RTWH AACHEN

Department of operation research

Erasmus master thesis 2013-2014

Development of the application for inventory management, automation and optimization

project report

Author:
Pavlo Zhdanov

Supervisor:
Dr. Marco Lübbecke

June 9, 2014

RWTH AACHEN UNIVERSITY

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Goals and motivation

As a Industrial Engineering student, the author had several purposes and requirements to deal with during the thesis. There were several topics that were found interesting to learn and others that deserve to be understood better. Some of the personal goals were to learn how to make an application in one of the most used languages, learn how to work with databases, especially with SQL, learn about optimization and algorithms, and get better knowledge in the fields of forecasting and stock management.

1.2 Topic selection

On the other hand, there were some academic requirements. The requisite from the home institution is that the topic has to be related to one of the master's relevant field and industrially oriented. Also, the topic must be related to the object of study of the host department for obvious reasons, i.e. containing some kind of optimization algorithm.

Obviously the best idea is to select something useful and at the same time interesting. The inventory management may not be the most original idea, and it is also not the one best suited with the Operation Research Department's original goals, since it is more focused on scheduling and logistics optimization problems with no experience in the field of inventory management.

But on the other hand, the development of an application for the warehouse management is covering quite well all author's personal goals for the thesis, as well as allowing him to acquire a better knowledge in the field of stock management, which is quite important element in all industry-related companies.

Also, due to the expected lack of external support, the author is assuming the

challenge to learn most of the information in the autodidactic way while the department's support would manifest itself when it comes to the optimization part.

1.3 Problem description

Today, every industrial company has to deal at some point with inventories. Inventory of raw material, pre-production, process-stock, pieces at any stage of development and finished goods. The truth is that during the supply chain the same pieces of matter are stored dozens of times in different forms and at different steps of its life, spending a great part of the fabrication period and life cycle in different inventories.

There are several purposes of having an inventory. The main is to reduce waiting time for the next stage of items life which could be the next step on the fabrication process, or customer demand. In the first case absence of inventory stops the process causing severe economical losses due to the paralysis during all steps of production. In the second stage is causes customers demand dissatisfaction, which also means financial losses at short and long-term.

There are also different types of stock according to their purpose: in-factory stocks, distribution centres, centralization or decentralization stock, majority, minority and local stocks. The purpose of this work is to design a self-managed distribution center, able to decide its own strategy and responsible to deal with providers and customers. This model is widely used and in many cases becomes a business model itself.

The costs associated to the inventory can be very elevated, especially in logistic oriented companies, up to the point of becoming the second more important cost after transport costs. A study by IDC Manufacturing Insights points that in some cases it is possible to reduce the stock level up to 25% in one year, or up to 50% in two years [3], therefore reducing the costs.

There are different ways to reduce inventory costs: reduce stock, reduce service-level, increase rotation, ABC policies, supply chain strategies, transport efficiency, packaging efficiency, stock design, automation, smart management, routes and location optimisation, etc...

This thesis will not overcome all of them, but will focus on in-stock times optimization and supply management. This decision is due to the fact that there are a lot of literature about supply chain, safety stocks, and Pareto distribution. Most of them are focusing on value of productions and on costs analysis. However when it comes to inside-warehouse transporting optimisation, it feel like there are not a lot of literature, cause these costs are relatively meaningless, and the "First available slot" policies work pretty well in most cases. Nevertheless, when it comes to huge

inventories with a very high rotation, even small improvement is able to affect the costs significantly.

The in-stock transport routes optimising can not be very relevant by itself, since the energy costs of these transport are nothing compared to global transport costs. But it affects other areas of inventory, leading to the improvement of inventory rotation, service level and times, and capacities of the warehouse. All with the same means and any additional cost, except for the costs associated to the algorithm development and implementation.

1.4 Objectives and structure of the thesis

The main focus of this thesis is implementing a functional application which could be adapted to any real warehouse and could take care of management and automation of its several functions. Also, the application will be used to run multiple simulation experiments, to optimize the functioning and to get some conclusions on the matter. Therefore the structure of this thesis is defined as follows.

Chapter 2 is dedicated to the definition of the problem and the goal, first approach to solve it, literature review of the required knowledge and software selection. Also there is the warehouse design: physical constraints, machinery choices, dimensions, times as well as basic functioning rules and assumptions.

Chapter 3 describes parts of the problem and the adopted solution. It contains an warehouse optimization and automation attempts, including routine tasks, supplies, forecasting and cranes movement algorithm.

In chapter 4 the development of the application is explained: the C# Application, the database background, and the connection and functions of both.

Chapter 5 is about the simulation mode, experiments performed and their analysis. Several tests are performed with variations on algorithm, attempting to improve the efficiency of that, and determine the relevant variables.

Chapter 6 provides a summary, a conclusion and series of reflections.

Chapter 2

Problem approach

Since the main goal of each company is profit, in this thesis it will be attempted to reduce inventory costs. However, since all the characteristics of the warehouse are invented, most of the costs are not easily available. Also, costs estimations are not intended in this thesis and would extend the work excessively. So that, the aim of this thesis is to reduce costs without including any cost into the calculations. Instead, the improvement will be measured and distances and durations reduction.

The objective of this thesis is to provide an application with two ways to reduce costs. On the one hand it provides management support, automatizing all the routine tasks and calculations and therefore also reducing the human resources costs. On the other hand, it will attempt to optimize several tasks, which include supply management, forecast, and in-factory movement management.

The purpose of the thesis is to develop a software which would help to manage, automatize and optimise the inventory. The optimising is meant to be a priority aspect to deal with. The purpose of the optimisation algorithm is to reduce time of retrieving and depositing articles into the warehouse, so that speed up in the high load moments, and optimize the stocks bottleneck, allowing higher stock rack exploitation. The reduction of this time, allows to deposit and withdraw products faster, which allows mayor rotation with same physical means. The mayor rotation with same costs implies an efficiency improvement, and therefore, cost reduction.

2.1 Literature research

Due to the economic importance of the inventories, there are plenty of books and researches done in this field. The greatest part of the literature in this field is focused on supply chain management, localization, supply optimization, safety stock policies, costs. This mostly deals with reducing stock, costs and global times, and are very helpful for the global inventory management part of this thesis, such as forecasting, supplies or experiments design.

Unfortunately, it seems that the in-stock positioning optimisation is not important enough, because it was not possible to find any competent guide to lean on for this thesis. There was found some superficial reports of big companies, and also there are consulting firms providing their consulting and software solutions, but after an exhaustive research, it was not possible to find anything more solid and proper for the in-stock items positioning, which is supposed to be an important part of this thesis. However beside meaning a huge challenge, it also brings the opportunity to get to some new and interesting conclusions.

In this situation, the decision was made to implement the algorithm based on the criteria seen in the other literature, trying to apply them with common sense, and discover the relevant variables and policies with exhaustive simulations analysis. Another parts of this thesis are not supposed to be so important, as far as it is not intended to bring nothing revolutionary new there. Therefore basically they will assume an implementation and configuration of well known mechanisms on practice.

Competition analysis

As mentioned above it was quite impossible to find any competent literature help when it comes to the field of internal stock logistics. There were made an information gathering around the companies offering similar service. First of them: "AZAP Procurement & stock optimization". To optimize the stock functioning they take into account the forecast demand, storage constraints(receipt, capacity) and supplier constraints(price, scheduling, etc). The next step is to generate an automatic forward supplies and stock plan, place some specific alerts and place a supply orders. Also they offer a complete range of functions that makes procurement agents' work easier and more effective: stock management according to the desired service-level; restocking frequency, limit and amounts; compliance with supplier's constraints(batch, minimum order) and warehouse's ones(capacity, structure); management and optimisation of stock, supply and transport costs.

Another company taken into account is "SysPro", which defines 4 important groups of actions.

1. CATEGORIZE PRODUCTS.
   Analyse and separate stock items according to their impact on profitability, revenue generation, historical supply and demand behaviour. This identifies critical items that require detailed forecasting and more attention then other items.

2. FORECASTING.
   The forecasting process enables an organization to forecast customer demand by item, making the future less unpredictable and being able to react appropriate. This process should be fine-tuned over time, using actual past data to ensure future forecast accuracy.

3. DETERMINING STOCK POLICIES.
   Stock policies are modelled to determine the best balance between customer service and inventory investment. A key outcome of the policy is the quantity of safety stock stored in the warehouse this is required to ride-out unpredictable fluctuations in demand caused by events such as strikes or natural disasters.

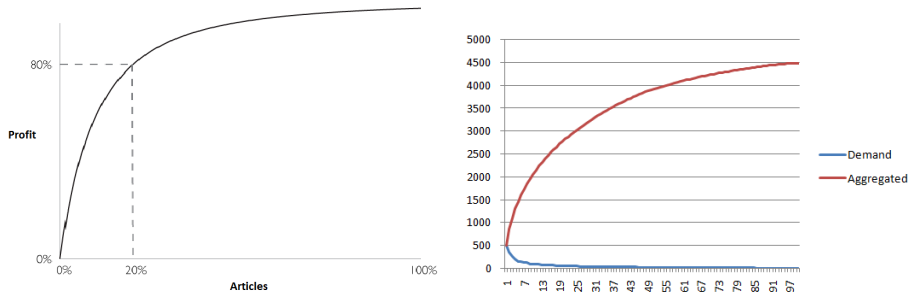
4. REPLENISH ACCORDING TO PLAN.
   Software solutions are available to ensure the timely replenishment of stock according to the forecasting policy.

This last firm also provides a free stock management manual about inventory optimization with some useful advices, but unfortunately without clues about their in-stock movements and placements policies.

ABC inventory policies

As in most of economic activities, in stock management the Pareto Principle, or the rule of 80% - 20% is widely applied. This rule was first mentioned by an Italian economist Vilfredo Pareto in 1906 to manifest the unequal distribution of wealth in his country. The 80% of wealth in Italy in 1906 was owned by just 20% of population. Curiously, this relation is still applicable nowadays and it can be seen in many situations.[8]

Of course the important side there is to differentiate the items according to their financial value in the enterprise, which is directly proportional to the demand, establish different treatment for each group of them, and threat them according to the their importance. For this purpose the ABC inventory categorization was invented. There are no strict rules, for example A group could contain 10% of items resulting in 70% of value, group B 20% of items resulting is 25% of value, and group C for the rest of items. So that, the most important articles from group A deserves more attention and more stock, since they are generating the most of income, and they would generate the most losses in case of break or customer dissatisfaction.



(a) 80-20 rule (b) Warehouse demand

Figure 2.1: Pareto distribution of demand.

ABC classification allows categorize products, and threat them according to their financial importance. Unfortunately the ABC system doesn't affect directly to the in stock-times reduction, since that times doesn't rely on costs or quantities, but on frequency and rotation index. However, a proper stock management and classification can reduce the amount of necessary stock in the inventory, which indirectly would reduce the withdrawal and deposit times.

Queues theory

Queues theory seems the most adaptable to design an algorithm for the cranes. It helps to predict the queue length and waiting time. However, it still quite far from this thesis' objectives, since changing the schedule doesn't directly affect the results, because the distance and the combination are much more important than the order of movements.

There are several possible policies for queue order:

- First Come First Served (FCFS) o First In First Out (FIFO)
- Last Come First Served (LCFS) o Last In First Out (LIFO)
- Service In Random Order (SIRO)
- Priority based order.
- Shortest job first. It is a legit policy, but it does allow some items get stuck inside the queue. Ordering queues by deadline could avoid it, or rather the use of Highest response ratio first.
- Highest response ratio first HRR. Priority = $\frac{time\_waiting+time\_run}{time\_run}$
- Fastest work end.

In the case of this thesis there is only one one queue and multiples servers. However, three different cases of queues can be distinguished, only one or another to be applied at same time.

1. Only withdrawal. ($deposit\ queue = 0$)
2. Only deposit. ($withdrawal\ queue = 0$)
3. Both deposit and withdrawal are needed. (combined trips)

In the case of WITHDRAWAL ONLY, when there are different articles in the queue, the best option could be FIFO, in order to properly fulfil the deadlines of withdrawal and avoid resource starvation. Another way is using AGING penalization , gradually

increasing priority of task of item, proportional to its time spent waiting in the queue or inversely proportional to their deadline. The order of withdrawals doesn't affect durations too much. What does really matter, is that the service time for same client are different in each server and varies from one moment to another. To select the server there will be used a highest response ratio, taking into account the best service time of each server, and the waiting queue on each server.

Same process is applied for DEPOSIT ONLY queues, but in this case there is no deadline to deposit, so pure HRR can be used. It is necessary to mention, that the shortest time for each crane is not necessarily the closest slot, since deposit has to fulfil with some constrains, like priority zone or corridor balance, which will be explained in the next part. There is no need to avoid resource starvation in this case, since if the temporary stock never empties, it doesn't matter what articles are in there.

The most complicated case is when both deposit and withdrawal queues aren't empty. It will be necessary to apply a priority algorithm with several elements to determine the sequence of movements.

David G. Kendall introduced a notation "A/B/C" for queuing problems in 1953. Later this notation was extended to A, B, X, Y, Z, V (1/2/3/(4/5/6))[12]. According to this notation, out problem could be defines as.

- A = distribution of arrivals (G, general)
- B = distribution of service (M, exponential)
- X = number of servers (5)
- Y = capacity of system
- Z = policy
- V = number of service states (0)

It is a non pre-emptive algorithm since it is not possible to interrupt and reassign the task. No processor sharing also. The demand is stochastic but depending on each separate product. And so is the distribution of service, which depends on warehouse level, and each product rotation. The algorithm is mean to be deterministic, there is an optimal value to be found, which remains the same in the same situation.

$$\lambda_{withdrawal} = \frac{20000}{28days*12hours} = 60\frac{pallets}{hours}$$

$$\lambda_{arrival} = \lambda_{withdrawal}$$

$$\lambda_{total} = 120\frac{services}{hour}$$

$$\frac{1}{\mu} = 40s \text{ (average time of single trip)}$$

$$\mu_{w,a} = 90\frac{services}{hour}$$

That makes impossible proper service, the process would be congested and the queues would tend to infinite. Luckily, there is the option of combined trips (deposit+withdrawal). That action has an average of 55 seconds according to the experiments, but takes two elements out of the queues. Therefore the time per action would be of 27.5 seconds. Therefore

$$\mu_{w+a} = 130.9\frac{services}{hour}$$

That mean, that there are no infinite queues, but actually it doesn't give a full picture of warehouses capacity, since in the high-load moments arrival rate will be higher then service rate, making necessary a pre-stock. The time of arrivals or withdrawals is uncertain, because the external transport schedule is not known with absolute precision. That makes impossible to make a load estimation during all the schedule. Instead there will be periods when cranes will not work, and peak times, when they will be totally busy.

Simulations

The simulation as a quantitative method has many advantages: its easy to understand, allows to recreate a complex problems without needs to simplify, it is flexible, and it allows to analyse the behaviour of the model in the real time. As the opposite, the simulation never returns an optimal value, and doesn't prove that there can't be better results or algorithms.

There are 6 basic steps to work with simulations [4]:

1. Definition of the problem
2. Construction of the simulation model
3. Test and validation of the model
4. Experiment design
5. Experiment
6. Analysis of results

The process is described better in the next chapter, and so are the inputs and outputs. The model is constructed as real as possible including the process that might affect the algorithm efficiency. However, many processes of warehouse that don't affect directly the algorithm efficiency are simplified.

During the validation it is important to observe the right functioning of warehouse by means of SQL tables. The factors to watch and the experiments are explained in the Simulation chapter.

2.2 Software selection

2.2.1 Coding language and compiler

Learning of program language is one of the main objectives of the thesis and coding of the application is going to be the part which most time require. Therefore there was an important selection between several possible options. Disposing of some knowledge of C and Basic, the choice was leaded toward other C based languages, although other options were considered. Also, author had a desire to use Visual Studio as a compiler, since it is free for students, provides tons of functionality and there are a lot of useful information on-line.

- Visual Basic. Since author already made applications in Basic, that would be the simplest choice, that was refused in favour of learning some other, more complex language.
- C. It still being the most used language such as Java, but the procedure-oriented code has several limitations and difficult codding comparing to object oriented languages. Also despite being the most used, C language is getting older and being substituted with other many superset languages, that allows to do the same, but also to do other things and to make it easier. Also it is not recommended at all for beginners.
- C++. Very complex and powerful language, but kind of hard and slow to start with.
- C#. It is faster and easier to learn and implement, good for developing on Windows and provides some of the best features from other languages.

  > "C# belongs to the Microsofts family of programming language and was developed in 2000 to be the part of the first ever release of Microsofts prestigious .NET framework. C# is very similar to Java in terms of capabilities. It has been said that C# combines the robustness of C++ with the advanced features of Java. Therefore, if you are good at Java, it is extremely easy to switch to C# and vice-versa[3].

- Java. As pros, it is object-oriented and it is the most widely used language, and there are a lot of jobs for Java programmers. As cons, a bit difficult and not so flexible as that could be C based languages, so that it needs more time to learn, and more time to code.

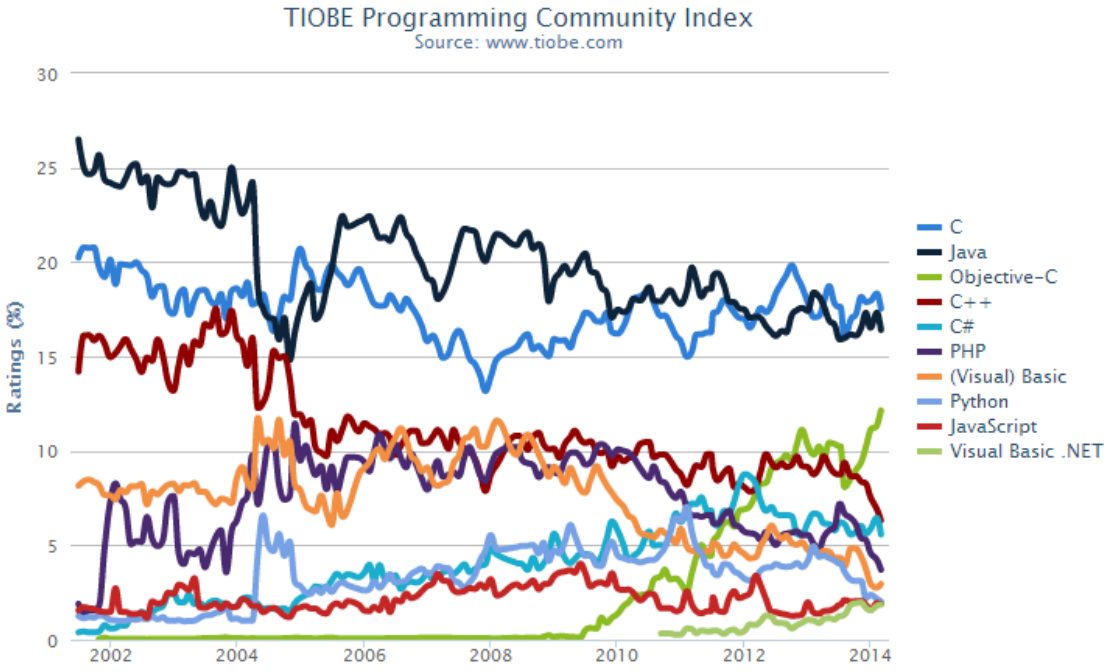


Figure 2.2: Evolution of usage of programming languages

Initially it was chosen to use C++, but after several sessions, the decision was changed in favour of C#. It seems that Microsoft onjective is to promote C# or Objective-C languages, limiting C++ features in latest versions of VB, making unable some necessary for the thesis functions. Moreover, C# seems faster to learn and to code in, and it is a good language to start with, in order to learn C++ or Java later.

2.2.2 Database

The list of considered applications include MS Access, MS Sql Server, and MySQL. Since the author never worked with a database, there were no enough knowledge to choose which one is more suited for the application. The SQL choice was obvious due to lots of applications using it. MySQL is a very robust and widely used SQL server, the second open-source relational database management system and it is one of most recommended programs for SQL, with a lot of help and information on-line, and quite good relationshit with C#.

2.3 Warehouse design

Due to the practical focus of the thesis, the ideal option would be to have some real warehouse data and dimensions. But all the attempts of getting such data was failed, so with tutors consent and to avoid losing much more time searching for a data, the choice was made to generate or invent author himself most of the data.

Since the thesis author lacks any sense of imagination, it is possible that this thesis may content a lot of round numbers. It must be said that since the data is invented, the exact numbers are not excessively relevant. They should not affect the experiment conclusion, and they may be changed to adjust the application for any real inventory. After a short investigation on similar warehouses, the main stock of the warehouse was chosen to have the following dimensions and characteristics:

- Length (in slots): 100
- Height (in slots): 10
- Width (in slots): 10 (rows)

Therefore the stock has a total of ten thousands of slots. The dimension in meters of the stock are calculated based on the slot dimensions in the next section.

There are 5 corridors between rows, each with access to two contiguous rows. Each corridor needs one stacker crane, attending two rows each crane.

The temporary stocks are a necessary element for any warehouse, but its calculation is out of scoop of this thesis.

2.3.1 Stock

Both European pallet and ISO pallet will be used in the warehouse. All the rack gaps need to be able to accommodate all of them, so it is necessary use the same stock dimensions for all of them. Possible dimensions of pallets:

- EUR 1200x800mm
- EUR 1200x1000mm
- ISO 1100x1100mm

The maximum length is 1200 mm and the maximum width is 1100mm. The maximum height agreed for the pallets is of 1100mm maximum. So the dimensions of rack gaps should be 1200x1100x1100mm, plus an extra space for safety and manipulation. Normally it is about 5cm in each direction, but these features will be specified according to selected rack structure and the features of the used stacker crane.

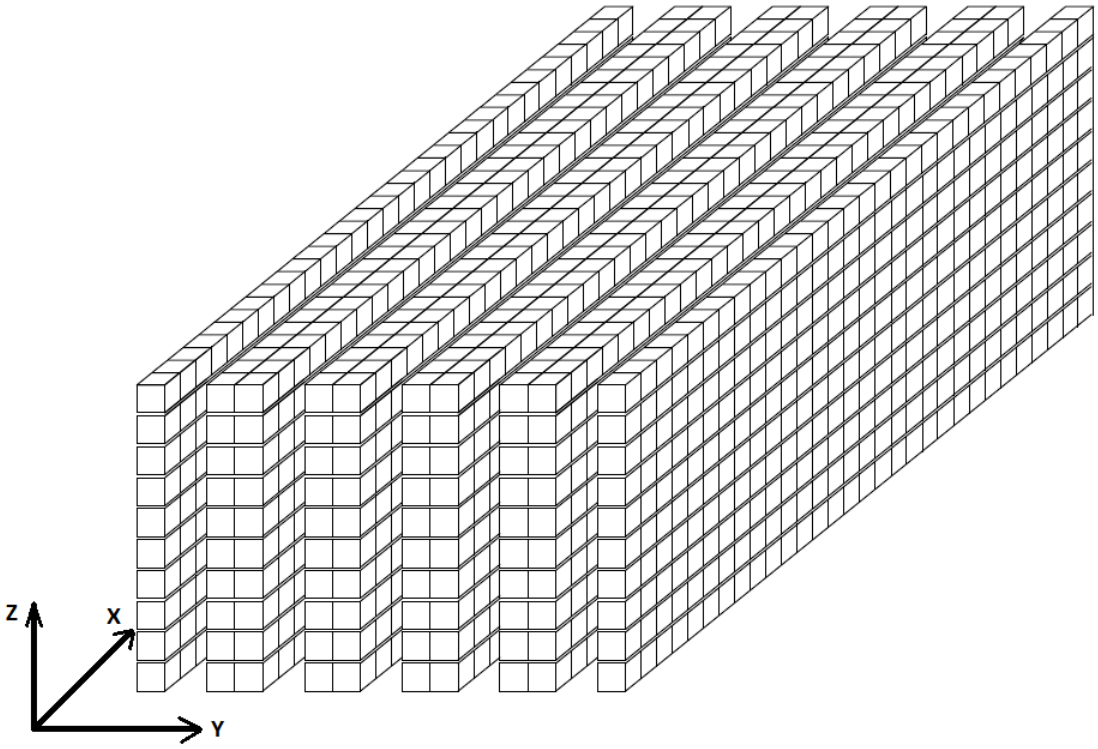


Figure 2.3: Inventory rack's sketch.

Therefore the space considered for each pallet is of 1.30 m x 1.25 m x 1.2 m (X,Y,Z), providing space for the structure of the stock rack, and allowing all the supported types of pallets. The width of corridors is about 1.5 meters warranting a safe passage of cranes with pallets. Therefore the overall dimensions of the stock structure are 130 x 20 x 12 meters.

2.3.2 Machinery

The effectiveness of the application of this thesis is maximized when all the components are automatized. So that, this software could be a "bridge" between an ERP system such as SAP, and the automate software in charge of all the warehouse hardware and machinery

The following elements are necessary for the functioning of automatic warehouse:

1. Stacker cranes to access the stock
2. Conveyor belts to move items between load/unload area, temporary stock and main stock
3. Bar-code readers connected to the developed application

Stacker cranes

The choice of stacker crane shouldn't be relevant for the purpose of the thesis. After inspecting few models on market, the following system was selected. [12]

Properties:

1. Max load up to 3000 kg
2. Axis X speed up to 4,2 m/s
3. Axis Z speed up to 1,2 m/s
4. Height: up to 36 meters
5. Only one axe moves at once.
6. One pallet at once.

2.3.3 Movements duration

Taking into account the speeds and dimensions, the time spent in each trip is calculated by the following equation:

$$Ta = X_a * \frac{X_u}{s_x} + Z_a * \frac{Z_u}{s_z} \tag{2.1}$$

where T_a = time to store in slot "a", X_a is the position of slot in axis X (depth, measured in slots), Z_a in the axis Z (hight), $s_{x,z}$ is the speed in these axes, and X_u, Z_u are the dimension of unitary slot in each axe.

The crane can only move in one axe at once. The time for deposit/withdrawal from any slot operation is of 8 seconds. The time for load/unload pallet at the start is of 3 sec. Therefore, the total time needed for an withdrawal/deposit operation is of

$$Ta = (X_a * \frac{X_u}{s_x} + Z_a * \frac{Z_u}{s_z}) * 2 + 8 + 3 \tag{2.2}$$

For the withdrawal+deposit trips the time is of:

$$Ti = T_a + T_b + (X_a - X_b) * \frac{X_u}{s_x} + (Z_a - Z_b) * \frac{Z_u}{s_z} + 3 * 2 + 8 * 2 \tag{2.3}$$

slot_id	row	depth	hight	crane	total_time	priority	occupied	reference	other	safety_stock
07631	A	66	9	1	34.4500	8	1	933		NULL
07632	B	66	9	1	34.4500	8	1	933		NULL
07633	C	66	9	2	34.4500	8	0			NULL
07634	D	66	9	2	34.4500	8	1	934		NULL
07635	E	66	9	3	34.4500	8	1	934		NULL
07636	F	66	9	3	34.4500	8	1	934		NULL
07637	G	66	9	4	34.4500	8	0			NULL
07638	H	66	9	4	34.4500	8	1	101		NULL

Figure 2.4: Times and other information in DB table "stock".

2.3.4 Articles

The warehouse will start operating with 100 different articles (in order to keep making all quantities round and understandable). However another articles could be added at any time.

The demand behaviour for each reference is invented by the author attempting to make it the most realistic and diverse possible. The only thing to keep in mind is that the inventory is considered high rotation, so the orders will be with frequency of no more than two weeks.

The demand patterns varies in quantity, frequency and trend. The following patterns and their combinations are possible:

- PROGRAMMED WEEKLY (1-7 days per week, similar amounts).
- NOT PROGRAMMED (similar amounts with similar intervals)
- Reasonably similar quantities on random date with large intervals.
- Stable, slowly raising or decreasing trend.
- NEW STAR articles, fast demand (and frequency) increment.
- DYING ARTICLES, decreasing of demand up to its extinction.
- COWS, high demand at stable level.

Seasonality is not taken into account since there are no data enough available. Also the high-rotation stock are not supposed to accommodate out of season products.

The demand will follow the Pareto distribution, and fit into ABC model, so it can be treated like that for the supply policies. For the analysis purposes, the articles

are named according to their turnover quantities, the lower is the number of article, the most important is its demand. However, since demand level of article may vary during the period of experiment, these numbering is only to give a general idea of what kind of item is, during monitoring SQL tables.

2.3.5 Flow rules

The flow course of articles in the warehouse begins with their entrance. All the articles are coming being packed in pallets, according to the size agreements. Several trucks are able to unload the incoming items in the unloading docks.

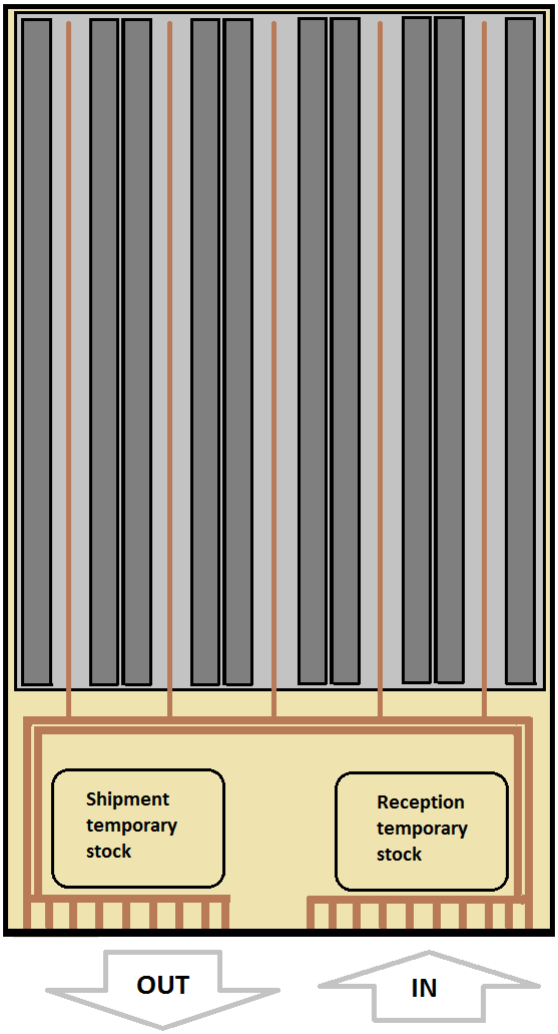


Figure 2.5: Drawing of the warehouse

After being unloaded from the trucks, all the articles are automatically placed on conveyor belts, which bring them to the temporary stock. On their way there, all the articles are identified by bar-code readers, so that allows to know at every moment the position of each container.

In practice all the warehouses has one zone for shipment and one for receiving items aside from the main inventory. That zones are designed to make a faster load-/unload on trucks, without need to wait till they are completely stored in the main stock, which takes more time and makes a bottleneck. This reduced the non productive time of external transport, and thereby the attached transportation costs, which are usually the higher costs of all the logistics chain. Both of these zones are considered for present warehouse, but without getting into a deep calculations and optimisation of them, since they are not the bottleneck of the process.

Once the article is identified, he is placed into ARRIVAL QUEUE in the database, while it is staying in the temporary stock.

The articles in ARRIVAL QUEUE are waiting to be assigned to the STACKER CRANES QUEUE, and once they are assigned to a crane trip, they leave the temporary stock and by means of conveyor belts they are directed to the selected CRANE QUEUE, depending on which row they have an assigned place.

The crane that deposits an item into the stock, normally (if needed) will retrieve another item from stock, to save time in another trip. The items to withdraw are picked from WITHDRAWAL QUEUE, which is filled according to orders received from the customers, and estimated time of empty trucks arriving to the warehouse.

Chapter 3

Solution description

3.1 Process description

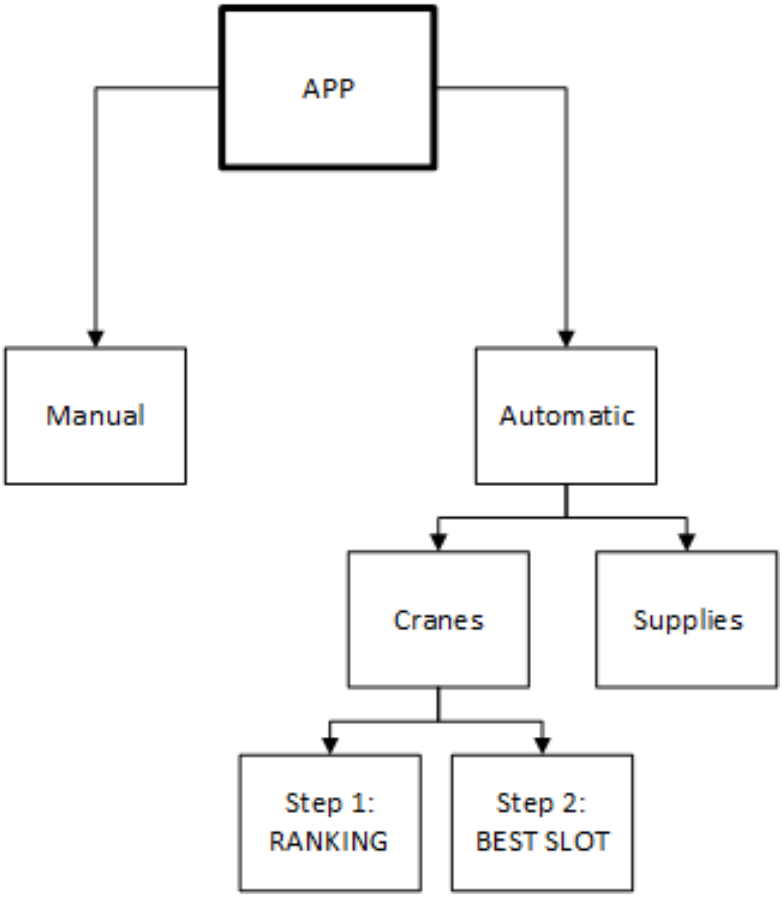


Figure 3.1: Applications parts.

The application consist of two main function: manual and automatic. The manual function is about communication with user through visual interface and allows to make decisions by his own. It allows user to manage suppliers, customers, articles and to note orders and place procurement orders. It also provides necessary information to make decisions.

The automatic process is able to manage stock by itself. This process consists of two main sub-processes. First sub-process controls the stacker cranes movements trying to optimize the withdrawal/deposit routes and times. The second sub-process is aimed to realise a global inventory control by controlling customer orders, procurements and stock load. Its goal is to make proper supply orders and keep stock at minimum necessary level.

3.1.1 Inputs and Outputs

There are several inputs to the system. The LEVEL 1 inputs are receiving goods from the supplier, or receiving an immediate request to withdraw pallets and send them away. The output for these inputs is the crane schedule with all the movements needed to fulfil the requests. These route is processed by the main process of the system, the algorithm and ranking system intended to optimize crane times. There are also a feedback path: inventory and cranes real-time state is updated due to the process, and this updated image are used again by the process to make decisions.

LEVEL 2 inputs are processed by another sub process. The entrance are customer orders. The process makes a statistical analysis, forecast, and releases supply orders as an output. All the information gathered during the process is also considered as an output and allows managers to use it to make some decisions.

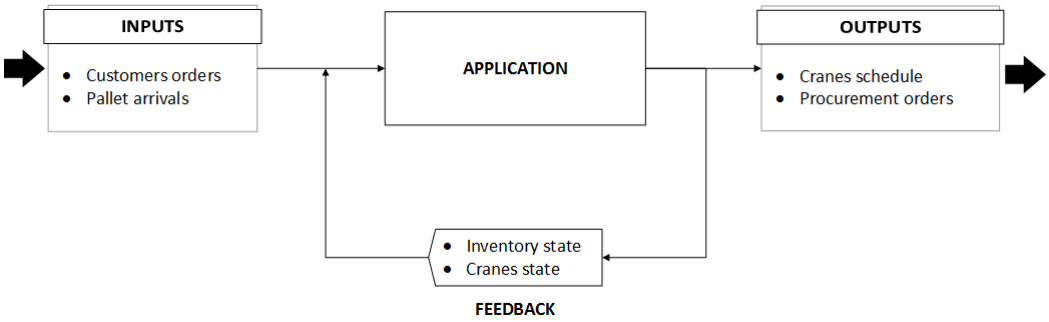


Figure 3.2: Inputs/outputs diagram.

3.2 Forecasting

As in every single economical activity, in stock management it is quite important to know as much as possible about the future. If the prediction is wrong, there might be severe financial consequences due to physical constraints and lack of means, or on contrary due to a continuous waste of that means due to over-dimensioning.

The forecasting part was resulting interesting to the author due to its hight market demand and wideness of application all over the work market. It is a huge tool, since in most cases when it deals with economical activities, the future is never certain. Also it was aimed to be an important part of the application.

On the other hand, having no access to real statistic data incapacitates to make a right judgement about the efficiency of forecasting, so finally the decision was made not to put too much effort in this part.

Moving average series should be enough for the purpose of this thesis. Improved with weekly seasonality, it allows to comprehend quite well the demand pattern and variation of articles. The yearly seasonality is disables due to lack of that long historic data and the very dynamic rotation of the warehouse.

There are two purpose of forecasting in this application.

- First, for the proper replenish program and good service-level.
- Second, for the items priority estimation.

The data from outcomes for past 56 days will be used for the prediction. Then, a 7-day moving average is used to define the trend, and make a linear forecasting for 14 days in advance.

The weekly component of demand is analysed separately to find the most and less probable days for demand.

Finally the trend and seasonality are aggregated together to display a daily forecast for next two weeks.

This forecast is enough for the short term predictions in a high-rotation stock. To deal with non-linear trends, the easy way is reducing historical data till it becomes linear in the short term.

On the other hand there might be articles with total or partial orders schedule agreed with the supplier. To meet with the service-level requirements and to provide more security, the higher value of forecasting and ensured orders is selected for the supplies calculation.

3.3 Supplies

There are two basic policies for supplies management: by replenishment level and by periodic replenishment. In order to make the application more dynamic, the management based on replenishment level is selected, despite not knowing the costs, which would allow to calculate the optimal supply batch quantity. Instead the optimal supply batch agreed with the providers is considered. This strategy is more secure, dynamic and more suitable for the high rotation stock. The disadvantage of placing triggers and continuous stock control is nullified by software automated management.

Although this warehouse doesn't deal with costs, it is interesting to make it as much realistic as possible. Therefore some constraints are included for each reference. There are values of MINIMUM ORDER and ECONOMIC BATCH set for each reference, which prevent the supply algorithm from cheating, and making some impossible in real life, even being optimal, replenishments order.

3.3.1 Procurement orders

The function of supplies is to reduce logistics costs attributed to the storage. Since the costs of transports and storage are not considered in this thesis, the objective of supplies is limited to keep storage level at minimum possible levels, considering constraints such as minimum order quantity, batch allowed and procurement time, and without violating the customer service-level agreement.

The constraints considered are:

- Minimum order quantity is the minimum quantity which provider allows to order. Each article has one according to its demand.
- Optimal batch quantity is the number of reference which is easier to order dye to packing, or transporting facilities. It is normally the cheapest option.
- Service rate = 100%. Reliability of suppliers is not considered in this thesis.
- Delivery time = around 4 days. The orders will be received on the fourth day in random time during the working time of stock.
- Service-level is of 3σ for category A products, and 2σ for B and C categories.

The procurement order is a triggered event. Every few minutes, there is a check of stock realised. To estimate the need of supplies, the values of current stock, predicted demand during the replenishment period, and supplies in the road are taken into account. A supply order is needed when:

$$I =< Max(Oi, Fi) + SS - Si \tag{3.1}$$

The amount ordered is:

$$S = \max(\max(Oi, Fi) - (Ii - SSi) - S_i; Sm) \tag{3.2}$$

where i = time of replenishment, O = ensured orders, F = forecast value, I = current inventory, SS = security stock, Si = supplies in process and Sm = minimum supply quantity.

3.3.2 Safety stock

Safety stock depends on required customer service level, on demand variation and on suppliers times and reliability. The purpose of safety stock is to reach the agreed customer service level, with the smallest amount of stock as possible. This part of stock has much lower rotation that the main stock, so that, it can be treated a different way that the main stock.

Here is where Pareto and ABC stocks model gain importance. Classifying all the articles into A-B-C categories, according to the demand quantities, is the only step in this project slightly related to the costs. Since the products of group A suppose the greatest part of incomes, they also deserve the greatest part of costs, since it is most important to avoid stock ruptures in these articles. Therefore a security level of 99% will be used for objects of group A, and a 95% level for the rest of the objects. This implies a higher safety stock for the objects of group A.

z	Service level
1.28	0.90
1.65	0.95
2.05	0.98
2.33	0.99
3.00	0.9986
4.00	0.9999

Table 3.1: Service level.

Standard deviation of demand:

$$\sigma = \sqrt{\sum_{n=1} \frac{(D_n - \overline{D})^2}{N-1}}. \tag{3.3}$$

Therefore, the safety stock will be recalculated periodically for each article according to the following equations:

$$SS_{B,C} = z_{95\%} * \sigma = 1.65 * \sigma \tag{3.4}$$

$$SS_A = z_{99\%} * \sigma = 2.33 * \sigma \tag{3.5}$$

where N = sample size, D = demand, σ = deviation, SS = safety stock and z = service level constraint.

3.4 Algorithm

The bottleneck of the warehouse system is normally situated in the cranes, because it is the most rigid part and its capacity is lower than of the rest of the system. Thus, when the rotation raises, the servers (cranes) capacity is saturated, becoming a bottleneck of the system.

The objective of algorithm is to optimize the routes of crane and to reduce the average time per trip. That increments the capacity of the cranes, and therefore, of the whole system.

The problem of working with real-time optimization is that is quite difficult to find an optimal value or sequence due to the very limited time for each decision, and to the constantly changing constraints. That means, not impossible, but worthless spend too much time looking for an optimal sequence, which will no longer be optimal after any of frequent external inputs. In this problem, to find a truly optimal value would be necessary to consider all the future known actions. So that would be possible to find an optimal sequence of movements for that known period of time. But the dynamic environment of the warehouse and the inexactitude of times does not allow that, since it would invalidate the sequences too often. Therefore it is probably better to not consider all the data (which also makes an analysis for each operation insanely long), instead it is better to order and choose some items first based on another arbitrary factor, such as deadline, without taking it into the calculations. And starting from that point the possibilities are more limited, and it is possible to find an optimal value (according to the algorithm) for one single trip.

Summarizing, there is no imperious need of finding optimal sequences, which would be very simple in static environments, such as making the production plan for the next week or some logistics problem. Instead the choice is taken to find just optimal singular trips, restricted by constraints(ranking, balance) that will allow to improve the sequence time, making the trips optimized for a long term.

3.4.1 The approach

To reduce the time spent by cranes, there are taken two different approaches.

On one side, STEP 1 or PHASE 1 is providing an objective categorizing of inventory products, in order to determine the behaviour of each different item to deal

with. That allows to threat each item in a different way and deposit it in the different places. That results in a classification of inventory items into several groups, which will affect their place to be deposited. This is a *long-term policy*, which may increase a short term movements duration, but after all the items inside warehouse are well sorted, the average times might be reduced.

On the other side, the STEP 2 takes into account the immediate factors, such as current cranes situation and stock state. Finally, based on several criteria and restrictions, such as ranking, stock load, cranes load and others, it chooses an optimal route for the next trip. The selection is made comparing all the possible options by means of SQL queries and C# commands.

3.4.2 Step One: Ranking

As it has been commented, when it concerns to classification of inventory, it's common to hear about Pareto distribution and the ABC stock policies. However, when it comes to reduce the in-stock times, the costs means nothing. And apparently, so do the demand levels. In first simulations the mistake was done of attempting to classify and order references according to their "value" for the company, which is based on their sells. But after a closer look at this, it doesn't make so much sense. Regarding the time reduction, it doesn't matter what is the demand and stock for certain reference. A reference with huge demand and high cost importance for the company has the same behaviour as a group of less important references. In other words, each pallet has to be treated according to its own importance, without being influenced by other pallets of same reference.

Logically, it is more important to consider the frequency of orders, and to a lesser extent the average stock, supplies frequency and rotation of the references. Considering that the stock, object of this thesis, is a high rotation inventory with static order.

In order to classify the inventory it is necessary to separate them into groups. The number of groups is irrelevant, thus to ease the process and make calculations more comprehensive, there will be a round number of 10 groups, grouped by percentile of 10%. That means that in a completely full stock each sequent 10% of items will belong to different group. That groups will be called RANKING GROUP, where 1 is the highest and the most important rank with more rotation and orders frequency, and 10 the lowest, for references with occasional and infrequent orders.

On the other side, the inventory slots will be divided in 10 group by percentile of 10%, according to the access time, time spent by crane to get from the entrance of stock rack to needed slot. That groups will be called PRIORITY GROUPS in order to distinguish them somehow prom RANKING GROUPS. These PRIORITY GROUPS are static and they never change.

However the RANKING GROUP are constantly recalculated. Depending of the

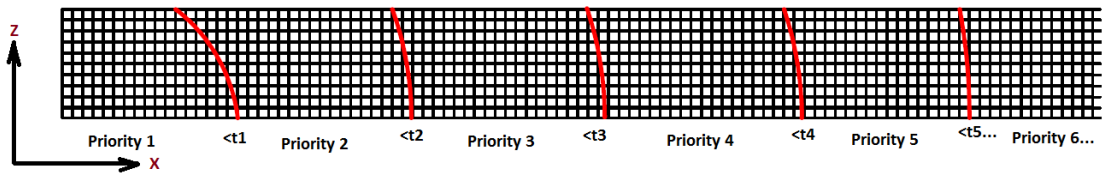


Figure 3.3: Priority zones of inventory.

level of activity, and of the stock level, there may be not all the RANKING GROUP used. In any case it doesn't mean that the product from ranking group n will belong to the priority group n. The final decision depends on more factor, as it will appear below. In practice it is impossible to split items in groups by percentile of 10 making a clean cut, and also the warehouse is never completely full, so sometimes it doesn't make sense to use all of the priority groups. There will be needed some rules and adjustments to work with this ranking system.

The purpose of classification items and inventory in groups is to assign for each item a zone in the warehouse. The items with higher rotation will be withdrawn and deposited more often, so it worth to place them closer to the entrance, which obviously will reduce global times.

But this policy has its counterpart. In order to place item with high ranking in the high priority slots, there is a need to maintain an amount of these slots reserved for these items. Which means that between withdrawal and replenishment these slots will be empty. That is an issue, since it will be obliged to place items in worse position, even being available better slots for them. This fact involves that not the optimal time will be chosen. In fact, the short-term optimality will be sacrificed in favour of long-term policies. That means that there will be a waste of time while depositing low references far away, but it will be compensated with benefits when it will come to deposits and withdrawals high-ranking references. However, it's important to maintain the balance between both and make sure to find an optimal point analysing simulation results.

3.4.3 First approach

If there was strict-scheduled warehouse, the future could be considered as certain, and there could be a way to realize a more complex, but much more efficient algorithm. However, with uncertain future, it is not possible to do that, so it is necessary to use several criteria to build a priority system, which may make more or less accurate decisions.

In the first consideration to the algorithm, to calculate the ranking of each reference there are several criteria to take into account:

- The rotation of item
- The trend or tendency of item demand
- The relative importance of item stock regarding the total stock
- The last withdrawal or order
- Future demand: forecast + ensured orders
- The relative position (rank) or item in the stock
- Number of references in warehouse
- Stock per reference in warehouse
- Average stock per reference
- Stock total
- Average stock total
- Last weeks demand
- Last weeks supplies

Some of these criteria was discarded soon enough, while other factors became more important:

- Frequency and rotation. The more rotation and withdrawal frequency has an item, the more trips it requires, the shorter might be that trips.
- Free space in each priority zone
- Current stock load. The maximum ranking assigned to an item might vary in function of the stock load. It would be 10 if the stock is full, but it does not make sense to deposit items in priority zone 10 if there are 6,7,8 and 9 empty.
- Predicted demand during the replenishment period
- Safety stock. If there are enough amount of selected article in stock, the leftover is just for safety and it is less likely to be used soon, so it is deposited far from entrance.

The next questions were:
How does each factor affects the efficiency of the algorithm if it does? When and how often are those variables recalculated? How can they be implemented into the algorithm?

To test the influence of each criteria it is necessary to create a model and run a simulation experiment to test it. The two factors to figure out is the best criteria to order references into the ranking, and the quantity of slots to reserve for each reference.

3.4.4 Step Two: Cranes scheduling

The objective in this step is to find an assign optimal trips to cranes. This is achieved comparing all the possible combinations of movements that meets the requirements, and select the one that would finish the job faster. It takes into account the ranking calculated in the previous step, as well as some other factors:

- Closest slot with selected reference
- Best available position to put the item in.
- Level of load of each corridor.
- Balance between the level of load for corridors
- Balance of work-load for each crane stacker.
- Balance of item between each corridor to fasten withdrawal operations
- Level of load of each priority zone.

Basically the function of this step is to compare the ARRIVAL and WITHDRAWAL queues, and search for the optimal pairs of them to compose each trip, choosing the route with smallest possible time, while keeping in mind the restrictions imposed by a long-term policies.

This step also has two missions. The first one makes adjustments and might vary the interpretation of the initial rank considering more restriction and performing some more light-weighted calculations, that requires a real-time inventory state. The second mission is to finally find the optimal combinations using SQL query requests.

The following factors might affect the final position, or even priority zone of the received item.

1. The balance between corridors. That is to provide faster order serving, which is not reflected directly in average time measures, but it is reflected in better order serving, which is also a priority goal. Well spread between corridors allows cranes to spread the load and reduce the time to finish each order when it is urgent. Simulation shows that, normally this is achieved quite automatically, since the reference arrived normally spreads between corridors using all cranes to get deposited. Otherwise, one option is to penalize the cranes with overload of selected reference comparing to others corridors.

2. Immediate crane balance and load queue for each crane. This is achieved with "span", waiting time till the crane is fried is considered in the calculation in order to find the fastest job finish time instead of fastest trip. If the optimisation of the total time is more relevant, there is also the option to penalize less, a fixed amount per each trip on given crane. Both options will be tested in simulation.

3. If priority zone to deposit is full, the item will be attempted to deposit in the next priority zone. Normally the article will be moved to the immediately next farer priority zone. If it occurs in the zone 10, it will be placed randomly.

4. If the deposit part of withdrawal-deposit trips has to store the item far away, does not makes sense to withdraw a reference close to entrance, but the one which is more far away, because the crane already has to do all the way there, and so it leaves the better placed withdrawal item for another, more suitable trip that would give more advantage. Therefore instead of ordering by total-time, it can be ordered by "time-between" or "total-time" + "time-between". All these combination will be tested in the simulation.

5. Other exception: no empty slot for deposit, inventory rupture, simultaneous deposit and withdrawal of same reference. These events have to be considered and treated in code, and also they are excluded from results analysis, because they alter average measures.

Normally ranking and minimum total-time are the most important pair of criteria from each step, since the combination of two prevents model from making too bad decisions and helping to converge the result toward the optimal values. However it is important to weight each in a correct way. The other ones are less important adjustment.

Optimal scheduling

There are two queues to analyse: withdrawal queue (items that require to be withdrawn from the stock and to be send to the customer) and deposit queues (items that arrive to the stock and need to be stored into the inventory).

That allows three possible situation: DEPOSIT ONLY(when the withdrawal queue is empty), WITHDRAWAL ONLY (when the deposit queue is empty) and COMBINED TRIPS (when none of queues is empty). Also there are 5 servers or stacker cranes, each one with its own mini-queue (this mini-queue never raises 3, the queues are maintained at low levels until some other trips are ended, so that it is possible to recalculate the optimal trips with new known conditions by that time).

The optimal sequence of trips implies the minimum time to end all the operations.

$$S_o = \min_N \sum_{i=0}^{N} t_i \tag{3.6}$$

where S_o = optimal sequence, t_i = time of each trip and N = number of operations to realise.

However, as it was already mentioned, this method doesn't result viable due to extent amount of time to process all the possible combinations. Therefore, and to

find a compromise between optimality and speed, it might be simplified and the final sequence will be composed by sum of optimal single times.

$$S_o = \sum_{i=0}^{N} T_o \tag{3.7}$$

When it is only withdrawal or only deposit situation, the fastest time is the time to the closest slot. However, the optimal time is found comparing the queue of the 5 cranes and the best times of operation in each corridor.

$$T_o = \min_{i=\{1:5\}} (\min(t_{run_i}) + t_{wait_i}) \tag{3.8}$$

where i = number of corridors $i \in 1:5$, t_{run} = time of operation and t_{wait} = waiting time for each crane.

When it comes to combined trips, there are quite more iterations to do:

$$T_{run_o} = \min_{m,n,p,q} (t_{m,n} + t_{p,q}) \tag{3.9}$$

where m = number of references in withdrawal queue, n_m = number of pallets in stock per each reference, p = number of references in deposit queue, q_p = number of available slots for these references.

In other words, for each slot occupied by one of the references to withdraw, it looks for the optimal time comparing to each empty slot that meets the requirement for each item to deposit.

That comparison, taking into account all the restriction, was just too heavy and took loots of time. Imagine there are 3 references in the withdrawal queue. Imagine there are 600 of these items in the warehouse. Also there are 5 different references to withdraw with different priority, making it 2000 available slots that meets the requirements. Now, for each of withdrawal slots the algorithm tries to find best deposit slot. There are over 1.200.000 and 3.000 request queries sent to the database. Does not takes an eternity, but still not viable for a real-time application. Not to talk about finding a sequences of several actions. Also, in bigger warehouses with more different items, the pool of combinations is exponentially increased making it even more hard and long-lasting process.

Therefore, only the first element from the WITHDRAWAL QUEUE is taken into account, making all possible deposit combinations. This prevent the WITHDRAWAL QUEUE from staling, picking the items with closer deadline, while for DEPOSIT QUEUE the order of deposits doesn't make any significant difference.

$$T_{run_o} = \min_{n,p,q}(t_{m,n} + t_{p,q}) \tag{3.10}$$

where n = number of slots in stock occupied with given article,p = number of references in deposit queue, q_p = number of available slots for these references. That gives one optimal trip per each corridor. Taking into account the queue on each corridor.

Chapter 4

Application development

The application development part of this thesis somehow became the most hard and time-requiring part. This happened due to the learning process and the coding itself. The application counts with over 8.000 lines of code and almost 200 SQL tables. The following software were used during the making of the thesis: Microsoft Visual Studio Ultimate 2010, MySQL Workbench 6, Texmaker, Autocad 2012, Photoshop CS, Paint, MS Office 2007, MS Visio 2013.

4.1 C# Application

4.1.1 Operation modes

The program consists of two parts, each one purposed for one of the work modes. The first one is the MANUAL MODE, which is designed for the every-day normal routine functioning, interacting with the user in charge of warehouse and with other automated parts of the warehouse. This mode also provides some extra mechanism to substitute any missing supposedly automated functions such is ERP system or bar-code readers, which are not included in present application, allowing manual input by users.

The SIMULATION MODE or automatic mode allows to run the program automatically, to simulate the behaviour of the application and of the warehouse in the future and analyse the efficiency of the algorithms. This mode needs historical data to run. This data is represented by database tables *ref_i*, invented by the author to simulate some pattern of demand. The tables represents daily demand for each of the 100 different articles, for 4 month before the set start of the experiments (invented historical data, for a forecast purposes) and for 4 month in the future (the orders to process by application). The software then generates withdrawal/pick-up orders for random time of selected day in the working schedule. Finally it transforms all the list into C# arrow, and runs the application till the end. The resupplies, and arrivals are generated automatically at each step.

4.1.2 Interface functions

The application provides an user interface which allows to introduce all the inputs and displays any required information.

The list and hierarchy of windows and their function:

- Main window
  - Manual mode
    - DEPOSIT PALLET: register all the incoming pallets and add them to the arrival queue. In practice this function is assigned to bar-code reader.
    - WITHDRAW PALLET: select pallets to withdraw.
    - SUPPLY ORDERS: manually place a procurement order.
    - CUSTOMER ORDERS: registers an incoming order. In practice can receive this information from an ERP system.
    - NEW ARTICLE: add new product.
    - ARTICLE ANALYSIS: displays all kind of useful information about selected article, such as description, provider, current stock, average position in stock, demand, etc...
    - STOCK ANALYSIS: displays the levels of busyness of stock by priority zone and corridor.
    - FORECAST: attempts to make a prediction of demand for selected product.
  - Simulation mode
    - SIMULATION WINDOW: allows to monitor the simulation, and analyse the results.
    - CHART
    - DATABASE CONSULT: read data manually from any table from the database.
  - Configuration
    - DB CONNECTION
    - MISCELLANEOUS

There are three examples below: Simulation window, Articles management and Forecast windows:

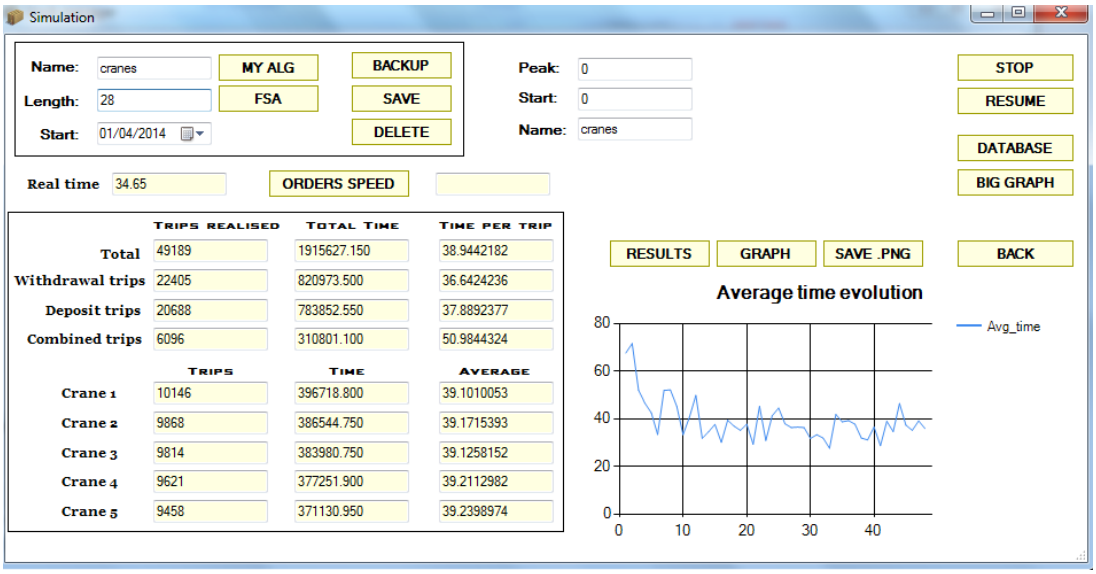


Figure 4.1: Example 1: Simulation window.

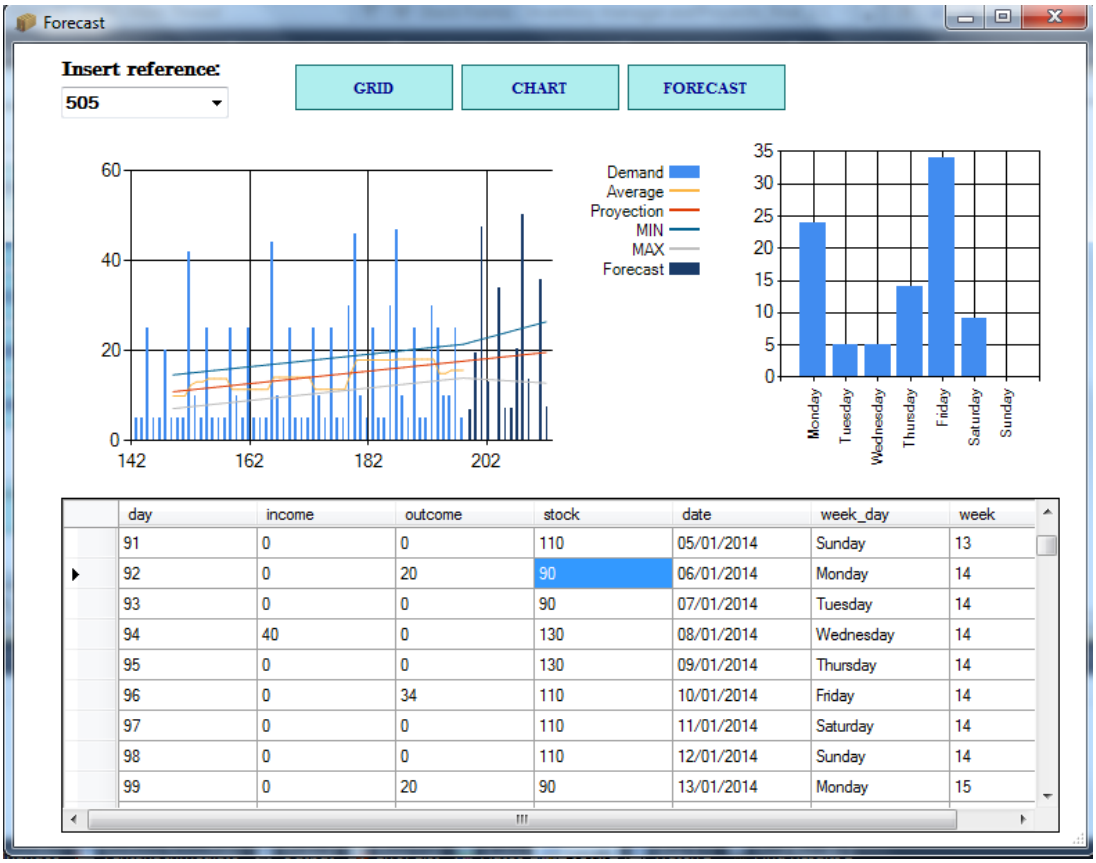


Figure 4.2: Example 2: Forecast window.

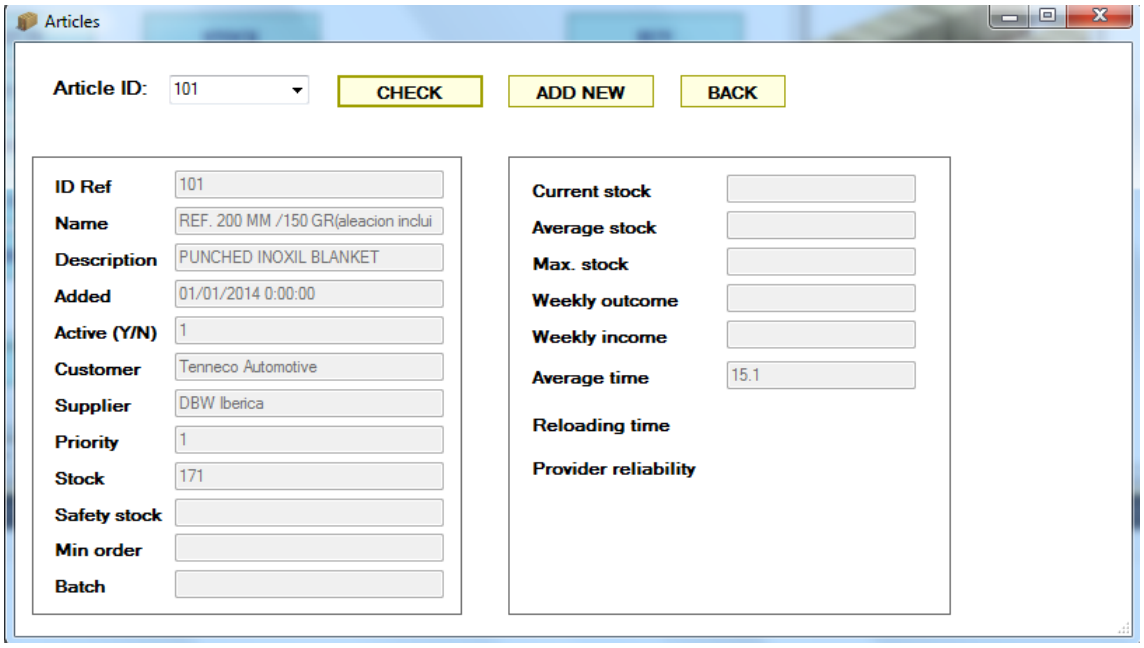


Figure 4.3: Example 3: Articles management window.

4.1.3 Simulation mode

The differentSIMULATION modes share most of working classes between them, as well as with the MANUAL mode, but the simulation itself has it own sequence, much more complex that the manual mode.

The methods are quite simple to operate in manual mode, but it becomes very hard to put together, coordinate and synchronize all together to run the simulation properly. Moreover it is necessary to rewrite the code for many of them due to the incompatibility and differences with the manual functioning.

On the other hand in few cases it is necessary to choose between two options, and sacrifice one of the properties in favour of others, to reach the needed balance.

Code

The simulation starts with a button click from the SIMULATION window. Previously is is necessary to configure all the settings for the simulation, such as importing the historical demand and all the future invented demand, fill the orders with FILL ORDERS function, fill the actual stock levels for each reference, set start date and mode. Thereupon the cursor jumps to the SIMULATION class and begins all the process.

First of all, the method SIMU is responsible to retrieve all the data from database table ORDERS. The data array formed from "reference name", "quantity" and "date

& time" of the orders, skipping the days with no orders. On the next step this array is converted to another array with only "reference name" and "date & time" fields, setting all the quantities to 1. This new data array is sent to the next method called THREADING, and it will be a static array during all the execution of the simulation. The THREADING method is the most important method during all the simulation. It keeps a count of all the orders, and it stays alive during all the simulation.

Going a few steps forward, it is necessary to say that the whole simulation is a sequential process, with the static pre-programmed part composed by customer orders, and dynamic and constantly refreshing part composed by supplies orders. Another option in consideration was to make several threads for each control of orders, of supplies and of cranes state. By several reasons, like the difficulty to keep the right executing order with many threads, this option was discarded in favour of one sequential thread.

There is a global timer called TIME_NOW which starts on the date& time set for the simulation start. This timer only moves ahead to the next event when triggered. The types of event are the following:

- Withdrawal order for an item in inventory
- Arrival of items
- End of each cranes movement

The IF loop compares the TIME_NOW DateTime variable, and performs actions needed for each option. If at the TIME_NOW moments there are no any more action to perform, TIME_NOW is set to the next action moment.

There are two triggers that activates the first sub-process. The first one triggers when any item needs to be withdrawn from the stock, either because an external transport has arrived to pick it up, or it is scheduled to arrive soon enough. The second one triggers when an external transport (truck) brings item to be stored in the inventory. Once one of the triggers (or both of them) is activated, the process of trips scheduling starts.

If it comes to WITHDRAWAL ONLY PROCESS, references and quantities, as well as deadlines of each order are moved into the WITHDRAWAL QUEUE. Once this queue stops being empty, the algorithm becomes active and basically programs cranes to withdraw an necessary amount of necessary reference in less possible time. This is achieved combining all the cranes and withdrawing the closest pallets.

In case of DEPOSIT ONLY the process is reversed. The pallets are identified at the entrance to warehouse with bar-code reader. That adds is to a virtual DEPOSIT QUEUE. Then the algorithm tries to place them on the best position according to their priority, corridor balance and some other restrictions. In this case the urgency

is not so important, since there are no external transport awaiting, but it is still important to optimize times, to avoid large queues and bottleneck at this stage.

The most relevant and complicated case is the MIXED case, when both DEPOSIT and WITHDRAWAL queues are not empty. Therefore the cranes has to do both withdrawal and arrival in same trip in order to reduce times, and when this happens is usually the most critic moment in these operations, which generates most important bottlenecks.

The issue in this process is that there is a conflict between WITHDRAWAL and DEPOSIT process, because the WITHDRAWAL needs to hurry up and take that pallets out of the stock, and the DEPOSIT process thinks on more long-term objectives, and wants to deposit the pallets in the right slot, to optimize the stock structure and make the withdrawals easier in future.

Each movement of cranes is stored in the tables to keep track on items position and have the real-time scan of the warehouse. After the algorithm assigns a trip to any crane, this trip is displayed in the real-time database table "cranes queue" as undone yet. However it does actualizes the inventory partially. The reference to withdraw is marked as it is not longer there, so it is prevented from participation in future calculations. The reference in which any item is going to be deposited is marked as no longer empty for the same reason. Although these changes are done before the trips is finished, the conflicts are avoided because each position is served from one crane only, and another trip can be programmed for this crane only after the actual one. In the deposit and withdrawal queues, the items are marked as completed as soon as they are assigned to any trip. The trip is marked as done only after it is finished, so it frees the crane and allows it to look for another movement.

There are also some scheduled processes. They are quite a heavy-weighted procedures, and there is no real need to update them with more frequency.

Each 50 trips, the function CHECKSTOCK is triggered. This function revises the stock and last movements, and places supply orders if necessary.

ORDERSRANKING is another scheduled procedure with same frequency. Is revises the stock and last movements and updates the ranking.

Finally, at the end of every day, the tables of each reference are updated. These tables store the historical value for number of orders, demand, supplies, stock load, safety stock and other values, allowing to make most of the calculations in the application, such as rotation, orders frequency, ranking, necessary safety stock, trend and others.

4.1.4 Classes

Most of the methods of the application are organized in a few classes according to the similarity of their functions. The most important are:

- CONNECTIONS 1: contains methods for searching and reading from the database. Retrieves data from DB and fills all the grids, charts, datasets, datasources and arrays for the posterior operations.
- CONNECTIONS 2: methods for input into the database. Update, Insert and Create queries.
  - ORDER_RANKING: recalculates the ranking and updates the table "articles".
  - INSERT METHODS: inserts supplies, orders, new items, etc...
  - UPDATE METHODS: updates all the queues and stock
- CALCULATE: collects method of calculations, such as algorithm step2, checkstock and supplies.
- SIMULATION: loop for the simulations.
- CALCULATE MANUAL: contains the methods from CALCULATE class, but also provides loops for the manual functioning, without accessing the SIMULATION class.

Connexions and Connexions-2 are working classes, containing method shared by the rest of classes.

There are variants of Simulation and calculus classes, adapted each for different stock policies: SIM_RAND, CALC_RAND...

4.1.5 Methods and functions

Main simulation loop

There are main loop FOR($i < n$) in SIMULATION class which runs till it finishes all the orders in the selected for the experiment time interval (n).

When the program runs is manual mode, the infinite loop is situated directly in Calc_manual class, simplifying the program quite a lot. There is no more TIME_NOW trigger, instead the $DateTime.Now$ parameter is used, representing the actual time.

The WHILE loop inside the FOR loop runs till it reaches any time associated action. The simulation has its own DateTime variable TIME_NOW independent from manual process and actual time. Also there are four time triggers, which are activated when TIME_NOW reaches them.

- TIME_WITHDRAWAL
- TIME_ARRIVAL
- TIME_TRIP_END
- TIME_EXECUTE

After receiving the statical array of orders, the simulation enters into a loop until it completes all the number of orders programmed (COUNT). This number of orders depends of start day of the simulation and the duration, both selected by the user before the experiment.

The TIME_NOW cursor runs from TIME_NOW = TIME_START till TIME_NOW = TIME_END. At every loop run the cursor values is compared to the value of the triggers. Each time it reaches a trigger, the corresponding routine is started. The triggers are also updated on each lap, and so is the status of cranes. The ranking and supplies are updated each 100 laps, to avoid to slow down the simulation.

```
for (int i = 0; i < count; i++)
{
   DateTime time_withdrawal = Convert.ToDateTime(list[1][i]); // time
       when a truck comes for any item

      bool a = true;
   while (a == true)
   {
      trip_ending = con.trip_ending();
      if (trip_ending[3].Count > 0)
         time_trip_end = Convert.ToDateTime(trip_ending[3][0]);

      contador++;

//SCHEDULED TASKS
      if (contador % 100 == 0 || contador == 0)
      {
         //recalculate ranking
         con2.orders_ranking(time_now);

         //resupply
         for (int arr = 0; arr < listnames.Count(); arr++)
         { calc.checkstock(listnames[arr], time_now); }
      }
      //clear cranes
      free_cranes = 0;
      list_cranes[0].Clear();
```

```
list_cranes[1].Clear();
list_cranes[2].Clear();

//cranes availability
string t_now = time_now.ToString("yyyy-MM-dd HH:mm:ss");
for (int i_c = 0; i_c < 5; i_c++)
{

   string crn = (i_c + 1).ToString();
   string query_cranes = "SELECT crane, timeend, COUNT(*) FROM
       cranes WHERE crane = " + crn + " and timeend >= '" + t_now
       + "' ORDER BY timeend DESC LIMIT 1;";
   list_cranes[0].Add((i_c + 1).ToString());//------ crane number
   if (this.OpenConnection() == true)
   {
      MySqlCommand cmd = new MySqlCommand(query_cranes,
          connection);
      MySqlDataReader dataReader = cmd.ExecuteReader();
      while (dataReader.Read())
      {
          list_cranes[1].Add(dataReader["timeend"] +
          "");//------time when crane free
         list_cranes[2].Add(dataReader["COUNT(*)"] + "");//------
             operations left

      }
      dataReader.Close();
      this.CloseConnection();
      cmd.Dispose();
   }

   if (list_cranes[2][i_c] == "0")
   {
      list_cranes[1][i_c] = time_now.ToString("yyyy-MM-dd
          HH:mm:ss");
      free_cranes++;
   }
   else
   {
      DateTime t1 = Convert.ToDateTime(list_cranes[1][i_c]);
      if (t1 < time_now)
         list_cranes[1][i_c] = time_now.ToString("yyyy-MM-dd
             HH:mm:ss");
      else list_cranes[1][i_c] = t1.ToString("yyyy-MM-dd
          HH:mm:ss");
   }
   Int32.TryParse(list_cranes[2][i_c], out cranes_avail[i_c]);
}
```

```
int oper = cranes_avail.Sum();
time_execute = time_now;


          //+ + + + + ++ +
if (time_now.Day == dia)
{
   con2.update_refxxx(time_now);
   dia = time_now.AddDays(1).Day;
}

if (time_now >= time_arrival)
{
   if (supplies[0].Count == 0)
   {
      con2.arrival2(sup_id);
      supplies = con2.arrival(time_now);
      if (supplies[0].Count > 0)
         time_arrival = Convert.ToDateTime(supplies[1][0]);
      else time_arrival = Convert.ToDateTime("01/01/2020
          00:00:00");
   }

   else
   {
      con2.arrival1(supplies[0][0], supplies[1][0]);

      sup_id = supplies[2][0];
      supplies[0].RemoveAt(0);
      supplies[1].RemoveAt(0);
      supplies[2].RemoveAt(0);
   }
}



          //+ + + + + ++ +
if (time_now >= time_withdrawal.AddHours(-1))
{
   con2.withdrawal(list[0][i], list[1][i]);

   //update trips time_array
   trip_ending = con.trip_ending();
   a = false;// goes for next FOR i
}

//+ + + + + ++ +
```

```
        if (time_now >= time_trip_end)//end of any trip
        {
           int trip_id;
           Int32.TryParse(trip_ending[0][0], out trip_id);
           con2.execute_trip(trip_id);
           trip_ending = con.trip_ending();
           if (trip_ending[3].Count > 0)
              time_trip_end = Convert.ToDateTime(trip_ending[3][0]);
           else
              time_trip_end = Convert.ToDateTime("01/01/2020 00:00:00");
        }

        if (time_now >= time_execute && oper <15)
        {
           calc.step2("", "", time_now, list_cranes);
        }

        if (time_now < time_withdrawal.AddHours(-1) && time_now <
            time_trip_end && time_now < time_arrival)
        {
           supplies = con2.arrival(time_now);
           if (supplies[0].Count > 0)
           time_arrival = Convert.ToDateTime(supplies[1][0]);
           else time_arrival = Convert.ToDateTime("01/01/2020 00:00:00");
           trip_ending = con.trip_ending();
           if (trip_ending[3].Count > 0)
              time_trip_end = Convert.ToDateTime(trip_ending[3][0]);
           else time_trip_end = Convert.ToDateTime("01/01/2020 00:00:00");

           if (time_trip_end <= time_withdrawal.AddHours(-1) &&
               time_trip_end <= time_arrival)
           {
              time_now = Convert.ToDateTime(trip_ending[3][0]);
           }
           else if (time_withdrawal.AddHours(-1) >= time_arrival)
           {
              time_now = time_arrival;
           }
           else
              time_now = time_withdrawal.AddHours(-1);
        }

        if (i == count)
           a = false;

     }
```

Step two

Basically the cursor leaves the infinite loop when at least one of the queues is not empty and there are cranes available. Then the cursor jumps to the CALC class to activate STEP 2. In STEP 2 there is a IF-ELSE LOOP, depending of the state of DEPOSIT and WITHDRAWAL queues, making four possibilities possible.

- if ((arrival_queue.Count == 0) && (withdrawal_queue.Count == 0))
- else if (withdrawal_queue.Count == 0)
- else if (arrival_queue.Count == 0)
- else

If both queues is empty, there is nothing to do. If one of them is full, either withdrawal or deposit is needed and the corresponding sub-method is activated. And if both of them are full, both are needed simultaneously.
After selecting an optimal trip, it is added to the cranes schedule. Each path calls the NEW_TRIP method to execute the trip and update all the queues and tables. Also, each path returns coordinates of selected slots back to user.

```
string [] values = {deposit_slot, deposit_reference, withdrawal_slot,
    withdrawal_reference, start of trip, end of trip, time spent}

con.new_trip(values);
return values;

return
```

Also there are PENALTIES defined, to take into account another factors, like crane busyness or article balance between corridors.

- CRANES_QUEUE
- CRANES_PENALTY
- CORRIDOR_PENALTY

Deposit and withdrawal trips

In case of DEPOSIT ONLY trips, since there is normally no special urgency to deposit items (contrarily to withdrawal), the algorithm retrieves the ranking of item to deposit and searches for an empty slot in the selected zone. If there is no slot in selected priority zone, it searches in the next worse priority zone. The best time from each crane is added to the waiting queue of each crane, and the least resulting

time wins the assignment.

Also there are option to implement small corrections, penalizing the crane decision if the selected corridor contains much superior amount of selected item.

The WITHDRAWAL ONLY trips code is quite similar to the code of DEPOSIT ONLY. First it finds the closest slot with needed article for each corridor. Then it compares the waiting queue of each cranes and sets the trip to the crane where waiting time plus travel time is minimum.

Combined trips

When it becomes to COMBINED TRIPS, the number of possible combination is increased exponentially, that is if there are 100 possible withdrawal movements and 100 deposit ones, it ends up in around 2000 possible combinations. Moreover, it takes more time to calculate each one of them. To reduce the number of possible combination there is a grid applied to the inventory rack, splitting it in 50 zones. Therefore only one (the closest) withdrawal slot per zone is considering, reducing considerably execution times without barely affecting the efficiency.

```
else
{
   int [] cranestr = {1,2,3,4,5};
   int [] rankstr = {1,2,3,4,5,6,7,8,9,10};
   List<string>[] list = new List<string>[5];
      list[0] = new List<string>();//-----------------slot_id
      list[1] = new List<string>();//-----------------crane
      list[2] = new List<string>();//-----------------depth
      list[3] = new List<string>();//-----------------hight
      list[4] = new List<string>();//-----------------total time

   for (int i = 0; i < 5; i++)
   {
      for (int i2 = 0; i2 < 10; i2++)
      {
         string query = "SELECT slot_id, crane, depth, hight, total_time
             FROM stock where reference = " + withdrawal[0][0] + " AND
             crane = " + cranestr[i] + " AND priority = " + rankstr[i2] +
             " order by total_time asc LIMIT 1;";

         if (this.OpenConnection() == true)
         {
            MySqlCommand cmd = new MySqlCommand(query, connection);
            MySqlDataReader dataReader = cmd.ExecuteReader();
            while (dataReader.Read())
```

```
            {
               list[0].Add(dataReader["slot_id"] + "");
               list[1].Add(dataReader["crane"] + "");
               list[2].Add(dataReader["depth"] + "");
               list[3].Add(dataReader["hight"] + "");
               list[4].Add(dataReader["total_time"] + "");
            }
            dataReader.Close();
            this.CloseConnection();
         }
      }
   }

//----------
   List<string>[] list3 = new List<string>[5];
   list3[0] = new List<string>();//-----------------slot_id
   list3[1] = new List<string>();//-----------------crane
   list3[2] = new List<string>();//-----------------depth
   list3[3] = new List<string>();//-----------------hight
   list3[4] = new List<string>();//-----------------total_time

   int [] cranestr2 = {1,2,3,4,5};
   int [] rankstr2 = {1,2,3,4,5,6,7,8,9,10};

   for (int iii = 0; iii < arrival[0].Count; iii++)
   {
      for (int i = 0; i < 5; i++)
      {
         for (int i2 = 0; i2 < 10; i2++)
         {
            string query3 = "SELECT slot_id, crane, depth, hight,
               total_time FROM stock where reference = " + arrival[0][0]
               + " and crane = " + cranestr2[i] + " and priority = " +
               rankstr2[i] + " order by total_time asc LIMIT 1;";

            if (this.OpenConnection() == true)
            {
               MySqlCommand cmd = new MySqlCommand(query3, connection);
               MySqlDataReader dataReader = cmd.ExecuteReader();
               while (dataReader.Read())
               {
                  list3[0].Add(dataReader["slot_id"] + "");
                  list3[1].Add(dataReader["crane"] + "");
                  list3[2].Add(dataReader["depth"] + "");
                  list3[3].Add(dataReader["hight"] + "");
                  list3[4].Add(dataReader["total_time"] + "");me
                  }
                  dataReader.Close();
```

```
            this.CloseConnection();
            cmd.Dispose();
         }
      }
   }
}

List<string>[] list4 = new List<string>[6];
list4[0] = new List<string>();//-----------------slot_id_deposit
list4[1] = new List<string>();//-----------------reference_deposit
list4[2] = new List<string>();//-----------------total_time
list4[3] = new List<string>();//-----------------slot_id_withdrawal
list4[4] = new
    List<string>();//-----------------reference_withdrawal
list4[5] = new List<string>();//-----------------crane

string crane;
int glob_ind = 0;

//BIG BUCLE
//for each position of item to withdraw...

for (int i2 = 0; i2 < arrival[0].Count; i2++)
{
   for (int i = 0; i < list[0].Count; i++)
   {
      crane = list[1][i];
      priority = con.ranking(arrival[0][i2]);
      int prior2;
      priorityfull2:
      string query4 = "SELECT slot_id, total_time, (((ABS(" +
          list[2][i] + " - depth))*1.3/4)+((ABS(" + list[3][i] + " -
          hight))+ " + list[4][i] + " + total_time)) FROM stock
          WHERE occupied = 0 AND (crane = " + crane + ") and
          priority = " + priority + " ORDER BY (((ABS(" + list[2][i]
          + " - depth))*1.3/4)+((ABS(" + list[3][i] + " - hight))+ "
          + list[4][i] + " + total_time)) ASC LIMIT 1;";

      if (this.OpenConnection() == true)
      {
         list4[5].Add(crane);
         MySqlCommand cmd = new MySqlCommand(query4, connection);
         MySqlDataReader dataReader = cmd.ExecuteReader();
         while (dataReader.Read())
         {
            list4[0].Add(dataReader["slot_id"] + "");//-----
                slot_id_deposit
```

```
                list4[1].Add(arrival[0][i2]);//----- reference
                list4[2].Add(dataReader["(((ABS(" + list[2][i] + " -
                    depth))*1.3/4)+((ABS(" + list[3][i] + " - hight))+ "
                    + list[4][i] + " + total_time))"] + "");//---
                    total_time
                list4[3].Add(list[0][i]);
                list4[4].Add(withdrawal[0][0]);
            }
            dataReader.Close();
            this.CloseConnection();
            cmd.Dispose();
        }

        if (list4[0].Count == 0)
        {
            Int32.TryParse(priority, out prior2);
            prior2++;
            if (prior2 == 11)
            prior2 = 1;
            priority = prior2.ToString();
            list4[5].RemoveAt(glob_ind);
            goto priorityfull2;
        }
        else if (list4[5].Count > list4[0].Count)
        {
            Int32.TryParse(priority, out prior2);
            prior2++;
            if (prior2 == 11)
               prior2 = 1;
            priority = prior2.ToString();
            list4[5].RemoveAt(glob_ind);
            goto priorityfull2;
        }
        glob_ind++;
    }
}

con.rowcount_noss(arrival[0][0]);//------------n of product X in
    each row (EXCLUDED SS)
con.RowCount(arrival[0][0]);//-----------------n of product X in
    each row (INCLUDED SS)
decimal[] list5 = new decimal[list4[2].Count];

for (int i = 0; i < list4[2].Count; i++)
{
   Decimal.TryParse(list4[2][i], out list5[i]);
}
```

```
decimal[][] time_dec = new decimal[3][];
time_dec[0] = new decimal[list5.Count()];//time to go
time_dec[1] = new decimal[list5.Count()];//time go
time_dec[2] = new decimal[list5.Count()];//time total

DateTime[] time_end = new DateTime[5];
TimeSpan tduration = new TimeSpan(0, 0, 0);
double span = 0;
decimal best_time = 1000;
decimal real_time = 1000;
string withdrawal_slot = "0";
string withdrawal_ref = withdrawal[0][0];
string deposit_slot = "0";
string deposit_ref = "0";
int craneid = 0;
int real_crane = 0;

for (int i = 0; i < list5.Count(); i++)
{
   decimal time1 = list5[i];
   Int32.TryParse(list4[5][i], out craneid);
   time_end[craneid - 1] =
       Convert.ToDateTime(cranes_queue[1][craneid - 1]);

   if (time_end[craneid - 1] <= time_now)
   {
      span = 0;
   }
   else span = (time_end[craneid - 1] - time_now).TotalSeconds;
   time_dec[0][i] = Convert.ToDecimal(span);
   time_dec[1][i] = Convert.ToDecimal(list5[i]) + 3 +3+8;
   time_dec[2][i] = time_dec[0][i] + time_dec[1][i];

   if (time_dec[2][i] < best_time)
   {
      deposit_slot = list4[0][i];
      withdrawal_slot = list4[3][i];
      deposit_ref = list4[1][i];
      best_time = time_dec[2][i];
      real_time = time_dec[1][i];
      real_crane = craneid;
   }
}

if (real_time == 1000)
   {
   con2.new_trip(real_crane, 0, "BREAK", "BREAK", "",
       withdrawal_ref, time_now, 0);
```

```
        }
      else if (deposit_ref == withdrawal_ref)/////update cranes
      con2.new_trip(0, 0, "DIRECT", "DIRECT", deposit_ref,
          withdrawal_ref, time_now, 1);

      else con2.new_trip(real_crane, real_time, deposit_slot,
          withdrawal_slot, deposit_ref, withdrawal_ref, time_now, 0);
   }
}
```

Check stock

This is a periodically scheduled method, responsible of maintaining each article stock at proper level. This level is the minimum level possible, taking into account all the existing constraints, and the service level required for each type of article. This method analyses the current stock, safety stock, the historic demand, the ensured orders, the forecasting and ordered supplies, using the combination of them to determine if the current stock is enough to meet the customers demand, or it is necessary to order supplies to providers.

```
public int[] checkstock(string refer, DateTime time_now)
  {
  Connexions con = new Connexions();
  Connexions2 con2 = new Connexions2();

  int stock = 0;
  int orders = 0;
  int foreweek = 0;
  int security = 0;//safety* stock
  double deviation = 0;
  int[] week = new int[8];
  int historic = 0;
  int supply = 0;

  supply = con.supplies(refer, time_now, 4);
  string[] stringstock = new string[2];
  stringstock = con.averagetime(refer);
  Int32.TryParse(stringstock[1], out stock);

  /// security stock////
  string[,] data2 = con.forecast(refer);
  int[] data = new int[56];
  for (int cont = 0; cont <= 55; cont++)
     {
     Int32.TryParse(data2[2, cont], out data[cont]);
  }
```

```
   ///weekly demand and deviation
   int j = 0;
   for (int i = 0; i <= 55; i = i + 7)
   {
      week[j] = data[i] + data[i + 1] + data[i + 2] + data[i + 3] +
          data[i + 4] + data[i + 5] + data[i + 6];
      j++;
   }

   for (int cont2 = 0; cont2 <= 7; cont2++)
   {
      historic = historic + week[cont2];
   }
   double promedio = historic / 8;
   double[] rest = new double[8];
   double restsum = 0;

   for (int cont3 = 0; cont3 <= 7; cont3++)
   {
      rest[cont3] = Math.Abs(week[cont3] - promedio);
      restsum = restsum + rest[cont3];
   }
   deviation = Math.Sqrt(restsum);
   security = Convert.ToInt32(Math.Ceiling(deviation * 3));
   orders = con.checkorders(refer);
   foreweek = con.checkordersweek(refer);

   int[] answer = new int[5];
   answer[0] = stock;
   answer[1] = orders;
   answer[2] = foreweek;
   answer[3] = security;
   answer[4] = supply;

   con2.update_rest(refer, answer, time_now);
   if (answer[0] <= (orders + security - supply))
   {
      this.resupply(answer, refer, time_now);
   }
   return answer;
}
```

Supply

If the previous method determines the need of resupplies, it calls the supply method, which determines the quantity to order, depending of the. As described in CHAPTER 3 it uses the combination of historical data and ensured orders to make a forecast demand. Then it analyses the warehouse levels, supplies on the way and the time required to resupply. Finally it considers the optimal bath and the minimum order quantity agreed with the suppliers.

```
public void resupply(int[] answer, string refer, DateTime time_now)
{
   Initialize();
   double supply_amount = 0;
   supply_amount = answer[2] + answer[3]-answer[4] - answer[0];
   string min_order = "0";
   string batch = "0";
   double i_order = 0;
   double i_batch = 0;
   string query = "SELECT min_order, batch from refs where id = " + refer
       + ";";
   if (this.OpenConnection() == true)
   {
      MySqlCommand cmd = new MySqlCommand(query, connection);
      MySqlDataReader dataReader = cmd.ExecuteReader();
      while (dataReader.Read())
      {
         min_order = (dataReader["min_order"] + "");
         batch = (dataReader["batch"] + "");
      }
      dataReader.Close();
      this.CloseConnection();
      cmd.Dispose();
   }
   Double.TryParse(min_order, out i_order);
   Double.TryParse(batch, out i_batch);
   string datetimenow = time_now.ToString("yyyy-MM-dd HH:mm:ss");
   string datetime = time_now.AddDays(4).ToString("yyyy-MM-dd HH:mm:ss");
   string supplier = "Default";
   string notes = "Automatic supply";

   Connexions2 con2 = new Connexions2();
   if (supply_amount > i_order)
   {
      double supply = Math.Ceiling(supply_amount / i_batch)*i_batch;
      int supply_amount2 = Convert.ToInt32(supply);
      con2.supply(refer, supply_amount2, datetime, datetimenow, supplier,
          notes);
}}
```

Other methods

CONNECTIONS 1 methods:

- AVERAGE_TIME: average positioning of the item in stock.
- SELECT: retrieves all the data about a reference: id, description, admission date, active (y/n), customer, supplier, ranking, current stock, average stock, maximum stock, past outcome (customer demand), past income, optimal batch, minimum order quantity, safety stock, orders frequency, average order quantity
- COUNT_TOTAL_STOCK: total stock load level.
- COUNT_REF_STOCK: current stock for selected item.
- FORE1: returns quantities ordered in past 4 weeks for given reference.
- FORECAST: retrieves all the data from past 8 weeks.
- FORECAST2: retrieves all the past data to fill the datagrid.
- CHECKORDERS: checks orders for next four days (during resupply period).
- RANKING: return the rank of the article.
- COUNT_SAFETY_STOCK: safety stock amount for selected article.
- STOCK_POSITIONING: amount of pallets of selected article placed in the assigned or better priority zone.
- CORRIDOR_COUNT: quantity of the selected article in each corridor.
- CORRIDOR2_COUNT: load of each corridor.
- ZONES_AVAILABILITY: number of free slots in each priority zones.
- FREE_SPACE_COUNT: counts empty slots in each row for given priority area.
- ROWCOUNT_NOSS: regular stock(no safety) of given article in each row.
- WITHDRAWAL & ARRIVAL: retrieves withdrawal or arrival scheduling.
- TRIP_ENDING: retrieves cranes scheduling.
- SUPPLIES: retrieves supplies scheduling
- SIM_RESULTS: applies several filters to get all the necessary data at the end of simulation. This data is represented in the simulation window.
- ORDERS_SPEED: calculates the average time of order completing.

- GRAPHS: searches and applies filters to data and displays it on the graphics.

CONNECTIONS 2 methods:

- INSERT_REFS, INSERT_ORDERS, INSERT_SUPPLIES: updates the appropriate tables with new articles, orders, or supply orders, either created manually or automatically.
- TABLEGENERATOR: creates and manages article-tables (ref101-ref999).
- WITHDRAWAL & ARRIVAL: inserts new articles into withdrawal or arrival queues.
- NEW_TRIP: when the optimal route is calculated, adds it to the cranes scheduling.
- EXECUTE_TRIP: updates the dynamic tables (cranes, arrivals and withdrawals) after each trip is finished.
- UPDATE_STOCK: updates stock each time any pallet is deposited or withdrawn.
- UPDATE_REFXXX: at the end of the day updates every article-table with values of input, output, av.stock, etc...
- ORDERS_RANKING, ORDERS_RANKING2: analyses stock situation, past orders, supplies and forecasting to create a ranking system and assign a rank to each article in warehouse.
- FILL_ORDERS, FILL_REFS, FILL_STOCK, DELETE_DATA: a set of methods to reset all the tables at the end of each experiments, and set them to initial values.
- SAVE_RESULTS: stores the results of each experiment.
- SIM_START: keeps register of the configuration of each experiment.

4.1.6 Debugging and memory management

After checking all the code, fix the compile errors, the SQL queries errors, and after all the code works fine and does what is supposed to do, there are still possible a few problems.

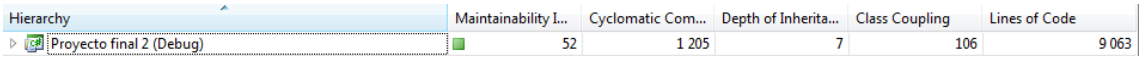
The out of memory exception. The program becomes to consume much and much resources after the machine executing it is running out of memory available and the application crashes. This happens dye the fact that some object are still consuming the memory, even when they are not used any more. The VS2010 has a "Garbage Collector" which closes all useless resources and frees memory. But there are also some not disposable objects that require an extreme precaution, such as connections to database, datasets, and data readers. Also it helps switching to x64

build mode, which allows to use more memory. After some adjustment the program doesn't crush any more and the memory consume is stable at 15.000-20.000 bytes.

Also there was a problem with "Time Out" connection. This can happen if something is wrong with the connection to database, sometimes when the process of simulation is paused and started again. To fix this issue, the Time Out is set to 10 minutes, and the connections are made through try/catch mechanism, preventing the application from crash.

4.1.7 Project dimensions

Final size of executable is around 10 Mb, consuming around 17Mb of RAM memory while running. Metrics results are the following:



Hierarchy	Maintainability I...	Cyclomatic Com...	Depth of Inherita...	Class Coupling	Lines of Code
Proyecto final 2 (Debug)	52	1 205	7	106	9 063

Figure 4.4: Project metrics

4.2 Data

4.2.1 Data flow

4.2.2 Database structure

As the database tables are connected and managed by means of C# application, there are no any rigid structure or triggered queries inside the database. The approximate connection of the most important tables are shown in the figure 4.5 Database scheme.

Connections

As mentioned above there are two main classes containing working methods. Both provide a connection with the Database.

- CONNEXIONS - stores OUTPUT queries (retrieves data from DB; SEARCH, SORT and SELECT information)
- CONNEXIONS 2 - stores INPUT queries (saves data into DB; UPDATE, INSERT and CREATE data)

The connection is established with the following methods. In the Connexions, Connexions2 and each other class that need a direct connection to the database the following code has to be declared.

First, the connexion properties are declared as global variables, so they can be used by each method and query. They properties are "server name", "database name", "user ID" and "password". These values can be changed from the application if the connection to the database is changed.

To configure the connection it is necessary to set proper values in "Initialize" method (from "Miscellaneous" in main window), so if the database host is changes, it is only necessary to change the configuration up here.

In this case, the application is connected to a database named "databasepfc", on a local host. Finally, the "connectionString" is initialized with all provided configurations. To start operating with a database, each method has to initialize the connection first calling this "Initialize()" method.

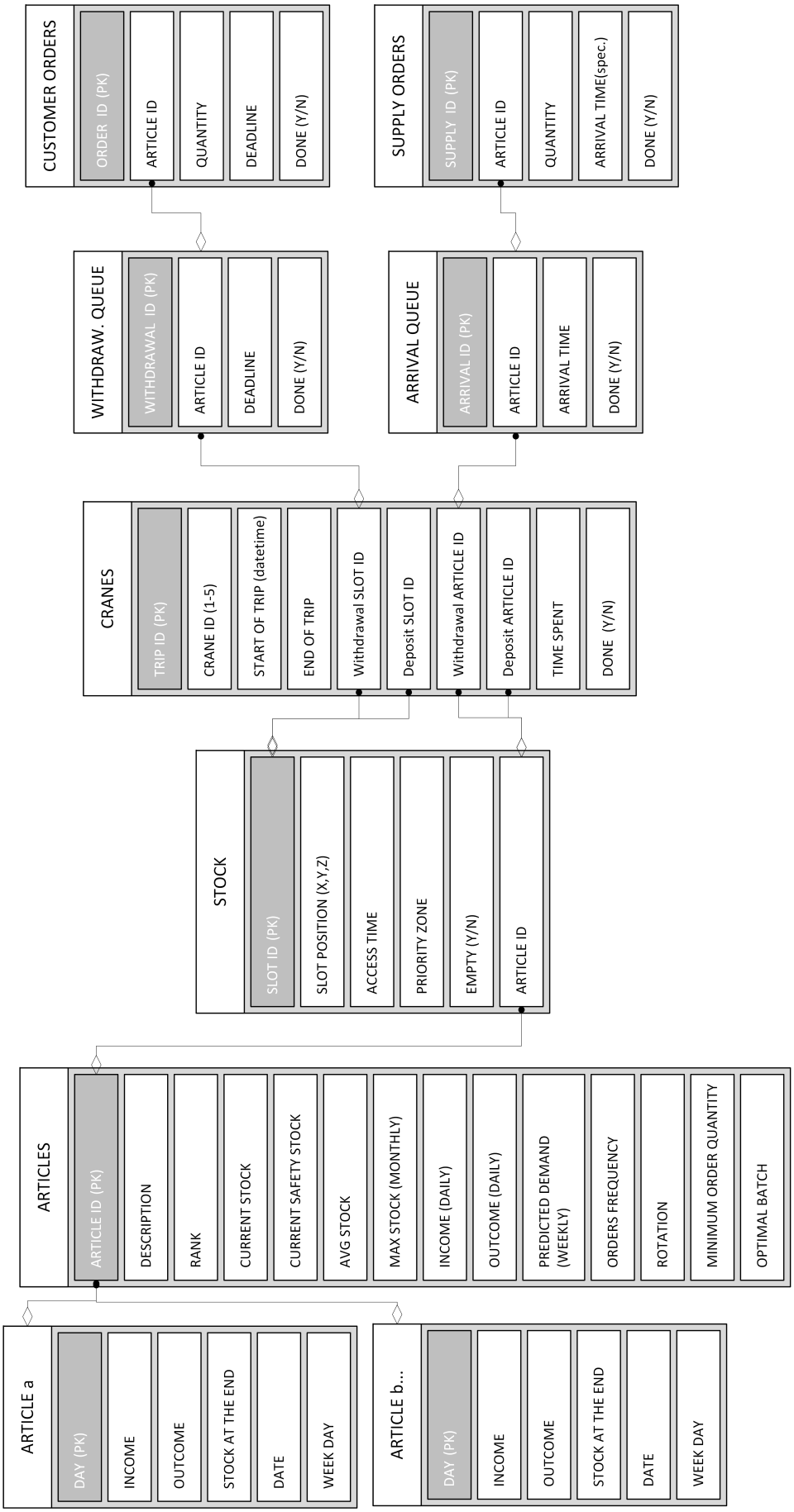


Figure 4.5: Database scheme

```
public void Initialize()
{
   server = "localhost";
   database = "databasepfc";
   uid = "root";
   password = "root";
   string connectionString;
   connectionString = "SERVER=" + server + ";" + "DATABASE=" +
   database + ";" + "UID=" + uid + ";" + "PASSWORD=" + password + ";";
   connection = new MySqlConnection(connectionString);
}
```

To open the connection the OpenConnection method is used. If the client is unable to connect to database or unable to login, the user has to know it. The connections are opened before each operation and closed right after, in order to avoid stacking static object in application memory.

```
public bool OpenConnection()
{
   try
   {
      connection.Open();
      return true;
   }
   catch (MySqlException ex)
   {
      //handling errors
      //0: Cannot connect to server.
      //1045: Invalid user name and/or password.
      switch (ex.Number)
      {
         case 0:
            MessageBox.Show("Cannot connect to server. Contact
                administrator");
            break;
         case 1045:
            MessageBox.Show("Invalid username/password, please try again");
            break;
      }
      return false;
   }
}
```

4.2.3 Tables

The SQL tables are in charge of storing all the data. There are three main function or types of tables.

- Static tables, such as STOCK, REFERENCES, CUSTOMERS, SUPPLIERS. These tables has a fixed number of rows and an object as a primary key (supplier_ID, article_ID, slot_ID).
- Dynamic tables, such as CRANES, WITHDRAWALS, ARRIVALS, SUPPLIES, ORDERS and REF[101-999]. These table are updates after each corresponding process. The number of lines is variable and the primary key is process or time associated.
- Simulation results. The results of stock, cranes, supplies, arrivals and withdrawals are saved after each simulation. Also there are a global table updated with the significant results from each experiment.
- Initiate tables. Holds an original state of dates and allows refresh the main tables after experiment, so the next experiment starts in same conditions.

4.2.4 SQL queries

Most of the queries are used from C# environment: SELECT, UPDATE, INSERT, CREATE, etc... Some specific ones are written inside MySQL database and called from C# application:

DELETER: Sets all the necessary tables to the initial condition before the next experiment. This query allows all the experiments start in equal conditions.

SETUP: Sets the initial conditions of warehouse to another point, such as the end of another experiment.

BACKUP: Saves all the necessary tables at the end of each experiment for the subsequent analysis.

An example below explains the process of analysing resulting times of experiment 6, preventing to take into account the null rows (stock ruptures), transitory period during the first 3.000 movements (observations from graphics) and a strange singular peak in the 17th value. The resulting 4 values are the total average time, average time of double trips, and average time of only withdrawal or only deposit trips:

```
SELECT count(*),avg(time) FROM databasepfc.6c where trip_id > 3000
and (trip_id < 16000 or trip_id > 17000) and time != 0;

SELECT count(*),avg(time) FROM databasepfc.5c where trip_id > 3000
and (trip_id < 16000 or trip_id > 17000)
and time != 0 and deposit_slot != '' and withdrawal_slot != '';

SELECT count(*),avg(time) FROM databasepfc.5c where trip_id > 3000
and (trip_id < 16000 or trip_id > 17000) and time != 0 and deposit_slot = '';

SELECT count(*),avg(time) FROM databasepfc.5c where trip_id > 3000
and (trip_id < 16000 or trip_id > 17000) and time != 0 and withdrawal_slot = '';
```

Figure 4.6: SQL queries examples

The following query displays the distribution of deposit movements across the priority zones:

```
SELECT COUNT(*)/(SELECT COUNT(*) from 10cra where deposit_slot ="") as movements,
FLOOR(withdrawal_slot/1000)+1 as prior_zone from 10cra  where deposit_slot =""
group by FLOOR(withdrawal_slot/1000)+1;
```

Count(*) displays number of empty slots for each priority zone. This query is used at some moments to monitor the stock load, and avoid empty slots in high priority zones:

```
select count(*), priority from stock where occupied = 0 group by priority;
```

The results are displayed in the figure 4.7 Simulation analysis:

count(*)	priority
32	1
279	2
152	3
83	4
234	5
538	6
654	7
214	8
18	9
633	10

(a) Zones business

movements	prior_zone	movements	prior_zone
0.2025	1	0.1952	1
0.2219	2	0.1883	2
0.1467	3	0.1434	3
0.0968	4	0.0988	4
0.0785	5	0.0939	5
0.0489	6	0.0690	6
0.0555	7	0.0938	7
0.0581	8	0.0641	8
0.0524	9	0.0252	9
0.0386	10	0.0283	10

(b) Trips over zones

Figure 4.7: Simulation analysis

Chapter 5

Simulation and results

5.1 Considerations

5.1.1 Measuring

There are many factors that affect the results and the efficacy of the algorithm. The simulation is a valuable tool that allows to simulate different policies, find out how each factor affects to the results and modify it trying to improve its efficiency.

These are the consideration to monitor and make sure that they are appropriate, in order to trust the posterior analysis of results:

1. RANKING. The ranking has to have some logic distribution, according to the level of stock, and the emptiness of separate priority zones. So that if there are no slots empty in first three priority zones, it is logical that the best ranking possible is 4. On the other hand, if the stock is half empty, makes no sense to store items past fifth priority zone.

2. CRANES. The cranes balance has to be adequate. It is inadmissible that any of them taking a break while clients queue isn't empty.

3. STOCK. In order to provide high efficiency, it's necessary to avoid that too much of high-priority slots spend too much time empty.

4. SUPPLIES. Supplies are released properly, reducing stock breaks, but keeping stock levels as small as possible.

5. Incoming and outgoing orders are server properly and within the deadline.

6. MOVEMENTS DISTRIBUTION per priority zones. The aim is to achieve that the greatest part of movement succeed in closest priority zones.

It is necessary also to remove the "DIRECT-TRIPS" (pallet that move directly from the entrance to the exit, without passing by the inventory rack) for comparison, since it allows some models to "cheat" in some way, or just avoid count it as trip. The same applies for a non-server trips (in case of inventory rupture), which may be analysed separately without affecting average times.

The copies of stock final position, and cranes movements are saved after every simulation. Also there are values that are displayed and saved in a table after each simulation. It is possible to monitor values during the simulation from the "Simulation" window.

5.1.2 Results analysis

The values to be measured and analysed are:

- Total time elapsed by cranes.
- Number of trips realized
- THE AVERAGE TIME PER TRIP, which is the objective to be reduced. The less time, the better the result. There are three average time values, for the three possible cases: withdrawal only, deposit only, and withdrawal+deposit. These times has to be compared separately in order to have a precise result. Also it is important to take only values when the system is stabilized.

$$t_{average} = \frac{\sum t_i}{n_{trips}}; i \in n$$

  where t_i = time spend by one of the cranes in the trip i, and n = total number of trips
- Last three values separately considering the withdrawal trips only
- Last three values separately considering the deposit trips only
- Last three values separately considering the combined trips only
- Last three values separately for each crane to assure the fulfilment of the cranes balance.
- THE REAL TIME, takes into account the number of pallets transported instead of number of trips released. Considers each combined trip as 2 separate trips, since it realises two functions at time.

$$t_{real} = \frac{\sum t_i}{n_{withdrawal}+n_{deposit}+n_{combined}*2}$$

- Average time of finishing withdrawing a customers order. Since the start of first trip till the end of last trip, until the order is completely withdrawn from warehouse. The smaller values are better.

$$t_{order_{avg}} = \frac{\sum t_{order}}{n_{orders}}$$

$$t_{order} = t_i - t_f$$

where t_i = the moment where withdrawal order is activated and t_f = the moment where the last action to complete the order is finished.

5.1.3 Start conditions and end results

It is important to set the same set-up conditions for the experiments to compare them. In order to experiment with several conditions and scenarios there will set different conditions. However the analysis of results must be made between experiment of same condition pull. Therefore there are two pools: the standard pool, containing most of experiments, and the double-length pool, for the experiments six and nine. They are also have different initial set-up explained more forward.

Another question was "which conditions would be fair to compare algorithm and non-algorithm models. Therefore both experiment pool has different initial stock load and distribution, but the simple and right answer to that question is that no matter the initial set-up, the results are quite equal once the experiment is stabilized past the transitory period.

These are initial inventory values common for both scenarios:

1. The date of simulation is set to 2014-04-01.
2. There is data enough to have a 4 month of real time, but since it would take too long, the simulation will be stopped when it reaches a stable level of results, normally 30.000 - 40.000 movements.
3. There are some supplies set for the first few days, to prevent the emptiness of the stock, until the automatic supplies arrives (4 days of replenishment time).
4. Stock state is the same before experiments.

Other values are different for each scenario:

SCENARIO 1: 70% stock load, stock sorted randomly, supplies starts the same day, old demand patterns.

SCENARIO 2: 85% stock load, stock sorted by demand in the closest slots, modified demand patterns, supplies start 4 days before the rest. Only for experiment 6 and 9 (both 60.000 movements).

5.1.4 Number of simulation or length of simulations

Since the application takes some time to process all the possible options and choose the best one, and there are many choices to take, it requires an important amount of time to run the experiment. That time could reach three days to simulate all the four month demand in case of the most heavy weighted version of algorithm. The best option when it is possible is to test as much as possible, trying different combinations of methods and variable values keys, but the authors was limited by the processing capability of his machine which isn't the fastest in the world. Whatever the case, it is a good idea to manage the time smartly and optimize the speed of execution. That is, if the simulation results are doing good, it is a good choice to give it more time in order to obtain a more versatile results. On contrary, when it becomes obvious that the experiment is going wrong, and there is small chance to obtain a good result, it is better to stop and reprogram the policies of algorithm.

Of course that is applicable to different simulation. There is no point to run several times a simulation to confirm the experiment, since the deterministic algorithm would return the same result. Moreover, the practice shows that even with slightly different start set-up, if the policies are the same, the result values end up to converge. However to obtain the reliable, stable and converged results the there is a need to run the experiment until steady values.

5.1.5 Performance or efficiency

The more complex is the algorithm and the more possibles option it has to compare, the more time it takes to calculate the results. First attempts was just too time-expensive, even optimizing the code. So that the algorithm required several simplification in order to fasten it without harming in a relevant way the optimality of the results.

First of all, some hard procedures have to be scheduled to succeed once every n movements or every n minutes. These procedures are RANKING and RESUPPLY. Both have to read and analyse large amounts of data to make a decision. But these decision are not necessary to be made in real-time before each crane movement. Scheduling them to each 20, 50 and 100 movements didn't result in efficiency decrease and seemed to be enough to speed up the application.

5.1.6 First slot available policies

This is the most simple tactic which proven to be very efficient in most of cases, without requiring any additional efforts.

Its efficacy is based on the higher use of closest areas, which are always full, allowing the short trips there.

"WITHDRAWAL ONLY" and "DEPOSIT ONLY" modes for this policy are relatively easy to set. The question is to choose the shortest trip, or the first available crane. The problems come where its needed to make a COMBINED "WITHDRAWAL-DEPOSIT" trip and it is impossible to deposit in the same withdrawal slot, due to the fact that the transport crane only can hold one item. In these cases the crane is forced to make a long trip to deposit the item in the first available slot, due to the fact that all other slots in the proximity are full. This happens even if the withdrawal reference is in very close position, and it is fast to withdraw. Moreover, the deposited reference could be a "high-priority" item, and could be needed very soon. This issue could be partly improved by an algorithm that would make a decision in which cases realising two trips is faster than only one, depositing this way the item in the recently vacant slot. But of course, this implies a use of an algorithm, adding more complexity and turning this into a "not-so-easy" model.

There was two options possible: if the warehouse has an information system with current stock positioning, either it doesn't and only knows where are necessary references stored. Both method, with slight algorithm improvement, and a totally "blind method" will be tested.

5.1.7 Improved policies

To estimate and rate the would be great to know about the results achieved by other similar applications developed by another industrial consulting companies.

One way to calculate the efficiency of the algorithm is to split it in several steps, and make simulation of each different step. For example in order to estimate the efficiency of the first step (sorting and ranking), it is possible to simulate it versus the non-ranking model, but this non-ranking model still share the same STEP TWO.

To estimate the 2nd step efficiency there is no absolutely right way, because there is no any other active model to compare with. It is necessary to program this another model by ourselves, so to invent a "non-optimizing" model, it is necessary to use part of programmed code, just to make it work. But it is needed to make sure to take the less possible resources to just make it work, and not optimization.

5.2 Experiments

5.2.1 Experiment 1: order by demand quantity

The first simulation attempt done with the designed algorithm. Articles ranking based on the demand, the higher demand, the greater rank of the article. Reserves slots for demand proportion to total demand.

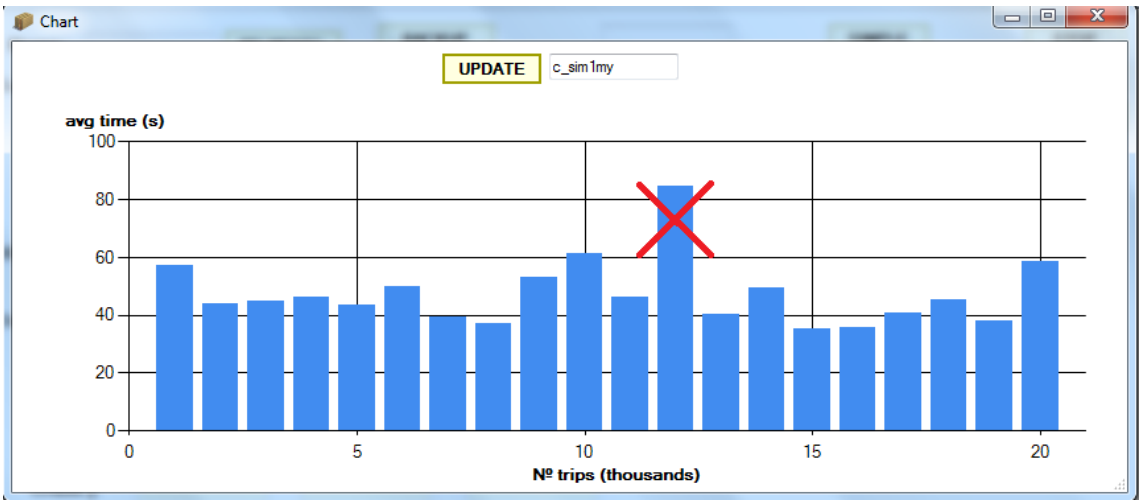


Figure 5.1: Experiments 1

Results:

1. Execution time = 10 h
2. Simulation time = 18 days
3. Simulated trips = 20.700 trips
4. Average time = 47.35s
5. Real time = 40.43s

The results are very poor comparing to the following algorithms. There are still some problems with the software, that is why the short scope of simulation. However, after comparing with the Random simulation, makes one think and realise, that ranking by demand is a totally wrong idea, because it doesn't affect the rotation times and so that storing high demand items in high priority zones only implies reserving huge amounts of slots worthlessly. In fact, high-demand items behaviour is compared to the aggregated demand of several low-demand items. Instead, what does matter is the frequency and the rotation of the article. After checking the stock load in different phases, it is obvious that high priority zones pass too much time empty. Also it doesn't make any sense to use all 10 priority zones, when the stock is full empty.

5.2.2 Experiments 2-3: first slot available

Withdraw operation from closest slot with the first available crane. Deposit operation with the first available crane to the first free slot in the corresponding corridor. Combined operations: first available crane which meets with the requisites(there are at least one of items to withdraw in corresponding corridor, and there are at least one free slot to deposit). Of course the deposit and withdrawal are from/to the closest possible slot.

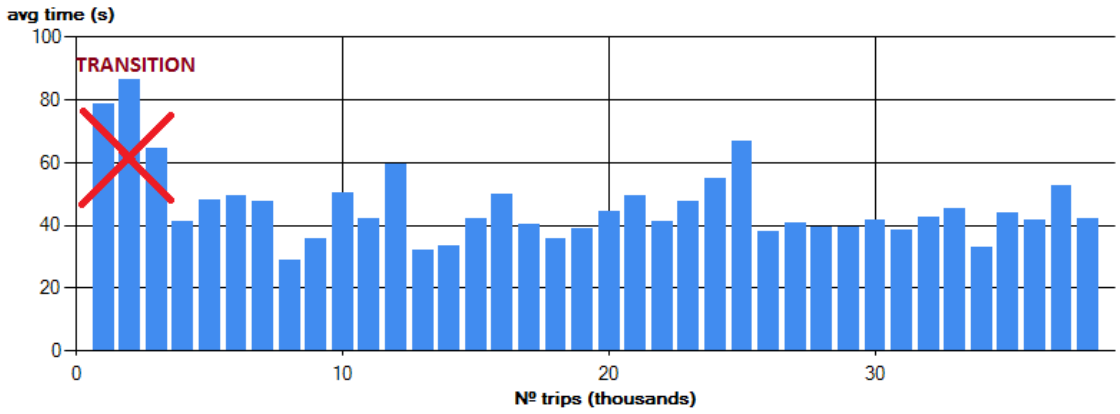


Figure 5.2: Experiments 2-3

- Average time = 42.7s
- Real time = 38.7

Results are better than in the previous experiment, which shows how wrong was the first ranking attempt. These results will provide point of reference for the following attempts to improvement.

5.2.3 Experiment 4: ordering by frequency

In this experiment items RANKING is ordered by frequency of orders and then by demand descendent. STEP TWO is simplified and it doesn't take into account all the slots any more. Only as much as 50, the closest one per each priority zone and corridor. This reduces significantly the experiment durations.

- Average time = 40.9s
- Real time = 34.05s

The results are significantly better than in the first ranking attempt, but this doesn't mean nothing since the first attempt used obviously wrong policies. There is still existing too much of free space during important amount of times in high

priority locations.

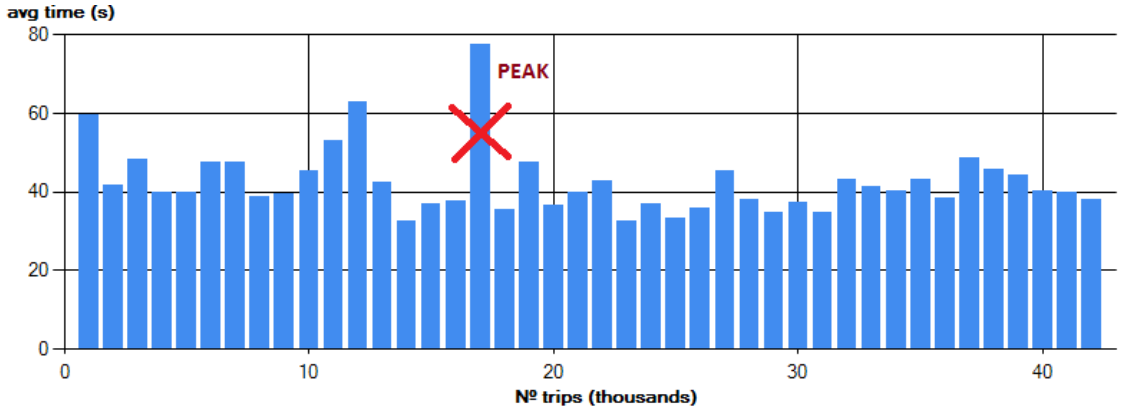


Figure 5.3: Experiment 4

The amount of space reserved is still depends on demand and not in supplies function. Also ordering by frequency does not consider current stock and demand levels.

5.2.4 Experiment 5: combined trips

In this experiment several changes to the STEP TWO are applied. Now the algorithm does not select the fastest trip for COMBINED TRIPS. Now it also considers the distance between withdrawn and deposited pallets, so if it is necessary to deposit or withdraw an item far away in low priority zones, the algorithm will try to take some advantage from this situation. That is, if one of the items (either the one to withdraw or the one to deposit) requires going to PRIORITY ZONE 8, the other item of this withdraw-deposit pair will be picked/placed from the zone 8 or 7, even if it is possible to pick/place it from the PRIORITY ZONE 2. Due to that, the item from the PRIORITY ZONE 2 still available for the next operations.

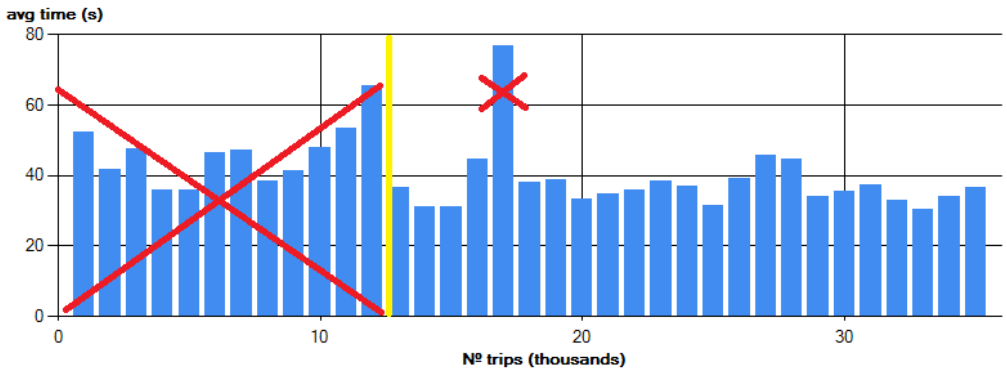


Figure 5.4: Experiment 5

This doesn't increase significantly the short-time results, but may potentially improve the long-time results. In fact, as it can be observed, this does significantly affects the results in positive way.

- Average time = 36.23 s
- Real time = 32.6 s

5.2.5 Experiment 6: free slots amount

In this experiment ranking is ordered by demand frequency descendent, and then by demand quantity ascendant, in order to not reserve too much space in high-ranking slot for "fat demand" articles. The quantity to reserve presents as a very important variable to determine.

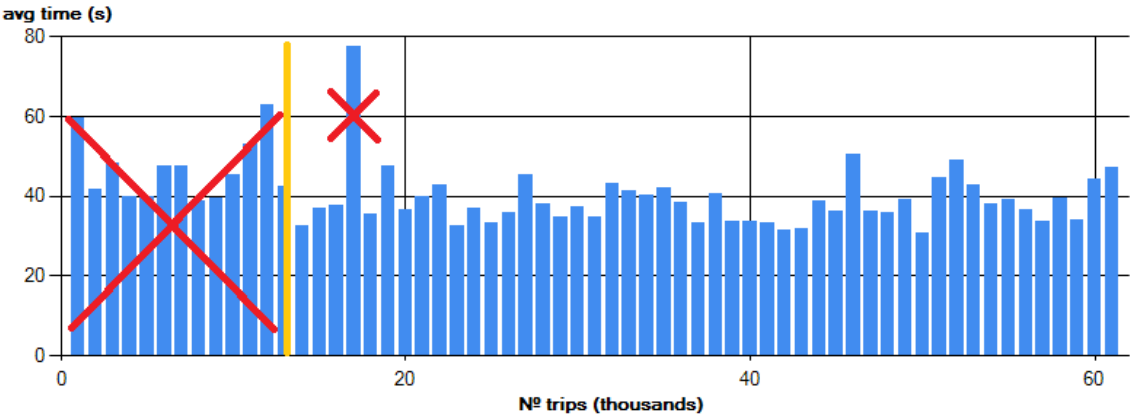


Figure 5.5: Experiment 6

In the Table 5.1 and Figure 5.5 the example of simulation results analysis is shown. The objective results to consider are the ones after deleting abnormal values such as the values during the transitory period, in which the natural re-ordering of warehouse occurs, and the singular peaks.

Trip	Average	w-out transit.	w-out peak
Total	40.7	39.18	38.38
Double	54.98	52.34	52.32
Withdrawal	37.78	36.49	36.5
Deposit	39.15	37.93	35.6

Table 5.1: Results noise reduction

- Average time = 38.38s
- Real time = 34.4s

5.2.6 Experiment 7: based on supplies prevision

In the Experiment 7 several changes to the ranking system are applied: now it has a more close and more real-time treat with reality, but more relying on FORECAST and on SUPPLIES ARRIVAL SCHEDULING. The ranking group now only relies on a real-time number of available slots in that group. The ranking is still ordered by demand frequency descendant, but from now, the criteria of SAFETY STOCK is applied. If the stock is already bigger than the predicted demand for replenishment period, there is no need to put more of this article to the high priority zone. Instead it goes to the tail of the ranking representing a safety stock. As the last, but not the least change, the maximum amount of slots to reserve per article depends from now on the expected withdrawal of this article in the same day.

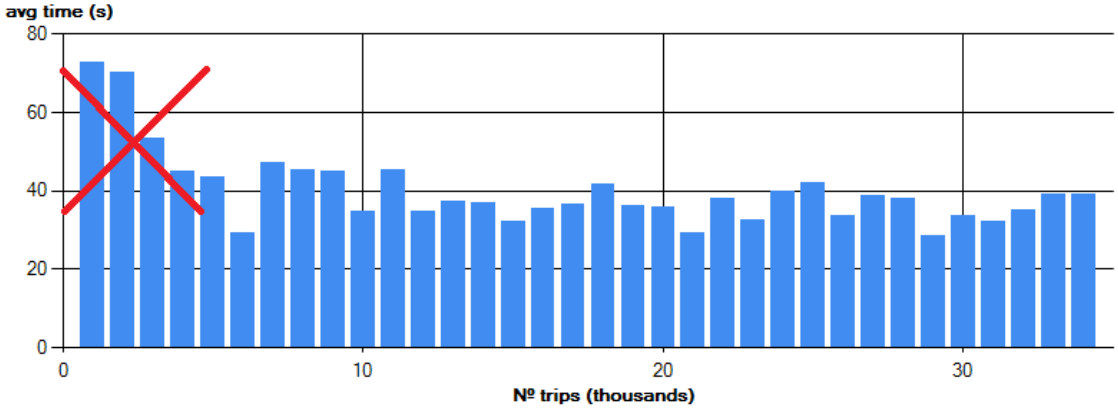


Figure 5.6: Experiment 7

- Average time = 37.23 s
- Real time = 33.25 s

5.2.7 Experiment 9: first slot available

Experiment 9 is another FSA (first slot available) experiment, but with some modifications to the Experiments 2 and 3. Now the application is a bit smarter and knows the closest available slots, and in which corridor they are. This means that the warehouse has to count with some kind of management software, which provides the closest free slot positions and the closest slot with needed item for each corridor.

The changes made doesn't affect significantly the other FSA results. However, these results are more reliable, since the double length of experiment.

- Average time = 42.6 s
- Real time = 39.4

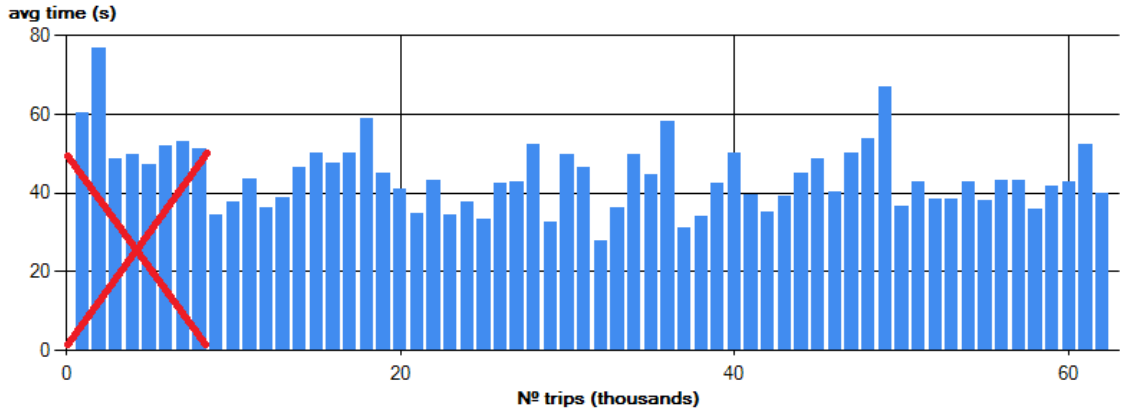


Figure 5.7: Experiment 9

5.2.8 Experiment 10: improved FSA

In this experiment most of the STEP TWO mechanics are incorporated. The model still not considering the ranking, but does now think a bit more in mid/long-term by considering stock load, cranes work load, load of each corridor and distribution of each item in each corridor. Also the combined trips are improved as in the experiment 5, now it also takes into account the distance between withdrawn and deposited object.

Results:

1. Average time = 42.56
2. Real time = 37.25

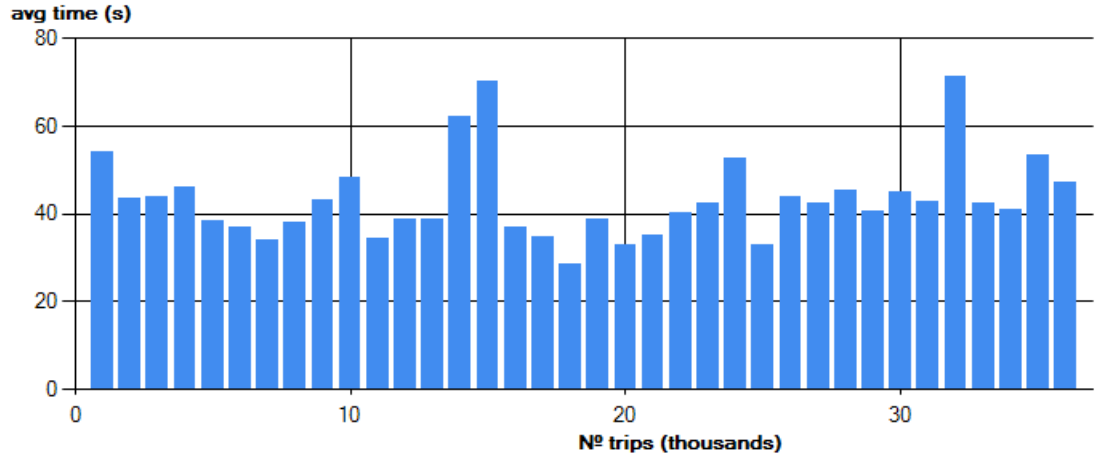


Figure 5.8: Experiment 10

This time there a slight improvement can be noticed. However is not enough to make conclusions big conclusions. The average time is almost the same, but there is

a slight improvement in real time. That means that in certain situation (high-load and overload) the adjustment made for this version of algorithm may result a bit better than in the previous version.

5.2.9 Experiment 11: corridors load balance

For this simulation, corridors where the number of pallets of selected article to deposit is much bigger than in the others corridors, are slightly penalized. Doesn't affect results significantly, but still being one of the best results.

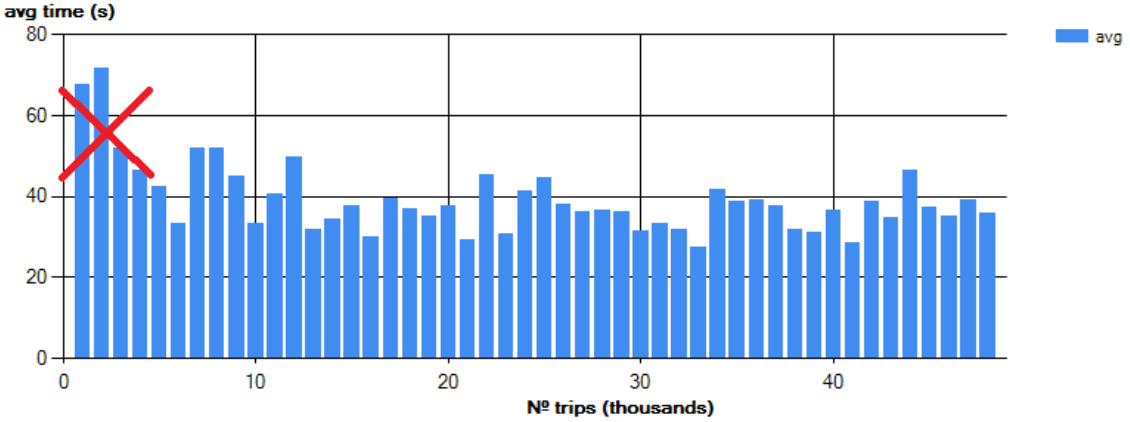


Figure 5.9: Experiment 11

- Average time = 36.01
- Real time = 32.3

5.2.10 Comparison

To compare several experiments results, the data has to be treated and filtered before. However, some visual comparative is provided below to make an idea of results behaviour. The lines represent the evolution of the average trip time, one point per thousand of trips.

In the figure 5.10 the comparison between experiments 6 and 9 is provided. These are the experiments with double length, providing the most reliable results.

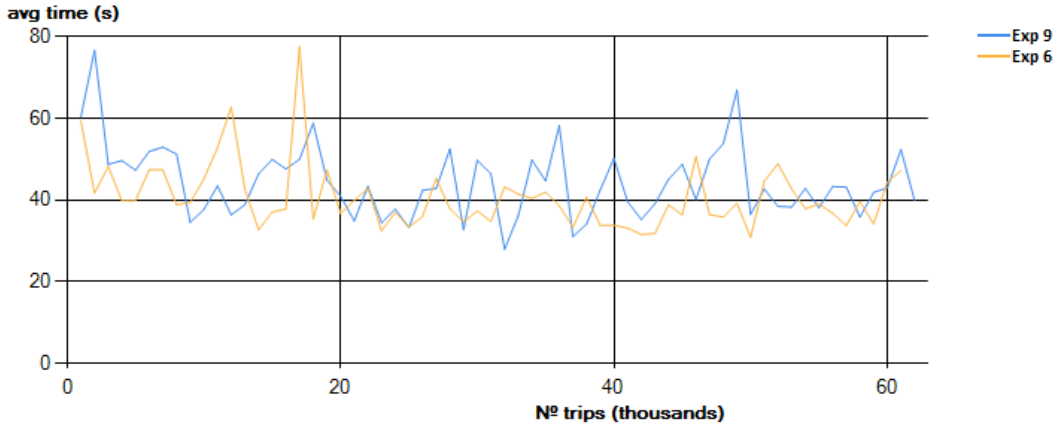


Figure 5.10: Double length experiments comparison (6 and 9)

The rest of experiments can be observed on figure 5.11:

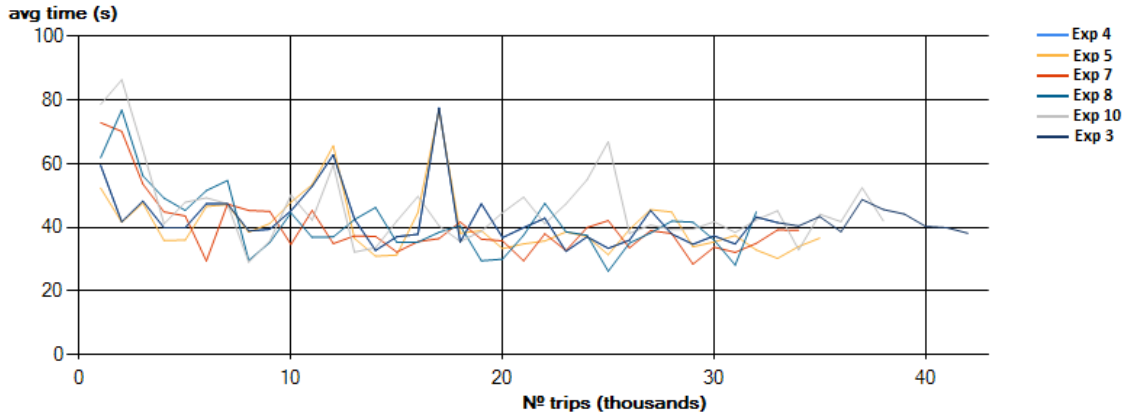


Figure 5.11: Experiments comparison

For the better understanding, all the values are shown in the following Table 5.2:

Experiment	**Deposit**	**Withdrawal**	**Combined**	**Average**	**Real**	**Order**
Exp1	41.05s	41.51s	90.6s	47.46s	40.43s	674.4s
Exp2FAS	41.06s	41.49s	48.78s	42.56s	36.44s	450.5s
Exp4 Freq	37.44s	39.45s	59.3s	40.9s	34.03s	402.18s
Exp5 Freq+	37.6s	36.7s	52.0s	38.93s	32.6s	419.45s
Exp6 Freq+cr	35.6s	36.5s	52.32s	38.4s	32.67	422.18s
Exp7	33.8s	36.75s	50.24s	37.23s	33.25s	415.39s
Exp8	39.8s	35.33s	39.8s	37.4s	34.6s	341.0s
Exp9FAS	44.2s	40.67s	43.52s	42.6s	39.4s	427.16s
Exp10FAS+	39.8s	44.1s	53.9s	42.7s	37.25s	439.69s
Exp11Alg	32.35s	35.76s	49.89s	36.01s	32.29s	423.43s

Table 5.2: Results comparison

The results show that with the ranking system and optimization algorithm is possible to achieve trip-duration reduction of around 10% regarding to the First Available Slot policies.

The most important is to order the items according to their orders frequency and rotation. Other important factors, that improved the results significantly, were:

- The balance of cranes work load. Proper cranes rotation is necessary for the proper functioning of the rest of the elements.
- The distance between withdrawal and deposit slots in combined trips.
- Withdrawals forecast, for the quantity of slots to reserve. However the quantity of slots to reserve still not clear. Obviously the less slots are empty(reserved) in high ranking zone (and for the less time), the better are the results.
- Supplies forecast. Knowing with one day precision the arrivals of supplies, it is possible to reserve slots for part of the amount of pallets incoming that day and avoid keeping too much empty slots.

Other factors were significantly less important:

- The balance of corridor load. This may occur because the corridors are already quite balanced.
- Safety stock. This can be caused due to the infrequent rotation of safety stock and relatively low amounts of this.

- Demand quantity. The higher demand level of item doesn't affect its ranking, since its behaviour is similar to the behaviour of a group of items with less demand. In fact, reserving slots for high demand items only makes things worse, since it reserves huge amounts of slots in high priority zones for a potentially slow article.

Chapter 6

Summary and conclusions

This project work has dealt with several parts of inventory management science. In author's opinion putting effort optimizing inside warehouse times worth more when it deals with a significant high-rotation applications, such as important centres of distribution. On other hand the managing part of this thesis could be adapted and would be useful in any warehouse or center of distribution, automatizing and slightly optimizing the processes. It is also possible to add the financial module to manage the costs or even adjust the application to work together with some ERP application such as SAP.

Also, if the warehouse has a big difference between valley-time demand and peak-time demand, another algorithm is possible, to reposition items in valley-time (during night hours for example)in order to improve service times in peak period.

Another interesting options, considered at start of this thesis, but not implemented due to time limitation, is to build an algorithm for a pick-up zone of a warehouse, where the transport could carry several articles at once, orders are composed by several articles and the algorithm would analyse typical order composition in order to find a correlation between items, and thus, use this correlation to position closer some items.

6.1 Algorithm efficiency

The designed algorithms proven the times reduction of around 5-10% depending on the experiment set-up and algorithm variation. It is hard and pointless to estimate exact improvement, since they vary depending on several factors and hardly depends on stock, demand patterns, promptness and other factor. So that, the algorithm can be very useful in some cases and become a total waste of effort in other.

Obviously the effort of developing such an application becomes more useful when it the inventory itself causes very important part of costs of the company, making it an important objective to work on. These is valid for all others stock improvements.

The in-stock movement algorithm efficiency highly depends on heterogeneity of rotation patterns of demand. The more diverse are the demand patterns, frequency and rotation, the more effect does the classification and different treatment for each of them.

The stock load also affect results. In fact, the more full is the inventory, the more is average time for any policies. But the difference between policies results is also much marked. The more full is stock, the less free space in high priority, the more distance between different rank items, the more effective is the algorithm.

The demand, costs, Pareto and ABC system doesn't really matter when it comes to times optimisation.

There are many factors that can be simplified without affecting to much the efficacy of the algorithm, but greatly improving the execution time of this.

The precision of forecast is also crucial for the algorithm. The more certain is the future, the more efficient could become such an algorithm. If the times of arrival and withdrawal were known with better precision, it would be possible to reserve slots with more efficiency, reducing the reserved slots to a necessary amount in precise time. Therefore, a company with a strict arrivals/withdrawals schedule, would benefit more from similar algorithm.

6.2 Personal achievements

In the original purpose the project was set to be an half practical and half research project. However the application coding required more time than expected, so finally it turned into almost a coding project that includes different quantitative mechanisms with industrial application.

However, the author feels satisfied with the amount of knowledge acquired during the development of this thesis, since this "coding experience" may result very helpful in the future, either coding again in C# or another language, or dealing with database management. On the other hand author regrets not to learn too much about scientific approach to optimization and forecasting.

References

[1] *http://www.sistemasalmacenajes.com.ar/mtranselevadoresg.htm.*

[2] *MySQL 5.0 Reference Manual.* DEV, 2012.

[3] William Brandel. *Inventory Optimization Saves Working Capital in Tough Times.* Computerworld, August 24, 2009.

[4] Juan Jose Guarch Bertoln Eduardo Vicens Salort, Angel Ortiz Bas. *Cuantitative Methods Vol.I.* UPV Publications, 1997.

[5] Andres Carrion Garcia. *Time series analysis and forecasting techniques.* UPV, Statistics Department, 1997.

[6] Ricardo de Navascues y Gasca Jordi Pau i Cos. *Integral logistics manual.* ed. Diaz de Santos, 2001.

[7] Chuck LaMacchia. *Ten Ways to Reduce Inventory,While Maintaining or Improving Service.* The Progress Group.

[8] United Nations Development Programme. *Chapter 3, Human Development Report.* 1992.

[9] S.C.Sharma. *Operation Research: Inventory Control and Queuing Theory.* DPH Mathematic Series, 2006.

[10] John Sharp. *Microsoft Visual C# 2012 Step by Step.* Microsoft Press, 2012.

[11] Mike Snell. *Microsoft Visual Studio 2012.* Sams Publishing, 2012.

[12] H. C. Tijms. *H.C,Algorithmic Analysis of Queues, A First Course in Stochastic Models.* Chichester, 2003.

[13] Sean Wheller. *Supply chain, inventory management and optimisation.* SysPro, 2004.