculation is carried out to define the time interval of steel plate request from the sink based on the steel plate processing capacity of shipyard which is in unit of tons/year.

The calculations are done based on the assumption that an average steel plate has the weight of 2.58 tons, which is the average value for the 10 types of steel plates selected for our application. Table 4 below shows the time steps calculated for each case.

Table 4: Time steps for each case

Cases	tons/year	tons/second	plates/second	Time step (sec)
1	160000	0.0051	0.0020	509
2	100000	0.0032	0.0012	814
3	40000	0.0013	0.0005	2034

It can be seen that with decreasing capacity of steel processing, time interval between requests for new steel plates increases.

Based on the changing steel plate processing capacity of shipyard, adjustments are required to be done in MODEFRONTIER as well. Same loop and most of the variables are kept constant. It is not required to make changes in design variables such as number of steel plates allowed to be stored in a lot and number of active lots. In addition, the objective functions are kept constant as well. However for healthy evaluation of results and in order to prevent crashing of software, source frequency boundaries are needed to be altered in each case. This is due to fact that number of steel plates arriving to the stockyard per week would not be same for a shipyard with greater capacity and relatively smaller capacity.

Furthermore, it should be again stated that source frequencies in each cases are defined based on the request frequency from the sink created in QUEST which corresponds to the processing capacity in the model. Also it should be reminded that source frequency is used to define the arriving number of plates once a week. The plates that are stored in the initial buffer are distributed each week.

For all cases, lower and upper bounds are selected to be 25% less and higher than source frequency. The boundaries are given for each case in Table 5 below.

Table 5: Source freuquency boundaries for each case

Cases	Request frequency	Source frequency	
	Time step (sec)	Lower boundary (second)	Upper boundary (second)
1	509	381	636
2	814	610	1017
3	2034	1526	2543

It should be reminded that boundaries for source frequencies are also in the unit of seconds and defined as time steps. It has the meaning that lower values for source frequency means greater number of steel plates being created in the source arrive to the buffers in QUEST.

3.1. Case 1

The first case is run with the given values in the previous section. It will be seen from the results that 1064 iterations are completed. As stated, the case can be identified as a rather difficult one due to the fact that 5 design parameters are being altered within certain intervals and software requires more iterations and therefore more time reach to obtain a more valid Pareto diagram.

Regarding the optimization procedure, it may also be noted that each iteration takes approximately 3 minutes, which results in several hours to reach the desired number of iterations.

All values regarding the optimization procedure are taken from MODEFRONTIER. Graphs will be presented in this section regarding the change of design variables and obtained results. It should also be noted that there are several failures in the simulation due to the full loading of the active lots in any of the sections. When the full loading condition is obtained and no more place is left for storing new arriving plates, the case is identified as failure by the software. Thus, it can be observed in Figure 17 that almost 50% of the results are failures.

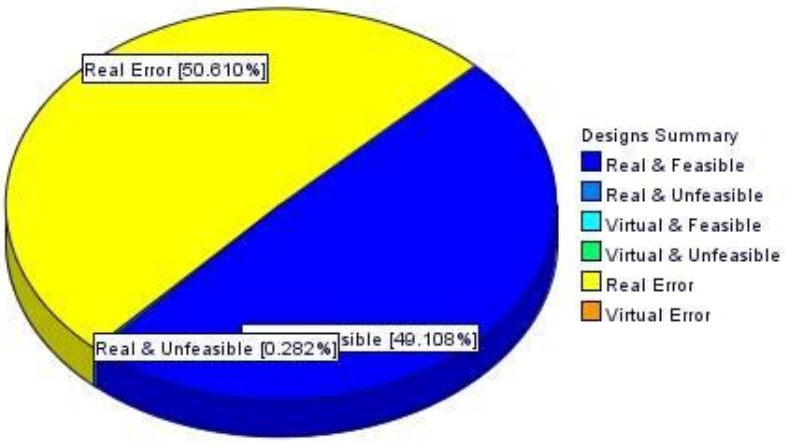


Figure 17: Percentage of errors and feasible results

However, it is also seen that number of failures decrease towards the last iterations which is a proof that optimization process starts to yield better results throughout the run. It is resulted from the nature of the algorithm as it tries to select and create healthier design variables with each new iteration.

Figures below shows the design variables used in each iteration. As mentioned before there are five design variables defined in the loop which are number of plates allowed in a lot, source frequency and three separate variables for number of active rows in each section.

Figure 18 shows the values used by software for number of plates allowed in a lot in each iteration.

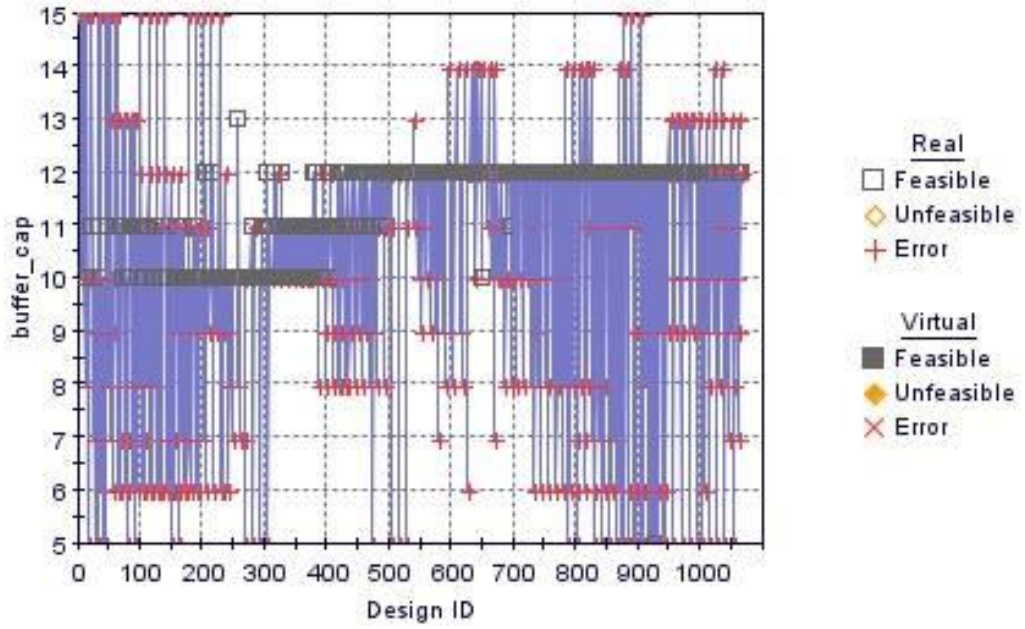


Figure 18: Number of plates allowed in a lot vs number of iterations

It can be seen that feasible results are generally obtained for “12” as number of plates that is allowed to be stored in a lot. It can also be seen that jumps of values occur several times due to the nature of genetic algorithm used. It can be discussed whether current of number of iterations is sufficient. Also as mentioned before number of errors tends to decrease as number of iterations increase.

Similarly Figure 19 shows the graph for change of source frequency.

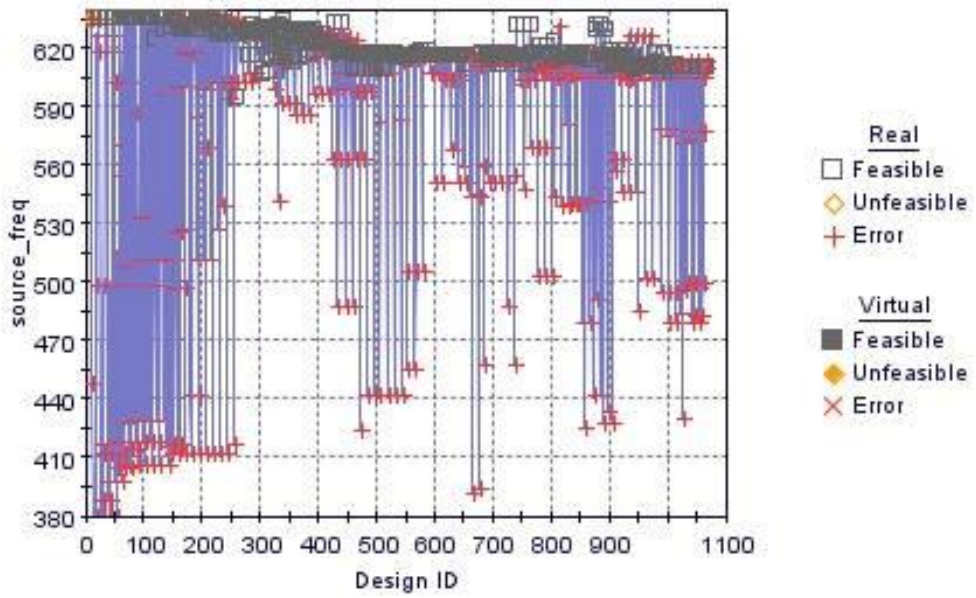


Figure 19: Source frequency vs number of iterations

It can be seen that feasible values for source frequency are grouped close to the maximum value that can be given within the defined limits. This fact is resulted due to the fact that lower source frequency value corresponds to high frequency of plates arriving to the stockyard which eventually causes early filling of the active lots and error of the simulation.

Figures 20, 21 and 22 below show the design variables, number of active rows in each section separately

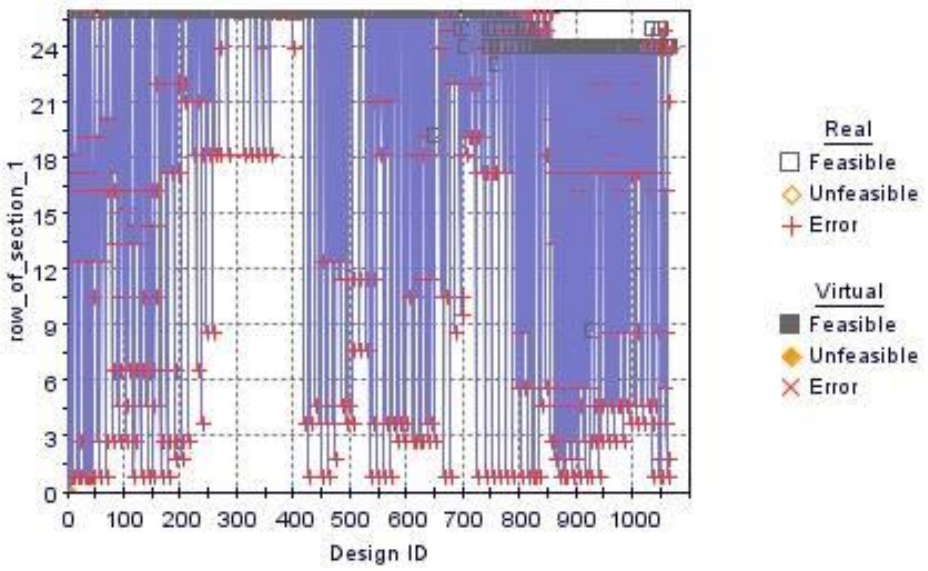


Figure 20: Number of active rows in section A vs number of iterations

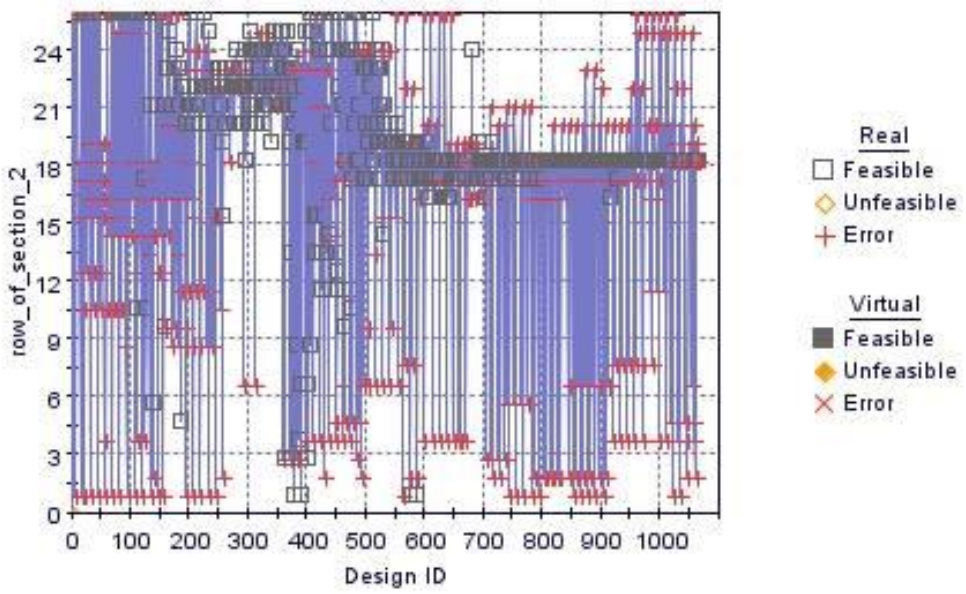


Figure 21: Number of active rows in section B vs number of iterations

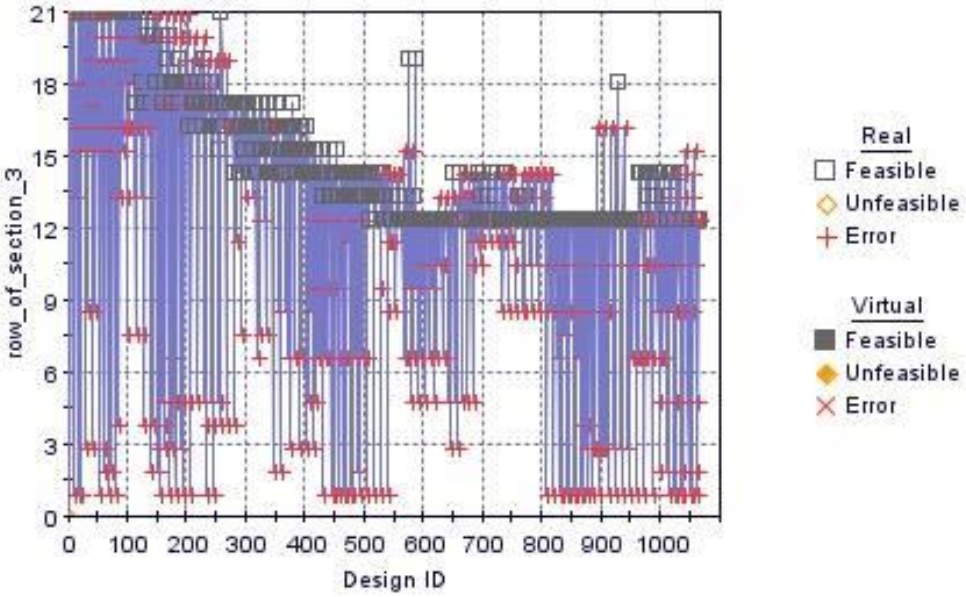


Figure 22: Number of active rows in section C vs number of iterations

It can be suggested for the cases that are failing, number of active rows in each section are rather small. This fact causes for early filling of active lots and results in the failure of simulation as well. However, it can be seen that most feasible results are obtained for 24 active rows in section A, 18 in section B and 12 in section C.

Also it should be noted that lower boundary for active buffers is set to be “1” in the application. It was initially desired to see the effect of closing a whole section in the

stockyard. However an error occurred during the run related with the cases in which one or two of the sections are closed. Therefore a change is made in order to solve the problem.

The graphs given so far are used for defining the feasible results. It is fair to state that converging variables are seen in the graphs. However for better evaluating the quality of the results that are obtained, graphs that are given below are considered to be rather enlightening. Figures below show the results for objectives with respect to number of iterations.

It should be noted that it only shows the ones that are feasible. There are no unfeasible results obtained from the procedure. However as mentioned before, 50% of the results are failures resulted from the simulation. The failures are not plotted in the graphs since they deliver no valid result.

Figure 23 shows WIP versus number of iterations.

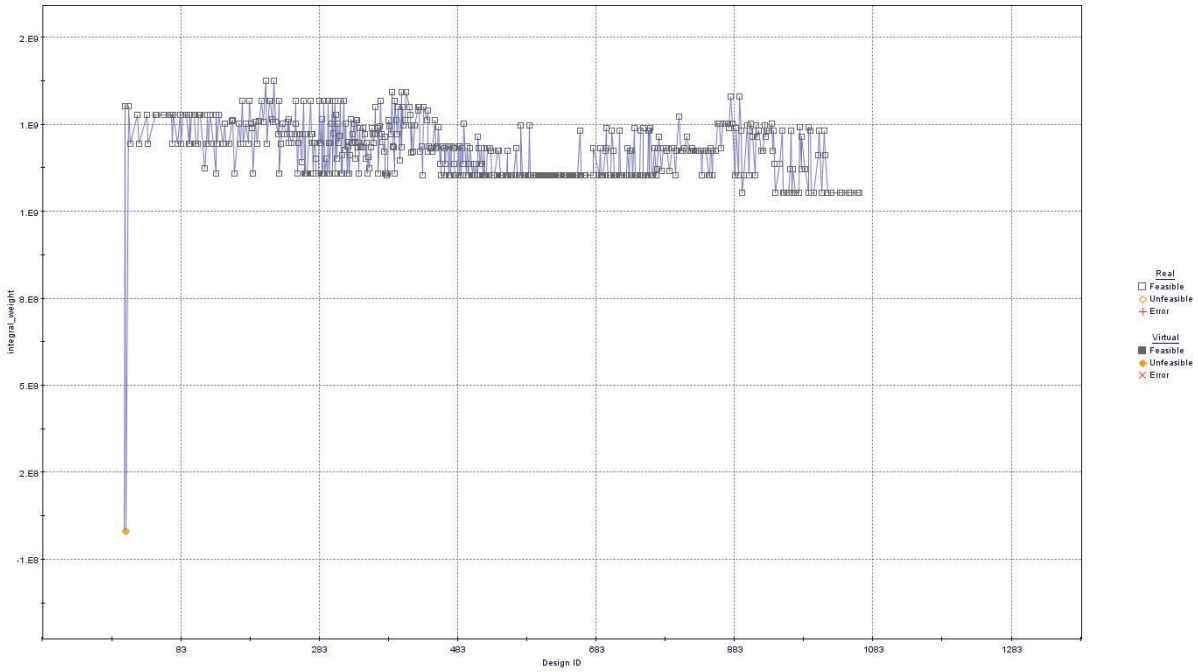


Figure 23: WIP vs number of iterations

It can be seen from Figure 23 that results show some convergence in the end. However a clear convergence is also seen between iterations 480 and 700. Therefore, it is fair to state that it is not possible to give a clear answer at this stage of run and more iterations are needed for having a more fit result. Still it is clear that the objective of minimizing the WIP is being done even with this number of iterations.

The evaluation of the values obtained will be presented later after the Pareto diagram.

Similarly Figure 24 shows number of active buffers values obtained in feasible solutions.

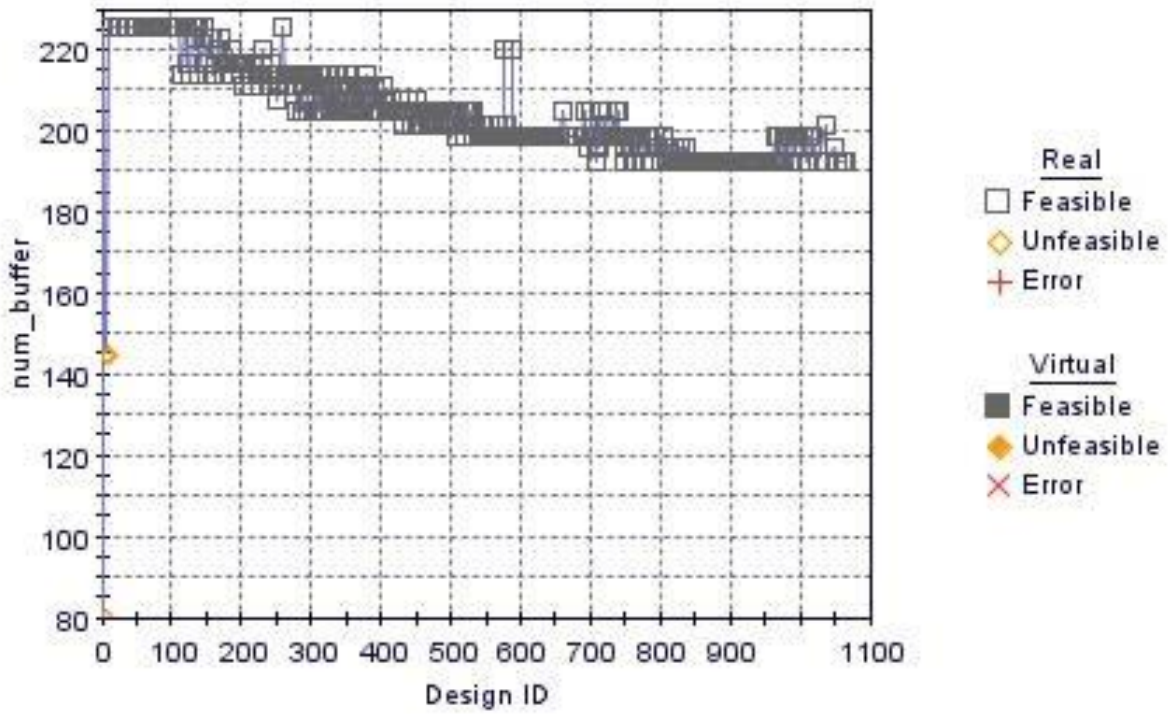


Figure 24: Number of active buffers vs number of iterations

It can be seen from Figure 24 that similar to what is already discussed for other objective function, WIP; there is a tendency to minimize the number of active buffers in the stockyard. Compared to WIP, jumps in the results are not big.

As it can be seen from Figures 23 and 24, number of feasible results has a tendency to increase throughout the run. Therefore it is fair to state that reaching a certain amount of iteration number will yield more valid results for the task.

Finally Figure 25 below shows the pareto graph for the case. As we have a limited number of iterations completed so far and rather less feasible results, it is not possible to suggest that the results are satisfying and may be taken as a source to give decisions regarding the design of the steel stockyard activities and layout. However it is still important to understand how the graph may be used for decision making in an ideal condition.

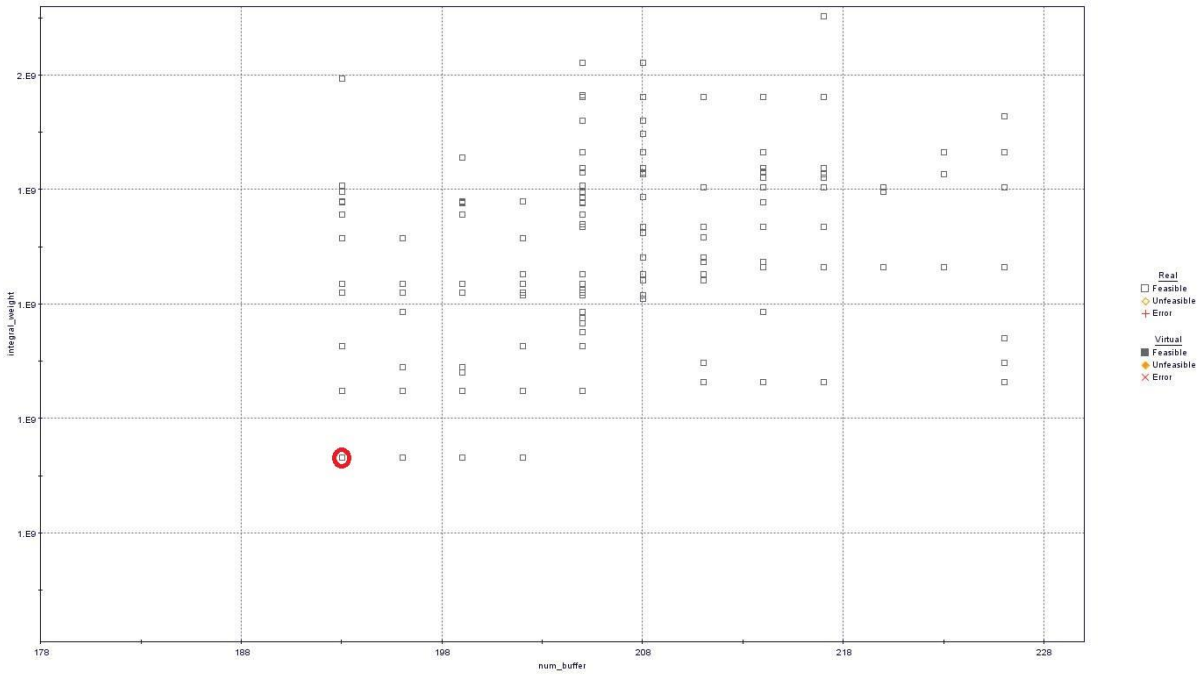


Figure 25: Pareto graph - Number of active buffers vs WIP

As already discussed, current number of iterations, 1064, is clearly not sufficient for our task. It can be also understood from the Pareto diagram presented. It was expected to have clear Pareto frontier in the graph with the most common and best results for our task. However it is seen that despite the fact that there are certain feasible results that can be selected from the results, no grouping of the result is seen.

It should be also noted several alternative give the same outcome as results. As described before, Pareto graph is a great tool showing every possible outcome of each alternative and creates a space to select among possible alternatives. Based on the solutions obtained so far, it can be stated that the point represented with red circle in the graph is the optimum one. Table 6 below shows the iterations that correspond to the selected results. It also covers the values of design variables and results for the objective function.

Table 6: Optimum results

	Iterations	953, 954, 958, 961, 966, 971, 989, 1009, 1022, 1037, 1050, 1061, 1063
Design variables	Number of active lots in section A	24
	Number of active lots in section B	18
	Number of active lots in section C	12
	Source frequency (second)	611
	Number of steel plates allowed in a lot	12
Objective functions	WIP (ton.second)	$1.17*10^9$
	Number of active buffers	193

Based on the results obtained, it can stated that current steel stockyard layout of the shipyard is not the optimum one. More efficient solutions regarding WIP and area covered can be obtained by changing the layout of the steel stockyard. Less area can be used for storage and free area may be used for alternative purposes. Also proper planning of the arriving steel plates may yield more efficient outcomes as well compared to initial condition.

It can be seen from the results that even such a high number of iterations is not sufficient to reach a certain conclusion in a complicated real industrial problem. Therefore for better evaluation of the case, more iterations are needed. It should be stated that, more iterations will be carried out and the results will be submitted as an "Addendum".

Furthermore, it may be also noted that several assumptions and approximations were done within the simulation. Deeper analysis of the results will be discussed in "Conclusion" section.

3.2. Cases 2 and 3

Due to the limited time of internship and thesis completion, these two cases were not conducted. As stated running the process takes significantly long time to reach the convergence.

Therefore, in case of completion of all cases, these cases will be presented in the "Addendum" as well.

4. CONCLUSION

Despite the fact that presented results are not totally ideal for the application and desired purpose of the work, it is possible to observe convergence of the result towards the minimum. An “Addendum” will be presented later when the runs are completed for all cases. Furthermore, it can be said that main initial objective of the work is fulfilled which is coupling a simulation software with an optimization tool and making it work.

Besides the inefficient results that are presented, it should be also noted that to reduce the optimization time required, 10 user defined values may be used instead of random DOE. These values may be selected from the ones that would not cause any failure in simulation due to full loading condition. Thus, MODEFRONTIER will be guided to better results that would yield less failures and greater number of valid results may be obtained in a shorter amount of time.

The results support the suggestion that coupling simulation and optimization would yield an efficient solution for real industrial problems. However it should be underlined that real industrial problems involve several issues and details which are difficult to simulate and optimize properly. The work clearly proves that the process of creating the simulation and coupling with an optimization tool is a complicated task to complete and requires great amount of time and work.

However it is also fair to suggest that both QUEST and MODEFRONTIER are powerful, capable and useful software. They offer great amount of help and user-friendly solutions for such applications.

To sum up, it is believed that this work may be regarded as a stepping stone for several future works in the field. It is clear that shipbuilding industry is headed in a direction that concepts mentioned within the work will become more important in near future.

5. FUTURE WORK

The thesis showed that coupling of simulation and optimization software is a useful and applicable approach. Therefore, it can be stated that works in this field will increase with further applications in shipbuilding as well as other industries.

Based on the some deficiencies of the work presented, some suggestions may be given to researchers for improving and carrying on with further works.

First of all, as it is stated and justified, simulation used within the thesis is not stochastic. Having completely correct and precise results were not the main objective of this work which is conducted in a limited time. For better results, simulation may be developed with stochastic data.

Another future step to develop the work may be implementing the piling and unpiling operations in steel stockyard. Again, due to the limited time of internship, this operation was not implemented in the simulation. This fact had a direct impact on the results. Therefore implementing this operation will significantly improve the work. Besides, other operations regarding plate processing may also be modelled and added to the simulation as well.

Furthermore, some additional design variables and new objective functions may be introduced. Crane and conveyor speeds are two options for design variables which may be taken into account as well. An objective that is related with cost directly may be implemented with some coding for cost calculation based on some assumptions and approximations.

Finally, the work shows that simulation and optimization software can be coupled and they can work together giving promising results. Therefore, same approach may be followed for other steps of shipbuilding process.

As it is already mentioned, it is believed that several works are going to be done and published in the field of simulation and optimization in the coming years and it is hoped that the thesis may be used by researchers to have some idea about the subject.

6. REFERENCES

Stopford, Martin. Maritime Economics. 2nd ed. London [etc.: Routledge, 1997. Print.

"Shipbuilding Statistics." (2015). The Shipbuilders' Association of Japan. Web. 2015.

Gebhart, Laurence P., and Robert G. Jarvis. "Productivity Improvement at the SENESCO Shipyard." Journal of Ship Production 19.3 (2003): 187-93. Print.

Park, C., J.-C. Park, G.-G. Byeon, H.-G. Kim, and J. Kim. "Steel Stock Management on the Stockyard Operations in Shipbuilding: A Case of Hyundai Heavy Industries." Production Planning & Control (2006): 1-12. Print.

Koenig, Philip C., Hitoshi Narita, and Koichi Baba. "Shipbuilding Productivity Rates of Change in East Asia." Journal of Ship Production 19.1 (2003): 32-37. Print.

Kolic, Damir, Niksa Fafandjel, and Albert Zamarin. "Lean Manufacturing Methodology for Shipyards." (2013). Print.

Lang, S., N. Dutta, A. Hellesoy, T. Daniels, D. Liess, S. Chew, and A. Canhetti. "Shipbuilding and Lean Manufacturing - A Case Study." The Society of Naval Architects and Marine Engineers (2001). Print.

Liker, Jeffrey K, and Thomas Lamb. "What Is Lean Ship Construction and Repair?" Journal of Ship Production 18.3 (2002): 121-42. Print.

Phogat, Sandeep. "An Introduction to Applicability of Lean in Shipbuilding." International Journal of Latest Research in Science and Technology 2.6 (2013): 85-89. Print.

Cha, Ju-Hwan, and Myung-Il Roh. "Combined Discrete Event and Discrete Time Simulation Framework and Its Application to the Block Erection Process in Shipbuilding." Advances in Engineering Software 41 (2010): 656-65. Elsevier. Web.

Cha, Ju-Hwan, Myung-Il Roh, and Kyu-Yeul Lee. "Integrated Simulation Framework for the Process Planning of Ships and Offshore Structures." Robotics and Computer-Integrated Manufacturing (2010): 430-53. Print.

Fernández, Rodrigo Pérez, and Verónica Alonso. "Virtual Reality in a Shipbuilding Environment." Advances in Engineering Software (2015): 30-40. Print.

Caprace, J.-D., C. Petcu, M. G. Velarde, and P. Rigo. "Optimization of Shipyard Space Allocation and Scheduling Using a Heuristic Algorithm." J Mar Sci Technol Journal of Marine Science and Technology (2013): 404-17. Print.

Bair, Frederic, Yves Langer, Jean-David Caprace, and Philippe Rigo. "Modelling, Simulation and Optimization of a Shipbuilding Workshop (2006)." Print.

Konak, Abdullah, David W Coit, and Alice E Smith. "Multi-Objective Optimization Using Genetic Algorithms: A Tutorial." Reliability Engineering & System Safety 91.9 (2006): 992-1007. Print.

Justesen, Peter Dueholm. "Multi-Objective Optimization Using Evolutionary Algorihms." (2009). Print.

Nilsen, Espen. "Parameterization and Multiobjective Optimization." (2013). Print.

Mitchell, Melanie. An Introduction to Genetic Algorithms. Cambridge, Mass.: MIT, 1996. Print.

Fishman, G. S., 2001. Discrete-Event Simulation: Modelling, Programing and Analysis. Newyork, US: Springer Verlag Newyork Inc.

7. APPENDICES

7.1. Appendix A

a) Plates of Steel Quality BV – GRADE A

Plate Dimensions (mm)			Material Specification	Plate Quantity				Weight				Total Weight
				Already Ordened			To be Ordened	Already Ordened			To be Ordened	
Thickness	Width	Length	Grade	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	(kg)
6	2440	12000	A	-	-	-	9	-	-	-	12.412	12.412
6	2750	12000	A	-	-	-	34	-	-	-	52.846	52.846
8	2440	12000	A	-	-	-	57	-	-	-	104.811	104.811
8	2750	12000	A	-	-	-	63	-	-	-	130.561	130.561
9,5	2750	12000	A	-	-	-	40	-	-	-	98.439	98.439
10	2440	9200	A	12	11	-	-	21.146	19.384	-	-	40.530
10,5	2200	9200	A	20	15	-	-	33.366	25.024	-	-	58.390
10,5	2440	9200	A	22	20	-	20	40.706	37.006	-	37.006	114.718
11	2200	9200	A	20	15	-	-	34.955	26.216	-	-	61.171
11	2440	9200	A	120	104	-	-	232.606	201.592	-	-	434.198
11	2440	12000	A	-	-	-	22	-	-	-	55.623	55.623
11	2750	12000	A	-	-	-	23	-	-	-	65.540	65.540
11,5	2200	9200	A	34	30	-	-	62.124	54.815	-	-	116.939
11,5	2440	9200	A	-	-	-	40	-	-	-	81.060	81.060
11,5	2750	9200	A	40	38	-	-	91.358	86.790	-	-	178.148
12	2200	9200	A	120	103	-	-	228.793	196.381	-	-	425.174
12	2440	9200	A	78	70	-	15	164.939	148.022	-	31.719	344.680

Plate Dimensions (mm)			Material Specification	Plate Quantity				Weight				Total Weight
				Already Ordened			To be Ordened	Already Ordened			To be Ordened	
Thickness	Width	Length	Grade	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	(kg)
12	2750	12000	A	-	-	-	13	-	-	-	40.412	40.412
12,5	1830	9200	A	53	46	-	-	87.558	75.993	-	-	163.551
12,5	2200	9200	A	16	13	19	17	31.777	25.819	37.735	33.763	129.094
12,5	2440	9200	A	101	89	-	-	222.474	196.041	-	-	418.515
12,5	2750	9200	A	9	-	2	-	22.343	-	4.965	-	27.308
12,5	2750	12000	A	-	-	-	12	-	-	-	38.858	38.858
13	2000	9200	A	-	-	-	21	-	-	-	39.432	39.432
13	2200	9200	A	9	9	-	-	22.343	18.589	-	-	40.932
13	2200	12000	A	-	-	-	22	-	-	-	59.271	59.271
13	2440	9200	A	-	-	-	37	-	-	-	84.760	84.760
13	2440	12000	A	-	-	-	20	-	-	-	59.760	59.760
13	2750	9200	A	27	24	57	12	69.710	61.965	147.166	30.982	309.823
13	2750	12000	A	-	-	-	28	-	-	-	94.294	94.294
13,5	2200	9200	A	20	15	13	19	42.899	32.174	26.851	40.754	142.678
13,5	2440	9200	A	11	-	22	-	26.168	-	52.336	-	78.504
13,5	2440	12000	A	-	-	-	28	-	-	-	86.883	86.883
13,5	2750	9200	A	10	10	-	13	-	26.812	-	34.855	61.667
14	2000	12000	A	-	-	-	14	-	-	-	36.926	36.926

Plate Dimensions (mm)			Material Specification	Plate Quantity				Weight				Total Weight
				Already Ordened			To be Ordened	Already Ordened			To be Ordened	
Thickness	Width	Length	Grade	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	(kg)
14	2200	9200	A	18	15	-	-	40,039	33,366	-	-	73.405
14	2440	9200	A	120	71	15	-	301,243	175,159	38,327	-	514,729
14	2750	12000	A	-	-	-	21	-	-	-	76,161	76,161
14,5	2440	9200	A	-	98	-	20	-	250,404	-	51,103	301,507
14,5	2750	12000	A	-	-	-	23	-	-	-	86,393	86,393
15	2000	12000	A	-	-	-	18	-	-	-	50,868	50,868
15	2200	9200	A	32	18	-	-	76,264	42,899	-	-	119,163
15	2200	12000	A	-	-	-	14	-	-	-	43,520	43,520
15	2440	9200	A	51	40	-	-	134,806	105,730	-	-	240,536
15	2440	12000	A	-	-	-	13	-	-	-	44,820	44,820
15	2750	9200	A	78	66	-	-	232,368	196,619	-	-	428,987
15	2750	12000	A	-	-	-	37	-	-	-	143,773	143,773
15,5	2000	12000	A	-	-	-	16	-	-	-	46,723	46,723
15,5	2440	9200	A	6	6	-	33	16,388	16,388	-	90,135	122,911
15,5	2750	9200	A	49	22	-	14	155,607	67,724	-	43,097	266,428
15,5	2750	12000	A	-	-	-	16	-	-	-	64,244	64,244
16	2200	12000	A	-	-	-	17	-	-	-	56,369	56,369
16	2440	9200	A	-	-	10	-	-	-	28,195	-	28,195

Plate Dimensions (mm)			Material Specification	Plate Quantity				Weight				Total Weight
				Already Ordened			To be Ordened	Already Ordened			To be Ordened	
Thickness	Width	Length	Grade	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	(kg)
16	2440	12000	A	-	-	-	14	-	-	-	51.486	51.486
16	2750	12000	A	-	-	-	36	-	-	-	149,213	149,213
16,5	2200	12000	A	-	-	-	18	-	-	-	61,550	61,550
16,5	2440	12000	A	-	-	-	14	-	-	-	53.095	53.095
16,5	2750	9200	A	-	42	-	-	-	137,633	-	-	137,633
16,5	2750	12000	A	-	-	-	10	-	-	-	42,743	42,743
17	2000	12000	A	-	-	-	16	-	-	-	51.245	51.245
17	2440	9200	A	6	6	-	-	17,974	17,974	-	-	35,948
17	2750	12000	A	-	-	-	30	-	-	-	132.116	132.116
17,5	2750	12000	A	-	-	-	17	-	-	-	77,067	77,067
18	2000	12000	A	-	-	-	16	-	-	-	54,259	54,259
18	2200	9200	A	43	31	-	-	122.976	88.657	-	-	211.633
18	2750	12000	A	-	-	-	23	-	-	-	107,247	107,247
19	2750	12000	A	-	-	-	9	-	-	-	44,298	44,298
20	2440	12000	A	-	-	6	4	-	-	27,582	18.388	45.970
20	2750	12000	A	-	-	2	7	-	-	10,362	36,267	46,629
20,5	2750	12000	A	-	-	2	9	-	-	10.362	47.795	58.157
21	2000	12000	A	-	-	-	10	-	-	-	39.564	39.564

Plate Dimensions (mm)			Material Specification	Plate Quantity				Weight				Total Weight
				Already Ordened			To be Ordened	Already Ordened			To be Ordened	
Thickness	Width	Length	Grade	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	(kg)
21	2200	12000	A	-	-	-	7	-	-	-	30.464	30.464
21	2440	9200	A	10	10	1	18	37.006	37.006	3.701	66.610	144.323
21	2750	12000	A	-	-	-	10	-	-	-	54.401	54.401
21,5	2440	12000	A	-	-	-	10	-	-	-	49.417	49.417
22	2440	12000	A	10	10	1	8	37.006	37.006	3.701	40.453	118.166
25	2440	12000	A	-	-	3	3	-	-	17.239	17.239	34.478
27,5	2440	9200	A	14	11	-	9	67.844	53.306	-	43.614	164.764
30	2440	9200	A	6	6	-	-	31.719	31.719	-	-	63.438
35	2750	12000	A	-	-	-	18	-	-	-	163.202	163.202
			TOTAL	**1.163**	**1.054**	**150**	**1.137**	**2.685.709**	**2.487.207**	**394.459**	**3.579.904**	**9.219.144**

b) Plates of Steel Quality BV – GRADE AH36

Plate Dimensions (mm)			Material Specification	Plate Quantity				Weight				Total Weight
				Already Ordened			To be Ordened	Already Ordened			To be Ordened	
Thickness	Width	Length	Grade	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	(kg)
13	2000	9200	AH36	18	16	3	-	33.799	30.044	6.872	-	70.715
13	2200	9200	AH36	-	34	-	-	-	70.227	-	-	70.227
13	2440	9200	AH36	9	8	-	15	20.617	18.327	-	34.362	73.306
13	2750	9200	AH36	-	-	-	22	-	-	-	56.801	56.801
13,5	2000	9200	AH36	18	16	-	-	35.099	31.199	-	-	66.298
13,5	2750	9200	AH36	-	40	-	-	-	107.247	-	-	107.247
13,5	2750	12000	AH36	-	-	-	35	-	-	-	122.401	122.401
14	2440	9200	AH36	27	16	-	-	66.610	39.473	-	-	106.083
14	2750	12000	AH36	-	-	-	21	-	-	-	76.161	76.161
14,5	2200	9200	AH36	-	17	20	26	-	39.165	46.076	59.899	145.140
14,5	2750	9200	AH36	18	-	36	-	51.836	-	103.672	-	155.508
14,5	2750	12000	AH36	-	-	-	22	-	-	-	82.637	82.637
15	2000	9200	AH36	27	24	5	-	58.498	51.998	11.916	-	122.412
15	2200	9200	AH36	18	16	-	-	42.899	38.132	-	-	81.031
15	2440	9200	AH36	32	30	9	-	84.584	79.298	24.582	-	188.464
15	2750	9200	AH36	80	66	-	26	238.326	196.619	-	77.456	512.401
15,5	2000	9200	AH36	20	4	14	-	44.776	8.955	31.343	-	85.074

Plate Dimensions (mm)			Material Specification	Plate Quantity				Weight				Total Weight
				Already Ordened			To be Ordened	Already Ordened			To be Ordened	
Thickness	Width	Length	Grade	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	(kg)
15,5	2000	12000	AH36	16	28	32	-	46,723	81,766	93,446	-	221.935
15,5	2750	9200	AH36	-	-	-	38	-	-	-	116,978	116,978
15,5	2750	12000	AH36	-	-	-	11	-	-	-	44,168	44,168
16	2000	9200	AH36	27	24	-	-	62,398	55,465	-	-	117.863
16	2200	9200	AH36	18	16	17	-	45,759	40,674	43,216	-	129.649
16	2200	12000	AH36	-	-	-	8	-	-	-	26,527	26,527
16	2440	9200	AH36	10	2	-	-	28,195	5,639	-	-	33.834
16	2440	12000	AH36	8	14	-	-	29,421	51,486	-	-	80,907
16	2750	9200	AH36	92	66	-	-	292,347	209,727	-	-	502.074
16	2750	12000	AH36	16	30	-	11	66,317	124,344	-	45,593	236.254
16,5	2440	9200	AH36	2	-	-	-	5,815	-	-	-	5,815
16,5	2750	12000	AH36	-	-	-	25	-	-	-	106.858	106.858
17	2200	12000	AH36	-	-	-	10	-	-	-	35,231	35,231
17,5	2750	9200	AH36	135	120	8	-	469,204	417,071	27,805	-	914,080
17,5	2750	12000	AH36	-	-	-	23	-	-	-	104,268	104,268
18	2440	9200	AH36	2	-	1	-	6,344	-	3,172	-	9,516
19	2200	12000	AH36	-	-	-	7	-	-	-	27,563	27,563
22	2440	9200	AH36	-	-	5	-	-	-	19,384	-	19,384

Plate Dimensions (mm)			Material Specification	Plate Quantity				Weight				Total Weight
				Already Ordened			To be Ordened	Already Ordened			To be Ordened	
Thickness	Width	Length	Grade	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	(kg)
22	2750	9200	AH36	-	-	32	-	-	-	139,818	-	139.818
22	2750	12000	AH36	-	-	-	7	-	-	-	39,894	39,894
25	2200	9200	AH36	-	-	3	5	-	-	11,916	19,861	31,777
28	2200	12000	AH36	-	-	-	3	-	-	-	17,408	17,408
			TOTAL	**593**	**587**	**185**	**315**	**1,729,566**	**1,696,856**	**563,220**	**1,076,657**	**5,066,298**

c) Plates of Steel Quality BV – GRADE B

Plate Dimensions (mm)			Material Specification	Plate Quantity				Weight				Total Weight
				Already Ordened			To be Ordened	Already Ordened			To be Ordened	
Thickness	Width	Length	Grade	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	(kg)
25	2200	9200	B	-	-	-	1	-	-	-	3.972	3.972
30	2440	11000	B	-	-	7	-	-	-	44,246	-	44,246
			TOTAL	**0**	**0**	**7**	**1**	**0**	**0**	**44.246**	**3.972**	**48.218**

d) Plates of Steel Quality BV – GRADE D

Plate Dimensions (mm)			Material Specification	Plate Quantity				Weight				Total Weight
				Already Ordened			To be Ordened	Already Ordened			To be Ordened	
Thickness	Width	Length	Grade	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	(kg)
40	2200	11000	D	-	-	-	1	-	-	-	7.599	7.599
60	2440	11000	D	-	-	-	2	-	-	-	15.402	15.402
			TOTAL	-	-	-	3	-	-	-	**23.001**	**23.001**

e) Plates of Steel Quality BV – GRADE DH36

Plate Dimensions (mm)			Material Specification	Plate Quantity				Weight				Total Weight
				Already Ordened			To be Ordened	Already Ordened			To be Ordened	
Thickness	Width	Length	Grade	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	Rev. A of Document 2E20-101	Rev. B of Document 2E20-101	Rev. 0 of Document 2E20-101-511	Rev. B of Document 2E20-101-511	(kg)
31,5	2440	9200	DH36	-	-	-	2	-	-	-	11.102	11.102
35	2200	9200	DH36	-	-	-	2	-	-	-	11.122	11.122
40	2750	12000	DH36	-	-	-	1	-	-	-	10.362	10.362
			TOTAL	-	-	-	5	-	-	-	**32.586**	**32.586**

7.2. Appendix B

```
#include <include_str_parthistory.inc>
#include <include_lib_dir.inc>
#include <include_str_elements.inc>
#include <agv.inc>
/* Preparing the code*/
-- Set extern routines
EXTERN
  open_output_stream : Routine   : Integer
-- Set global Variables
BCL_VAR
  bcl_msg : string
  bcl_status : Real
-- Set user attributes to be used
User_attrib
  pweight : real -- user atribute for weight of the parts
/*****************************************************************************/
                        /*-Code starts*/
/****f* scl/sample_weight_storage()
 * NAME
 *   sample_weight_storage()
 * SYNOPSIS
 *   *  sample_weight_storage()
 * FUNCTION
 *  Sample the weight storaged in all buffers in some point of simulation.
 * INPUTS
 *
 * RESULT
 * Real
 * AUTHOR
 *   $Author: GMonteiro $
 * CREATION DATE
 *   $Date: 2015/12/07 11:39:41 $
 * HISTORY
 *   $Revision: 1.9 $
 * SOURCE
 */
Routine sample_weight_storage() : real
VAR
the_element : Element
the_part    : Part
icount      : Integer
weight      : real
Begin
/*********************************Code Start*********************************/
if( first_element == NULL ) then
     exit
endif
the_element = first_element
----------------------------Loop for seek all buffers----------------------------
 while( the_element <> NULL ) do
        if (the_element->element_type == BUFFER) then
          for icount = 1 to the_element->num_out_parts do
             if (the_element->out_parts[1] <> NULL) Then
               the_part = the_element->out_parts[icount]
               weight = weight + the_part->pclass->pweight -- get weight
               --write(weight,cr)
```

```
            endif
          endfor
        endif
     the_element = the_element->next_element
endwhile
---------------------------------finish the code------------------------------
return weight
--write("Final weight = ",weight,cr)
End
/*****************************************************************************/
/****m* scl/track_weight
 * NAME
 *   track_weight
 * SYNOPSIS
 *   * track_weight()
 * FUNCTION
 *   run the simulation for simtime and get number_samples samples of weight
 *   change the consts if needed
 * INPUTS
 *
 * AUTHOR
 *   $Author: GMonteiro $
 * CREATION DATE
 *   $Date: 2015/12/07 11:39:41 $
 * HISTORY
 *   $Revision: 1.9 $
 * SOURCE
 */
Procedure track_weight()
CONS
weight_outfile 'REPORTS\\weight.csv'
number_samples 1000
simtime 2678400
VAR
the_time         : integer
the_weight       : real
sim_step         : real
icount           : Integer
out_stream       : Integer
outfilepath      : String
Begin
/*********************************Code Start*********************************/
--------------------------set the bases of the code--------------------------
outfilepath = libdir + weight_outfile
out_stream = open_output_stream(outfilepath)
the_time = 0
sim_step = (simtime / number_samples)
----------------------------check for inconsistence---------------------------
if (sim_step == 0 OR sim_step < 0 ) then
  write ("FATAL ERROR IN SIM_STEP, PLEASE CHECK!!")
  exit
endif
-----------------------------------first run----------------------------------
  exec_bcl_cmd ("RUN " + str("%g",sim_step))
  the_weight = sample_weight_storage()
  the_time = sim_time
  --write (the_time,the_weight,cr)
  write (#out_stream,the_time,';',the_weight,cr)
-------------------------------get all samples--------------------------------
for icount = 1 to  number_samples by 1 do
```

```
    exec_bcl_cmd("CONTINUE FOR "+ str("%g",sim_step))
    the_weight = sample_weight_storage()
    the_time = sim_time
    if (signal_status( 666 ) ==1 ) then   -- signal 666 was reserved only for this
      --write("CATCH THE ERROR!!!!!",cr)   -- debug only
      --sim_warning "Error in the cranes!!!"
      write (#out_stream,the_time,';',"Storage full!",cr)
      delay 5  -- give time for write the file
      SUSPEND_LOGIC
      exit
    endif
    --write (the_weight,cr)
    write (#out_stream,the_time,';',the_weight,cr)
endfor
---------------------------------finish the code------------------------------
END
/*****************************************************************************/
/****m* scl/calculate_integral_weight()
 * NAME
 *  calculate_integral_weight
 * SYNOPSIS
 *   * calculate_integral_weight()
 * FUNCTION
 *  calculate the discrete integral using the retangle approx for work in progress.
 * INPUTS
 *
 * AUTHOR
 *   $Author: GMonteiro $
 * CREATION DATE
 *   $Date: 2015/12/07 11:39:41 $
 * HISTORY
 *   $Revision: 1.9 $
 * SOURCE
 */
Procedure calculate_integral_weight()
CONST
weight_file 'REPORTS\\weight.csv'
return_file 'REPORTS\\intregral_weight.txt'
VAR
inputfilepath    : String
outputfilepath   : String
curr_line        : String
fail_check       : Integer
time1            : string
weight1          : string
time2            : string
weight2          : string
dt               : integer
dweight          : real
integral         : real
Begin
/*************************************Code Start*********************************/
---------------------------set the bases of the code---------------------------
inputfilepath  = libdir  +  weight_file
outputfilepath = libdir  + return_file
open file inputfilepath for text input as 7
read_line(#7, curr_line)
fail_check = 0
-------------------------------calc the integral-------------------------------
while (curr_line <> $EOF) do
```

```
     read_line(#7, curr_line)
     scan_str(curr_line, ';', time1, weight1)
     read_line(#7, curr_line)
     scan_str(curr_line, ';', time2, weight2)
     if (weight1 == "Storage full!" OR weight1 == "Storage full!") then
                        fail_check = 1
    endif
     dt = val(time2) - val(time1)
     dweight = val(weight2) - val(weight1)
     integral = integral + (dt*dweight)
endwhile
--------------------------------finish the code--------------------------------
close #7 -- close stream channel
open file outputfilepath for text output as 9 -- open file to write
if (fail_check == 0) then
    write (#9,integral,cr) -- write the file
    else
    write (#9,"Storage full!",cr) -- write the file
endif
END
/*****************************************************************************/
```

7.3. Appendix C

Buffer_na	buffer_bin	buffer_section	row	col
Delivery	1	0	0	0
Buff_1	1	1	0	0
Buff_2	1	2	0	0
Buff_3	1	3	0	0
Minibuff1	1	1	0	0
Minibuff2	1	2	0	0
Minibuff3	1	3	0	0
BuffA1	1	1	1	1
BuffA2	1	1	1	2
BuffA3	1	1	1	3
BuffA4	1	1	2	1
BuffA5	1	1	2	2
BuffA6	1	1	2	3
BuffA7	1	1	3	1
BuffA8	1	1	3	2
BuffA9	1	1	3	3
BuffA10	1	1	4	1
BuffA11	1	1	4	2
BuffA12	1	1	4	3
BuffA13	1	1	5	1
BuffA14	1	1	5	2
BuffA15	1	1	5	3
BuffA16	1	1	6	1
BuffA17	1	1	6	2
BuffA18	1	1	6	3
BuffA19	1	1	7	1
BuffA20	1	1	7	2
BuffA21	1	1	7	3
BuffA22	1	1	8	1
BuffA23	1	1	8	2
BuffA24	1	1	8	3
BuffA25	1	1	9	1
BuffA26	1	1	9	2

It should be noted that whole file is not given here. However this appendix is presented to represent the respective file and show how the binary coding is integrated for activating and deactivating lots.

7.4. Appendix D

```
#include <include_str_coordinates.inc>
#include <include_str_objects.inc>
#include <include_str_buffers.inc>
#include <include_lib_dir.inc>
EXTERN
  route_do_unload    : Routine : Integer
  read_buffers       : Routine : List : @Buffers
  open_output_stream : Routine   : Integer
  buffer_rows        : Array [] of Integer
Const
bufferinfilepath 'DATA\\Buffers.csv'
/****f* scl/
 * NAME
 *
 * SYNOPSIS
 *   *
 *   *
 * FUNCTION
 *
 * INPUTS
 *
 * RESULT
 *
 * AUTHOR
 *   $Author: JMoita $
 * CREATION DATE
 *   $Date: 2015/11/17 13:18:21 $
 * HISTORY
 *   $Revision: 1.20 $
 * SOURCE
 */
Routine get_buffer_from_DB(buffer_name : String): @Buffers
VAR
 the_buffer        : @Buffers
 buffers_list      : List : @Buffers
 bufferinfiledir   : String
 current_buffer    : @Buffers
 icount      : Integer
 num_buffers : Integer
Begin
  bufferinfiledir = libdir + bufferinfilepath
  buffers_list = read_buffers (bufferinfiledir)
  num_buffers = list_item_count(buffers_list)
  list_rewind (buffers_list)
  for icount = 1 to num_buffers by 1 do
    current_buffer = list_get_item(buffers_list)
    if (current_buffer->buffer_name == buffer_name) then
      the_buffer = current_buffer
    endif
```

```
  endfor
  return the_buffer
End
/****m* scl/create_a_buffer
 * NAME
 *   Create a Quest source
 * SYNOPSIS
 *   * create_a_buffer(object ; eclasslib )
 *   * create_a_buffer(@Objects ; string )
 * FUNCTION
 *   Create a Quest Buffer
 * INPUTS
 *   * @Object - structure containig the parameters of an object
 *   * @eclasslib - string representing the library containins the object class data
 * AUTHOR
 *   $Author: JMoita $
 * CREATION DATE
 *   $Date: 2015/11/17 13:18:21 $
 * HISTORY
 *   $Revision: 1.20 $
 * SOURCE
 */
Procedure create_a_buffer(object : @Objects ; eclasslib: string )
var
 the_coord_opos    : @Coordinates
 the_size_osize    : @Coordinates
 the_rotation_orot : @Coordinates
 the_buffer        : @Buffers
 buffer_name       : String
 buffer_rows_var   : array[3] of Integer
 bb                : Integer
Begin
  buffer_name = object->oname
  the_coord_opos = &object->opos
  the_rotation_orot = &object->orot
  the_size_osize = &object->osizes
  buffer_rows_var = buffer_rows
      Switch object->odescr
        case 'Stkbuff':
           the_buffer =  get_buffer_from_DB(buffer_name)
           bb = the_buffer->buffer_bin
        switch the_buffer->buffer_section
            case 1:
              if (buffer_rows_var[0] <> -1) then
                  if (the_buffer->row > buffer_rows_var[0]) then
                    bb = 0
                  endif
              endif
            case 2:
              if (buffer_rows_var[1] <> -1) then
                  if (the_buffer->row > buffer_rows_var[1]) then
                    bb = 0
                  endif
```

```
            endif
          case 3:
            if (buffer_rows_var[2] <> -1) then
                if (the_buffer->row > buffer_rows_var[2]) then
                  bb = 0
                endif
            endif
        endswitch
          if (bb <> 0) then
            exec_bcl_cmd("CREATE BUFFER CLASS '" + object->oname + "' NUMBER OF ELEMENTS 0
GEO '" + object->ogeopath + "'")
            exec_bcl_cmd("SAVE ECLASS '" + object->oname + "' TO '" + eclasslib + "in.ecl'")
            set_eclass_dimensions(eclasslib + "in.ecl" , eclasslib + "out.ecl" , the_size_osize )
            exec_bcl_cmd("CREATE ELEMENT '" + object->oname + "_1' OF BUFFER CLASS '" + object-
>oname + "' AT " + str("%g",the_coord_opos->coordx) + ", " + str("%g",the_coord_opos->coordy) + ", " +
str("%g",the_coord_opos->coordz))
            rotate_element (object->oname , the_rotation_orot)
            exec_bcl_cmd("SET DEVICE '" + object->oname + "_1' RENDER TO $" + object->render)
            exec_bcl_cmd("SET '" + object->oname + "' OUT TYPE TO PULL")
                  exec_bcl_cmd("SET '" + object->oname + "' CAPACITY TO 10")
                  exec_bcl_cmd("SET '" + object->oname + "' ROUTE LOGIC TO 'Default Pull Route'")
            exec_bcl_cmd("SET '" + object->oname + "' REQUEST INPUT LOGIC TO 'Default Request
Input'")
          endif
        case 'Buffer':
          write(object->oname,cr)
          the_buffer =  get_buffer_from_DB(buffer_name)
          write (the_buffer->buffer_name,cr)
          if (the_buffer->buffer_bin <> 0) then
            exec_bcl_cmd("CREATE BUFFER CLASS '" + object->oname + "' NUMBER OF ELEMENTS 0
GEO '" + object->ogeopath + "'")
            exec_bcl_cmd("SAVE ECLASS '" + object->oname + "' TO '" + eclasslib + "in.ecl'")
            set_eclass_dimensions(eclasslib + "in.ecl" , eclasslib + "out.ecl" , the_size_osize )
            exec_bcl_cmd("CREATE ELEMENT '" + object->oname + "_1' OF BUFFER CLASS '" + object-
>oname + "' AT " + str("%g",the_coord_opos->coordx) + ", " + str("%g",the_coord_opos->coordy) + ", " +
str("%g",the_coord_opos->coordz))
            rotate_element (object->oname , the_rotation_orot)
            exec_bcl_cmd("SET DEVICE '" + object->oname + "_1' RENDER TO $" + object->render)
          endif
        case 'Minibuffer':
          write(object->oname,cr)
          exec_bcl_cmd("CREATE BUFFER CLASS '" + object->oname + "' NUMBER OF ELEMENTS 0
GEO '" + object->ogeopath + "'")
          exec_bcl_cmd("SAVE ECLASS '" + object->oname + "' TO '" + eclasslib + "in.ecl'")
          set_eclass_dimensions(eclasslib + "in.ecl" , eclasslib + "out.ecl" , the_size_osize )
          exec_bcl_cmd("CREATE ELEMENT '" + object->oname + "_1' OF BUFFER CLASS '" + object-
>oname + "' AT " + str("%g",the_coord_opos->coordx) + ", " + str("%g",the_coord_opos->coordy) + ", " +
str("%g",the_coord_opos->coordz))
          rotate_element (object->oname , the_rotation_orot)
          exec_bcl_cmd("SET DEVICE '" + object->oname + "_1' RENDER TO $" + object->render)
          exec_bcl_cmd("SET '" + object->oname + "' OUT TYPE TO PULL")
         exec_bcl_cmd("SET '" + object->oname + "' IN TYPE TO PULL")
         exec_bcl_cmd("SET '" + object->oname + "' ROUTE LOGIC TO 'Default Pull Route'")
```

```
            exec_bcl_cmd("SET '" + object->oname + "' REQUEST INPUT LOGIC TO 'Default Request Input'")
          DEFAULT :
            exec_bcl_cmd("CREATE BUFFER CLASS '" + object->oname + "' NUMBER OF ELEMENTS 0
GEO '" + object->ogeopath + "'")
            exec_bcl_cmd("SAVE ECLASS '" + object->oname + "' TO '" + eclasslib + "in.ecl'")
            set_eclass_dimensions(eclasslib + "in.ecl" , eclasslib + "out.ecl" , the_size_osize )
            exec_bcl_cmd("CREATE ELEMENT '" + object->oname + "_1' OF BUFFER CLASS '" + object-
>oname + "' AT " + str("%g",the_coord_opos->coordx) + ", " + str("%g",the_coord_opos->coordy) + ", " +
str("%g",the_coord_opos->coordz))
            rotate_element (object->oname , the_rotation_orot)
            exec_bcl_cmd("SET DEVICE '" + object->oname + "_1' RENDER TO $" + object->render)
        Endswitch
   logger("BUFFER element '" + object->oname + "_1' created with success",1)
End
/*******/
/****m* scl/storage_buffer_route_logic
 * NAME
 *   Storage Buffer's route logic
 * SYNOPSIS
 *   * storage_buffer_route_logic()
 * FUNCTION
 *   Is the route logic from the Storage Buffer that will retain the parts of third ship until
 *   the firt ship is done. Then the Storage Buffer will send the parts for preerection.
 *   It uses the signal to sync
 * INPUTS
 * AUTHOR
 *   $Author: JMoita $
 * CREATION DATE
 *   $Date: 2015/11/17 13:18:21 $
 * HISTORY
 *   $Revision: 1.20 $
 * SOURCE
 */
Procedure storage_buffer_route_logic ()
var
   num_unloaded : Integer
Begin
   wait until signal 1 ON
   -- signal 1 means that ship one is done. See 'end_process_logic()' in QB_process
   --Needs to be modificated later. This is not general.
   -- These two lines below are the simple route logic from a buffer
   num_unloaded = route_do_unload()
   route_unload_parts( num_unloaded, 1, 1 )
End
/*******/
/****m* scl/
 * NAME
 *
 * SYNOPSIS
 *   *
 *   *
 * FUNCTION
 *
```

```
 * INPUTS
 *
 * AUTHOR
 *   $Author: JMoita $
 * CREATION DATE
 *   $Date: 2015/11/17 13:18:21 $
 * HISTORY
 *   $Revision: 1.20 $
 * SOURCE
 */
Procedure count_stkbuffers()
CONST
buffer_count_file 'REPORTS\\num_buffer.txt'
VAR
out_stream         : Integer
outfilepath       : String
buffer_count : Integer
the_element : Element
the_buffer        : @Buffers
buffer_name       : String
Begin
outfilepath = libdir + buffer_count_file
out_stream = open_output_stream(outfilepath)
buffer_count = 0
if( first_element == NULL ) then
     exit
endif
the_element = first_element
 while( the_element <> NULL ) do
        if (the_element->element_type == BUFFER) then
          write (the_element->name,cr)
          buffer_count = buffer_count + 1
        endif
     the_element = the_element->next_element
endwhile
write (#out_stream,buffer_count,cr)
END
```

7.5. Appendix E

It should be noted that this is a rather long code controlling the creation of the simulation. Therefore it is shortened to show related sections with the application. The variable determining lines are high lightened.

```
/* Add every include file here */

#include <include_lib_dir.inc>
#include <include_str_ships.inc>
#include <include_str_coordinates.inc>
#include <include_str_contaccessories.inc>
#include <include_str_elements.inc>
#include <include_str_joins.inc>
#include <include_str_element_assembly.inc>
#include <include_str_models.inc>
#include <include_str_objects.inc>
#include <include_str_objects_connection.inc>
#include <include_str_cranes.inc>
#include <include_str_street_nodes.inc>
#include <include_str_street_segments.inc>
#include <include_str_street_paths.inc>
#include <include_str_street_agvs.inc>
#include <include_str_street_decpts.inc>
#include <include_str_scenarios.inc>
#include <include_str_bayplan.inc>
#include <include_str_container.inc>
#include <include_str_Stowingschedule.inc>
#include <include_str_Arrivalcond.inc>
#include <include_str_labors.inc>
#include <include_str_vessels.inc>
#include <include_str_platforms_process.inc>
```

BCL_VAR

bcl_msg : string

--CONST

--$DEF_STR_LEN 500

/* Here include all extern functions to be utilized*/

EXTERN

read_ships : ROUTINE : List : @Ships

read_elements : ROUTINE : List : @Elements

read_joins : ROUTINE : List : @Joins

read_elements_assembly : ROUTINE : List : @Elementassembly

read_models : ROUTINE : List : @Models

read_objects : ROUTINE : List : @Objects

read_objects_connection : ROUTINE : List : @Objectconnection

read_cranes : ROUTINE : List : @Cranes

read_streetnodes : ROUTINE : List : @Streetnodes

read_streetsegments : ROUTINE : List : @Streetsegments

read_streetpaths : ROUTINE : List : @Streetpaths

read_streetagvs : ROUTINE : List : @Streetagvs

read_streetdecpts : ROUTINE : List : @Streetdecpts

read_bayplans : ROUTINE : List : @Bayplans

read_scenarios : Routine : List : @Scenarios

read_containers : ROUTINE : List : @Containers

read_Stowingschedules : ROUTINE : List : @Stowingschedules

read_arrivalconds : ROUTINE : List : @Arrivalconds

read_contaccessories : ROUTINE : List : @Cont_Accessories

read_labors : Routine : List : @Labors

```
read_vessels              : Routine : List : @Vessels
read_platforms_process    : Routine : List : @Platforms_Process
count_nbr_conected_block  : Routine       : Integer


/* Here we define all the global variables*/

Var

  SCHEDULE_name  :  String  -- a global variable for the schedule file name
  buffer_rows    : Array [3] of Integer
--------------------------------------------------------------------------------
/********************************** Main Procedure ***********************************/
--------------------------------------------------------------------------------

/****m* scl/main
 * NAME
 *   Run the main procedure of the software
 * SYNOPSIS
 *   * main()
 *   * main()
 * FUNCTION
 *   Run the main procedure of the software
 * INPUTS
 * RESULT
 * AUTHOR
 *   $Author: GMonteiro $
 * CREATION DATE
 *   $Date: 2015/11/10 16:32:52 $
 * HISTORY
 *   $Revision: 1.98 $
 * SOURCE
 */
```

```
procedure main1()

/* First thing to do is to add all DATA file as const variables*/

Const
  shipinfilepath 'DATA\\Ships.csv'
  elementinfilepath 'DATA\\Elements.csv'
  joininfilepath 'DATA\\Joins.csv'
  elemassinfilepath 'DATA\\Elements_Assembly.csv'
  modelinfilepath 'DATA\\Models.csv'
  objectinfilepath 'DATA\\Objects.csv'
  objconninfilepath 'DATA\\Objects_Connection.csv'
  cranesinfilepath 'DATA\\Cranes.csv'
  streetnodesinfilepath 'DATA\\Street_Nodes.csv'
  streetsegmtsinfilepath 'DATA\\Street_Segments.csv'
  streetpathinfilepath 'DATA\\Street_Paths.csv'
  streetagvsinfilepath 'DATA\\Street_Agvs.csv'
  streetdecptsinfilepath 'DATA\\Street_Decpts.csv'
  bayplansinfilepath 'DATA\\Bayplan.csv'
  scenariosinfilepath 'DATA\\Scenarios.csv'
  schedulepath 'SCHEDULES\\Schedule.DAT'
  containersinfilepath 'DATA\\Container.csv'
  stowingschedulesinfilepath 'DATA\\Stowingschedule.csv'
  arrivalcondsinfilepath 'DATA\\Arrivalcond.csv'
  contaccessoriesinfilepath 'DATA\\Containers_accessories.csv'
  laborsinfilepath 'DATA\\Labors.csv'
  conveyorinfilepath 'DATA\\Conveyors.csv'
  processinfilepath 'DATA\\Process.csv'
  vesselsinfilepath 'DATA\\Vessels.csv'
  platformsprocessinfilepath 'DATA\\Platforms_Process.csv'
[……..]
  modelid 6
  scenarioid 20
```

```
shipyard_year_capacity 40000

sqr_number 100 --model_grid
sqr_length 10 --model_grid
time_interval 3600


    SCHEDULE_name  =  rightstr(schedulepath,len(schedulepath)-10)  -- get the schedule name

    laborgeolib = libdir + geopath + 'DEFAULTS\\labor'
    laborcontrollerlib = libdir + geopath + 'DEFAULTS\\Labor_controller'

    partgeolib = libdir + geopath + 'PARTS\\'

    conveyor_config_file =  libdir + conveyorinfilepath
    process_config_file = libdir + processinfilepath

    buffer_rows[0] = -1
    buffer_rows[1] = -1
    buffer_rows[2] = 1

logger("End reading the databases", 1)
```

[……]

/**********End reading the databases**********/

```
    create_packing_unpacking_process_from_scenariosdb (scenarios_list, scenarioid, 'PackingOutPanPlan1',
'UnpackingInPreErection1', elements_list, joins_list, 'Bloc Assembling', 'S', 1)

    create_packing_unpacking_process_from_scenariosdb (scenarios_list, scenarioid, 'PackingOutpreerection1',
'UnpackingInDryDock1', elements_list, joins_list, 'Bloc Erection', 'B', 1)

    create_packing_unpacking_process_from_scenariosdb (scenarios_list, scenarioid, '',
'UnpackingInPreErection1', elements_list, joins_list, 'Bloc Erection', 'B', 0)
```

```
case 6:
  -- Create the eclasses

  logger("*************************", 1)
  logger ("Creating the elements",1)

  create_pclass_from_scenariosdb(elements_list, scenarios_list, scenarioid, 'P' , 'PC_' , '' , '')

  create_eclass(objects_list,modelid,eclasslib)
  change_source_logic_ssy()

  -- Create the cranes

  logger("*************************", 1)
  logger ("Creating the cranes",1)

  create_cranes_from_db(cranes_list, modelid, eclasslib, objects_list)

  logger("*************************", 1)
  logger ("Setting the connections",1)

  create_connections(objects_list, objconn_list, modelid)

  pfrac_source_pull(elements_list, 'P')


  pfrac_sink_pull(elements_list, 'P')

  exec_bcl_cmd("READ ECLASS FROM 'C:\\simula_cnpq-23-2013\\eclasses\\1_2.ecl'")
  exec_bcl_cmd("READ ECLASS FROM 'C:\\simula_cnpq-23-2013\\eclasses\\1_3.ecl'")
  exec_bcl_cmd("READ ECLASS FROM 'C:\\simula_cnpq-23-2013\\eclasses\\2_2.ecl'")
  exec_bcl_cmd("READ ECLASS FROM 'C:\\simula_cnpq-23-2013\\eclasses\\2_3.ecl'")
```

```
exec_bcl_cmd("READ ECLASS FROM 'C:\\simula_cnpq-23-2013\\eclasses\\3_2.ecl'")

exec_bcl_cmd("READ ECLASS FROM 'C:\\simula_cnpq-23-2013\\eclasses\\3_3.ecl'")


exec_bcl_cmd("SET 'Source2' IAT TO 509")


exec_bcl_cmd("SET 'Sink1' IRT TO 509")


exec_bcl_cmd("CREATE UNLOAD PROCESS 'wait_to_unload'")

exec_bcl_cmd("SET PROCESS 'wait_to_unload' PART PRE_REQUIREMENT TO ANY PART CLASS
1189")

exec_bcl_cmd("SET 'Delivery' UNLOAD PROCESS to 'wait_to_unload'")


  case 7:

    -- Create the pclasses


    logger("**************************", 1)

    logger ("Creating the parts",1)

[……..]
```

7.6. Appendix F

```
REM /*****************************************************************************/
REM This is the batch to control QUEST
REM Just to you know, when i use the command REM, i'm making a documention comment
REM And when i use :: i'm commenting a code line (except with the command IF)
REM/*****************************************************************************/
REM So let's start...
REM This is the new batch for quest
REM It should be more maintanable now
REM/*****************************************************************************/
REM Just for sure, i'm now re-compilling all the SCLs archives from the project
cls
::CALL C:\SIMULA_CNPQ-23-2013\GO_BCL_Compilator_Otimization.bat
::CALL C:\SIMULA_CNPQ-23-2013\GO_Configs.bat
cls
REM/*****************************************************************************/
REM let's start cleaning wheatever we have on the console now
cls
REM/*****************************************************************************/
REM done, now let's get the path where i'm and set all the default paths as variables. NOTE
REM Maybe is needed to change these vars if the location is not the default
cls
:: the location of the batch
::SET var_path=%~dp0
::
:: the location of deneb quest
::SET var_quest_path=C:\deneb\quest
::
::test vars
::ECHO %var_path%
::ECHO %var_quest_path%
```

```
cls
REM/*****************************************************************************/
REM Now let's summon QUEST, and pass all the BCL commands via console, and create a log file in the process.
REM Note that the QUEST program MUST BE INSTALLED IN THE LOCAL "C:\deneb\quest\quest"
REM Note also that this part of the code i just copied from the existing quest.bat
REM since is needed the license for quest work
REM/*****************************************************************************/
REM *Not my code star here*
cls
@echo off

set DENEB_PRODUCT=quest

set DENEB_PATH=C:\deneb

set CNPQ_BCL_PATH=C:\SIMULA_CNPQ-23-2013\BCLMACROS

if "%DENEB_PROD_DIR%" == "" set DENEB_PROD_DIR=%DENEB_PATH%\quest
if "%TMP%" == "" set TMP=C:\tmp
if "%TMPDIR%" == "" set TMPDIR=C:\tmp
if "%LM_LICENSE_FILE%" == "" set LM_LICENSE_FILE=1700@serverlabsen.peno.coppe.ufrj.br
if "%P_SCHEMA%" == "" set P_SCHEMA=%DENEB_PROD_DIR%\parasolid
if "%COP_CONFIG_LIB%" == "" set COP_CONFIG_LIB=%DENEB_PROD_DIR%\COP\

rem The message directory where quest_msg is written &
rem the home directory from where .qpthfig is picked can be set
rem set DELMIA_DEFAULT_MSG_DIR=%TMPDIR%
rem set HOMEDRIVE=C:/
rem set HOMEPATH=deneb

set VIEWER_PROG=C:\Program Files\Internet Explorer\IEXPLORE.EXE
```

```
set DENEB_DOC_VIEWER=%VIEWER_PROG%
%DENEB_PROD_DIR%\docs\%DENEB_PRODUCT%_HOME\HOMEPAGE.html

if "%TMPDIR%" == "" goto TMPDIR_ERROR

if not exist %DENEB_PROD_DIR%\quest.exe goto EXEC_NOT_FOUND

set OLD_PATH=%PATH%

set PATH=%DENEB_PROD_DIR%\lib;%DENEB_PROD_DIR%\bin;%PATH%

rem Set up path for 3d expert

set PATH=%DENEB_PROD_DIR%\3dexpert\intel_a\code\bin;%PATH%

rem Set up script for the UG direct server

if not exist %DENEB_PROD_DIR%\UG\UGDSetup.bat goto SKIP_UG

call %DENEB_PROD_DIR%\UG\UGDSetup.bat

:SKIP_UG

rem Set up script for the CATIA V5 server

if not exist %DENEB_PROD_DIR%\V5\V5Setup.bat goto SKIP_V5

call %DENEB_PROD_DIR%\V5\V5Setup.bat

:SKIP_V5

rem Set up script for the PROE direct server

if not exist %DENEB_PROD_DIR%\PROE\PROEDSetup.bat goto SKIP_PROE

call %DENEB_PROD_DIR%\PROE\PROEDSetup.bat

:SKIP_PROE

copy %CNPQ_BCL_PATH%\start_ot.bcl %DENEB_PATH%\quest\

cd /d %DENEB_PROD_DIR%

echo initializing and running %DENEB_PRODUCT%

%DENEB_PRODUCT%.exe CONFIGS\SIMULA_CNPQ-23-2013 CONFIGS\quest -b <start_ot.bcl>
errorfile.txt
```

```
rem Restore original path

set PATH=%OLD_PATH%
set OLD_PATH=
goto CLEAN_EXIT

:TMPDIR_ERROR
@echo Error using TMPDIR=%TMPDIR%
goto ERROR_EXIT

:EXEC_NOT_FOUND
@echo Cannot find %DENEB_BIN_DIR%\quest.exe
goto ERROR_EXIT

:ERROR_EXIT
pause

:CLEAN_EXIT
cls
```