POLITECNICO DI MILANO

Facoltà di Ingegneria Gestionale

Master of Science in Management Engineering

FACILITY RELAYOUT PROBLEM: DESIGN AND SIMULATION OF A REAL CASE STUDY, AND PROPOSAL OF ALGORITHMS TO SOLVE FLPs.

Supervisor: Prof. Margherita Pero

MSc Thesis of:

LEONARDO ZANONI

ID. No. 919474

Academic Year 2019 – 2020

ACKNOWLEDGMENT

Foremost, I would like to express my sincere gratitude to my tutor Prof. Tommaso Rossi for the support of my thesis, for his patience, knowledge and for the opportunity to apply theoretical concepts to a real company.

I wish to thank the company's management for the possibility to see their offices and productive plant and get their data in order to complete this thesis.

Last but not least, I would like to thank my supervisor Prof Margherita Pero for the possibility to finish my academic career with her.

The credit for this finish line goes all to my family.

To my parents, for always wanting the best for me and pushing me to accomplish every goal.

To my sister, for simply being amazing and a guide model.

To my grandmother, for her love.

To my brother-in-law, for making the time spent in family more precious.

They are and will always be my main supporter.

I would like to thank all my friends. They are the source from which I get my energy. I want to thank all my friends of Montecatini Terme, especially my adored 0 defects, and all my friends of Marina di Carrara. I want also to thank all my friends of Milano, both from Politecnico and not: they are the best memories associated to my journey in the last 5 years in Milan; especially, I want to thank the amazing people of CDS which has supported me for the last year and half; in particular, I am grateful to the guys of 201A/B and 210A for their tireless presence.

I want to dedicate this thesis to my lovely aunt that has always known that I would have reached this goal.

ABSTRACT

Facility Layout Problem is a category of problems concerning the optimization of the allocation of productive and non-productive areas inside a plant, mainly optimizing the flow of raw materials, work in progress, finished products, tools and operators between the considered areas. In order to design an optimal plant layout, some methodologies have been developed in the last decades. These methodologies are perfect to solve problems with a small number of facilities and are good enough to find a suboptimal solution in problems with a huge number of solutions. In this paper will be presented some of the main resolution methods to facility layout problems and some new approaches.

The first aim of this thesis is to apply a hybrid methodology to a real company facing a facility relayout problem.

While developing a solution to a facility layout problem is a good habit to develop a simulation model in order to verify the efficiency and discover eventual inefficiencies of the designed plant layout. The simulation model of a factory can be represented with a discrete event simulation model.

The second aim of this thesis is to develop a model able to represent the company and evaluate the new layout proposed for the company.

In order to study facility layout problems, more papers, some merely theoretical and other also applied to real case study, have been visioned. In order to study discrete event simulation has been taken as reference only the most important book on the subject.

In addition to the real case study which has seen the application of both FLPs concepts and Simulation concepts, will be presented some algorithms developed through Python programming language to solve two categories of facility layout problems. The development of that model has been inspired while analysing the new frontiers of methods to solve FLPs. Since these algorithms are untied from the real case study (fulcrum of this paper), they are introduced only as an enrichment of this paper.

KEYWORDS: Facility Layout Problem, Discrete Event Simulation, Relayout

ASTRATTO

Per problemi di layout dell'impianto (Facility Layout Problem – FLP) si intende una categoria di problemi riguardanti l'ottimizzazione dell'allocazione delle aree produttive e non all'interno di un impianto, principalmente ottimizzando i flussi di materie prime, semilavorati, prodotti finiti, attrezzi e operatori scambiati tra le aree in questione. Per creare un layout ottimale dell'impianto sono state create diverse metodologie negli scorsi decenni. Queste metodologie sono perfette per risolvere problemi con un piccolo numero di aree da allocare e sono invece valide da riuscire a trovare soluzioni subottimali in problemi con un alto numero di aree. In questo documento saranno presentati alcuni dei principali metodi di risoluzione dei problemi di layout dell'impianto e alcuni nuovi approcci. Il primo obiettivo di questa tesi è l'applicazione di una metodologia ibrida a un'azienda reale che sta affrontando un problema di relayout dell'impianto produttivo.

Mentre viene sviluppata una soluzione a uno di questi problemi è buona abitudine sviluppare un modello di simulazione per verificare l'efficienza della soluzione trovata ed eventuali inefficienze del layout. Il modello di simulazione di una fabbrica è rappresentabile come simulazione a eventi discreti.

Il secondo scopo di questa tesi è di sviluppare un modello di simulazione in grado di rappresentare l'azienda e valutare il nuovo layout proposto.

Per poter studiare i problemi di layout dell'impianto sono stati visionati più documenti, alcuni puramente teorici e altri anche applicati a casi reali. Per poter studiare la simulazione a eventi discreti è stato preso come referenza solamente il principale libro sull'argomento.

In aggiunta al case study sul quale sono stati applicati i concetti sia di FLP che di simulazione, verranno presentati alcuni algoritmi sviluppati con il linguaggio di programmazione Python, per risolvere due categorie di problemi di layout dell'impianto. Lo sviluppo di tali modelli è stato ispirato analizzando l'analisi delle nuove frontiere di metodi risolutivi per i FLP. Dato che questi algoritmi sono slegati dal case study (fulcro di questa tesi), sono introdotti solo al fine di arricchire questo paper.

SUMMARY OF THE PAPER

- PURPOSE

In this paper some of the main classic resolution methods for facility layout problems (FLPs) and some new approaches to the problem will be presented. The first aim of this thesis is to apply a hybrid methodology to a real company facing a facility relayout problem.

After having designed a solution, to verify the efficiency and discover eventual inefficiencies of the designed plant layout, the need is to develop a simulation model.

The second aim of this thesis is to develop a model able to represent the company and evaluate the new layout proposed for the company.

In addition to the real case study, which has seen the application of both FLPs concepts and Simulation concepts, some algorithms developed through Python programming language to solve two categories of facility layout problems will be presented.

- INTRODUCTION

Facility Layout Problems (FLPs) is a category of problems concerning the optimization of the allocation of productive and non-productive areas inside a plant, mainly optimizing the flow of raw materials, work in progress, finished products, tools and operators between the considered areas.

Inside FLPs are included many categories of problems and every of them will be analysed in the theoretical part of this thesis. The real case FLP that will be treated in this thesis is a mix between relayout (we can talk about relayout because the company is already operating and the productive and logistic areas already exists and are well defined) and greenfield design (we can talk about greenfield design because the company can benefit from the free of a new building of its property, previously rented to another firm). The company is so interested in knowing how is better to use the new building. The evaluation of the new layout will be predominantly on moving some part of production or warehouse or both to the new plant, but there will be also the possibility to evaluate a relayout of the not moved productive and logistic activities inside the already used plants.

The analysed company is a firm mainly working in two completely different markets: solutions for temperature control and management systems in industrial, professional and domestic fields, and mosquitos' nets and screens. The company is facing the relayout process simultaneously to the

drafting of this paper and will start to effectively move productive and logistic areas in about one year from this moment.

After having found the best layout according to the adopted FLPs methodology, the effectiveness of the new proposed layout (also the discarded ones) will be analysed through the study of the simulation model. The results obtained thanks to the model inserted into the simulator will be used for final discussions and comments on the layouts with the aim to choose the best possible layout for the company.

- METHODOLOGY USED TO SOLVE FACILITY RELAYOUT PROBLEM

In order to solve the relayout problem of the company there is the need to have as input a file containing the picking list of production carried in a considerable amount of time and there is also need for info about the dimensions of the plant and the actual location of machineries. Thanks to these data, it is possible to apply a hybrid methodology in which a feasible layout is manually-originated and its total cost is calculated through a standard matrix of costs multiplied by the distance matrix between warehouse and productive areas.
For every analysed layout were calculated:
- X-position and Y-position of the Centroids of the productive areas in a grid system

- Rectangular distance between warehouse/s and each productive area

- The total cost of the layout
To calculate the X-position and Y-position of the Centroids, the same standard weighted area method used when calculating the centroid of a figure composed of rectangles has been used.
In some cases, the rectangular distance could have not been applied due to physical constraints that forced the handling route to follow a longer path. In that cases new points were added to calculate the total travel path.
The total cost of layout is obtained by summing all the rectangular distances of areas *i* multiplied by the kgs exchanged in 1 year.

Simultaneously to the quantitative analysis, a qualitative analysis has been carried out. While drafting each possible new layout, observations on the benefits and maluses of the solution were made.

After finishing the qualitative and the quantitative analysis on each possible scenario, a summing up table has been drafted. Thanks to that table it was possible to easily discard the worst scenarios and

focus only on the most relevant ones in order to choose, after a deep comprehension of the few remained scenarios, which was the best layout at the moment.

- SIMULATION MODEL

A simulation model is composed of modules and relations which are used to describe the components of the system and their interactions and behaviours.

In order to create a model, three main types of data should be available:

- The logical flow of the activities: this kind of data is used to obtain the Petri Nets Graphs representing all the company's activities (the focus is on internal logistic and productive activities). In addition to that, also the data about all the resources exploited in these processes should be available.
- Layout of the plant (productive and non-productive areas) and data about material handling
- All the data concerning operative times

The first two kind of data are the same one needed to design the optimal layout solution of the FLP.

In the model presented in this thesis some assumption and simplification choices have been made; the limits of the proposed model are a consequence of some missing data about company's productive and logistic operations.

The software used to develop the simulation model is Arena Simulation (by Rockwell Automation). All the modules used are shown and explained in this paper.

The proposed model can be divided into 5 main areas (explained in chronological order):

- Truck arrival and assignment of goods between cross docking and warehouse
- Picking for production and unloading of raw materials from warehouse
- Reception of raw materials, production and withdrawal of finished products
- Picking for sale and packaging
- Truck loading and truck departure

- RESULTS

After the completion of the model, it was possible to run it to simulate the behaviour of the company. Three different simulations of the same model set with three different kinds of data were made: respectively, the model run with the characteristics of the actual company configuration and with the two most valuable scenarios discovered with the hybrid methodology used to solve the facility relayout problem of the company.
After the run of the simulation model set with the three different scenarios, the initial focus of the analysis has been on the availability (1 - average utilization) of the transport operator.
The two new scenarios see a reduction of the average utilization of the transport operator of 31% and 35% in respect to the actual configuration. The reduction in both the cases is noteworthy.

After having confirmed the effective reduction of the average utilization of the transport operator, it was possible to look for other improvements. The focus moved on: other resources/operators' average utilization and the queues' length and average numbers of entities in queues.
The analysis revealed that there is no evidence of a correlation between the reduction of the average utilization of the transport operator and the insignificant variation in these two kinds of variable of interest. The variation of these measures is attributable to the stochasticity of the model.

Given the fact that there is no evidence for other quantitative benefits, except for the transport operator, the choice on which layout is the best should depend only on the qualitative benefits that it is able to bring.
After having checked the solutions (given by the hybrid method to solve FLP), through the simulation model, the final optimal layout was chosen.

- ALGORITHMS PROPOSED

This paper also treats an argument uncorrelated with the real case study but correlated with FLPs. In fact three algorithms to solve two family of facility layout problems are proposed.

The first algorithm solves unequal area (facilities can have different areas) greenfield (new layout from scratch with no constraints inside the plant) FLP that works allocating randomly the facilities into the grid system. For each solution found the algorithm checks the feasibility of the solution. If it is a physically feasible layout, the algorithm calculates the total material handling cost, if it is not a

feasible solution, the algorithm searches for another solution until a feasible one is found. The best layout is the one presenting the lowest total material handling cost.

The second algorithm solves equal area greenfield FLPs. Differently from the first algorithm, given a location, the algorithm assigns a facility to it. Each layout originated is a feasible layout. It is able to find a suboptimal solution for problems with high number of facilities in few minutes.

The third algorithm starts from the second algorithm and adds a local search operator that, given the suboptimal solution found, searches a better solution in its neighbourhood exchanging the locations of two facilities at every iteration and saving the new solution obtained if it is better than the starting one.

INDEX

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Plant layout design is the activity to choose how to position machineries and the non-productive areas inside a production plant. This task belongs to the industrial management world and combines qualitative and quantitative analysis in order to find the best layout possible in order to optimize factory's operations. Plant layout design can be made both on operating plants and in plants that will be opened in the future.

Given the fact that a good plant layout gives the possibility to save money, its design has always been crucial for the industrial factory and in the last decades has risen in importance and its tools and methods have been used also in other fields unrelated to the productive world for example in the hospitals' layout design.

There are multiple benefits from a plant layout design, mainly quantitative benefits but also qualitative ones. These benefits include: reduced work-in-process inventory, reduced waiting time, higher quality, and reduced material handling costs among machineries and other minor benefits. [1]
In a globalized world, especially in the following years that the economy will be turbulent after the economic crisis deriving from corona virus disease 2019, companies should try every possible action in order to be competitive. Unfortunately, in the Italian industrial landscape plant layout design has not found a great popularity yet.

The main aim of a plant layout design is to allocate machineries in the way that minimizes the cost of transportation/handling of Work in Process, Raw Materials, Finished Products, Tools and Operators.
Another purpose is to potentially reduce the working force as a consequence of a reduction in wasted time caused by all the inefficiencies brought by a poorly designed layout. For examples: bad machineries allocation or bad routing schedule in assembly lines with one operator working on more machineries.
A qualitative aspect reached by a good plant layout design is the optimization of the working space (move away machineries from offices, bring near the locker room and canteen, increase safety moving pipes from operators, …) and a potential job enlargement, both enhancing operators satisfaction level.

The annual money saved from the inefficiencies can be reinvested in new projects or used to reduce the selling price. Both of the choices will contribute to increase the competitiveness level of the company.

In order to check the feasibility of a new design, there is the need to represent the future real system through a simulation model and check through a simulation how the supposed design will work when realized in the real world and if it satisfies the monetary targets set (i.e. in a Net Present Value logic the machinery allocation cost to shift from the actual layout to the new layout should be repaid, in a reasonable time frame, from the yearly money saved thanks to the optimization of the plant). A new proposed layout and a simulation model developed through Arena and representing the assembly lines of a real case study (an Italian SME in Novara's province) will be seen in this thesis.

To enable the reader to better understand the paper, a brief description of the chapters is given below.

First Chapter

In this part will be analysed the scientific papers chosen to better represent the theoretical concepts at the base of a factory layout design.

At the beginning, will be deeply explained which are the benefits of a plant layout design and how they are reached.

Consequently, will be shown in detail some methods to allocate machineries inside a plant. There will be a structured explanation on how they work, when they can be used and their pros and cons.

After that will be presented some allocation algorithms; these algorithms, present in the available review, exploit concepts and theories from other fields. Even if the most used methods are powerful enough to solve almost every layout design with a small number of facilities, the study of new algorithms has given the possibility to improve the results of this field of industrial engineering for problems with high number of facilities.

Second Chapter

In this section will be introduced the theoretical concepts regarding the simulation. Will be explained what a simulation is and what a Discrete Event Simulation is, starting from the real system and going through the conceptual model and the computational model and arriving then at the solution. During the explanation of the conceptual model will be taken into consideration the Petri Nets (place/transition (PT) nets), instead, while explaining the computational model will be taken into consideration some software's functionalities.

Third Chapter

This part will be used to apply the theoretical concepts seen in the first chapter to a real study case. The company analysed is characterized by the production of completely different products and the

need to move part of the production / warehouse to a new building. The focus of this chapter will be to analyse some new possible future layouts and highlight their qualitative aspects in order to choose, at the end of the chapter, the best layout. Given as input in this phase, the Petri Nets Graphs representing all the company's logistic and productive activities will be shown and commented.

Fourth Chapter

This chapter will apply the theory seen in the second chapter to the real case study. Starting from the Petri Nets Graphs and the layouts presented in the third chapter, and having gathered all the other data needed, will be possible to realize the simulation model.

Will be presented the model made with Rockwell Automation – Arena Simulation software ®, explaining all the peculiarities of the model and of the software itself.

After the run of the model will be possible to comment the results obtained for each layout studied.

Fifth Part

Starting from the allocation algorithms presented in the literature review, in this chapter will be proposed three algorithms to solve two different FLPs.

All the algorithms proposed are developed in Python language. While the first one is an algorithm to solve unequal area FLPs, the last two can solve only equal areas FLPs.

After these five chapters, there will be a conclusion paragraph in order to recap the legacy of this thesis.

1. FACILITY LAYOUT PROBLEM

1.1 INTRODUCTION

Analysing the state of art of the Facility Layout Problem (FLP), it is clear that there are two main approaches to design new optimized layouts, based on robustness and/or flexibility.

The first approach is called Dynamic Facility Layout Problem (DFLP). The dynamicity of this approach is given by the fact that it considers multiple production periods and layout design. The concept at the base is that a company will change production mix and will insert new products and remove others, doing so the optimal layout will change for every production period. The aim of DFLP is to minimize the total cost of material handling and machinery handling during more time periods. Handling of machineries can be highly expensive and disruptive; i.e. when the plant should be shut down and the production should be completely stopped.

The second approach is called Stochastic Facility Layout Problem (SFLP). The stochasticity is given by the fact that most of the companies have a variable product mix and demand and in order to originate a flexible layout to respond to high variability DFLP's methods are not efficient. As input for a Stochastic FLP are given random variables with known parameters (routing, material handling costs, …) and statistical parameters (demand mean, demand variability, …).

Briefly: the difference from DFLP and SFLP is that the first one has fixed demand inside a period and the demand changes from period to period, while the latter has random demand inside a period and the demand does not change from period to period.

In the literature is sometimes mentioned a third approach, called the "fuzzy approach" that can be based on fuzzy numbers or linguistic patterns [2]. Due to the scarcity of the literature about this topic, and also due to the fact that is most of the time associated with Genetic Algorithms, we will not focus on it but just mention it for completeness. So, in the next pages we will go in deep in just the two main approaches indicated above.

1.2 STOCHASTIC FACILITY LAYOUT PROBLEM

As already stated, this kind of approach addresses the stochasticity of the demand and of product mix. For what concern this approach there is not a single formulation of the problem, but there are some criteria used as main objectives. We will consider the three main criteria used as objectives that are

Flexibility (for future changes), Robustness (to uncertainty), Dominance (optimality). In *figure 1* is possible to see a schematization of the approach's three main criteria.

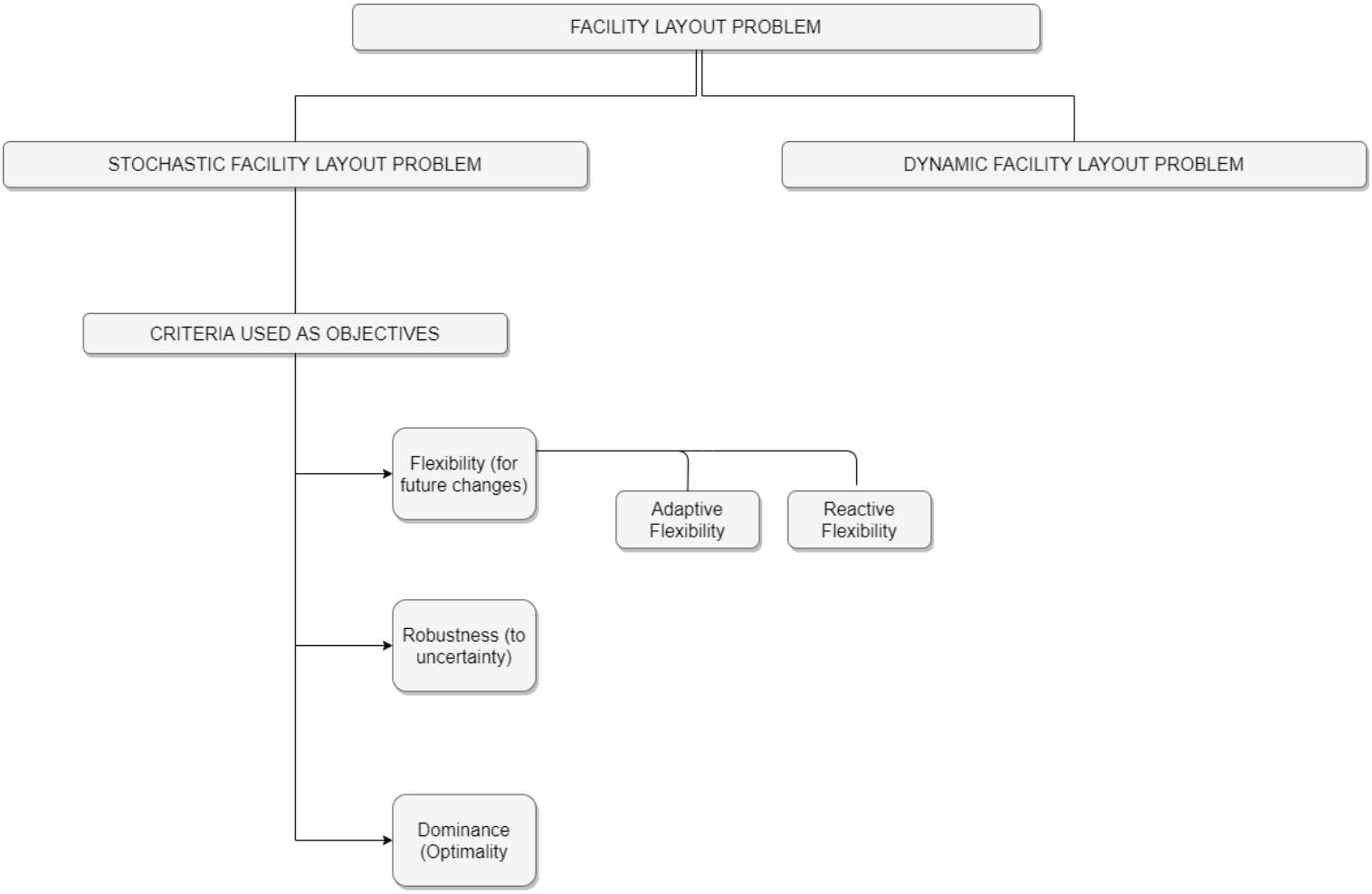

Figure 1. Scheme of Stochastic Facility Layout Problem

A facility layout is said to be robust if it is able to perform well over a multiplicity of different scenarios and outcomes. A facility layout is said *flexible* if it is able to rapidly adapt to changes without considerably affecting performance. E.g. When a layout has a vacant space used to strategically prevent a demand increase, the layout is flexible because it can rapidly increase demand using that space but it may not be robust due to the fact it may not be optimized in the initial time period.

1.2.1 Flexibility

The first study on the variability of demand in a plant has been made by Shore and Tompkins in 1980 [3]. Doing so, they were the first to put the focus on the concept of flexibility in a plant layout while the demand is not deterministic but stochastic.

According to them, the flexibility is the possibility of a layout to respond to current and future product mixes. In order to calculate the flexibility of a layout they brilliantly introduced a coefficient of penalty for each layout that represented the effectiveness to adapt to different demand levels. With this methodology they were able to insert in the standard computation of layouts the variability factors.

Some years after Shore and Tompkins, Gupta [4] released a paper in which he used simulation to solve SFLP. The stochasticity is inserted in the flow matrix between departments. Using a Monte Carlo Simulation he generated randomly the flow between all pairs of departments; doing so he took into consideration that flows between departments could be not deterministic. The nominal flows are considered independent, normally distributed, with known mean and variance. His study is limited to departments with square shape and equal size and does not consider future relayout costs that can have a huge impact and so are not negligible. Taking as input the results of Monte Carlo Simulation and with the use of CRAFT heuristic, was computed for each layout generated the distances between all the couple of departments. After having collected all the data on the distances between all the couple of all departments on the various scenarios, is computed the average and with the averages is associated to each pair a penalty value associated to their distance. The layout with the smallest penalty value is considered the most flexible layout.

The first to conceive a new type of flexibility based on the cost of a future relayout were Bullington and Webster [5] in 1987. Their approach differs from the previous ones: the flexibility of a plant layout to adapt to the changes in demand is called Reactive Flexibility, while the flexibility of a plant layout to adapt to a relayout is called Adaptive Flexibility. While reactive flexibility is based on material handling costs, adaptive flexibility is based on relayout costs (handling of machinery, production stoppages, …).

Four years after the paper of Bullington and Webster, Savar [6] developed the first simulation algorithm usable to generate and evaluate layouts in a stochastic environment. The peculiarity of this algorithm is that its objective function includes calculations on both reactive and adaptive flexibility; in fact there is a weighted sum of material handling costs, proximity ratings and relayout future costs.

Obviously, each information used later in the objective function is originated by a simulation made with stochastic input data; mean and variance known.

1.2.2 Robustness

Robustness is the capability to perform well over a multiplicity of different scenarios and outcomes. In mathematical terms, is the defined as the frequency with which a layout falls within a percentage of the optimal layout solution in different production/demand scenarios. In other terms the most robust layout is the one that more often is close to be optimal while varying the production/demand. Note: is not assured that for a specific production scenario the most robust layout is the optimal one, or it can even be inefficient.

Rosenblatt and Lee [7] developed the idea of robustness in 1987. The problem taken into consideration is a single period stochastic FLP. As foundation to their formulation, they said that it is difficult and time wasting to find the exact value of the joint probabilities of the different scenarios and so they preferred to represent the demand as in PERT/CPM models and so as a 3-point random variable. They used Quadratic Assignment Problem (QAP) to solve the procedure.

1.2.3 Dominance

The concept of dominance in SFLP is formulating criteria to be used to determine which alternative layouts are efficient, after that a modified QAP formulation is used to solve the SFLP single period version, while for a multi period version has been introduced a DP formulation. Both the contributions come out from the searches of Kouvelis and Kiran [8][9], which used this concept after having assigned to each production scenario a probability of happening.

1.3. DYNAMIC FACILITY LAYOUT PROBLEM

The Dynamic Facilities Layout Problem has the objective to minimize the overall costs given by the sum of the flow costs between departments over a series of discrete time periods and the cost of the machineries handling due to the relayout between the considered time period. Usually the departments' sizes are considered as unitary and the problem is solved as a modified Quadratic Assignment Problem. [10]

The main problem of DFLP is the dynamicity and so the estimation of the future changes in the layout and the schematization of them into different and subsequential discrete scenarios.

DFLP born in 1983 with the paper of Nicol and Hollier [11] in which they described the necessity to introduce to the static version of the FLP the fact that during the years, the changes in the layout of a plant are multiple and their costs are not negligible. The management should consider this aspect while implementing a new design. DFLP considers the trade-off between material handling costs and rearrangement costs in order to find the optimal scenario for each time frame.

Note: material flows between departments change over time, that's why there is the need to change the layout. The material flow in each period is considered as fixed and not variable as in the Stochastic FLP.

The Quadratic Assignment Problem is a combinatorial optimization problem in the branch of optimization and operation research, introduced by Koopmans and Beckmann to solve the facilities location problems. [12]

Urban [13] has expressed DFLP in a QAP is the following form:

Objective function:

$$Minimize: \sum_{t=1}^{T}\left[\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N} f_{ikt}d_{jl}x_{ijt}x_{klt} + \sum_{i=1}^{N} s_{it}y_{it} + r_t z_t\right]$$

Subject to:

$$\sum_{i=1}^{N} x_{ijt} = 1 \qquad \forall\, j,t$$

$$\sum_{j=1}^{N} x_{ijt} = 1 \qquad \forall\, i,t$$

$$x_{ijt},\ \ y_{ijt},\ \ z_t \ \ \in \ \ \{0,1\} \quad \forall\, i,j,t$$

Where:

N = total number of departments and locations

f_{ikt} = flow between department *i* and department *k* at the period *t*

d_{jl} = distance between locations *j* and location *l*

s_{it} = variable rearrangement cost of moving department *i* at the beginning of the new period *t*

r_t = fixed rearrangement cost of any rearrangement at the beginning of the new period *t*

$$x_{ijt} = \begin{cases} 1 & if\ department\ i\ is\ assigned\ to\ location\ j\ in\ period\ t \\ 0 & otherwise \end{cases}$$

$$x_{klt} = \begin{cases} 1 & if\ department\ k\ is\ assigned\ to\ location\ l\ in\ period\ t \\ 0 & otherwise \end{cases}$$

$$y_{it} = \begin{cases} 1 & if\ department\ i\ is\ relocated\ at\ the\ beginning\ of\ period\ t \\ 0 & otherwise \end{cases}$$

$$z_t = \begin{cases} 1 & if\ arrangement\ are\ made\ at\ the\ beginning\ of\ period\ t \\ 0 & otherwise \end{cases}$$

In order to solve the QAP there are 4 main categories of algorithms / methods, in *figure 2* it is possible to see a scheme of the 4 main algorithms / methods categories and their subcategories.

In the next pages will be briefly explained at first Exact Methods and Hybrid Approaches and then there will be a focus on some Heuristic Methods and Metaheuristic Methods.

The choice to focus only on Heuristic Methods and Metaheuristic Methods is given by the importance they have in the literature: they are the most studied at the moment because they do not have limitations in number of facilities and give suboptimal results. Is to be said that in the future maybe there will be a focus on Hybrid Approaches.

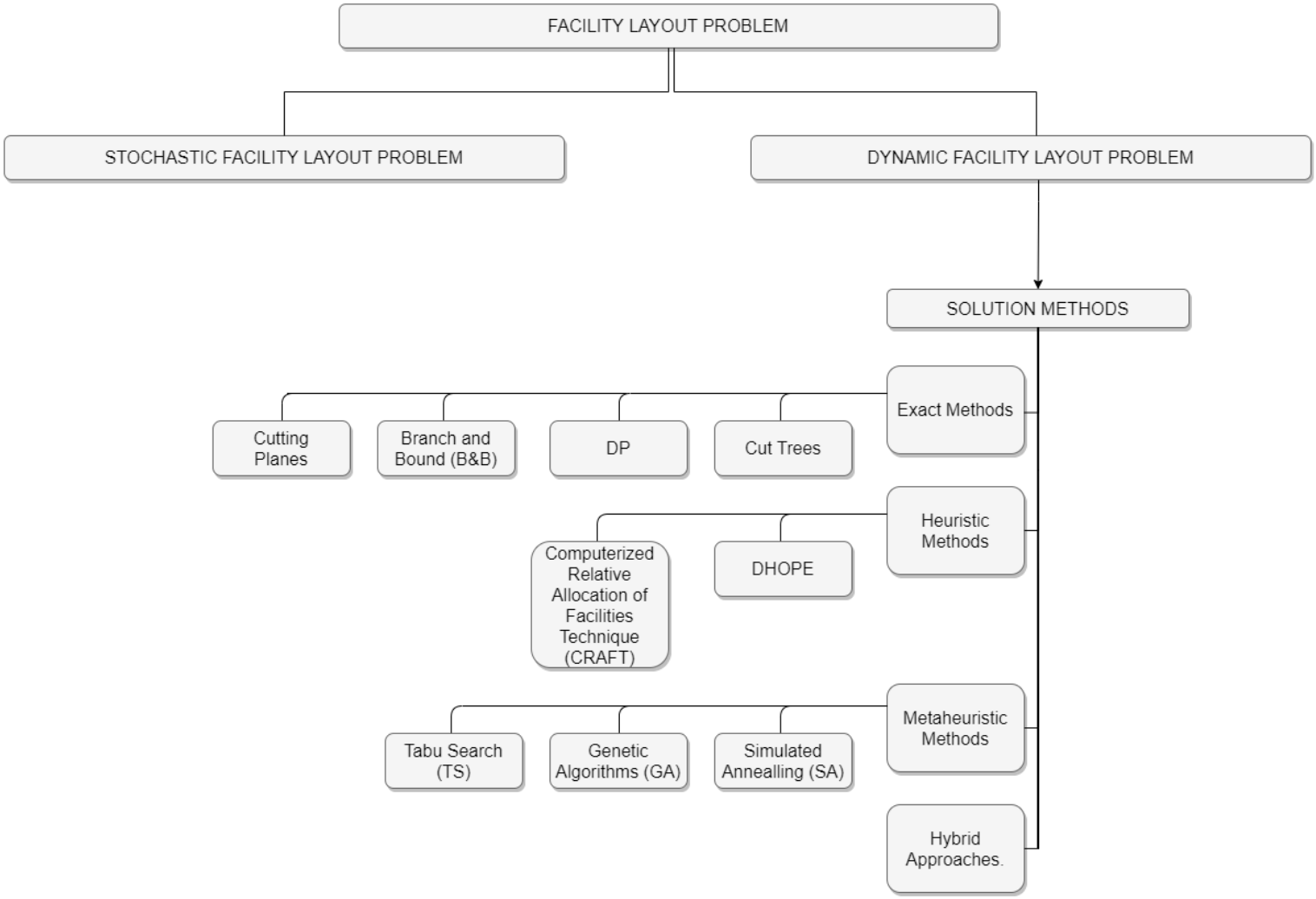

Figure 2. Scheme of Dynamic Facility Layout Problem

1.3.1 Exact Methods

Exact methods are procedures or algorithms that allow the optimization of the plant layout. The only reason that the other already mentioned methods exist, is that exact methods are limited in the maximum number of departments able to analyse simultaneously in order to find the optimal solution. The limitation is at about 9 departments and can vary based on the model used and the solution approach adopted. In *table 1* are shown some models present in literature.

Paper	Model	Year	Max Dept.
Montreuil [14]	FLP0	1990	-
Meller and Gau [15]	First MIP Model	1996	6
Sherali et al. [16]	Polyhedral approximation	2003	9
Castillo et al. [17]	MINLP-ϵ accurate scheme	2005	9
Anjos and Vannelli [18]	Two-phase mathematical programming	2006	-
Liu and Meller [19]	First Sequence pair in MIP	2007	9

Table 1. Examples of Exact Methods' models [20]

The solution approaches can be divided into two categories: the ones searching for optimal solutions in plants with departments with equal areas and the ones able to find the optimal solution also in plants with departments with unequal areas.

Inside the pool of all the possible methods and all the algorithms of Linear Programming, Mixed integer Programming, Two-Phases Programming and more, Lacksonen and Enscore [21] compared 5 procedures: Computerized Relative Allocation of Facilities Technique (CRAFT), Cutting Planes, Branch and Bound (B&B), DP (Dynamic Programming) and Cut Trees.
After the generation of some problems and the resolution of that set of problems through all the five algorithms, it resulted that the most effective one is the Cutting Planes algorithm.

Even if cutting planes algorithm has been demonstrated to be the best algorithm it will not be analysed in this thesis. The focus instead, inside the paragraph of Heuristic Method, will be on Computerized Relative Allocation of Facilities Technique (CRAFT) that is probably the most widely used algorithm, and it has also been applied to solve the business case present in this thesis.

1.3.2 Hybrid Approaches

Being aware of the reality that exact solution approaches are not practical for large problems, Hybrid Approaches have been studied. Tendentially, are mixes of Metaheuristic Methods and Exact Methods, trying to take from the first one the possibility to intervene on large-scaled problems, while trying to take from the second one the possibility to arrive at the optimal solution.

Erel et al. [22] invented a new hybrid algorithm divided into three phases. In the first phase are generated the viable layouts and are chosen the ones that seem to be close to be optimal. In the second phase there is another problem formulation applied on the set of chosen layouts and is solved with Dynamic Programming (DP). In the third phase the final solution is improved with local improvement procedures. This algorithm has been demonstrated to be at least as competitive as the Metaheuristic Methods that will be shown later.

1.3.3 Heuristic Methods

An algorithm is said to be heuristic when it is designed to solve a problem in a faster and more efficient way than exact methods, sacrificing optimality, accuracy and precision in favour of speed. So this kind of algorithm tries to search for a sub-optimal solution when it is too difficult to solve the whole problem in an exact way, finding the only optimal solution.

Kochhar and Heragu [23] developed an algorithm for the multi-floor DFLP. Multi-floor DFLP is when you do not have just one floor but a multiplicity of them. It is also the case in which a company operates in more than one building close together. The algorithm developed was called Dynamic Heuristically Operated Placement Evolution (DHOPE). Starting from an initial layout for the first period, this algorithm tries to minimize the sum of the material flow costs and the rearrangement of machineries for the subsequent period. In two iterations the algorithm is able to find the near-optimal solution for two consecutive periods.

1.3.3.1 Computerized Relative Allocation of Facilities Technique (CRAFT)

CRAFT is tendentially used to solve single period relayout FLP but it can also be used for Dynamic FLP given that it is possible to take into consideration possibility of future changes.

The main reasons for relayout are:

- change in production volume (and so new layout due to the needed expansion)
- change in process technology (and so new layout due to new machineries)
- change in product produced (and so new layout due to new machineries)

CRAFT Methodology can give a 23% savings of material handling cost, increasing productivity and reducing efforts made by operators [24].

Steps of CRAFT

1. Insert as input the following information:
    a. # of departments
    b. # of interchangeable departments
    c. Initial layout
    d. Cost and Flow matrices
    e. Area of departments
2. Compute the centre of gravity of each department (the centroid of polygons composed by rectangles).
3. Create the distance matrix using centroids. The distance can be computed as rectilinear or Euclidean.
4. Compute the total cost of the layout as the sum of the element in the matrix obtained by the product between cost, flow and distance matrices.
5. Find all the possible pairwise substitutions of departments. The substitution is admitted if the departments have at least a common border.
6. Compute the centroids and the new total cost for each possible pairwise substitution.
7. If there is a pairwise substitution that reduces the total costs, it is the new layout. If there are more solutions reducing the total cost, the one with the minimum cost is the new layout. In both cases go back to step 5. If there are no more pairwise substitutions reducing the total cost the process is ended and the solution is obtained.

1.3.4 Metaheuristic Methods

Metaheuristic Methods are mathematical optimization methods that through a search algorithm may provide a sufficiently good solution to an optimization problem. Is used when there are incomplete or imperfect information or, as in our case, when there is a limited computation capacity. Metaheuristics algorithms investigate only a subset of the all solutions in order to find between them the optimal solution that is close to be the optimal one.

One way in which a Metaheuristic algorithm can work is finding a final subset of solutions that is usually chosen taking at first a random subset, and later choosing in the originated subset a new final subset composed by the best solutions of the first subset. Random solutions in the closeness of the solutions in the final subset are evaluated, and within them is chosen the best solution that is considered to be a close-to-be optimal solution.

In *table 2* is possible to see some solution approaches present in the literature.

Paper	Solution	Year
Tate and Smith [25]	GA and Penalty function	1995
Tavakkoli-Moghaddain and Shayan [26]	GA	1998
Li and Love [27]	GA	2000
Logendran and Kriausakul [28]	Tabu search Algorithm	2006
Liu and Meller [19]	Finding initial Solution by GA for MIP	2007
Scholz et al. [29]	Tabu Search with Slicing tree	2009
Komarudin [30]	Ant Colony with Tabu Search	2010
Scholz et al. [31]	Tabu Search with Slicing tree	2010
Baykasoglu and Gindy [32]	Simulated Annealing	2000
Balakrishnan et al. [33]	GA	2003
Dunker et al. [34]	GA and Dynamic Programming	2005
Baykasoglu et al. [35]	Ant Colony with Budget constraint	2006
McKendall and Shang [36]	Hybrid Ant System	2006
Balakrishnan and Cheng [37]	GA and Simulated Annealing	2006
Kulturel-Konak [38]	Review dynamic and stochastic FLP	2007
Drira et al. [39]	A survey in Facility Layout problem	2007
Balakrishnan and Cheng [40]	Forecast Uncertainty and Rolling Horizons	2009
McKendall and Hakobyan [41]	Tabu Search and Boundary Search	2010
Sahin et al. [42]	Simulated Annealing with Budget limits	2010
Pillai et al. [43]	Robust Optimization Approach	2011
Xu and Li [44]	Fuzzy Environment with Swarm Optimization	2012
Chen [45]	A new data set and Ant colony Optimization	2013
Mazinani et al. [46]	GA	2013
Pourvaziri and Naderi [47]	GA	2014
Ulutas and Islier [48]	Clonal Selection Based Algorithm	2015

Table 2. Example of Metaheuristic Methods' model [20]

In the following pages will be taken into consideration three well known metaheuristic algorithms that are applied to DFLP. In order: Tabu Search (TS), Genetic Algorithm (GA), Simulated Anealling (SA).

1.3.4.1 Tabu Search

Tabu Search (TS) was developed in 1986 by Glover [49] and is a metaheuristic approach able to solve dynamic facility layout problems. TS starts from the current solution x_t and moves in the feasible space to the best solution in its neighbourhood $N(x_t)$ in each iteration *t*. The new solution is simply called x_{t+1} and becomes the new x_t in the following iteration. One problem occurring is caused by the uncertainty of finding a new better solution in $N(x_t)$; to solve this problem and avoid repetition of the same solutions, at each solution are applied some specifications and that specification are saved in order to easily check if the solution has been already generated. This method takes the name of Short Term Memory (STM).

Other two mechanisms are used by TS: Diversification and Intensification. Diversification obliges the algorithm to search in a vast area of solution of the feasible space of solution before converging to a final solution. Intensification obliges the algorithm to search in the $N(x_t)$ of the solutions with specifications more desirable [50].

Step 1: Initial solution

In order to initialize the algorithm, some feasible initial solutions should be created. In order to do that it is possible to create random layouts or use some algorithms able to create solutions. Also for this algorithm the GIGO (garbage in, garbage out) rule is valid. It is way better to start with a good initial layout solution originated by other algorithms instead of starting from random ones. From the paper of Heragu & Kusiak which developed the ABSMODEL [51], the number of initial solutions can be set at $2^{n/2}$ where *n* is the number of departments.

In this phase, after the generation of the initial solutions, is calculated the objective function of all of them and it is stored in a memory called Adaptive Memory (this is the base for the short term memory explained above) [52].

Step 2: Neighborhood structure

There are many possible neighborhood structures. This step is really important in the TS method. Defining a neighborhood structure means choosing how the algorithms generates a new solution x_{t+1} for every initial solution x_t. One easy approach could be a binary movement: as seen in the CRAFT method, the TS can choose two facilities and exchange their locations. This assures that the new solution is in the feasible space. The new solution is accepted only if it improves the objective function

in respect of the actual solution. The new solution should also be inserted in the AM: the indices of the facilities *i* and *j* just exchanged are inserted in the tabu list for the next ϑ iterations. Doing so in the next ϑ iterations the algorithm will not exchange them again unless is the only exchange that improves the solution. Since finding a better solution in a $N(x_t)$ is time consuming and can also be impossible, there is a limitation of α possible iteration made. Both α and ϑ are chosen by the plant layout designer.

Step 3: Intensification, selection

Intensification is searching in $N(x_t)$ the good quality solutions for constructing better solutions closer to the optimal one. In this phase there is an intensive use of AM.

AM is structured as a memory containing all the layout analysed with their objective function value associated. The memory is ordered in ascending way from the solution with the best objective function to the worst one.

After having an ordered list of layouts in the AM, it is applied to a process to select a solution x_t in order to search for x_{t+1}. The process is done through a probabilistic function that obviously gives priority to the better solutions but assures that is not chosen only the best one and that there is the possibility to choose one of the worst ones.

As the iterations go on, the AM removes some uncompetitive solutions found in previous iterations. This reduces diversification property but increases intensification property.

One example of probability function in the literature is given by Rochat and Taillard [52] and assigns to the *i-th* worst solution the probability to be choose equal to:

$$\frac{2i}{|L| * (|L| + 1)}$$

With | L | equal to the length of the list stored in the AM. So, the probability of the best solution to be chosen is:

$$\frac{2|L|}{|L| * (|L| + 1)} = \frac{2}{(|L| + 1)}$$

While for the worst solution the probability become:

$$\frac{2}{|L| * (|L| + 1)}$$

Step 4: final intensification

As the algorithm has reached the number of solutions found / iteration made by the designer, it can start the last phase called final intensification. In this phase the best solution present in AM is chosen. To that solution is applied the neighbourhood structure for all the possible combinations (in the case of binary movement, are evaluated all the changes of adjacent facilities). If a new solution is accepted, the final intensification starts again from the new layout obtained. When no better layout are obtained, the TS has reached the end. The last layout obtained is the final solution of the algorithm.

1.3.4.2 Genetic Algorithm

Genetic Algorithm is a metaheuristic algorithm that is inspired by Darwin's theory on natural evolution where the fittest individuals are the ones that are selected for reproduction and to perpetuate the specie. Is used to solve both constraints and unconstrained problems. GAs are applied to a wide range of problems of the real world which have a high level of complexity such as immunology. [53]

In the next pages will follow a detailed description of the phases of a Genetic Algorithm. Customized as follows [20]:

- *3*× *n* matrix to demonstrate each layout as a chromosome. Each population is made of some chromosome
- To produce each population we start from the best solution of the precedent population
- The algorithm will finish when there will be no more improvement in the objective function

In *figure 3* is shown the flowchart of a general Genetic Algorithm.

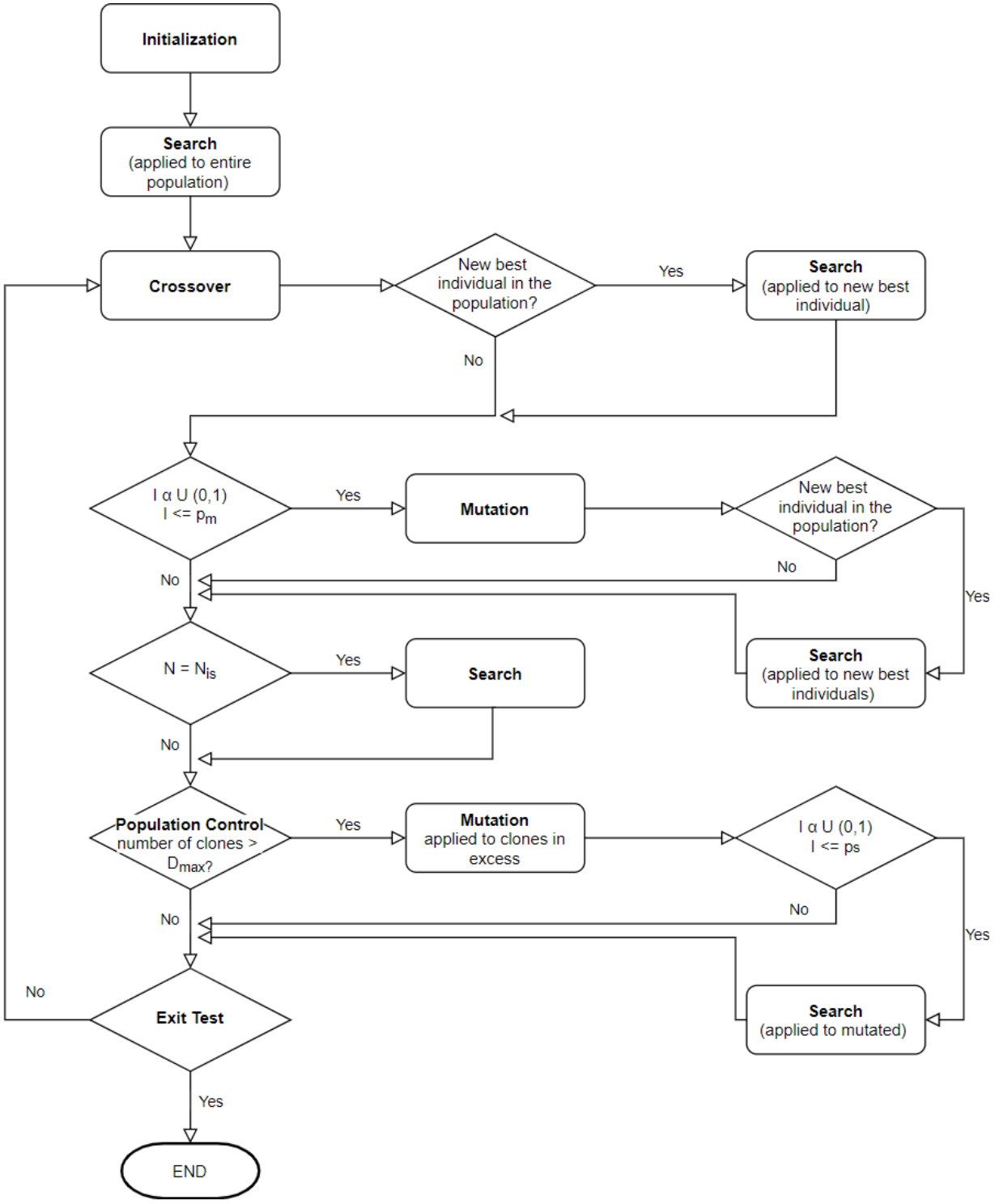

Figure 3. Example of one point crossover

Initial solution

As it happens for TS also for GA it is better to start from an initial promising solution (Garbage In, Garbage Out) for the initialisation of the algorithm. At least is recommended to start with a solution in the feasible area and later improve the solutions iteratively as explained later. Starting from a feasible solution increases the possibility to find better solutions in the consequent iterations.

As it can result obvious, one solution that, with high probability, can be considered a good starting point, is the one in which departments with higher flows are placed close together. To originate this initial solution in which two departments with highest flow between them are put close together is for example possible to apply two methods inside the slicing three method [20]:

1 – Put the two departments at the end of the department sequence and consequently put the number for slicing them at the end of the slicing sequence; doing so they will be close together in the final layout.

2 – Oppositely, it is possible to put the two departments at the beginning of the department sequence and invert their order in the slicing sequence. Doing so in the final layout the two departments will be allocated close together.

The number of possible solutions after creating an initial solution is drastically reduced.

Crossover

Crossover is an operator that is used to create a new population. The creation of a new population is a requisite of each iteration of GA. Crossover chooses two chromosomes (layout) and, as happens in a real-world genetic process, with transmission of characteristics from both the chromosomes, two offspring are created. The sons have some characteristics from the first chromosome and others from the second chromosome.

There are two main types of crossover: one point and two points crossovers. In a one point crossover, a point is selected randomly and the genes before the point will be transmitted to the son in an association parent 1 to child 1 vice versa. The genes after the point are instead transmitted to the son in an association parent 1 to child 2 and vice versa. There is then the need to correct the genes: is unlikely that in the genes before the chosen point present in parent 1 (or 2) there are any genes after the chosen point present in parent 2 (or 1). To avoid this kind of error, the repeated genes are substituted. The substitution is made by taking the actual sequence of genes, removing the doubled

genes and then inserting at the beginning of the sequence the missing genes. An example is provided in *figure 4*.

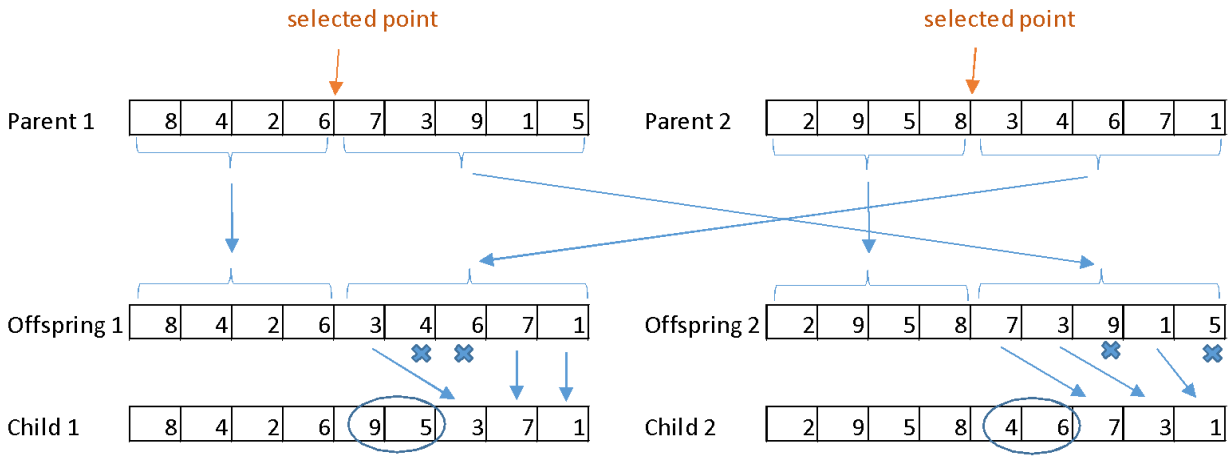

Figure 4. Example of one point crossover

Note: the repetition in offspring 1 are the missing genes in offspring 2 and vice versa.

Two points crossover is similar to one point crossover. In this case there are selected 2 points and, as happened before, genes inside the two points are transmitted from parent 1 to offspring 1 and the same for parent 2 and offspring 2. The genes outside the points are transmitted from parent 1 to offspring 2 and vice versa. After that the correction phase should be applied, as it happens in one point crossover. A figure showing the procedures will be redundant.

Mutation

Mutation is an operator that works on one parent solution in order to generate an offspring by randomly modifying some genes in the parent's solution. The scope of a mutation is to create a new population and escape from the local optima.

The mutation works by choosing two numbers N_1 and N_2, these two numbers will be the positions of the genes to be exchanged inside the parent solution. At the end, it originated a solution called "offspring made by mutation operators".

Migration

Migration is a step which consents to reduce the number of layouts considered. In each iteration a defined number of populations are generated, and the GA runs in each population separately. In order to maintain a fixed amount of populations, the migration operator chooses the best chromosomes

within each population and these chromosomes will migrate to the next population. The other chromosomes will be cancelled. This mechanism avoids the exponential growth of chromosomes to be analysed and also ensures continuous improvement.

Dynamic calculation of the number of operators

In order to choose in every iteration the number of operators to use in the next iterations, some analysis should be made. Every iteration has many operators as a number obtained with a conversion rate applied to the average percentage improvement of the objective function of the solution in the previous *k* iterations.

When the percentage of improvement is equal to 0 it means that the algorithm is stuck into local optima. In order to go on, the GA should be improved in the number of mutation and/or migration operators. This guarantees more randomness in the algorithm and that increases the probability to exit from local optima.

Instead, when the percentage of improvement is pretty high, less mutation and/or migration operators are used. That is due to the fact that a continuous improvement means that the solutions found are always closer to the local optimum and an excess in randomness could reduce the possibility to find that local optimum.

Tuning GA parameters

The tuning phase is characteristic of metaheuristic algorithms. In tuning phase is chosen which parameters used in the algorithm should be fixed parameters and decide which other parameters should be variable inside a predetermined range of values. For example, the number of operators is not fixed but variable inside certain boundaries as explained above. The dynamicity of certain type of parameters can be used to recreate stochasticity or other behaviours not predictable a priori. One fixed parameter in GA is for example the initial population size; obviously it could not be a random number as it is the starting point of the algorithm.

Objective Function

The objective function of every DFLP problem is based on materials handling costs and rearrangement costs. In addition to the standard objective function formulation, in this problem there is the need to give a penalty to the solutions that are not feasible. The penalty objective function was introduced by Komarudin [30] and can be described as: $p_{inf}(V_{feas} - V_{all})$. The resulting objective function is as follow:

$$\sum_{j=1}^{n}\sum_{i=1,i\neq j}^{n} f_{ij}d_{ij} + p_{inf}(V_{feas} - V_{all}) + \sum_{i=1}^{n} Re_i \times ReCost_i$$

With:

i , *j* departments

n number of departments

p_{inf} number of infeasible departments

V_{feas} best amount calculated for a layout in which all departments are feasible

V_{all} best amount calculated for a layout regardless of its feasibility

Re_i is 1 if department *i* is rearranged, 0 otherwise

$ReCost_i$ is the cost of rearranging department *i*

1.3.4.3 Simulated Anealling

If we refer to Anealling in nature we are referring to a phenomenon of solids in which a solid that is heated and then cooled reaches a nearly globally minimum energy state. This physical phenomenon has inspired the Simulated Anealling (SA) algorithms. The algorithm simulates a small random displacement of an atom that results in a change in energy ΔE.

If ΔE < 0 the energy state of the new configuration is lower and so the new-found solution is accepted. If ΔE > 0 the energy state of the new configuration is higher and so the new-found solution is tendentially rejected but there is a probability that it is still accepted and that probability is based on the Boltzmann probability factor:

$$P = e^{\left(\frac{-\Delta E}{k_b T}\right)}$$

Where:

k_b is the Boltzmann constant,

T is the current temperature

The probability to accept a new worst layout (ΔE > 0) decreases as the temperature decreases; as it happens in nature where the probability of an atom to have a displacement decreases as the temperature decreases.

To simulate the solid anealling, the algorithm starts with a high temperature and in every iteration the temperature is decreased until, arrived at a certain temperature (and so after a certain amount of iterations) the algorithm is finished and the quasi optimal solution is found.

In the next pages we will go deep into a Simulated Anealling algorithm to solve Grid System Unequal Area FLP (GSUA-FLP) proposed by Nima Moradi [54]. The proposed algorithm has been chosen due to its ease. To simplify we will not apply it to Dynamic or Stochastic problems but to a general Facility Layout Problem.

GSUA-FLP with SA

Unequal area problems have been already explained: the unequal area means that departments have their own area specification, tendentially are considered rectangular shaped or composed by sum of rectangles with characteristic Width and Length. Equal area means that every department has the same width and length. Equal area is a subset of unequal area problems.

Grid System approach is the opposite of continuous approach. In the latter, each department can be located in each position on the site floor while it satisfies the constraint of non-overlapping with other departments. Instead, in grid system approach, the site floor is divided into unitary square units that together create a grid in which departments will be located. For example, the grid system can be composed of squares 1m×1m and the width and length of each department approximated per defect or excess to integer numbers.

In *figure 5* are represented a grid system and a continuous system.

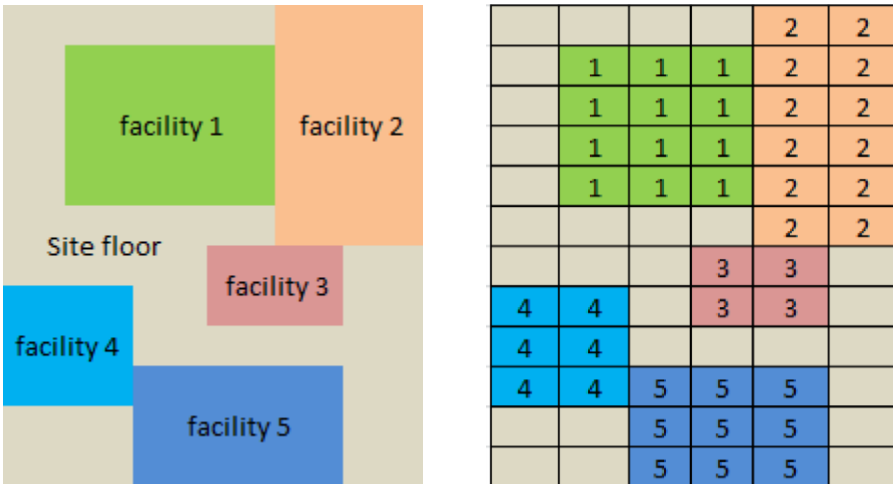

Figure 5. On the left: example of Continuous System. On the right: example of Grid System

In order to formulate the problem, some assumptions have to be made:

- Facilities are rectangular shaped
- The grid system is composed by 1×1 square meter
- The rotation of facility is not permitted
- The distance between facilities is calculated with Euclidean distance from the CoG (center of gravity)
- Facilities must be all inserted in the site floor without overlapping
- For simplification we will use a single period horizon
- Facilities can be located freely
- The problem is not multi floor and the single floor is represented by a grid system
- The objective function is a minimization of the total material handling costs.

The formulation of the problem results to be:

$$Minimize: \sum_{i=1}^{N} \sum_{j=i+1}^{N} f_{ij} d_{ij}$$

Subject to:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} A_{ij} = 0 \qquad \forall\, i \neq j$$

$$u_i + l_i \leq X_U$$

$$u_i \geq X_L$$

$$v_i \leq Y_U - 1$$

$$v_i - b_i \geq Y_L - 1$$

$$u_i, v_i \; integer$$

$$A_{ij} = max[0,1 + min(ui + li - 1, uj + lj - 1) - max(ui, uj)] \times max[0,1 + min(vi, vj) - max(vi - bi + 1, vj - bj + 1)]$$

$$d_{ij} = \sqrt{\left(\left(u_i + \frac{l_i}{2}\right) - \left(u_j + \frac{l_j}{2}\right)\right)^2 + \left(\left(v_i - \frac{b_i}{2} + 1\right) - \left(v_j - \frac{b_j}{2} + 1\right)\right)^2}$$

With:

N = number of departments

d_{ij} = Euclidean distance between department *i* and *j*

A_{ij} = common area between departments (overlapping)

f_{ij} = flow between departments

X_U , X_L = upper and lower limit of length of the site floor

Y_U Y_L = upper and lower limit of width of the site floor

l_i , b_i = length and width of department *i*

u_i , v_i = reference point of department *i* in the grid system

Note: the last two equations indicate how to calculate common area (A_{ij}. In the first equation is in fact described the non-overlapping of facilities) and the distances between departments (d_{ij}).

Note: u_i and v_i are the reference point of department *i* in the grid. u_i can be seen as the coordinate on the x-axis, while v_i can be seen as the coordinate on the y-axis. In *figure 6* are represented two departments (dept. 1 and dept. 2) which have coordinates u_1 , v_1 = (0,7) and u_2 , v_2 = (3,2)

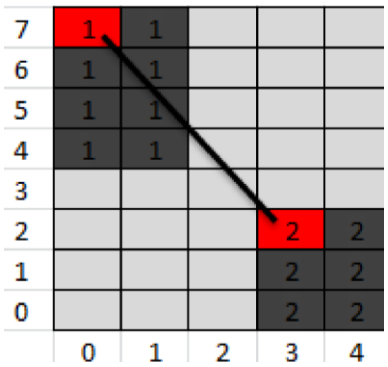

Figure 6. Example of reference points ui , vi in the Grid System

Algorithm formulation of SA to solve GSUA-FLP

Recap: SA needs an initial solution to start its iterations. The initial solution can be found randomly or greedy. After the initialization of a solution, neighbourhood solutions are generated. If the new solution is better it is accepted, if it is worse it is accepted with a probability factor given by Boltzmann. The probability to choose a worse solution reduces the probability to be stuck into a local optimum. At every iteration a new solution is generated, the best solution is registered in the memory, the temperature is cooled. When the temperature reaches a certain level the algorithm finishes. The parameters needed to start the algorithm are described in *table 3*.

Parameter	Notation
Initial Temperature	T_{max}
Final Temperature	T_{min}
Cooling Rate	α
Maximum iteration at each temperature	max_iteration

Table 3. Parameters for SA

The next two algorithms are the SA algorithm proposed by Nima Moradi [54]. The first algorithm is the SA algorithm, while the second one is the solution generation phase and the local search operator. The local search operator is designed to change the location of two random facilities at each iteration. The algorithm continues to exchange random locations until the exchange is feasible or has reached the maximum number of iterations. Algorithm 2 is able to explore the whole space.

ALGORITHM 1 (Simulated Anealling)

1. $s = s_0$ // initial solution with ALGORITHM 2//
2. set T_{max}
3. set T_{min}
4. set *max_iteration*
5. set α
6. $T = T_{max}$ // set the initial temperature as the maximum temperature//
7. **While** ($T \geq T_{min}$):
8. { $n = 1$
9. **While** ($n \leq max\_iteration$): //at a fixed temperature//
10. { apply ALGORITHM 2 //generation of neighbor solution (s') or local search//
11. ΔE = f(s') – f(s) //calculation of the objective function for s and s'//
12. **If** $\Delta E \leq 0$:
13. { $s = s'$ //the new solution is accepted as the best one// }
14. **Else**:

{ accept $s = s'$ with probability $P = e^{\left(\frac{-\Delta E}{k_b T}\right)}$ }

15. $n = n + 1$

16. }
17. $T = \alpha \times T$ //cooled the temperature with the coefficient of cooling α //
18. }
19. Display (*s*) //print the best solution//

ALGORITHM 2 (Initial solution generation algorithm and local search)

1. $r = 0$ //index of infeasibility//
2. **While** ($r = 0$):
3. { **For** ($i = 1$ to $i = n$):
4. { $u_i = \text{rand}(X_l , X_u - l_i)$
5. $v_i = \text{rand}(Y_l - l + b_i, Y_u - 1)$ //random allocation of department i in the grid. Go back to *figure 6* to see again u_i and v_i//
6. }
7. **For** ($i = 1$ to $i = n$):
8. { **For** ($j = 1$ to $j = n$):
9. { $S[n,n] = 0$ //is a matrix n*n which represent the allocation of facility to the site floor. Is now set to 0 because for the moment no facilities have been allocated to the site floor//
10. } }
11. **For** ($i = 1$ to $i = n$):
12. { **For** ($j = 1$ to $j = l_i$):
13. { **For** ($z = 1$ to $z = b_i$):
14. { $S[u_i + j, v_i + z] = i$ //facilities are located with their # at the site floor
15. } } }
16. g = 0 //g will count the number of 0 in matrix $S[n,n]$
17. **For** ($i = 1$ to $i = n$):
18. { **For** ($j = 1$ to $j = n$):
19. { **If** ($S[n,n] = 0$):
20. { g ++ }
21. **Else**: continue
22. } }
23. **If** ($g = A_s - \Sigma A_i$):

24. { r++ } //this condition calculates the non-overlapping of departments and so the feasibility of the solution//
25. **Else**: continue // If the solution is not feasible, another one is searched//
26. }

1.4 SDFLP AND FUTURE OF FLP

1.4.1 SDFLP – Stochastic and Dynamic Facility Layout Problem

In 1992 Palekar et al. [55] solved the SDFLP: they found a way to solve FLP with both stochasticity and dynamicity properties. All the other problems can be seen as a particular case of this SDFLP.

SDFLP is a quadratic integer program. The stochasticity of demand inside a period is given by a coefficient simulating the probability of that demand to happen and so every scenario is grouped as optimistic, most likely, or pessimistic with a determined occurrence for these outcomes. After having estimated the inter-departmental flow matrix in each period, conditional changes from *t* to *t+1* are modelled as a Markov Chain. Depending on the number of departments can be used the DP approach or heuristic/metaheuristic methods.

1.4.2 Future of This Topic

The tendency is to always search for the best algorithm theoretically able to find not a sub-optimal solution but explicitly the optimal one. The idea is to extend the property of exactness of the Exact Methods to the Metaheuristic and Hybrid methods that are able to solve bigger-size problems and are not limited to 9 facilities. Metaheuristic and Hybrid methods can also overcome the dichotomy between dynamicity and stochasticity of a problem and solve both.
To give an example of the new kind of papers on the subject, is possible to cite the latest paper released at the moment of the drafting up (autumn 2020): A new hybrid approach based on discrete differential evolution algorithm. [56]

The branch of industrial engineering analysing ways to design plant layout is still under study and the participation to the advancement of research is brought on by a variety of Universities and researchers and that will make the quality of algorithms rising always to a higher level.

While DFLP and SFLP are focused on *greenfield design*, and so designing a new facility without constraints deriving from the existing facility, there is another branch of FLP that is Facility Re-Layout Problem (FRLP).

FRLP is a subset of DFLP because it also takes into consideration the rearrangement costs; in addition to a standard DFLP has some physical constraints caused by the factory area that is not always a rectangular one and can have unavailable spaces such as columns or forklift's warehouse aisles and so on.

In the real world, FRLP is more common than DFLP or SFLP(in fact more companies make a relayout instead of a layout). The focus of researchers has started to follow the requirements of the market and so it is easier to find papers about FRLP instead of DFLP or SFLP in the last years.

In this thesis will be shown an application of CRAFT to a FRLP.

2. DISCRETE EVENT SYSTEM, PETRI NET & DISCRETE EVENT SIMULATION

In this chapter will be discussed three different but strictly correlated topics:

- Discrete Event System
- Petri Net
- Discrete Event Simulation

The aim of this introduction is to briefly explain these topics, explain their relations and the goals of doing a system analysis.

Discrete Event Systems are a combination of components that act together in order to perform a function that will not be possible individually. These systems are not defined on a continuous interval of time but only when a state of the system changes and so only when a transition happens.

Petri Nets is a mathematical model used to rigorously describe high-complexity discrete event systems. Petri Nets can be also represented with a graphical model that can be easily comprehended by any user. Petri Nets make also possible the study of reachability of states, and can study the system also through linear-algebraic techniques.

Discrete Event Simulation is a tool that has been developed to study high-complexity discrete event systems; in fact, analytical solutions of a DES are difficult to be extracted. Simulation is a process through which a system is first recreated through a model and then analysed numerically in order to find useful data to estimate the quantities of interest.

The combination of these three topics allow the study of the most important and most analysed systems. Any manufacturing company or service company can be modelled with a discrete event system and so can be represented through a Petri Net model and a Petri Net graph, and later on can be studied through the simulation tools.

The goals of studying a real-world system with system theory (and so exploiting Petri Net model and Simulation tools) are 5:

-Modelling and Analysis: understand how the system works

-Design and Synthesis: understand which parameters are relevant and which are not for the system

-Control: select the inputs in order to obtain the predetermined result

-Performance Evaluation: understand the actual performances of the system

-Optimization: study the system in order to determine the optimal system behaviour

2.1 DISCRETE EVENT SYSTEM

Qualitatively speaking, a system is “a combination of components that act together to perform a function not possible with any of the individual parts” - IEEE Standard Dictionary of Electrical and Electronic Terms.

Important to highlight that a system is composed of interacting components and also that the system performs a function and so is not a constraint to be a physical object. For example, a system could represent an economic mechanism, species behaviour, social dynamics, and so on.

In order to develop techniques to design, analyse and control the performances of a system, there is the need to structure a mathematical model.

The first thing to model are the input variables. These are variables controllable and modifiable and represent some real measure such as temperature of a process, quantity of materials and so on. These variables can be in function of the time. Input variables are expressed as:

$$\{u_1(t), \dots, u_p(t)\} \qquad t_0 \leq t \leq tf$$

Then we assume that we can measure the output of the system we are modelling while varying the input variables and so we define the output variables as:

$$\{y_1(t), \dots, y_m(t)\} \qquad t_0 \leq t \leq tf$$

The relation between the input variables and the output variables is determined by what happens inside the system. We define this mathematical relationship between input and output with the function g():

$$y_1(t) = g_1\big(u_1(t), \dots, u_p(t)\big) \;\; , \;\; \dots \;\; , \;\; y_m(t) = g_m\big(u_1(t), \dots, u_p(t)\big)$$

So in the mathematical form, the system model obtained is:

$$y = g(u) = \big[g_1\big(u_1(t), \dots, u_p(t)\big), \dots, g_m\big(u_1(t), \dots, u_p(t)\big)\big]^T$$

The whole system theory not useful to comprehend what a DES is, will be skipped.

In *figure 7* there is a representation of a generic system.

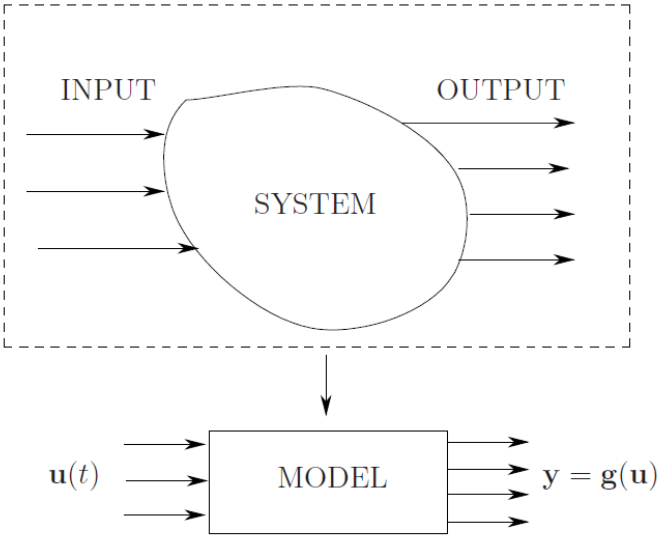

Figure 7. Example of a generic System

The state space X of a system is the set of all possible values that the state may take. When the state space of a system is described by a set of natural numbers, and the system is observed only at discrete points in time, where for events we are implying the state transitions occurring at every discrete interval of time.

For "event" we intend a specific action taken or as a spontaneous occurrence determined by nature law (physical, chemical, …). An event is described with the letter *e* and the set of events with the letter *E*.

2.1.1 Differences Between Systems

In this section will be explained some differences between systems.

- Static vs. Dynamic Systems: we refer to a static system when the output does not depend on the pasta values of input. We refer to a dynamic system when the output depends also on the value of the past inputs.
- Time-Varying vs. Time-Invariant Systems: if the behaviour of a system does not change with time, the system is a time-invariant system. If the behaviour changes, we are dealing with a time-varying system. The property to be a time-invariant system implies a property called stationarity: thanks to this property if we use the same input for a system in two different moments in time, we should expect the same output.
- Linear vs. Nonlinear Systems: if a system satisfies the condition $g(a_1u_1 + a_2u_2) = a_1g(u_1) + a_2g(u_2)$ is a linear system; in the other cases is a nonlinear one.

- Continuous-State vs. Discrete-State Systems: if the state variables can be only elements of discrete set is a discrete-state system, while if the variable can range in $\mathbb{R}$ or $\mathbb{C}$ is a continuous-state system.
- Time-Driven vs. Event-Driven Systems: if the state changes with time we are dealing with a time-driven system. If the state changes only when an event forces an instantaneous state transition, we are dealing with an event-driven system.
- Deterministic vs. Stochastic Systems: if there are one or more variables that are not fixed but depend on a certain probability distribution, the system is said to be stochastic. In order to characterize the system behaviour, a probabilistic framework is required.
- Discrete-Time vs. Continuous-Time Systems: in a continuous-time system all the data are defined for all possible values of time, while in a discrete-time system data are defined only at discrete points in time.

In *figure 8* are briefly summarized the major system classification.

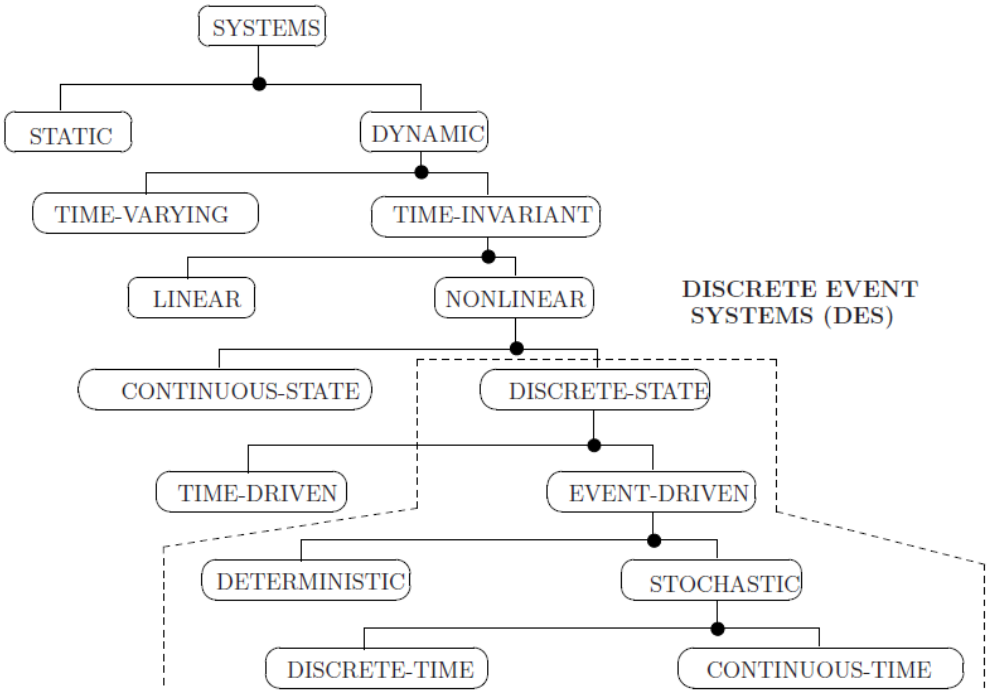

Figure 8. Major System Classification

2.1.2 Discrete-State Systems

In discrete systems the state changes as time changes. Events happen with every "clock tick", while continuous events happen with the time changing. The discrete time unit *k* appears as an argument of inputs, states, and outputs. The transition happens instantaneously with a clock tick. The clock time is not the only way an event may happen.

Assume that there is a clock that measure the time in the system; there are then two possibilities:

1. At every clock tick an event *e* belonging to set *E* occurs. If nothing happens, we can still say that a null event has happened with the clock tick and the state of the system has not changed.
2. At various times instant not necessarily coinciding with a clock tick, some event *e* occurs.

While in the first case the transition is synchronized with the clock tick, in the second case the transition is asynchronous. In the first case the cause of the transition is the clock tick, while in the second case is an external cause not included in the schematization of the system.

The first case is called *time-driven*, while the second case is called *event-driven*. This distinction will be useful while simulating.

Discrete Event System (DES) is a discrete-state, event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time.

2.1.3 Queueing Systems

In this section will be shown some concepts that will be at the base of the simulation, especially of a manufacturing system.

For queueing we refer to a common behaviour of the systems that are usually designed and build, that affirms that in order to use certain resources there is the necessity to wait. A queue system is good to represent service companies such as a postal service or a bank service, but also manufacturing companies that use storages between machineries working with different paces or finished products waiting in the warehouse for the truck that brings them to the final customer. Example which can be carried are infinite; there are always three basic elements that are used to represent a queueing system:

- *Resources:* are the elements which provide the service. Due to the fact that a resource gives a service to the object processed, sometimes it will be better to refer to resources as *servers*.
- *Entities*: are the object waiting in their quest to use a resource. Entities are processed by the servers.

- *Queue*: is the logical space in which the entities wait for the service to be provided. A queue is a set of entities waiting.

2.2 PETRI NETS

Petri Nets is a widely used model developed by C. A. Petri in the early 1960s. Petri Nets are used to explicitly represent the transition function of a Discrete Event System, in fact, following some specific rules, Petri Nets are able to manipulate events. In order to enable and event, this model exploits explicit conditions. This gives the possibility to also represent complex Discrete Event Systems with complex control schemes. To be better understood, a Petri Net is not only described with the mathematical model, but it can also be represented by a graphical model that is called Petri Net Graph. This graphical model is intuitive and can easily show a lot of structural information about the Discrete Event System.

The reason for the success of Petri Nets are the techniques developed to study the system through the model. These techniques comprehend reachability analysis and linear-algebraic techniques, and both can be applied to both timed and untimed Petri Nets.

In order to create a Petri Net there are two main steps:

1. The Petri Net Graph (named also Petri Net Structure) should be created
2. In order to create the complete Petri Net model with its dynamics, should be added to the Petri Net Graph:
    a. The initial state
    b. The set of marked states
    c. The transition labelling function

In the following pages will be explained the mathematical and the graphical model of the Petri Nets.

2.2.1 Notation and Definitions Of Petri Nets

In Petri Nets, events are represented by *transitions*. After determined conditions happen, the transition can occur. The information about these conditions are contained in *places*. Places and transition are the only two types of nodes. Places and transitions are connected by *arcs*. Petri Net is a bipartite graph: that means that arcs do not connect place with place or transition with transition but can only connect place and transition or transition and place. Arcs are oriented. The orientation of the arc

determines the relation between the place and the transition. Doing so, if the place is the input of the transition it is used as a condition required for the transition to occur. If the transition is the input of the place, the place is later associated with the conditions generated by the realization of the transition.

Petri Net Graph is a bipartite weighted graph constituted by:

$$(P, T, A, w)$$

Where

$P = \{p_1, p_2, p_3, \ldots, p_n\}$ is the set of Places

$T = \{t_1, t_2, t_3, \ldots, t_m\}$ is the set of Transitions

$A \subseteq (P \times T) \cup (T \times P)$ is the set of Arcs, places to transition and transition to places

w: A → {1,2,3, ...} is the weight function on the arcs

Assumption: there are not unconnected places or transition inside (*P,T,A,w*)

|P| = n

|T| = m

An arc is described with the form (p_i , t_j) or (t_j , p_i). The weight of that arc is a positive integer.

In a Petri Net Graph is allowed that multiple arcs connect two nodes: the weight of an arc is used to substitute the number of arcs connecting two nodes. For example a Transition describing a machine that works in batches of 10 units should be linked to a Place with 10 different arcs; that is inconvenient and in order to express the concept of ten pieces worked in batches we use the *weight* and so we use just one arc with weight 10.

Describing a Petri Net Graph is usually helpful to use a notation able to easily list the Input Places and the Output Places for a specific Transition.

$I(t_j) = \{p_i \in P : (p_i, t_j) \in A\}$ set of Input Places to transition t_j

$O(t_j) = \{p_i \in P : (t_j, p_i) \in A\}$ set of Output Places from transition t_j

The same notation can similarly be used to describe $O(p_i)$ and $I(p_i)$.

Graphically speaking, the standardization is to use circles to describe places and bars to describe transitions. In *figure 9* is shown a simple graph containing 2 places and 1 transition.

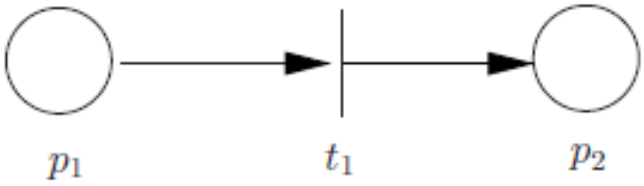

Figure 9. Example of a basic Petri Net Graph

In *figure 9 it is* possible to see two oriented arcs. The first one from p_1 to t_1 means that $p_1 \in I(t_1)$ and the second one from t_1 to p_2 means that $p_2 \in O(t_1)$. Moreover, the weight of the two arcs is 1 because it is assumed unitary when it is not diversely specified. So we can say that

$$w(p_1, t_1) = 1 \qquad \text{and} \qquad w(t_1, p_2) = 1$$

In order to make a complete description of the petri net we should also include that the weight of arcs not belonging to A are equal to 0:

$$w(pi, tj) = 0 \text{ when } pi \notin I(tj) \qquad \text{and} \qquad w(tj, pi) = 0 \text{ when } pi \notin O(tj)$$

In the case of *figure 9* we should say:

$$w(p_2, t_1) = 0 \qquad \text{and} \qquad w(t_1, p_1) = 0$$

This notation does not give additional information so we will not use it anymore.

The complete mathematical representation of the Petri Net Graph shown in *figure 9* is the following:

$P = \{p_1, p_2\}$
$T = \{t_1\}$
$A = \{(p_1, t_1), (t_1, p_2)\}$
$w(p_1, t_1) = 1 \qquad \text{and} \qquad w(t_1, p_2) = 1$

Note: it can happen that a transition has no inputs. In that case it means that it is a transition that occurs anyways. It can also happen that a transition has more places as inputs. In that case it means that the transition must wait for all the conditions of that place being verified before being able to occur.

Going back to the concept that Petri Nets are able to represent a Discrete Event System, we are missing how the conditions under which the events can occur are represented in Petri Nets.

The mechanism able to provide the information about the realization of the conditions is provided by assigning *tokens* to places. A token is used to highlight that the condition described by that place is verified. How the tokens are assigned to a Petri Net Graph defines a *marking*. Graphically a token is represented by a black dot positioned in correspondence with the Place in which it is assigned.

A marking x is a function x: P → $\mathbb{N} = \{0,1,2, ...\}$. So, x defines row vector $\mathbf{x} = [x(p_1), x(p_2), ..., x(p_n)]$ where n is the number of places in the Petri Net.

A Marked Petri Net is a five-tuple (P,T,A,w,x) composed by a normal Petri Net (P,T,A,w) and a marking (x). From now on, when we refer to a Petri Nets we are implying a Marked Petri Nets.

In *figure 10* is shown an example of two possible markings (x_1 and x_2) made on the Petri Net Graph of *figure 9*.

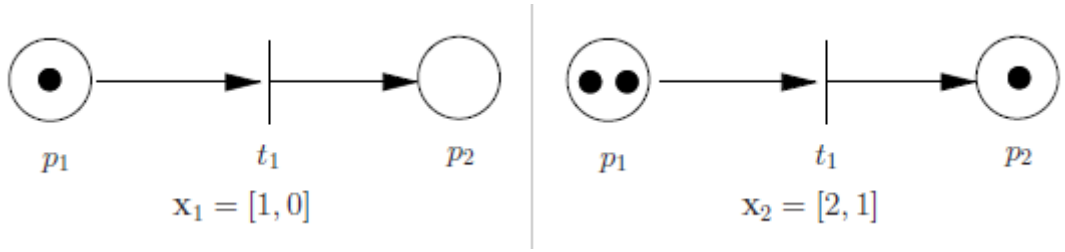

Figure 10. Example of a Marked Petri Net Graph

In a marking **x**, the *i-th* position ($x(p_i)$) represent the number of tokens assigned to the place i with $x(p_i) \in \mathbb{N}$. The number of tokens assigned to a place is potentially infinite, so the state space X of a Petri Net with n places is potentially $X = \mathbb{N}^n$.

At this point is still missing the concept of dynamicity of the system (remind: DES is an abbreviation of Discrete Event Dynamic Systems (DEDS)). In order to go on, is fundamental to introduce the concept of *enabled transition*.

A transition $t_j \in T$ is said to be *enabled* if:

$$x(p_i) \geq w(p_i, t_j) \text{ for all } p_i \in I(t_j)$$

Basically, in order for the transition t_j to happen, every p_i that is an Input Place of t_j, should have a number of tokens greater or equal than the weight of the arc (number of arcs) connecting p_i *to* t_j.

In *figure 11* is shown an enabled transition

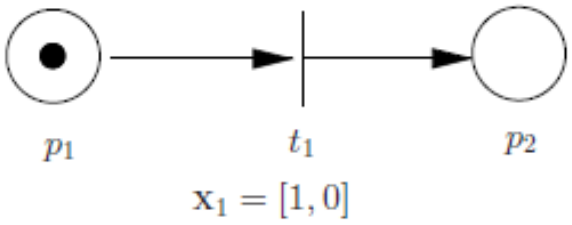

Figure 11. Enabled transition

Due to the fact that $x(p_1) = 1$ and $w(p_1, t_1) = 1$, the condition to be enabled is satisfied, in fact:

$$x(p_1) \geq w(p_1, t_1)$$

Having now defined the enabled transition is possible to continue and introduce the Petri Net Dynamics.

2.2.2 Petri Net Dynamics and State Equations

In Petri Net the mechanism to make a state transition happen is provided by moving tokens through the net. When tokens move, they change the state of the Petri Net. If a transition is enabled is said that can *fire* (it means that can occur). The state *transition function f* is the state changing through the firing of an $\mathbb{N}^n$ enabled transition

$f : \mathbb{N}^n \times T \rightarrow \mathbb{N}^n$ in (P,T,A,w,x) is defined for transition $t_j \in T$ iif:

$$x(pi) \geq w(p_i, t_j) \qquad \text{for all} \qquad p_i \in I(t_j) \tag{1}$$

If $f(\mathbf{x}, t_j)$ is defined, $\rightarrow \mathbf{x}' = f(\mathbf{x}, t_j)$ where:

$$x'(p_i) = x(p_i) - w(p_i, t_j) + w(t_j, p_i) \quad i = 1, \ldots, n \tag{2}$$

The first condition (1) is the condition of a standard enabled transition, so we define f only in the space of feasible events. The second equation (2) regulates the way the transition happens: a place p_i will lose as many tokens as the weight of the arc connecting p_i to t_j.

In *figure 12* is shown an example of a transition:

Figure 12. Petri Net Transition

In *figure 12*, on the left there is the initial state with an enabled transition (see description of *figure 11*), while on the right side there is the same graph after the firing of the enabled transition.

It is possible that p_i is both input and output of transition places. In that case p_i loses as many tokens as the output arcs but receives as many tokens as the input arcs.

Notes:

-The overall number of tokens is usually not constant. If there is a transition without inputs arcs, and so happens anyways without conditions of places, that node generates tokens and so it makes possible the number of overall tokens to grow to potentially $+\infty$. In the other case, if a transition does not have output arcs, it will absorb tokens every time unit and so is also possible the case that after a certain amount of period the resulting state is $\mathbf{x} = [0, \dots, 0]$.

-In the Petri Net is not specified the sequence in which the transitions fire. If there are more enabled events that can fire in the same moment, we must examine each possible sequence of transitions.

-Not all the states are reachable: the set of *reachable states R* is defined as:

$$R[(P, T, A, w, x)] := \{y \in \mathbb{N}^n : \exists s \in T^* (f(x, s) = y)\}$$

Where T^* is the extended domain of T and $s \in T^*$.

The state equation of a Petri Net is described starting from the equation of the transition function:

$$x'(p_i) = x(p_i) - w(p_i, t_j) + w(t_j, p_i) \quad i = 1, \dots, n$$

It has already implicitly said that the next state $\mathbf{x'} = [x'(p_1), x'(p_2), \dots, x'(p_n)]$ is given starting from the current one $\mathbf{x} = [x(p_1), x(p_2), \dots, x(p_n)]$ after a firing of a particular transition t_j.

Let us define the *firing vector* **u**: an *m*-dimensional vector of the form:

$$\mathbf{u} = [0, \dots, 0, 1, 0, \dots, 0]$$

where the only 1 is in the *j-th* position in correspondence with the *j-th* transition that is the one that is firing.

Let us also define the *incidence matrix* **A**: an *m*×*n* matrices whose (*j, i*) are given by the difference between the weight of the input arcs and the output arcs.

$$a_{ji} = w(t_j, p_i) - w(p_i, t_j)$$

The mathematical formulation of **x'** is now

$$\mathbf{x'} = \mathbf{x} + \mathbf{uA}$$

And so the transition function can be written as:

$$f(\mathbf{x}, t_j) = \mathbf{x} + \mathbf{uA}$$

Taking as reference the transition shown in *figure 12*:

$$x_1 = [1 \quad 0]$$

$$u = [1]$$

$$A = [-1 \quad 1]$$

$$x_1' = [1 \quad 0] + [1][-1 \quad 1] = [0 \quad 1]$$

To simply describe the matrix A: in the graph represented in *figure 12* we have 2 places and 1 transition, the resulting matrix is a 1×2 matrix. In position (1,1) is described the number of arcs from transition 1 to place 1 minus the number of arcs from place 1 to transition 1. In our case the number of arcs from transition 1 to place 1 is 0, while the number of arcs from place 1 to transition 1 is 1, so the resulting a(1,1) = 0 - 1 = -1.

The vector **u** is instead mono-dimensional due to the fact that we have just one transition and is equal to one because the transition is firing.

Vector $\mathbf{x_1}$ has already been explained contextual with *figure 10*.

2.3 DISCRETE EVENT SIMULATION

The first goal in the study of a dynamic system is to obtain a model. A model is composed of mathematical equations which are used to describe the components of the system and their interaction and behaviours. The study of that mathematical equations in a discrete event system is considered unfeasible.

Due to the fact that analytical solutions for Discrete Event Systems are really difficult to be extracted, a useful tool to overcome this problem revealed to be the simulation. Simulation is a process through which a system is first recreated through a model and then analysed numerically in order to find useful data to estimate the quantities of interest said above.

In the next sections the objective is to present the main components of discrete-event computer simulation and the way it can function and will also be introduced some software for model simulation.

It is important to remark that the scope of a simulation is the output analysis. For output analysis we intend the measuring of relevant data. It should be also considered that it is dangerous to use simulation tools to obtain data and use that data as valid: it is important to assign to data their statistical validity.

2.3.1 Event Scheduling Scheme

Is possible to see the simulation as a systematic way to generate sample paths of a stochastic Discrete Event System. Let's recall some concepts:

- E is an event set, e is a single event
- X is a countable state space
- $x_0 \in X$ at time $t = 0$ is a given state
- $\Gamma(x) \subseteq E$ for every state x is a set of feasible events
- $i \in \Gamma(x)$, with i a feasible event, is associated to a clock value y_i which represent the amount of time require for i to occur
- v_i is the event lifetime
- Stochasticity is supplied by the computer's random number generator

In t = 0, the random number regenerating mechanism supplies lifetimes and so it set $y_i = v_i$ for all $i \in \Gamma(x_0)$. The triggering event e' is the event with the smallest y_i and so it is the first to occur:

$$e' = arg\ min_{i \in \Gamma(x)}\{y_i\}$$

Now that the first event has occurred, it is possible to update the state of the system. The new state is x'. The actual value of a stochastic DES at the new state x' is given by the computer's random number generator (RNG).

The time spent at state x before passing to state x' is the interevent time and is defined by:

$$y^* = min_{i \in \Gamma(x)}\{y_i\}$$

The time is now updateable to *t'*:

$$t' = t + y^*$$

Is now also possible to update the clock values for all the feasible events in the state *x'*. There are two possible cases: the first case is when an event that was feasible in *x* remains feasible in *x'*. The clock value is simply updated as:

$$y_i' = y_i - y^*$$

The second scenario is when an event was not feasible in *x* but became feasible in *x'*. For these events a new lifetime supplied again with RNG is needed.

The event scheduling scheme is the procedure above with one modification: instead of maintaining the clock values y_i we want to maintain a Scheduled Event List (SEL). In order to do that, when an event is activated at some event time t_n , its next occurrence will happen at time $t_n + v_i$.

The mathematical form of SEL is:

$$L = \{(e_k, t_k)\}, \quad k = 1,2, \dots, m_L$$

With m_L is the number of feasible events at the current state.

SEL is ordered with the smallest scheduled time first; that means that the first event of the list is the triggering event for the transition to a new state. We assume now that the SEL is deterministic and there is not the possibility that an event scheduled not as first become, with a determined probability, the trigger event.

2.3.1.1 Event Scheduling Scheme Steps

Before starting the simulation procedure, the initialize function sets the time to 0 and the state to some initial value x_0. Then, the function provides to the RNG the distribution probabilities of event lifetimes. The RNG is now able to provide event lifetimes for all the feasible events at the initial state. Now the SEL is initialized and the simulation procedure can start. The procedure consists in a continuous loop of the following 6 steps:

1. Remove the first event of the SEL
2. Update the simulation time
3. Update the state according to the event just removed
4. Delete from SEL the events that are unfeasible for the current state

5. Add to SEL the events that were not in the list due to infeasibility and now have become feasible
6. Reorder SEL based on the smallest-scheduled-time-first scheme

The components and variables involved in the process of simulation are the following:

- State: a list of all the state variable
- Time: a variable that keep the information about the simulation time
- Scheduled Event List: a list where all scheduled events are stored with their occurrence time
- Data Registers: places in which date are stored for estimation purposes
- Initialization Routine: the function which initialized the data structure
- Time Update Routine: the function identifying the next event
- State Update Routine: the function updating the state based on the info of the Time Update Routine
- Random Variate Generation Routine: the function that, thorough computer's RNG, generates event lifetimes
- Report Generation Routine: the function computing quantities of interest based on data gathered
- Main Program: the function coordinating all the other functions. Is responsible for the alignment of initialization routine and all the other routines. It is also responsible for the terminating criteria

2.3.2 Simulation Of A Simple Queueing System

One of the main information about process performances that a simulation is able to gather, concerns queues. We will see which kind of information is possible to gather from an event scheduling scheme, in a single row, single server queueing system. Assume that the simulation starts at t = 0 and with no entities in the queue. The other assumption is that the criteria to terminate the simulation is not the time passed but the number N of entities processed, with N given as input. If we run this kind of simulation, we are able to estimate the subsequent performance of interest. Note that we are referring to a stochastic system.

Expected Average System Time

Give S_k the overall time that an entity k spend in the system, and S_N the mean of the average of all the S_k, then an estimate of S_N is given by:

$$\hat{S}_N = \frac{1}{N}\sum_{k=1}^{N} S_k$$

To compute this indicator we should keep track of the time in the system of every entity processed.

Probability That Customer System Times Exceed a Given Time

This indicator can be useful to simulate systems in which the time in line should not exceed a given deadline. That can be the case of a service company in which a long time in the queue can have negative effects on sales. The given time in the system is written as *D* (deadline). There is no way to assure that the deadline will be respected every time, but we can calculate the probability that given an entity it will not exceed the deadline. Define n_N = number of customers that stay in queue more than *D*, then we can estimate the probability that a customer system time do not exceed the given deadline as:

$$\hat{P}_N^D = \frac{n_N}{N}$$

Server Utilization

This indicator represents the percentage of time in which a server is utilized during the whole simulation. It can also be seen as a probability that a server is busy. Let *T(i)* be the total time during which the queue is long *i* entities, (*i* = 0,1,..) and so the total time of simulation can be seen as:

$$T_N = \sum_{i=0}^{\infty} T(i)$$

Thus the probability of a server to by busy is :

$$\hat{\rho}_N = \frac{\sum_{i=0}^{\infty} T(i)}{T_N}$$

Or given the fact that the server is always busy except when there are no elements in queue, the probability of a server to be busy can also be written as:

$$\hat{\rho}_N = 1 - \frac{T(0)}{T_N}$$

Mean Queue Length

Q_N is the average queue length we want to calculate. $\rho_N(i)$ is the probability that queue length is equal to *i* over the time interval required to serve *N* entities. Then Q_N is:

$$Q_N = \sum_{i=0}^{\infty} i \times \rho_N(i)$$

Before estimating Q_N is required to estimate $\rho_N(i)$ as:

$$\hat{\rho}_N = 1 - \frac{T(i)}{T_N}$$

Since it is valid for every *i*, it became clear that:

$$\hat{Q}_N = \frac{1}{T_N} \sum_{i=0}^{\infty} i \times T(i)$$

In *figure 13* is represented on a XY-graph an example of the queue behaviour in a one-server system during the time. a_i are the arrivals of new entities in queue, while d_i are the departures of the entities from the queue.

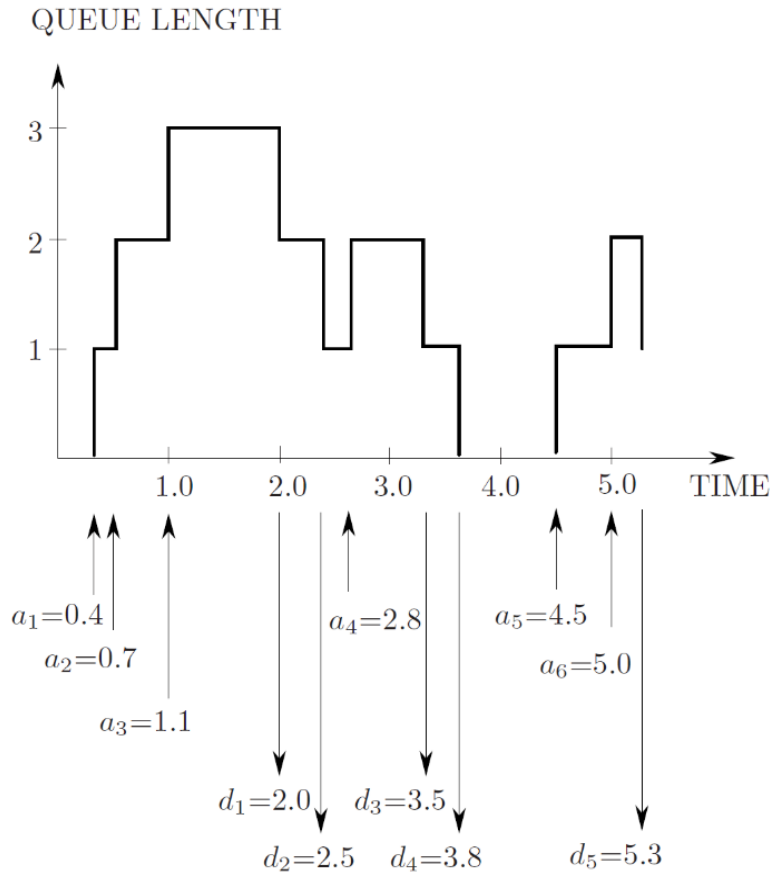

Figure 13. Example of a queue's behaviour in a simulation

2.3.3 Process-Oriented Simulation Scheme

As most of the system can be structured as the concept of the queueing systems (i.e. is possible to describe a manufacturing system with only a queueing system) it can be useful to develop some tricky way to manage them. Such models can be convenient to see entities as undergoing a process as they pass through the DES. This process is a sequence of queues and servers. The process-oriented scheme uses this concept of process to simulate what happens to each type of entity of interest during its routing.

The basic queueing system seen above (1 queue – 1 server) can be seen, for every entity that enter in the system, in the process-oriented scheme as:

1. The entity arrives
2. The entity enters the queue
3. The entity requests the service of the server. The entity remains in the queue until the server is ready to process it
4. Once the server is ready, the entity is processed
5. When the service has been provided, the entity leaves the server
6. The entity leaves the system

With step 1,2,5,6 as instantaneous actions, while step 3,4 with a stochastic time.

Differently from the event scheduling scheme, the process-oriented scheme sees the process as a sequence of functions. These functions can be of two kinds:

- Logic functions: are all the actions that happen instantaneously. These actions are made by the entities that trigger the logic function. Examples of these functions can be the arrival of the entity (in general all the instantaneous action of the steps seen above (step 1,2,5,6)) or action such the recording of the arrival time and so on.
- Time delay functions: are all the actions that do not happen instantaneously. We are referring to the actions of step 3 and 4 and so the waiting time and the processing time. This type of functions can be of two subtypes:
    - Specified time: the delay is fixed. We are referring to the service time; even if it can depend on a prespecified distribution and its value is determined by the random variate generator, it can be considered as fixed.
    - Unspecified time: the delay depends on the state of the system. The time waited in a queue for example does not depend only on the prespecified distribution of the

subsequent server/s but it depends also on the number of entities in the queue that is a quantity that varies during the simulation time.

Even if the queueing systems can be described and solved with the event scheduling scheme, the advantage of solving it through a process-oriented scheme is not only a matter of facility to think of customer/products as entities flowing through a network of interconnected queues and servers, but also a matter of simplicity in computer code needed to simulate such events. This aspect will be treated in the next section.

The main components of a process-oriented simulation scheme are 5. Three of them have already been explained and are: entities, resources, queues. The other two components are the following:

3 Attributes: are info that characterize a particular individual entity of any type. Every entity is attached with the attribute if required. The attributes are used to prioritize or diversify actions on a server. For example an entity can have an attribute that makes it the first element of a queue no matter of its arrival position; or an entity can have an attribute that will increase the time needed to be processed.

4 Process Functions: logic functions or time delay functions. The functions that register the times in which instantaneous and time delayed actions occurred.

2.3.4 Discrete Event Simulation Languages

The simulator for a specific Discrete Event System can be built both as event scheduling scheme or process-oriented scheme. In both cases it is possible to use languages such as C++. For ease, some simulation languages have been developed. The strength of a simulation language is that it is able to deliver building blocks (that carefully take into consideration the standard components such as random variate generation, data collection and processing) tailored to DES models. After having created these building blocks, the modelling of a system does not require programming skills. Many software-houses have developed simulation languages and are selling their simulation software worldwide. Most of the software possess built-in the random variate generator capable of recreating many statistical distributions. For more than two decades, software has been based on object-oriented programming and the users have a graphical interface in which they can easily model their systems.

Some software are also provided for graphical simulation that show for example the entities processed in a server, or the entities waiting in a queue. The simplicity of the simulation can lead the user to fallacious conclusions; it is highly recommended to do more simulation extended on a long time

period in order to be sure that there are not infrequent events that can drastically change the behaviour of the system.

Two software are now considered. Both of them were studied, but just one was used to simulate the business case discussed in the next chapters of this thesis.

SimEvents

SimEvents is embedded in MatLab and operates within Simulink. It allows time-driven and event-driven actions. The pros of SimEvent is the possibility to use Simulink libraries and MatLab programming code.

Arena

Provided by Rockwell Automation, Arena is able to graphically build models and animate the execution of a simulation. It allows simulation both on process-oriented or event scheduling schemes. The structure and the vastity of modules present in Arena make it the best simulation software for factory simulation. The choice on which software using has relapsed on this software.

The only reference used for Chapter 2 (DES, Petri Net, Simulation) is the book named “Introduction to Discrete Event Systems Second Edition” written by Christos G. Cassandras (Boston University) Stéphane Lafortune (The University of Michigan) in 2008. [57]

The book is all-encompassing of the subject. From that book it was chosen only the strictly chapters needed to comprehend what a discrete event system is, why we use Petri net and why it is important also to make a simulation model. The reason why the only reference chosen was this book is that it has a complete view on the subject chosen and also encompasses other correlated arguments such as: supervisory control, timed and hybrid models, stochastic timed automata, Markov chains, sensitivity analysis, and more.

3. FACILITY LAYOUT PROBLEM: A REAL CASE STUDY

3.1 INTRODUCTION

Chapter 1 and chapter 2 have been exploited to summarize the literature review about two main topics: Facility Location Problem and Discrete Event Simulation.

The first topic, FLP, is linked to many sub problems, which differ for some peculiarities but are all focused on optimizing the layout of the facilities inside a production/assembly plant.

Here, in chapter 3, we will take into consideration a real case study in which have been applied some of FLP's theoretical concepts and methods in order to design the new optimal layout.

The chapter will start with a brief overview of the company (the products and the plant's structure); after that will be explained: the data given as input to the methodology adopted, how the methodology works, the analysed layouts with relative pros/cons and the final comments on the chosen solution.

3.2 THE COMPANY

The analysed company is a firm mainly working in two completely different markets: solutions for temperature control and management systems in industrial, professional and domestic fields, and mosquitos' nets and screens. The company was born through the merger of two different companies founded in the early '80. The company has the administrative head office and one production plant in the province of Novara; this production plant is the focus of the analysis.

The actual production activities are carried in two different plants located close to each other:

i. First Plant: the ground floor is dedicated to mosquitos' nets and screens production. The first floor is dedicated to administrative and other offices.
ii. Second Plant: the left side is a 3200 m^2 warehouse. The right side is dedicated to temperature control systems production.

The production facility of temperature control systems, covering an area of 1,800 m^2, uses modern production systems to manufacture high-quality products, and is run using precise order planning and controls in all the manufacturing stages. The area is equipped with high-tech automated machinery for the injection moulding of thermoplastics materials from 45 to 300 tonnes, for the deposition of expanded polyurethane seals, and assembly and special process lines manned by expert technicians. Here are produced: fan filter units, compact fans, cross flow and external rotor fans for ventilation

and air filtering in the industrial automation, electronics, air conditioning, cooling and heating sectors, air-conditioning components, anti-condensation heaters and thermoelectric units, for the temperature control of equipment in industrial and domestic fields, mechanical control systems, thermostats and hygrostats, for regulating temperature and humidity in electrical and electronic equipment, internal lighting systems for electric panels.

The warehouse, with a structure covering 3,200 m^2, organized in a modern manner and supported by an IT system for warehouse process coordination, has increased the productivity of storage and handling activities, dedicating attention to improving the flexibility of logistic flows and goods traceability. The warehouse is mainly composed of narrow aisle warehouse (7-meter-tall racks), and only the most distant part from production is used as floor warehouse for picking for production activities and for packaging activities.

The production facility of mosquitos' nets, covering an area of 1500 m^2, is mostly composed of sequential assembly stations. Here are produced: flyscreens, roller blinds and braking devices.

3.3 PETRI NETS GRAPHS OF THE COMPANY

In this paragraph are shown and commented the Petri Nets Graphs that summarize all the processes occurring in the company: arrival of raw materials, stock of them, picking for production, productive activities, stock of finished products, packaging and departure of finished products.

In order to give a fast idea to the reader on how the company operates without the need to deeply analyse the Petri Nets Graphs, the graphs representing the processes will be briefly explained. At the end of this paragraph there will be a focus on all the employees involved in the material handling that will be crucial in the following paragraphs.

Figure 15: in this figure are represented the graphs showing 4 activities:

- How it is managed the arrival of a truck and the quality check procedure before the handling of materials
- How it is managed the departure of a truck containing finished products and the operators in charge of it
- The packaging process, happening before the trucks' departure
- The picking of finished products and their handling to the packaging area and their consequent delivery

Figure 16 and Figure 17: in these figures are deeply shown the processes concerning the two categories of products produced in the screens and mosquitos' nets area. The products' categories are called H- (*figure 16*) and NH- (*figure 17*). The two products share just the first activities and operators; after the first phases the two products go to different assembly stations managed by different operators. The two categories of product are linked because they have the same main raw material (nets) and the whole material handling is managed, with a Re-Order Point logic, by the same operator.

Figure 18: in this figure it is possible to see how all the products, except screens and mosquitos' nets are managed. It is possible to affirm that from a logical point of view the company operates with a star configuration with a warehouse in the middle. In fact, only screens use sequential assembly lines, while all the thermal products receive raw materials from the warehouse, assemble the products in the area (most of the times there are not material handling inside an area) and return the finished products to the warehouse (see *figure 14* for a graphical schematization of a star configuration). The logical flow in order to start the production starts in the warehouse where raw materials are picked. After the picking the raw materials are transported to the productive area. When the productive area receives the batch of raw materials the production can start. When a certain amount of products are finished, a logistic operator transport them from the productive areas to the warehouse.

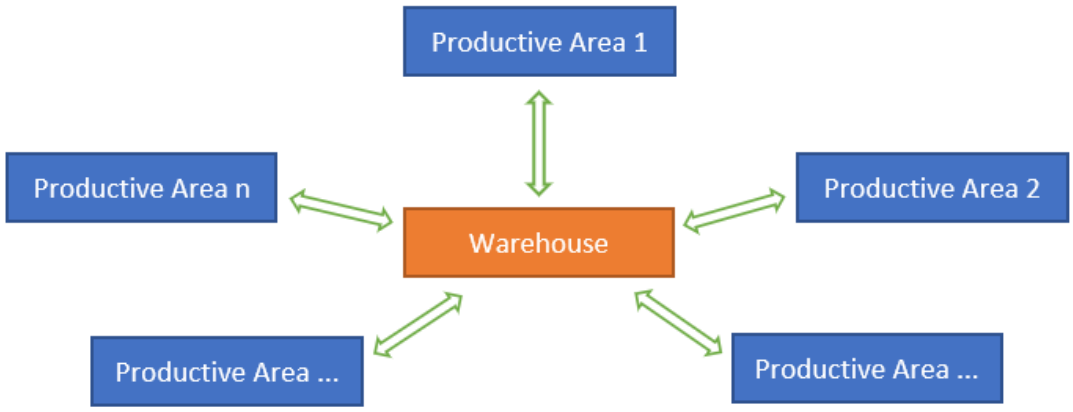

Figure 14. Scheme of a Star Configuration

Notes on Petri Nets Graphs' notation used:

-Big black full dot: it represents the refill through a Re-Order Point logic.

-Transitions are represented with a rectangle and not with a line.

-All the other notations remain the standard ones.

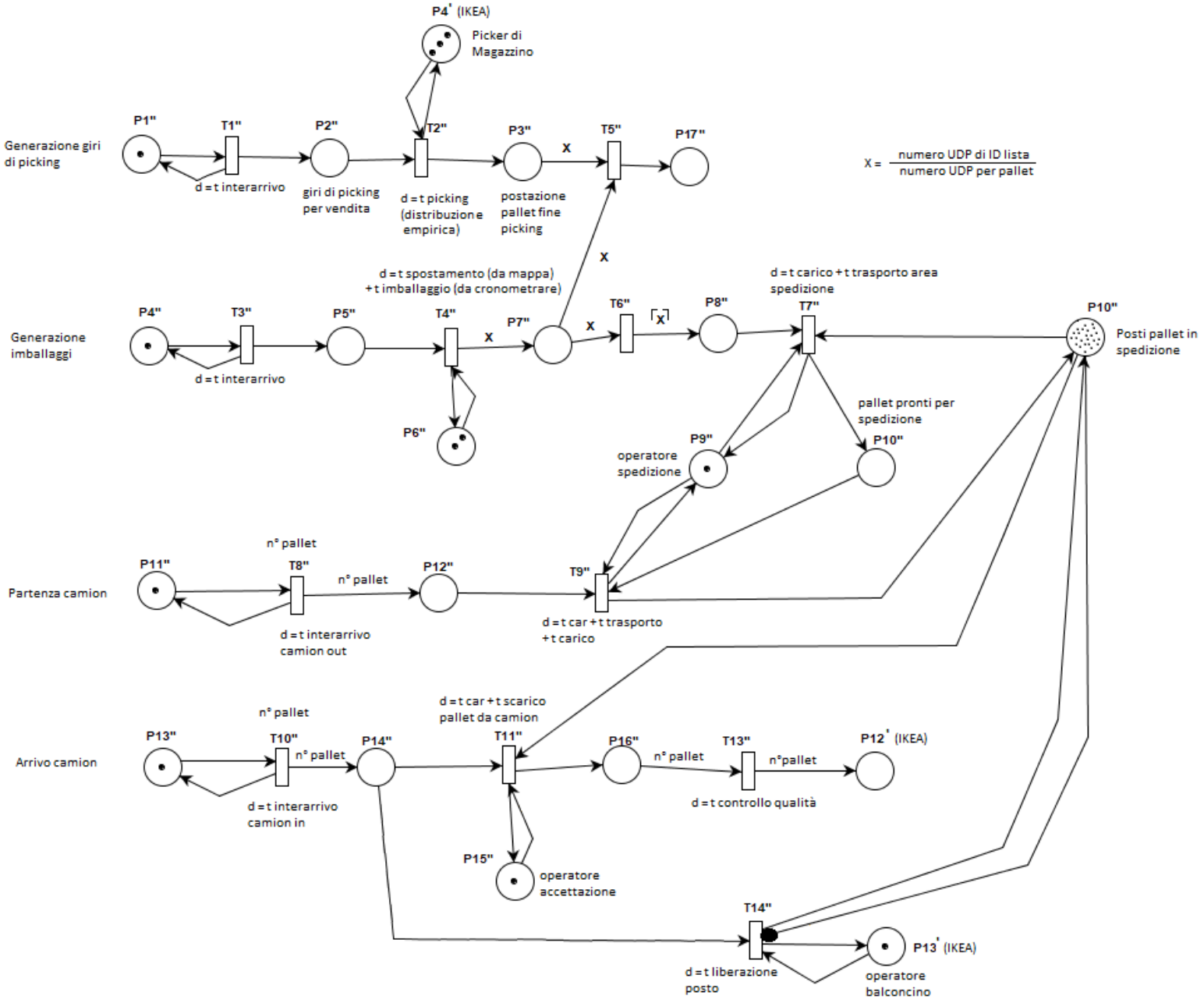

Figure 15. Petri Net Graph: Picking, Packaging. Arrival and Departure activities

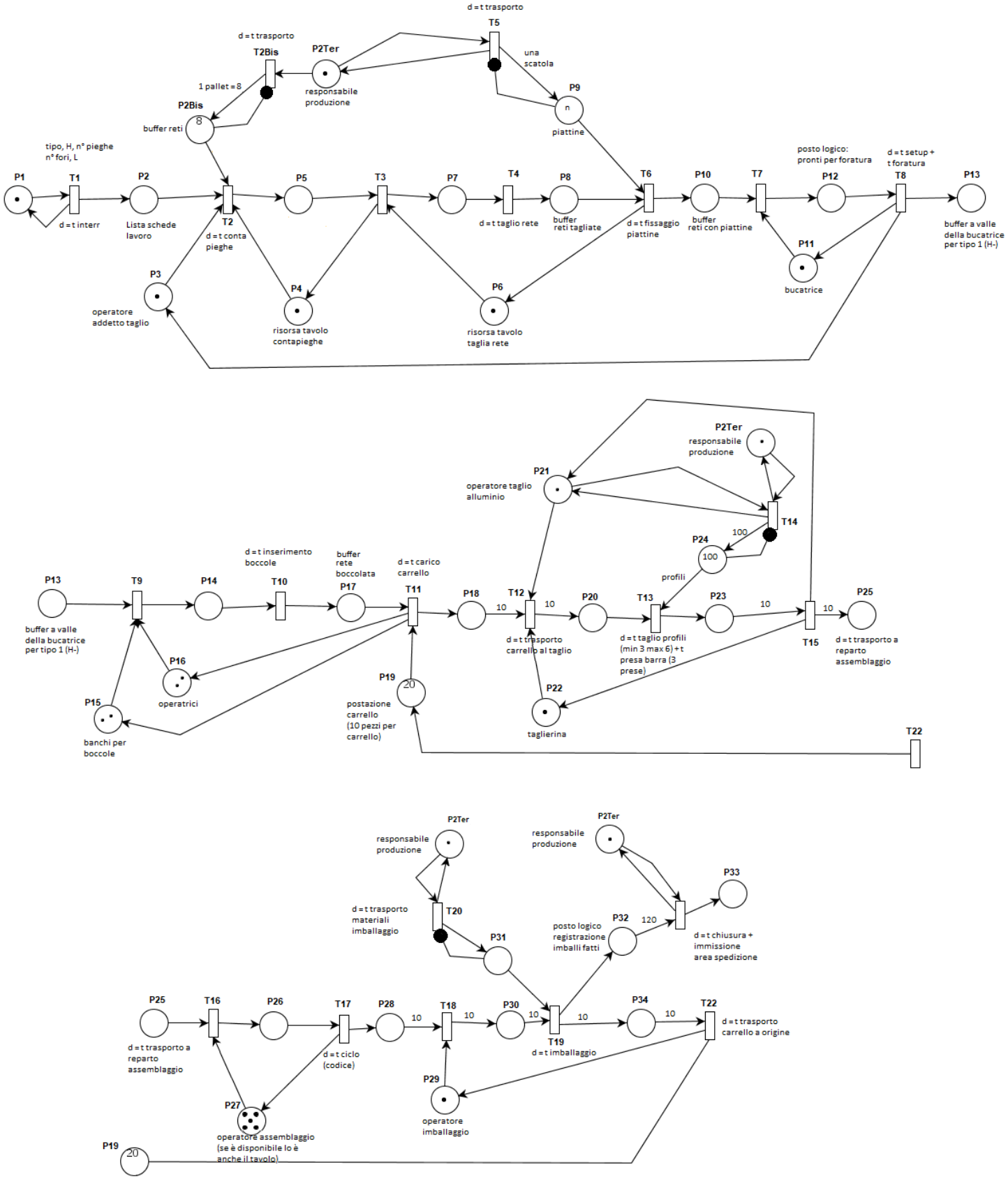

Figure 16. Petri Net Graph: Process for Screen Product H-

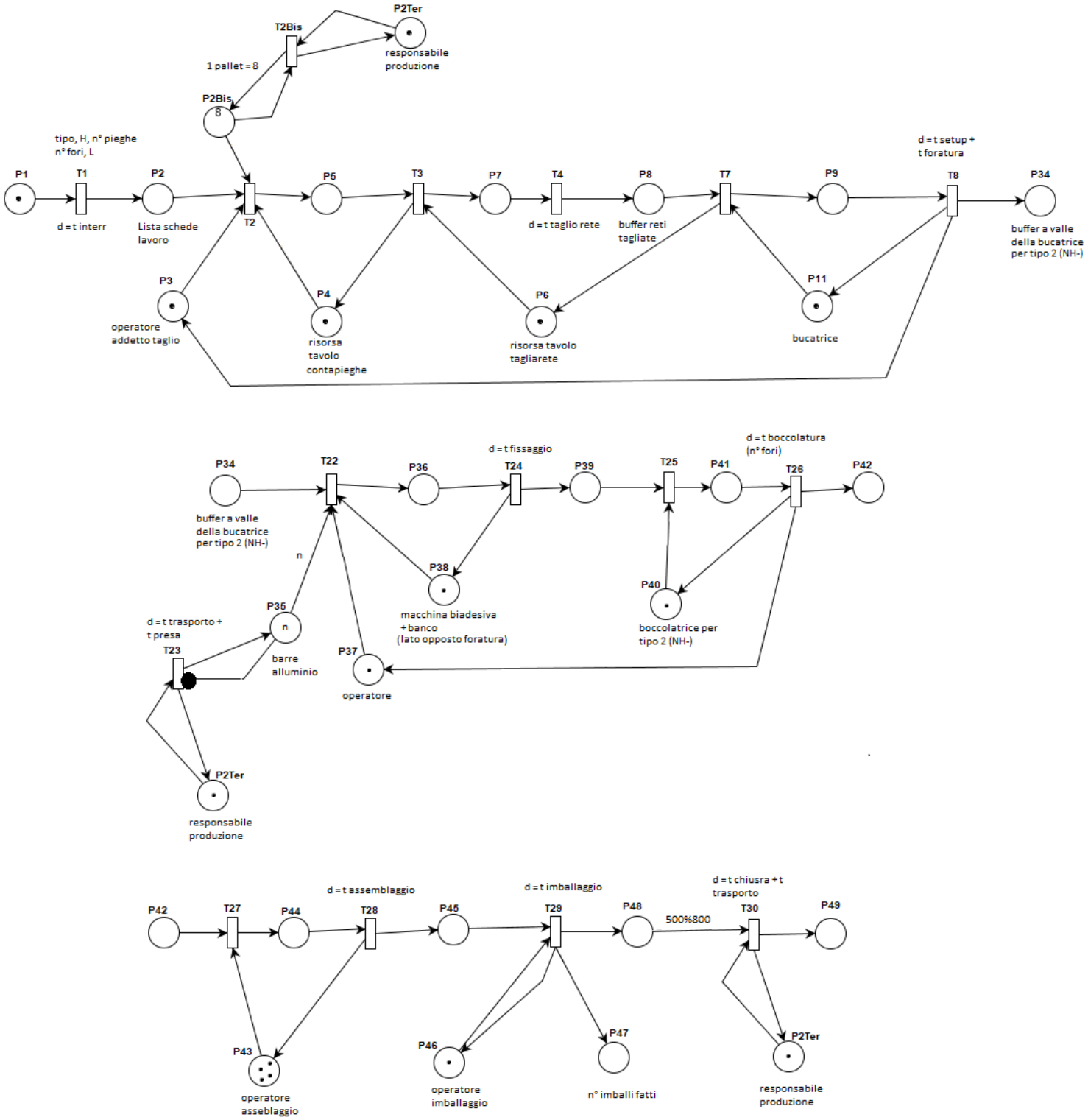

Figure 17. Petri Net Graph: Process for Screen Product NH-

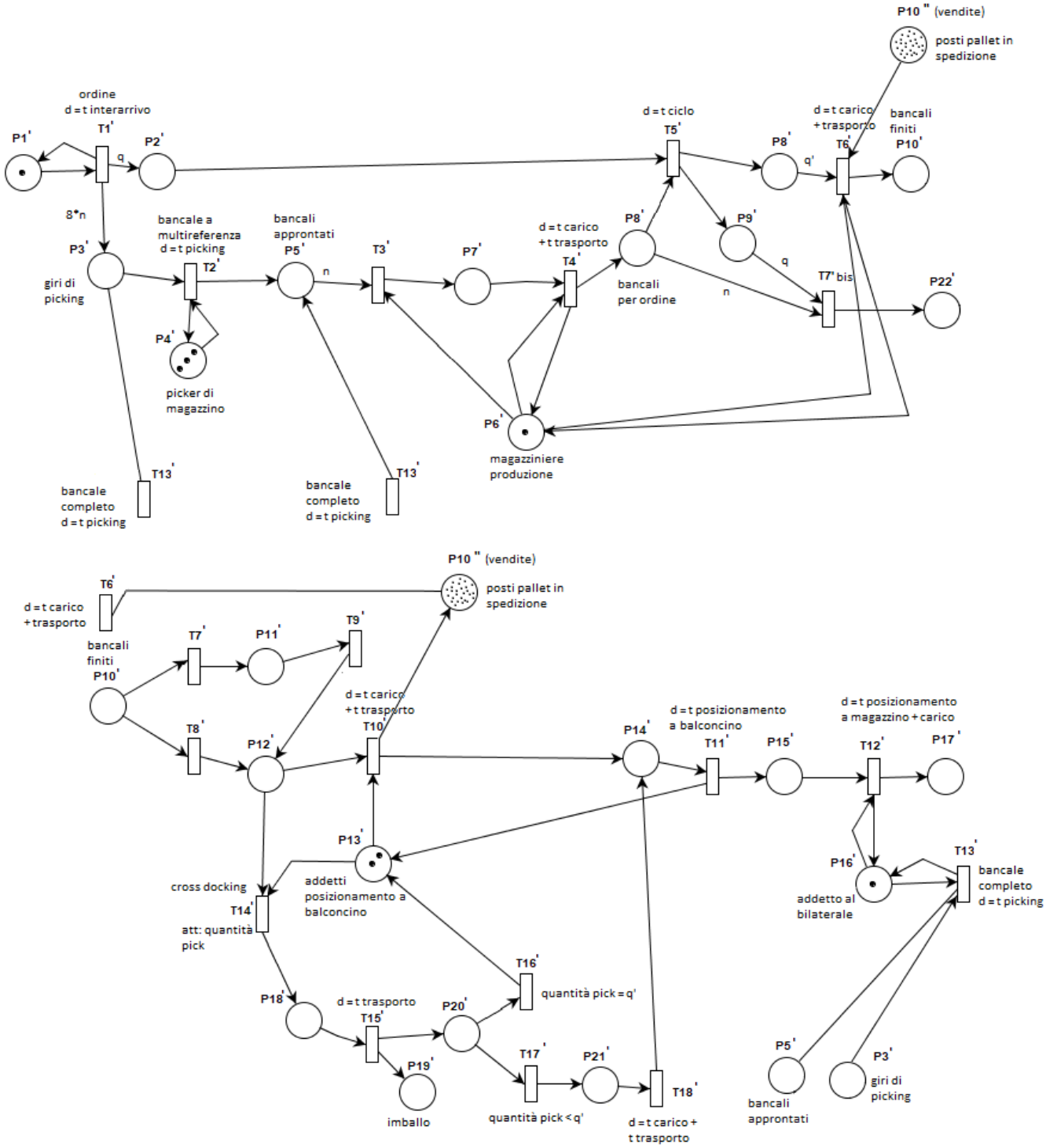

Figure 18. Process of picking and production

3.3.1 Logistic Operators

A brief description of all the logistic operators working in the company is here presented with the association to the indexes of the corresponding Places in the Petri Nets Graphs (from *figure 15* to *figure 18*):

- P2Ter – production manager. He is in charge of following all the productive operations. He is also in charge of refilling the raw materials of the screens and mosquitos' nets and preparing the delivery of those products.
- P15'' – acceptance operator. He is in charge of unloading the products arriving from the supplier and put them in the place used for quality check.
- P13' – two 'balcony' operators. Their main task is to support the narrow aisle forklift truck operator (known also as trilateral or bilateral turret truck), loading and unloading the truck. They take products from the narrow aisle forklift truck and they put the products in the floor picking area and vice versa. They are also in charge of taking the just arrived goods from the delivery area and bringing them to the narrow aisle forklift truck operator or to the cross docking area.
- P16' – narrow aisle forklift truck operator. He is in charge to store (and unstore) products in the 7-meter-tall narrow aisle warehouse.
- P4' – three warehouse pickers. They are in charge of two pickings: production and sale pickings. Both of the pickings start with taking the required goods (raw material for production and finished goods for sale) from the floor warehouse and bring them to the warehouse/production interchange area (picking for production) or to the packaging area (picking for the sale).
- P6' – transport operator. She is in charge of the material handling from the warehouse/production interchange area to the production and vice versa.
- P6'' – two packaging operators. They have to receive goods, package them and leave them in an area close to the package machinery.
- P9'' – delivery operator. He is in charge of taking the packaged products and bringing them to the delivery area. When a truck arrives, he has to take the goods from the delivery area and load the truck.

3.4 PROBLEM DEFINITION

As it was possible to infer, the company we are dealing with is about to face a Facility ReLayout Problem. In particular, the company is about to benefit from the emptying of a 2400 m^2 building of its property, previously rented to another firm.

The new plant is located between the two already used plants. In fact, the actual plants are 55 meters distant while the new building is 5 meters from the first plant and 17 meters from the second one.

The FLP to deal with falls within a particular category of problems: is a mix between relayout (the company is already operating and the productive and logistic areas are well defined) and greenfield design (the company can exploit a new building). The company is so interested in knowing how is better to use the new building; the evaluation will be mainly on moving some part of production or warehouse or both to the new plant and the evaluation of a relayout of the not moved production activities / warehouse inside the already used plants.

The overall plan floor of the operative plant is shown in *figure 19*.

The plant at the top side of *figure 19* is the one composed by warehouse + production of temperature control systems.
The plant at the middle of *figure19* is the plant which will be emptied shortly.
The plant at the bottom of *figure 19* is the plant dedicated to mosquitos' nets and screens production.

Figure 19. Floor Plan of the plants

The characteristics of the actual areas involved in the production of the different kinds of products are listed in *table 4.*

GROUPS	**AREA [m^2]**	**# Squares**	**ID**	**kg**
Presses Area	365	25	1	0
CLIP	66	5	2	126680
EDF	200	14	3	171292
IP55	50	4	4	7789
IK	20	2	5	0
Resistors	37	3	6	16759
Tables	90	7	7	246478
TCU	48	4	8	8294
TRT	34	3	9	6282
VFC	61	5	10	6362
Lean Zone Filters	37	3	11	25744
Screens and Mosquitos' Nets	933	64	13	0
Warehouse	3200	1	20	0

Table 4. Characteristic of production areas

In *table 4* has been made use of acronyms of the products' type; the deep comprehension of the type of product worked in a specific area is not relevant for the aim of this thesis.

In the first column of *table 4* there are the groups in which the machineries and tables with tools are grouped. The grouping methodology is made based on products produced in the productive area.

In the second column there is the total area used by the group comprehending the surrounding space attributable to the area (i.e. pathways, etc).

Third and fourth columns are important for the solving methodology; in fact, to better use the proposed method is important to schematize the actual layout into a grid. The new schematization is made by assigning to the areas a proportionate number of squares (column 3 is obtained with an approximation for excess of the total area divided by 14,75) and an index (groups in column 4 are indexed in increasing order).

In the following figure (*figure 20*) is shown the schematization of the actual disposition of the machineries and warehouse of the company. As can be easily comprehended, the schematization

comprehends the three plants. The numbers used in the scheme are the indexes assigned to groups in *table 4*.

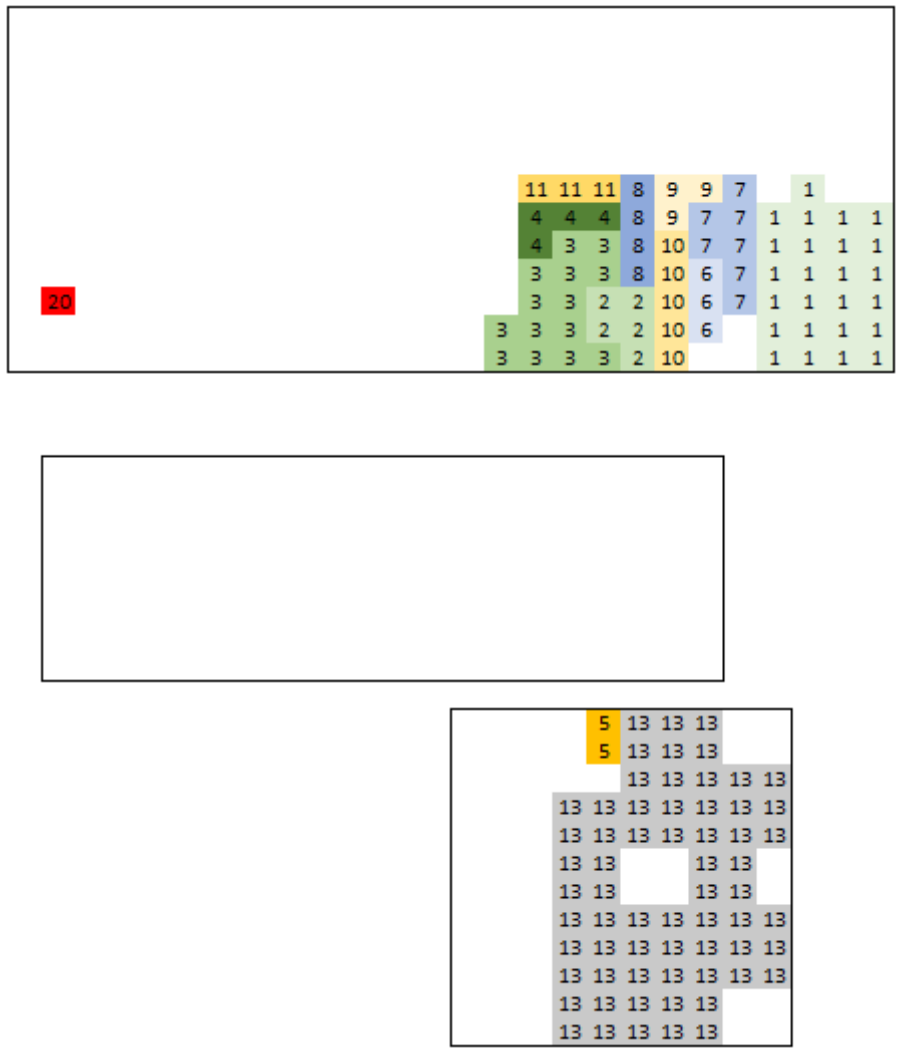

Figure 20. Scenario 1: As-Is Situation

Some clarifications and notes should be done.

- The apparently strange disposition of machineries inside the plant at the bottom of the scheme is given by physical constraints imposed by the presence of some columns
- In the schematization the upper plant seems 75% empty: in reality the left side is completely dedicated to warehouse even if 3200 m^2 have been schematize with just one square; the up-right side is instead used for a small part for some offices (quality, operations management) and restrooms, and for the main part for activities of reception of raw materials and departures of finished products.
- The warehouse has been schematized with one square but the square is not centred; this is due to the fact that the picking activities are carried in the floor picking in the last part of the

warehouse and the centre of gravity of the interchange area between production and warehouse coincide with where is put the square in the shown schematization. The remaining part of the warehouse (the one closer to productive areas) is composed of 7-meter-tall narrow aisle warehouse completely used for stock of finished goods or raw materials arrived from the supplier.

- Picking for production is inefficient: it requires 35% time more than picking for selling. A new location/solution for picking for production is one of the aims of the company.
- The total actual cost of the layout is 15.516.269. The explanation of how this cost has been evaluated will follow in the next paragraph.

3.5 METHOD EXPLANATION, NOTES AND ASSUMPTIONS

3.5.1 Method Explanation

In order to originate *table 4* and the schematization shown in *figure 20*, a file containing the picking list of 10 months of activities and a file containing the dimensions of the plant and the machineries have been provided by the company. Thanks to that, it is possible to apply a hybrid methodology in which a feasible layout is manually-originated and its total cost is calculated through a standard matrix of costs multiplied by the distance matrix between warehouse and productive areas.

For every analysed layout have been calculated:

- X-position and Y-position of the Centroids of the productive areas in a grid system
- Rectangular distance between warehouse/s and each productive area
- The total cost of the layout

To calculate the X-position and Y-position of the Centroids has been used the same standard weighted area method used when calculating the centroid of a figure made of rectangles. The rectangular distance is given by the formula: $|x_{warehouse} - x_{area\ i}| + |y_{warehouse} - y_{area\ i}| \quad \forall i$

In some cases, the rectangular distance could have not been applied due to physical constraints that forced the handling route to follow a longer path. In that case new points were added in order to calculate the total travel path.

The total cost of layout is obtained by summing all the rectangular distances of areas i multiplied by the kgs exchanged in 10 months of production.

3.5.2 Assumption and Notes

Assumption 1

Some assumptions were made while calculating the total cost of a layout. The only distance taken into consideration is in fact the one between warehouse and productive areas (to be precise we are talking about the distance between warehouse/production interchange area and productive areas). In paragraph *3.3.1 Logistic Operators* more travel distances were explained. A recap of the travel distances is here presented:

- Arrival/Delivery area → warehouse 7m (narrow aisle warehouse)
- Warehouse 7m ←→ floor warehouse
- Floor warehouse ←→ warehouse/production interchange area
- warehouse/production interchange area ←→ productive areas
- Floor warehouse → packaging area
- Packaging area → delivery area

As is possible to notice five distances out of six were not used to calculate a total cost. The assumption at the base is that in each scenario considered the relative distance between warehouse 7m, floor warehouse, warehouse/production interchange area and packaging area is not changed and so the handling cost between these areas is not differential and can be omitted.

In *figure 21* is shown a focus on two of the possible locations of the above listed logistic areas.

Figure 21. Location of logistic areas

In *figure 21.a* is shown the actual location, while in *figure 21.b* is shown the other main location for the warehouse/production interchange area and the relative new location of the other logistics areas. Is easy to note that the relative distances in *figure 21.a* and in *figure 21.b* are almost unchanged.

Assumption 2

The reader may have noticed that in the last column of *table 4* there are four groups with 0 kilograms. One of them is of course the warehouse (warehouse/production interchange area) which is used to calculate the material handling between it and the productive areas, and so its kgs can only be 0.

Another group with 0 kgs is the screens and mosquitos' nets one. This productive area, as explained in *paragraph 3.3*, does not exploit the same resources of the other productive areas (i.e. it is not served by the transport operator and does not use the same warehousing location). For this reason, it can be considered as an autonomous productive area not influenced by the relayout. It will just be considered as a space consuming area.

Another group presenting a value of 0 kgs is the IK group. IK products occupy a small space (raw materials warehouse close to productive area + productive area + warehouse of finished products close to productive area). This kind of product is characterized by a huge number of products per week, but the small dimension of the products makes it possible to refill raw materials only once a week and only with few travels. The material handling of this product has not been taken into consideration due to its irrelevance.

The last group with 0 kgs are the presses. In this area raw plastic material is moulded. As it happens for screens and mosquitos' nets, it does not use the same resources of other productive areas and so is not taken into consideration.

In the following pages will be presented and commented the layouts taken into consideration and studied through the hybrid methodology explained above.

3.6 PROBLEM RESOLUTION

Scenario 2

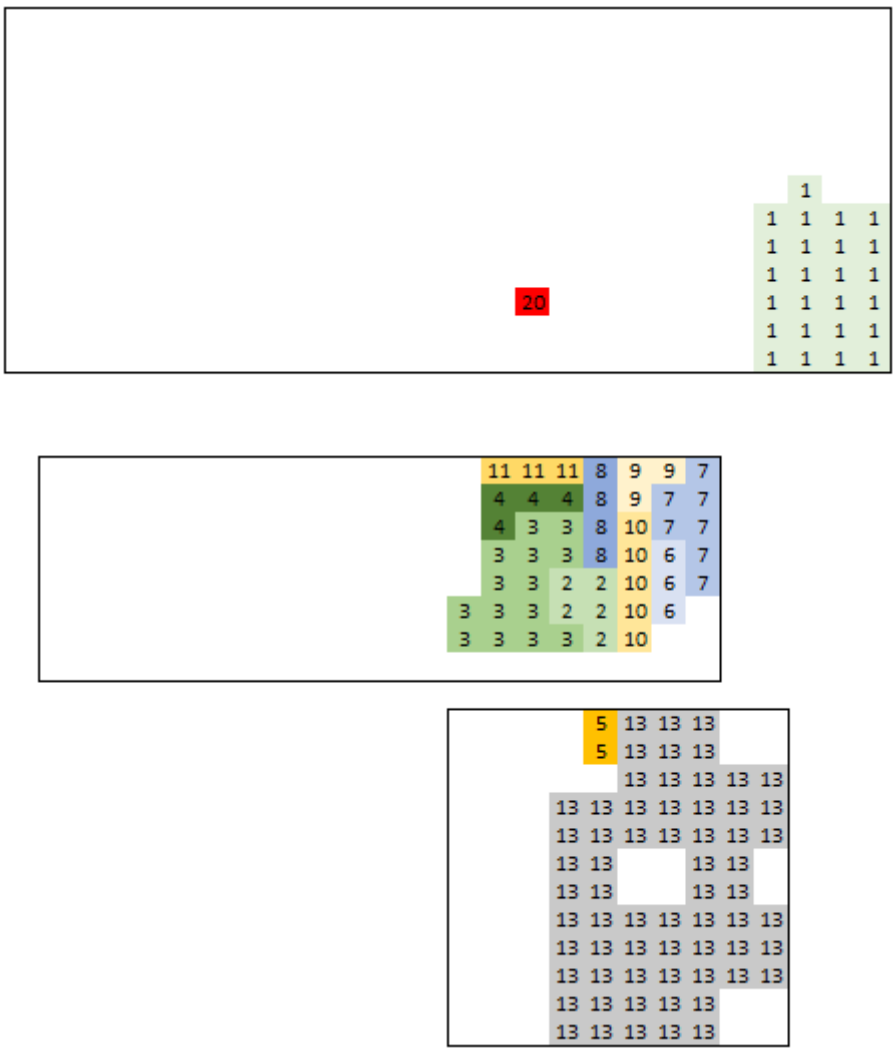

Figure 22. Scenario 2: New building dedicated to thermal assembly, new location of warehouse dedicated to production activities

Overall costs are 7.777.893. With this scenario production is located in three different locations and the warehouse of picking for production is moved closer to the production site.

Pros:

- There is the possibility to use double supermarket logic
- Presses are not moved, low investment

Cons:

- The production manager has to manage production and move materials between three different locations

Scenario 3

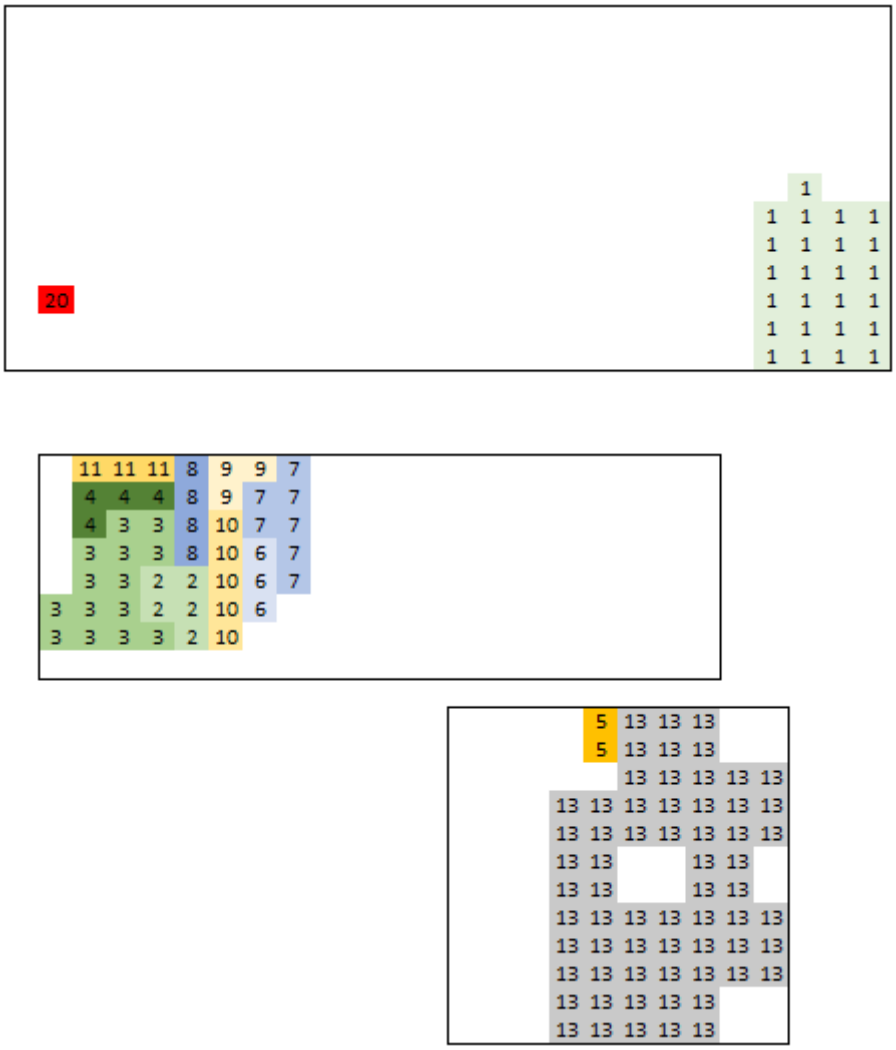

Figure 23. Scenario 3: New building dedicated to thermal assembly, actual location of warehouse dedicated to production activities

In this scenario the total cost is: 8.883.013. The warehouse for picking activities is not moved, but the thermal production is moved closer to it.

Pros:

- There is the possibility to use double supermarket logic
- Presses are not moved, low investment

Cons:

- The production manager has to manage production and move materials between three different locations

The pros and cons are the same of scenario 3, but the total cost is higher. Scenario 3 is worse than scenario 2.

Scenario 4

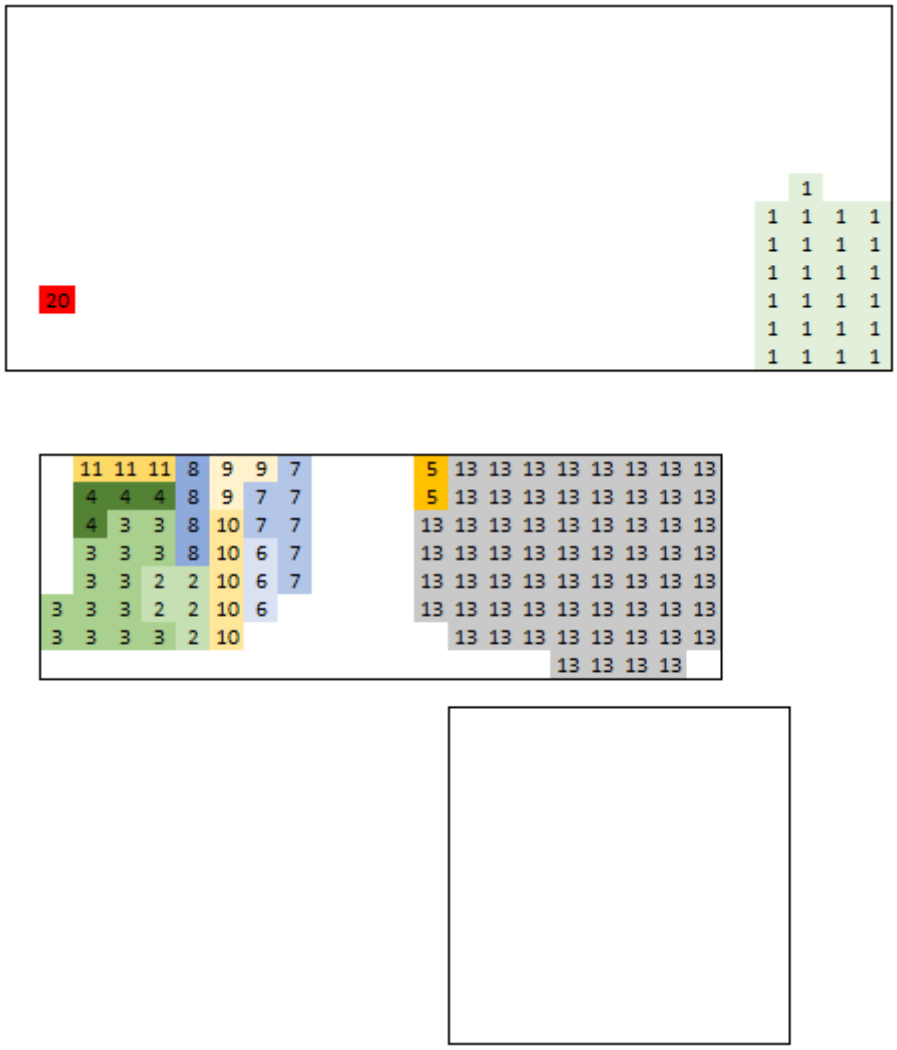

Figure 24. Scenario 4: New building dedicated to thermal and screens assembly

The total cost is 8.883.013. It removed the cons of the production manager to manage three different plants.

Pros:

- Presses are not moved, low investment
- Easy connection between warehouse and all the production areas
- There is the possibility to use double supermarket logic

Cons:

- Emptied a plant with physical constraints and so difficult to be used for new purposes, but it can be easily moved part of the production of screen back to the old plant if the demand of screens continues to grow

Scenario 4.1

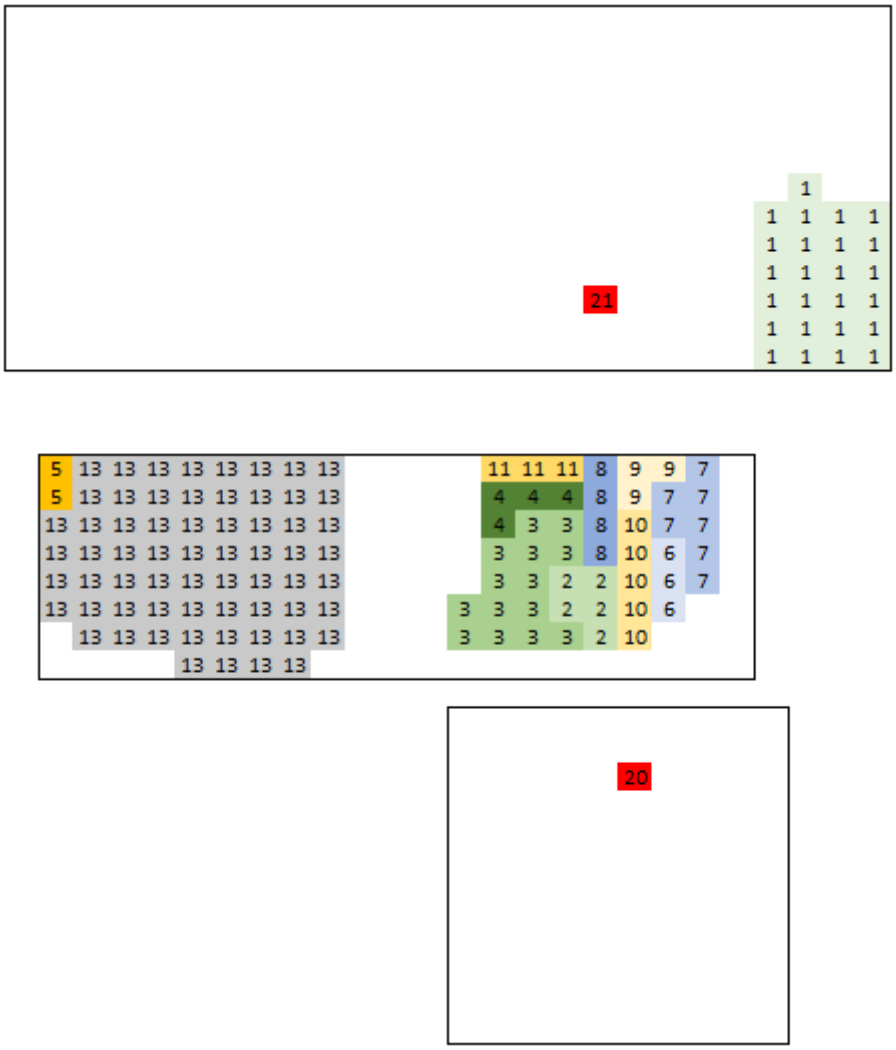

Figure 25. Scenario 4.1: New building dedicated to thermal and screens assembly and double warehouse

The scenario 4.1 has a total cost of 7.034.742. Differently from scenario 4, here the unused plant finds its utility serving as a second warehouse.

Pros:

- Presses are not moved, low investment
- Easy connection between warehouse and all the production areas
- There is the possibility to use double supermarket logic

Cons:

- Doubled the warehouse: risk of doubling the pallets
- Higher investment respect to scenario 4

Scenario 5

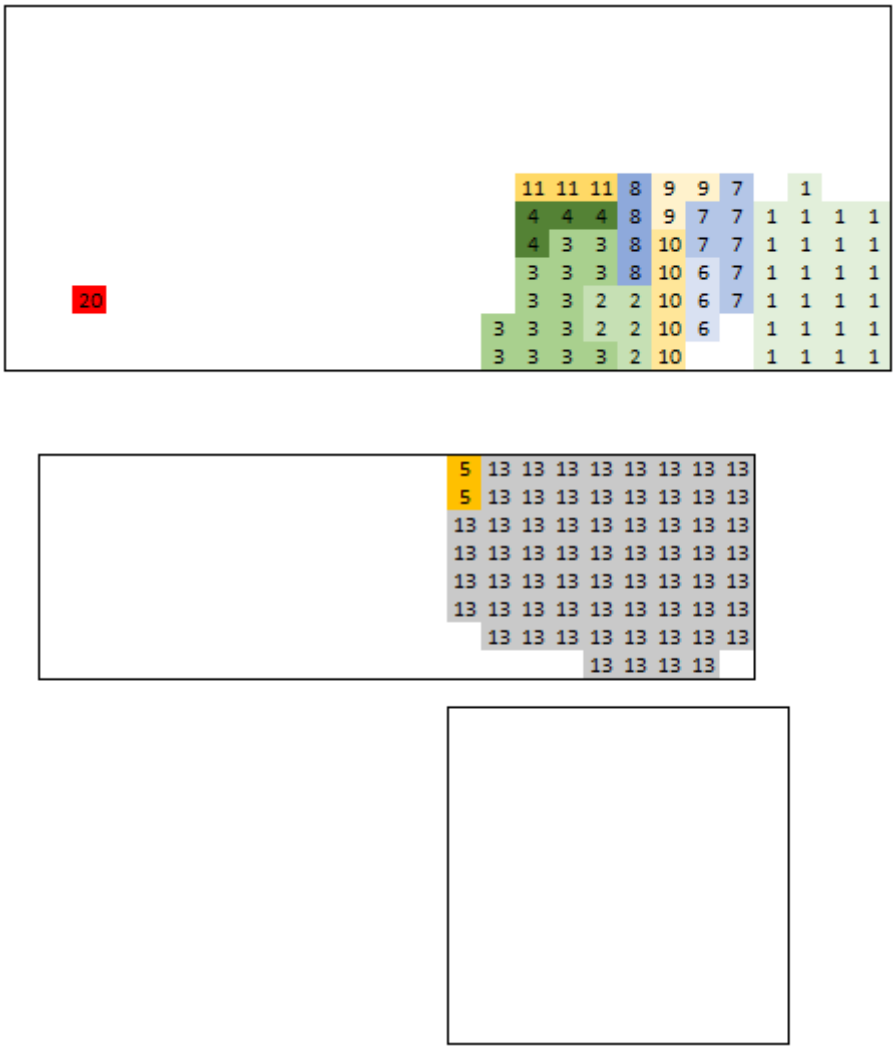

Figure 26. Scenario 5: New building dedicated to screens assembly

This scenario is characterized by a total cost of 15.516.269. The total cost is the same as the actual layout.

Pros:

- Possibility to easily enlarge the production of mosquitos' nets and screens in the new plant without any new investment in machineries handling

Cons:

- Emptied a plant with physical constraints and so difficult to be used for new purposes
- Total cost equal to the actual one

Scenario 6

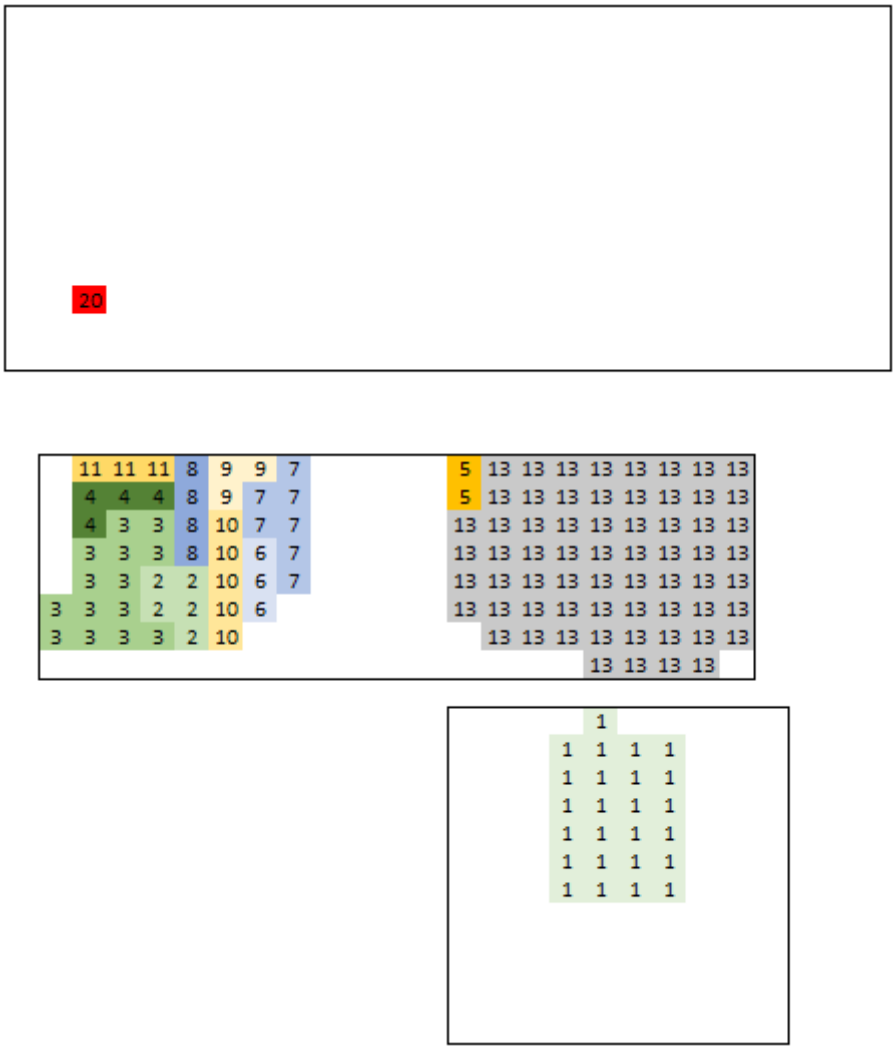

Figure 27. Scenario 6: New building dedicated to thermal and screens assembly, presses moved to the other old building

Scenario 6 presents a total cost of 8.883.013. This scenario frees the actual thermal control production plant.

Pros:

- Possibility to enlarge offices and restrooms
- Possibility to enlarge the warehouse

Cons:

- Move the presses, high investment

Scenario 6.1

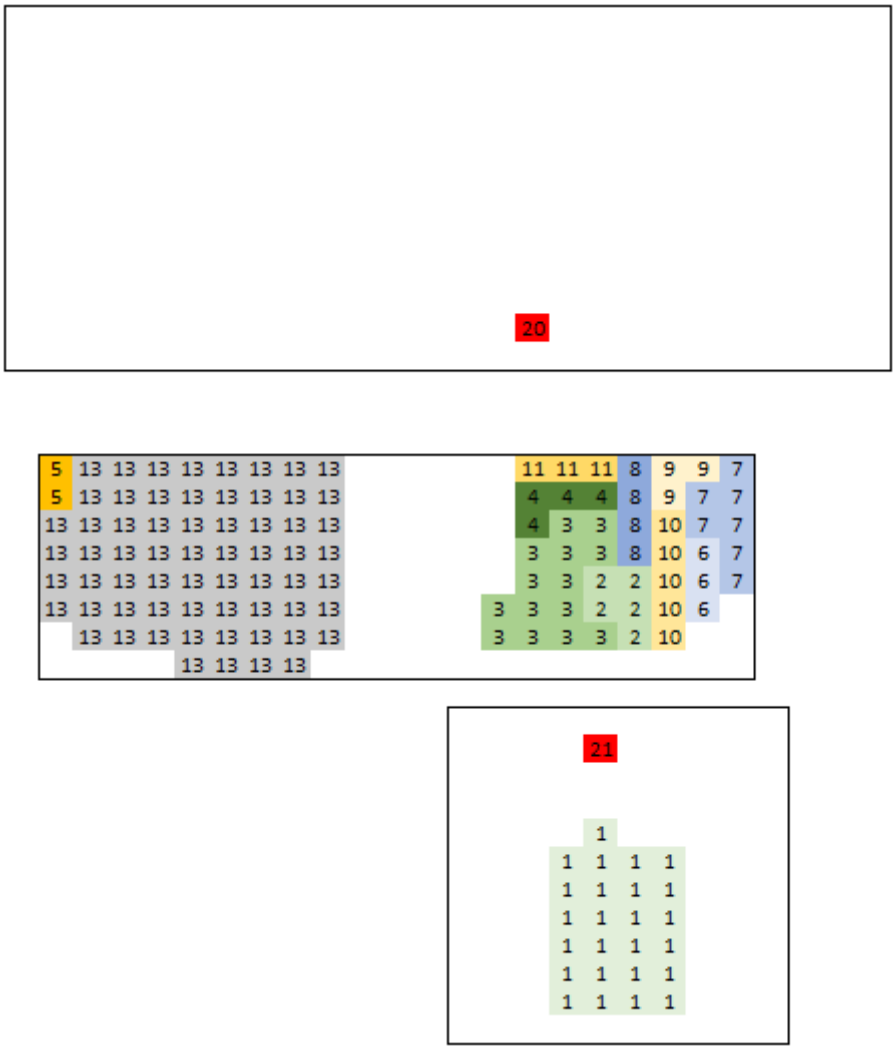

Figure 28. Scenario 6.1: New building dedicated to thermal and screens assembly, presses and warehouses for production activities moved in part to the other old building

This scenario, starting from scenario 6, evaluates the usage of two picking warehouses. The total cost is 8.606.634.

Pros:

- Possibility to enlarge offices and restrooms
- Possibility to enlarge the warehouse

Cons:

- Move the presses, high investment
- Doubled the warehouse: risk of doubling the pallets

Scenario 6.2

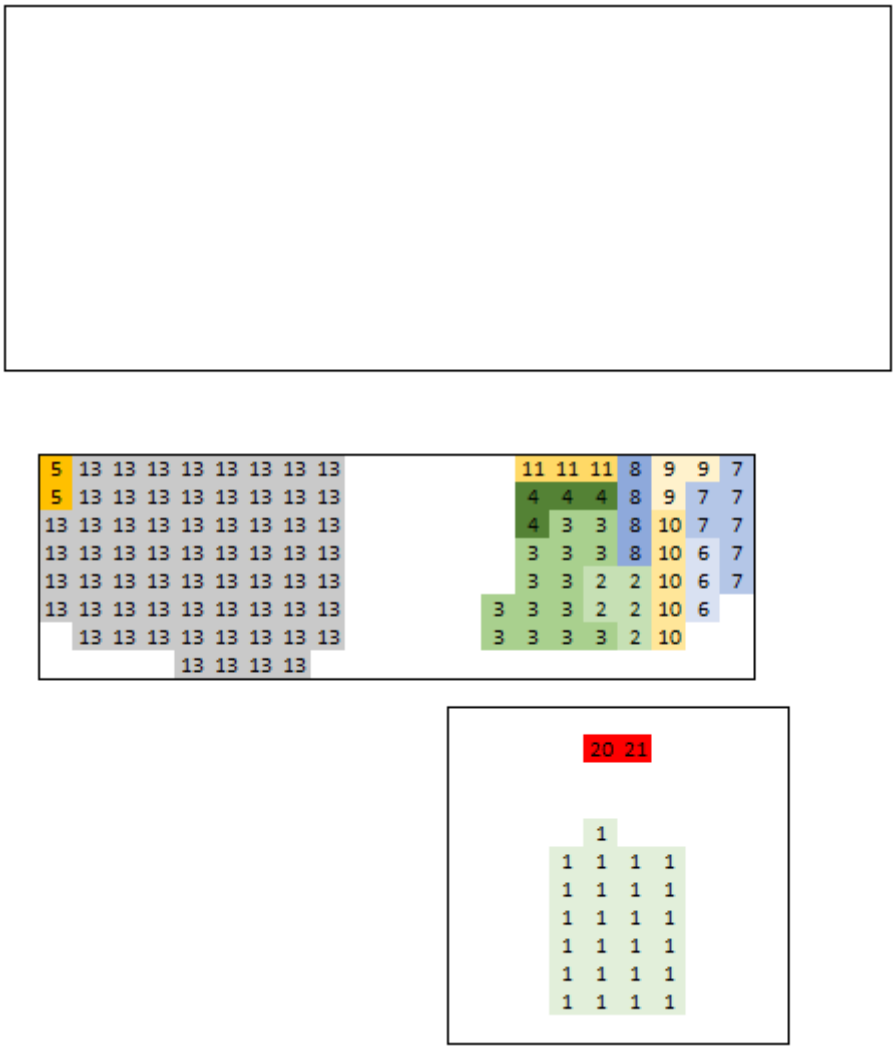

Figure 29. Scenario 6.2: New building dedicated to thermal and screens assembly, presses and warehouses for production activities moved all to the other old building

This scenario starts from the scenario 6.1 and tries to avoid the risk of doubling pallets. The total cost of this solution is 7.020.725.

Pros:

- Possibility to enlarge offices and restrooms
- Possibility to enlarge the warehouse

Cons:

- Move the presses, high investment
- Risk of not having enough space for warehouse in the actual mosquitos' screens plant

Scenario 7

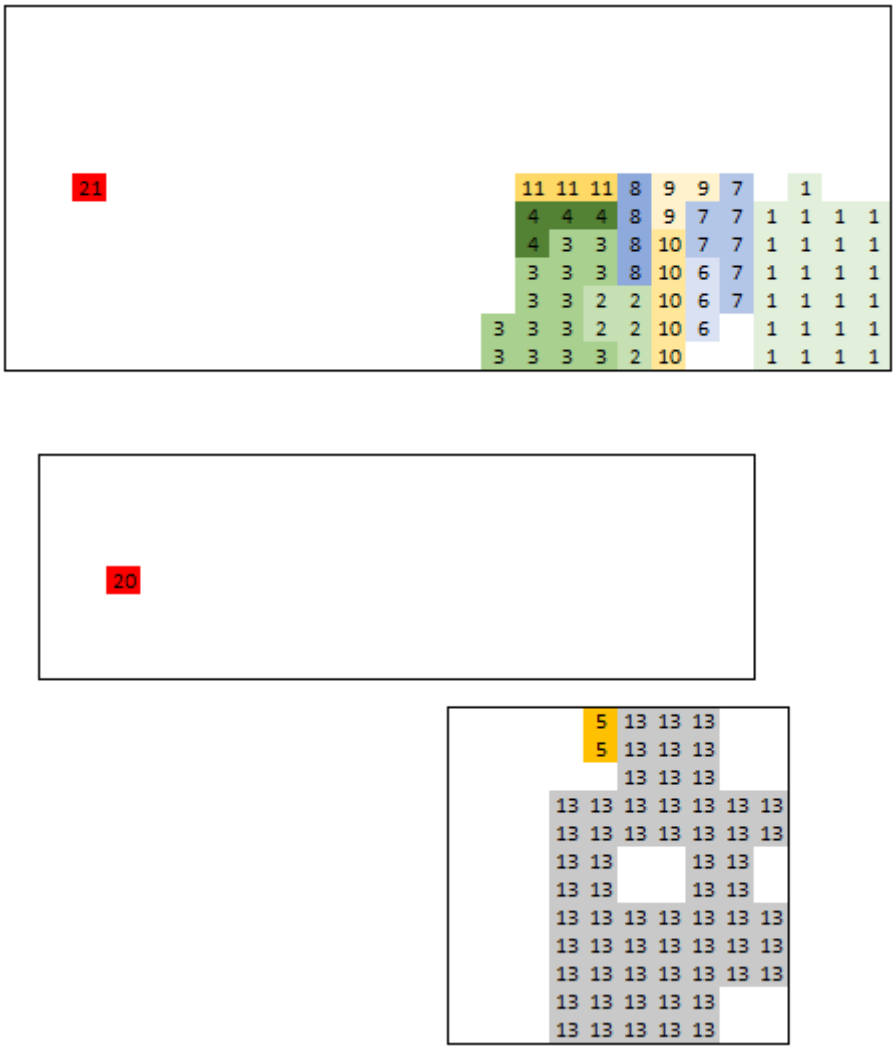

Figure 30. Scenario 7: New building dedicated to a second warehouse

This last scenario has a total cost of 17.577.015. Here is taken into consideration to move the picking warehouse to the new plant

Pros:

- No machineries handling is needed, investment cost almost 0

Cons:

- Total cost higher than the actual one

Summing up, nine new scenarios have been studied and analysed. In *table 5* it is possible to see a brief comparison between the actual scenario and the new scenarios, numerically ordered.

SCENARIO	**TOTAL COST**	**# of PROS**	**# of CONS**
Scenario As-Is	15'516'269	-	-
Scenario 2	7'777'893	2	1
Scenario 3	8'883'013	2	1
Scenario 4	8'883'013	3	1
Scenario 4.1	7'034'742	3	2
Scenario 5	15'516'269	1	2
Scenario 6	8'883'013	2	1
Scenario 6.1	8'606'634	2	2
Scenario 6.2	7'020'725	2	2
Scenario 7	17'577'015	1	1

Table 5. Sum-up of all scenarios

One limitation of all the FLPs methodologies is that they are not able to keep into consideration the qualitative benefits or malus deriving from a layout, so there is the need to study them apart. When a layout designer is studying a new possible layout, as it has been done here, he has also to keep into consideration some peculiarities of the layout that can bring new advantages or remove actual ones. If the layout would have been chosen only taking into consideration the total cost of the layout, the whole analysis carried out in this chapter, in which a qualitative analysis is shown close to the quantitative result would be useless. The solution is not assured to be Scenario 6.2 even if it presents the lowest total cost. In fact, *table 5* is used to assign a first visual impact of scenarios as described through three characteristics: Total Cost, Pros, Cons. The solution to be chosen, is the one that presents the best trade-off between these three characteristics.

In order to reduce the number of alternatives considered it is possible to make a pairwise comparison. An alternative *j* is discarded against an alternative *i* if:

$$totalcost_j > totalcost_i \;\; , \;\; \#Pros_j < \#Pros_i \;\; , \;\; \#Cons_j > \#Cons_i$$

Scenario 2 discards scenarios: 3, 5, 6, 6.1, 7.

Table 5 is then reduced to *table 6:*

SCENARIO	TOTAL COST	# of PROS	# of CONS
Scenario 2	7’777’893	2	1
Scenario 4	8’883’013	3	1
Scenario 4.1	7’034’742	3	2
Scenario 6.2	7’020’725	2	2

Table 6. Best scenarios remained after pairwise comparison

In *figure 21 (location of logistic areas)* were shown the real locations of the actual layout of logistic areas and the location of those areas in a new possible layout. Both solution 4.1 and 6.2 present a different location of that areas; in these scenarios the new logistic areas are inserted in the plant in which now are carried the productive activities for screens and mosquitos’ nets; in *figure 31* can be seen its representation.

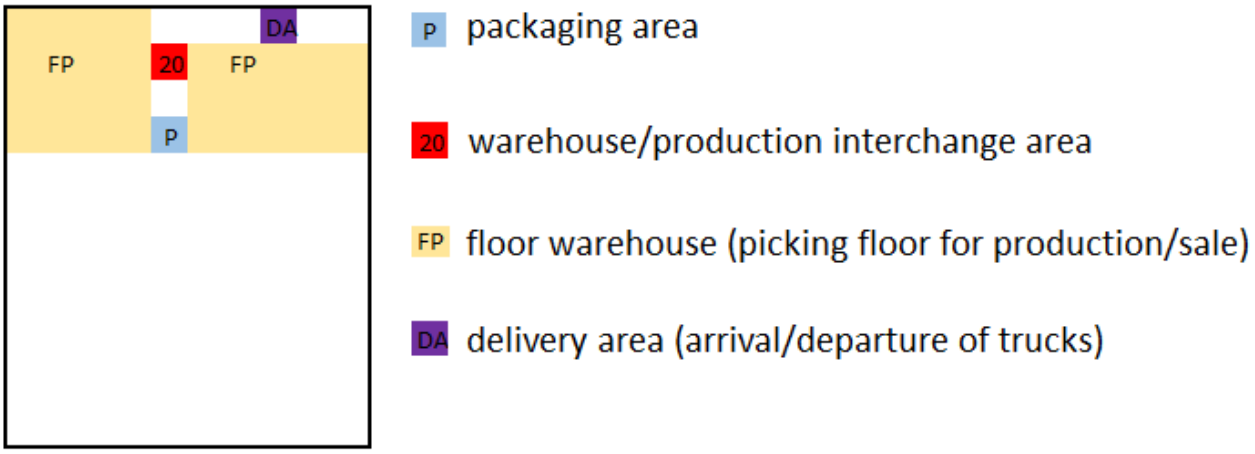

Figure 31. Logistic areas in scenarios 4.1 and 6.2

It is possible to notice that there is no warehouse in this configuration. This is due to the fact that in this configuration, the trucks arriving with raw materials for production unload the goods directly in the floor warehouse which will be used as a temporary warehouse for production activities and for a warehouse of packaged finished goods ready to be delivered with the first arriving truck.

This scheme is characterized by a high level of complexity in the management: should encompass lean techniques such as JIT, Kanban and furthermore the integration with suppliers and customers.

Another factor increasing the difficulty to apply this configuration is given by the fact that not all the goods arriving from supplier are raw materials: a noteworthy % of the goods arriving to the company

should not be worked in the productive areas but should be kept in stock for a small amount of time in a cross docking mentality because that goods will be sold in few hours/days. The truck cannot unload all the goods in the same place and this is particularly difficult and time wasting for the acceptance operator that would be forced to unload all the goods from the truck, keep only the production goods, load the truck again with the goods destined to cross docking, change the plant and unload the truck again.

The reduction in cost of solutions 4.1 and 6.2 is of about 10/20% in respect to scenarios 2 and 4 (see *Table 7*) and is highly probable not repaying the difficulties and the effort deriving from that kind of configuration.

Cost saving %	Scenario 2	Scenario 4
Scenario 4.1	-9,5%	-20,8%
Scenario 6.2	-9,7%	-21,0%

Table 7. Cost Saving Scenarios 4.1 and 6.2 in respect to Scenario 2 and 4

For that reason, Scenario 4.1 and Scenario 6.2 are discarded. Now only two scenarios are remained: 2 and 4. Going back to the pages in which are analysed pros and cons of the two scenarios, it is possible to notice that the pros of scenario 2 are the same as scenario 4, while the cons of scenario 2 is solved with scenario 4.

Furthermore, the only cons of scenario 4 "emptied a plant with physical constraints and so difficult to be used for new purposes" would be cancelled if the demand of mosquitos' screens will continue to grow in the next years because part of the assembly lines can be moved back to the old plant and there will be enough space to double all the lines, both in the new plant and in the old plant. (Note: the demand trend is increasing but it is impossible to forecast the demand of the next years due to market peculiarities).

After all that considerations, the scenario proposed as the best one at this moment, considering the trade-off between cost of material handling and pros and cons of the layout, is SCENARIO 4.

In order to make a complete analysis of all the scenarios, in the next chapter every scenario will be simulated because there is the possibility that one of the discarded scenarios presents some peculiarities that will lead to a further cost reduction becoming the best scenario possible for the future layout of the company.

It is important to underline that the utility function used to choose the best scenario changes as the designer changes. It is also important to adapt the utility function to the needs of the company analysed, putting the focus not on what the management think they want but on what they really need.

4. SIMULATION

4.1 INTRODUCTION

Simulation is a process through which a system is first recreated through a model and then analysed numerically in order to find useful data to estimate the quantities of interest.

A simulation model is composed of modules and relations which are used to describe the components of the system and their interactions and behaviours.

It is important to remark that the scope of a simulation is the output analysis. For output analysis is intended the measuring of relevant data. It should be also considered that it is dangerous to use simulation tools to obtain results and use that results as granted: it is important to assign to data their statistical validity.

In the previous chapter has been described the company through the Petri Nets Graph and after that, through an hybrid methodology has been chosen the layout that seems the best one to be used for the production plant of the company. The choice was made through a maximization of a utility function given by the trade-off between quantitative results and qualitative results (studies on layout benefits/maluses).
In this chapter will be studied the effectiveness of the recently found new proposed layout (and also for the discarded ones) through the study of a simulation model. The results obtained thanks to the model inserted into the simulator will be used for final discussions and comments on the layouts.

4.2 DATA GATHERING

In order to create a model, three main types of data should be gathered and analysed.

1. The logical flow of the activities:
   This kind of data was gathered through a visit in the company. The logical flow of activities was described in the Petri Nets Graphs (see paragraph 3.3). When analysing the logical flow of activities, and so the data used in a Petri Nets Graph, is mandatory to comprehend the group of tasks, the number and kind of operators in charge of the tasks and the sequentiality of the tasks.
   The groups of tasks identified in our case were: truck arrival, picking for production and transportation (the thermal product has the same operators in charge of the transportation of materials), production of the different kinds of products, picking for sale, packaging and truck departure.

2. Layout of the plant and data about material handling:

   Two different aspects fall inside this category: where is the real-world physical location in which Places and Transition of the just found Petri Nets Graphs take place (i.e. real location of the facilities in the plant, the plant's constraints and so on) and how these facilities are linked and connected (i.e. how the operators move between facilities, the travels paths, the travels lengths, the means of transport used, and so on).

   For the first category of data, it has been used the plan floor provided by the company. With that plan floor it was possible to see all the actual location of facilities (warehouse, picking area, packaging area, delivery area, productive areas, …) the plant constraints (i.e. walls and height of the roof). With these data it was possible to create new possible layouts and, also for the new layouts, it was possible to know in advance the travel paths and the relative lengths.

   For the second category of data, average data available on the internet have been used. The data of interest were two:

   - The average velocity of an average electric ride on a pallet jack (used by the transport operator, balcony operator, delivery operator): it resulted to be 12,5km/h.
   - The average velocity of an average narrow aisle forklift truck: it resulted to be 12,5km/h while moving in length and 0,5m/s while moving in height.

3. All the data concerning times:

   The company provided some Microsoft Excel's spreadsheets containing all the data about 10 months of: Truck arrivals, Picking for Production, Production, Picking for sale, Packaging, Truck departures.

   Truck arrivals: this spreadsheet contains info about the arrival of the trucks and the total number of units received. From this sheet has been derived the number of units per truck and the % of units stocked (97%) and the % of units sent to cross docking (3%).

   Picking for Production: this spreadsheet contains the ID of the picking for production and if the picking was Mono (the products needed in that picking were contained in just one pallet) or Multi (the products needed in that picking were contained in more than one pallet). With every ID were also encompassed the time of picking. From this sheet have been extrapolated the distribution of times for pickings for production of the Mono and Multi types and the % of occurrence of the two types of picking.

   Production: this spreadsheet is the most important one and also the one with more data available. It contains data about production in all the nine productive areas kept into consideration for the methodology applied to originate new layouts (the nine groups of

productive areas are CLIP, EDF, IP55, Resistors, Tables, TCU, TRT, VFC, Lean Zone Filters. See *table 4* for more info). The spreadsheet is structured as a production order and for every of them is known: the productive area, quantity of the order, number of pallets to fill the request, time to work a unit of product, moment of the request of the products from a productive area. In order to manage it a pivot table was created; from that table was extracted the distribution of quantities required by each productive area (means and standard deviations) and the distribution of times required for production in each productive area (means and standard deviations).

Picking for sale: as for the spreadsheet of picking for production this spreadsheet contains the ID of the picking for sale and if the picking was Mono (the products needed in that picking were contained in just one pallet) or Multi (the products needed in that picking were contained in more than one pallet). With every ID were also encompassed the time of picking. From this sheet have been extrapolated the distribution of times for pickings for sale of the Mono and Multi types and the % of occurrence of the two types of picking.

Packaging: this spreadsheet contains the seconds for each packaging. There is also the time in which the packaging has started. This spreadsheet will be commented in the next paragraph.

Truck departures: this file contains the number of boxes/packages/pallets for every departing truck. There is also the time in which the order was created and the time in which it was concluded. This spreadsheet will be commented in the next paragraph.

Note: in every spreadsheet there were some incongruous data and so it was necessary to apply data cleaning techniques in order to have reasonable data. These data cleaning techniques cannot assure the final quality of the data.

Moreover, it is important to underline that some relevant data were not available and have determined some modelling choices/assumptions illustrated in the following paragraph.

The unavailable data are:

- Data about the percentage of production picking order evasible from the floor picking (without recurring to narrow aisle warehouse).
- Data about typologies of raw materials inside the arriving trucks.
- Data about typologies of products delivered with a specific truck
- Data about which typologies of products are packaged in a specific packaging order

The absence of this data obliges to reduce the complexity of the model.

4.3 SIMPLIFICATION AND CHOICES OF THE SIMULATION'S MODEL

In this paragraph will be presented the simplification and the choice made before translating the real-world system into the simulation model and the reasons supporting these choices.

Choice 1.

It has been chosen to not simulate the IK, presses, screens and mosquitos' nets production process. This is due to the already explained fact (see *assumption 2 paragraph 3.5.2*) that this kind of products exploit different kinds of resources and due to the low logistic effort to sustain that kind of operations, it would have just complicated the model without adding value.

Choice 2.

The second choice is a simulation choice. In order to simulate picking for production and picking for sale it has been chosen to use a probabilistic function in order to decide if the picking would have required to visit more pallet locations or just one (Mono vs Multi). The data about the percentage of Mono/Multi picking were extracted by the Microsoft Excel's spreadsheets. In these spreadsheets were also present for every order of production if it was a Mono or Multi picking. If the model was structured to know for every order if it is Mono or Multi it would have become a more accurate model for what concerns past events but would have presented complications while simulating future possible events. Due to the fact that a model should be able to simulate also future scenarios, the model has been set with probabilistic and not deterministic criterion, reducing the precision on simulating past events but increasing the precision while simulating future possible events.

Choice 3.

One unsolvable problem is the impossibility to have clean data about packaging times and truck departures from the spreadsheets. For what concern packaging, it has been chosen to select a reasonable time distribution from the dataset of all the times available (an estimation of 36% of data available revealed to be inconsistent and for what concern the remaining 64% there is no guarantee about their validity); this will, under some terms, decrease the reliability of the model. For what concern the truck, a randomly chosen number of pallets were chosen as the number of pallets to complete a departing truck. Also this choice will influence the results.

Assumption 1.

The first assumption that simplifies the model concerns the temporal division of the 7-meter-tall warehouse (narrow aisle warehouse) and the floor warehouse. It has been chosen to create a model in

which the products requested in a specific production order are already present in the floor warehouse and do not require to wait for the operators to take the products from the narrow aisle warehouse. This is obliged from the fact that is missing the data about the percentage of production picking order evasible from the floor picking. The model is so structured to start as soon as possible (when at least one picker is available) the picking for production because products required are considered always available and at the same time the narrow aisle warehouse will replenish the floor warehouse with products that will be required in the future production orders.

Assumption 2

Given the fact that there is missing data about the typology of arrived products, packaged products and delivered products, the model acts for granted that the same number of entities entering the system, will be processed, packaged and delivered. This simplifies the model in the sense that the information about what is entering, what is processed and what is leaving in a specific moment in time is lost.

4.4 INTRODUCTION TO ROCKWELL AUTOMATION - ARENA SIMULATION

Arena's Modules:

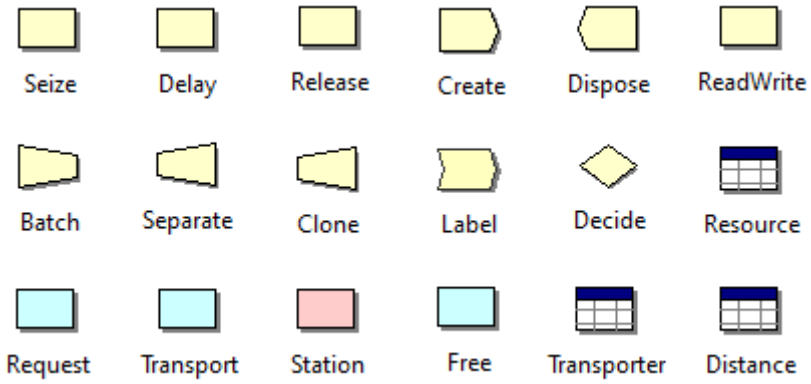

Figure 32. Arena's Used Module

Arena's modules can be used alone but are usually used in a specific order that can better module real world operations. For example, some of the most used sequence of modules are:

- Seize-Delay-Release used to model a machinery
- Create-ReadWrite used to create entities with attributes given as input from external files such as .CSV or .TXT files

- Station-Request-Transport-Station-Free used to model a transportation from one station to another
- Separate-…-Batch used in situation in which I have as input a group of entities and I want to process the entities singularly and then consider the entities again as a group

Here are briefly explained every module that will be used in the simulation model shown in the next chapter:

- Seize: it is used to model a queue. It has also as input the resource that will be occupied.
- Delay: it is used to model a time usage. It can model a wait or a service provision. It can be structured with deterministic time or stochastic time.
- Release: it releases the resource and the entity used in the Seize.
- Create: it creates entities. More Create modules can be present in a model. No modules can precede the Create module.
- Dispose: it receives and disposes the entities. It is used at the end of the life of the entity (truck departure in our case). No modules can follow the Dispose module.
- ReadWrite: it can both read from a file or write in a file. Every entity entering in this module represents a line of the file to read (or write). In each line more numbers/strings can be present: this module will practice a direct assignment between elements in the line and attributes of the entity.
- Batch: given a fixed or variable number *n*, this module group *n* entities into a single one
- Separate: given a group of *n* entities, this module separates the *n* entities.
- Clone: this module replicates its input. The entities passing through this module will proceed in the following modules and will also be replicated in another line starting with the Label module.
- Label: this module starts a cloned line.
- Decide: this module chose in which branch the entity has to go. The branches can be 2 or more. The choosing criterion can be random based on a probability to choose a branch, or can be based on condition (for example “if Attribute 1 = 0 choose branch n°3”)
- Resource: resources are all the factors needed when providing a service to an entity. Resources can be machineries, tools, or operators. Resources can be single or multiple based on the real number of resources present in the company. Resources can be used by more Seize. The simulator will keep into consideration when a resource is occupied and when is free to be used and doing so will act by consequence.
- Request: this module represent a queue of entities that are waiting to be transported

- Transport: this module uses a transporter to transport the entities waiting in the Request module from the departure Station to the arrival Station. Each Transport module is associated with a transporter.
- Station: is the representation of a physical place. It can receive entities transported from another Station module and can transport entities from it to another Station module.
- Free: this module frees the transporter occupied for the transportation. Now the transporter is able to satisfy other Transport modules.
- Transporter: is the operator able to handle the entities from one Station module to another ore. It is characterized by a velocity. Each Transporter can be associated with more Transport modules.
- Distance: here are described the distances between Station modules. The simulator knows the relative distances between each Station; the simulator is so able, knowing also the velocity of the Transporter, to calculate the time to transport entities from one Station module to another.

4.5 THE PROPOSED SIMULATION MODEL

The aim of the previous paragraphs of chapter 4 was to put the reader in the condition to be able to deeply comprehend the proposed simulation model that will now be presented.

In order to make the model easier to read, it has been decided to split the model into the 5 main areas (explained in chronological order):

- Truck arrival and assignment of goods between cross docking and warehouse
- Picking for production and unloading of raw materials from warehouse
- Reception of raw materials, production and withdrawal of finished products
- Picking for sale and packaging
- Truck loading and truck departure

4.5.1 Truck Arrival and Assignment of Goods between Cross Docking and Warehouse

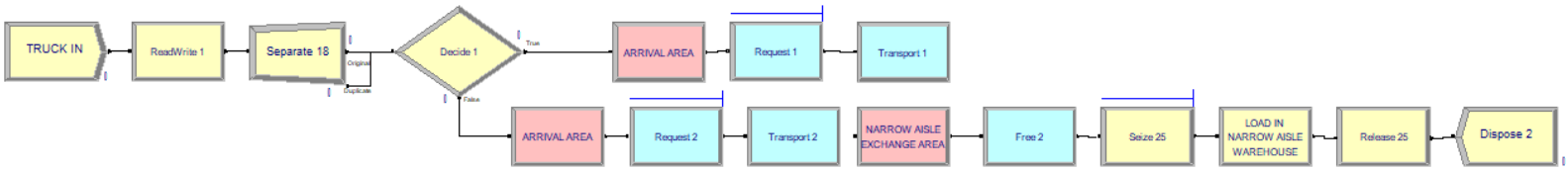

Figure 33. Truck Arrival and Assignment of Goods between Cross Docking and Warehouse

In a logical order, in *figure 33* is shown the first phase of the model. A Create module ("*truck in"*) originates entities following a normal distribution. The values of that distribution are derived from the Microsoft's Excel spreadsheets provided by the company (from now on we will refer to this file only as "spreadsheets"). A ReadWrite module will transform every truck arrived into a number of pallets which will be later split to cross docking or to narrow aisle warehouse according to a probability factor derived by the spreadsheets.

Pallets assigned to cross docking will be transported to the delivery area by the delivery operator (see *paragraph 4.5.5*), while pallets assigned to warehouse will be transported to the narrow aisle exchange area by the two 'balcony' operators and consequently will be loaded in the warehouse.

4.5.2 Picking for Production and Unloading of Raw Materials from Warehouse

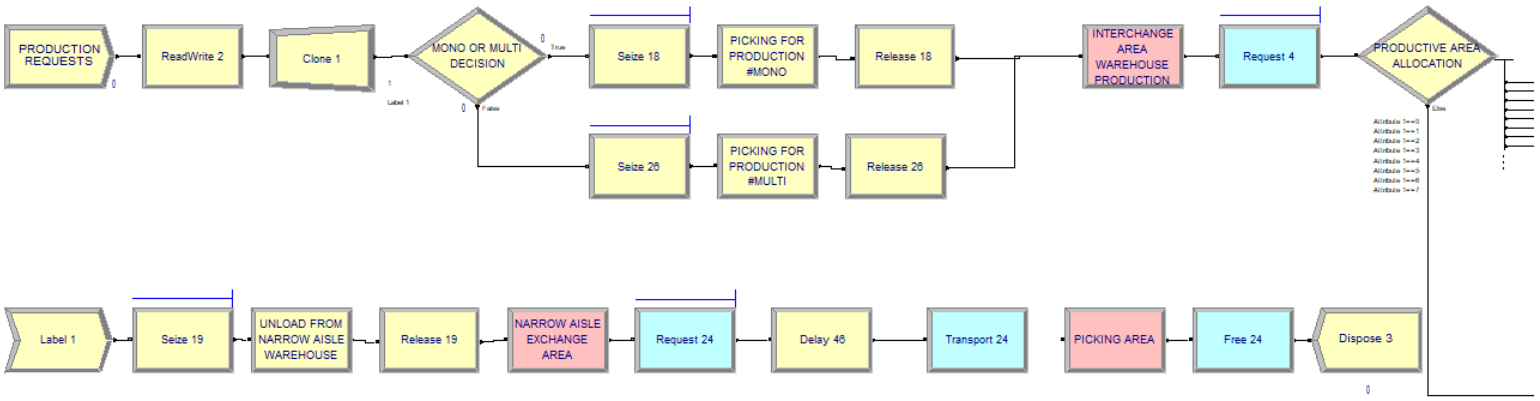

Figure 34. Picking for Production and Unloading of Raw Materials from Warehouse

A Create module generates entities following a normal distribution obtained by the spreadsheets. A ReadWrite file assigns to these entities two attributes: “production area”, the area in which they will be processed, and the “batch size”, that is the number of units of raw materials requested with that particular order. The file associated with the ReadWrite file is the real production order list of the first 10 months of 2020.

When a production request is sent from one of the nine productive area to the logistic operators two things happens:

- In the upper branch are carried the operations for the picking for production inside the floor picking warehouse and the order is placed in the interchange area between warehouse and production, making possible for the transport operator to transport raw materials to the productive areas requiring them.
- In the lower branch are carried the operations needed to unload a pallet from the narrow aisle warehouse and place it in the floor picking warehouse.

The way this model works encompasses the assumption made before (see *assumption 1 paragraph 4.3*). The assumption is that products required in the production order are already present in the floor warehouse and do not require to wait for the operators to take the products from the narrow aisle warehouse; furthermore, the model act as if the warehouse knows in advance the future order so it can replenish the floor picking with the right advance.

After the picking for production (can be Mono or Multi thanks to a probabilistic criterion) the request of transport will be split into the nine productive areas. The decision criterion is based on a condition associated with the attribute “production area” linked to each entity.

4.5.3 Reception of Raw Materials, Production and Withdrawal of Finished Products

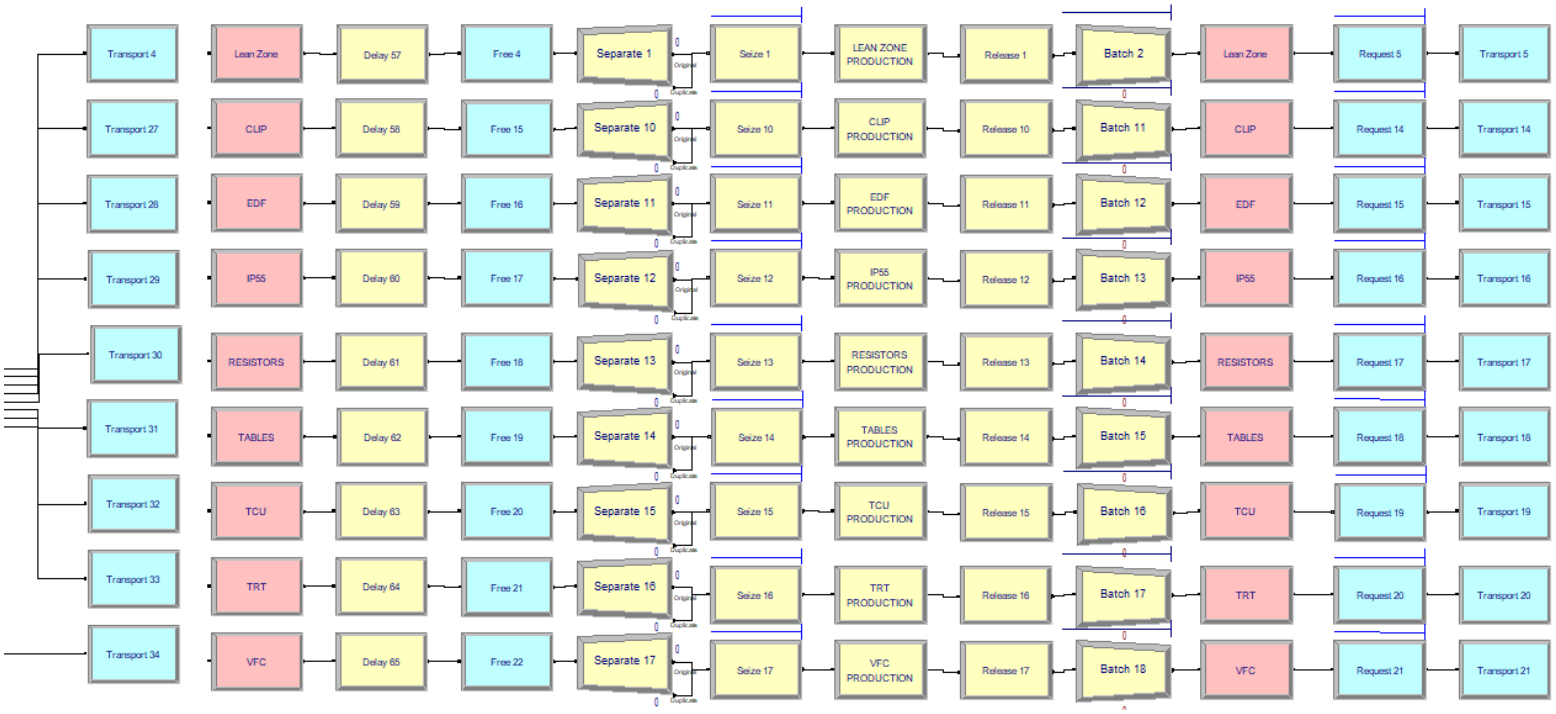

Figure 35. Reception of Raw Materials, Production and Withdrawal of Finished Products

Every branch shown in *figure 35* models a different productive area (areas are CLIP, EDF, IP55, Resistors, Tables, TCU, TRT, VFC, Lean Zone Filters). The transport operator is in charge of bringing raw materials to every station. The group of modules Separate-Size-Delay-Release-Batch is used to model the working process inside a productive area:

- Separate module: given as input an entity it creates a number of entities equal to the "batch size" attribute associated to the entity in input.
- Seize module: here the resources are occupied in order to process every entities one by one. The number of resources for each productive area are given by the spreadsheets.
- Delay module: after a time $t \sim N(\mu, \sigma^2)$, with values of mean and standard deviation extracted from the spreadsheets, the single entity is free to go to the next module
- Release module: resources are released and entities can go to the next module
- Batch module: when a number of entities equal to the "batch size", attribute of the entity input in the separate module, reach this module entities are batched in a single entity.

When products are batched into a new entity, the latter can request the transportation to the transport operator.

4.5.4 Picking for Sale and Packaging

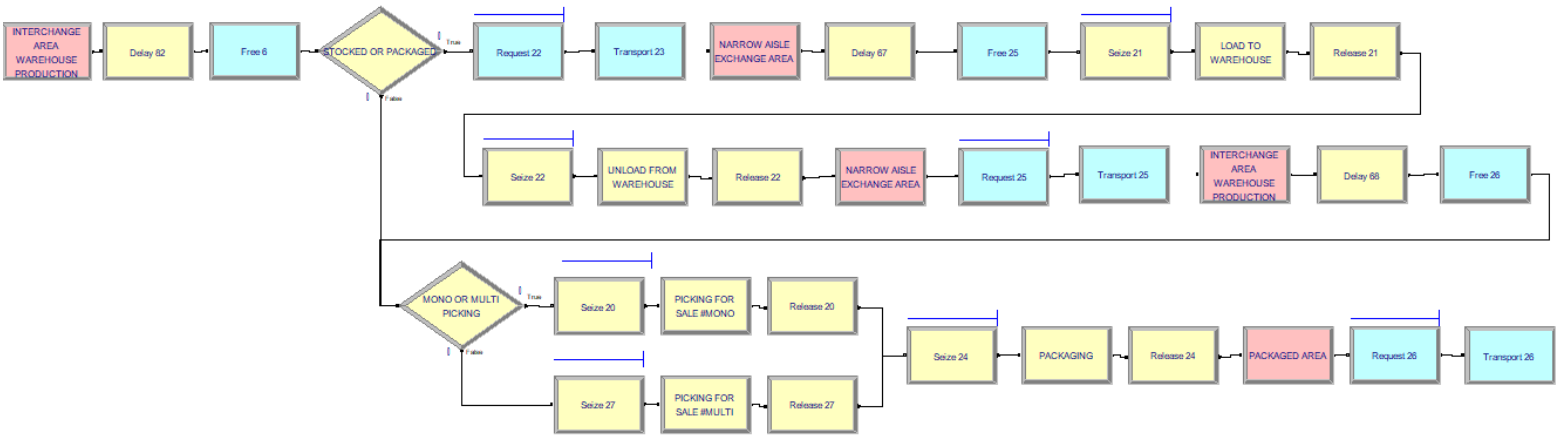

Figure 36. Picking for Sale and Packaging

The batch of products originated after the raw materials processing occurring in the productive areas are brought back to the interchange area between warehouse and production. Here two things can happen: the products stay in the floor picking warehouse if they will be picked soon for sale, or the products are brought to the narrow aisle warehouse.

The second branch of activities has been mapped sequentially as load and unload of the finished products in the narrow aisle warehouse; it is so required the assumption made on the asynchrony (see *assumption 2 paragraph 4.3*). A list of products packaged and delivered in every truck is not a given data. Thanks to this reason the model can work based on the fact that soon or later the finished products stocked in the warehouse will be unloaded and brought to the packaging area. The order in which products will be unload from the warehouse has lost its relevance.

When finished products are placed in the floor picking area, they can be picked (Mono or Multi) and brought to the packaging area (the time for transportation is encompassed in the distribution of the times of Mono and Multi picking for sale. The transporter is in fact the production picker and the distance between floor warehouse and packaging area is close to 0. For these two reasons the transportation has not been mapped).

When the products are packed are ready to request the transportation to the delivery area.

4.5.5 Truck Loading and Truck Departure

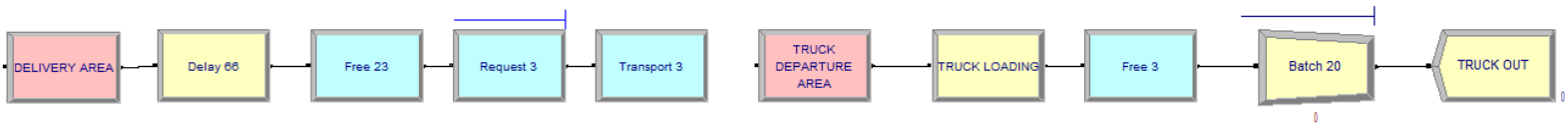

Figure 37. Truck Loading and Truck Departure

Entities are transported by the delivery operator from the packaging area to the delivery area. After that, packaged products are transported to truck departure area. Entities arriving at the truck departure area can arrive from the truck arrival area, if were entities arriving from a truck and assigned to cross docking, or from the delivery area if are finished products packaged. In both the cases entities can now be loaded on the truck. (operation made by the delivery operator). Once a truck is full (in order to module the truck fulfilment, a batch module has been used) it can leave the company (dispose module "truck out") and finally bring the finished products to the customers.

4.5 COMMENTS ON THE RESULTS OF THE SIMULATIONS

After the completion of the model, it is possible to run it to simulate the behaviour of the company. Three different simulations of the same model set with three different kinds of data were made: respectively, the model was run with the characteristics of Scenario As-Is, Scenario 4 and Scenario 2. Remember that Scenario 4 and Scenario 2 revealed to be the best layouts while analysing all the possible scenarios.

After the run of the simulation model set with the three different scenarios, the initial focus of the analysis switched on the availability (1 - average utilization) of the transport operator.
Scenario 4 sees a reduction of the average utilization of the transport operator of 31% in respect to Scenario As-Is.
Scenario 2 sees a reduction of the average utilization of the transport operator of 35% in respect to Scenario As-Is.
The reduction in both the cases is noteworthy.

After confirming the reduction of the average utilization of the transport operator, it is possible to look for other improvements. The focus must be on:

- Other resources/operators' average utilization

- Queues' length and average numbers of entities in queues

The comparison between Scenario As-Is and Scenario 4 / Scenario 2 has revealed that there is no evidence of a correlation between the reduction of the average utilization of the transport operator and the insignificant variation in the other resources/operators average utilization and queues length and average numbers of entities in queues. The variation of these measures between Scenario As-Is and Scenario 4 / Scenario 2 is attributable to the stochasticity of the model.

Some conclusions can be made. There is no evidence of a general quantitative benefit from a relayout, except for the average utilization for transport operators. The choice on which layout will be the best should depend only on the qualitative benefits that it is able to bring.

Even after this consideration SCENARIO 4 (see *figure 24*) is still the best possible scenario due to the fact that it remarkably reduces the average utilization of the transport operator and encompasses the best qualitative benefits that the company is searching for.

It should be said that the simplification made in order to create this model can not be overlooked. Also, this model is influenced by the GIGO rules (garbage in garbage out). The missing of some data may have influenced the results of the simulation. Due to the fact that the physical relayout of facilities will not start before one year from the drafting of this thesis, the company has time to get more precise data in order to create a simulation model that can assure the results of the future layout and can be also used for other What-If Analysis not concerning the relayout (for example the introduction of a new product or new resources, and so on).

5. PYTHON CODES TO SOLVE FLP

5.1 FIRST ALGORITHM: FLP WITH UNEQUAL AREA

The first python code proposed takes inspiration from the Grid System Unequal Area – Facility Layout Problem solved with Simulated Anealling proposed by Nima Moradi [54] and seen in the literature review. The algorithm takes inspiration just from how Nima Moradi has developed an algorithm to solve a GSUA-FLP and not from the concept of Simulated Anealling. In fact, the proposed algorithm does not search in the neighbourhood of the found solutions but at every iteration searches only for a new solution. Considering this aspect, the Boltzman coefficient was useless and even harmful and so has not been used.

The main concepts that have been exploited from the algorithm proposed by Nima Moradi are the number of iterations until the temperature is cold, and the random allocation of facilities to the grid.

The proposed python code has the objective to find a suboptimal solution, good enough to apply other more precise methods to find another solution even closer to the optimal solution.

The following algorithm is a greenfield design FLP (new layout from scratch with no constraints inside the plant) that works allocating randomly the facilities into the grid system. For each solution found the algorithm checks the feasibility of the solution. If it is a physically feasible layout, the algorithm calculates the total material handling cost, if it is not a feasible solution, the algorithm searches for another solution until a feasible one is found. After having obtained a new layout, its total cost is compared with the total cost of the actual optimal solution. If the new solution is better than the actual optimal one, the algorithm saves the new one as the optimal one, in the other case the algorithm goes to the next step. After a predetermined number of solutions found (better than the optimal or not) the algorithm stops and shows the optimal solution found (that is a suboptimal solution for the considered problem).

In the next pages will be presented in the following order:

- Python module: classofmachine (containing the class “machine” and the arrays)
- Python module: objective_function (containing the objective function)
- Python code: the proposed algorithm

Python Module: classofmachine

```
1. class machine:
2.
3.     def __init__(self, width, length, x, y, flow1, flow2, flow3, flow4, flow5):
4.         self.width = width
5.         self.length = length
6.         self.x = x
7.         self.y = y
8.         self.flow1 = flow1
9.         self.flow2 = flow2
10.        self.flow3 = flow3
11.        self.flow4 = flow4
12.        self.flow5 = flow5
13.
14. machine1 = machine(10, 10, 0, 0, 0, 50, 50, 0, 0)
15. machine2 = machine(10, 10, 0, 0, 0, 0, 0, 50, 0)
16. machine3 = machine(10, 10, 0, 0, 0, 0, 0, 50, 0)
17. machine4 = machine(10, 10, 0, 0, 0, 0, 0, 0, 100)
18. machine5 = machine(10, 10, 0, 0, 0, 0, 0, 0, 0)
19. machine_array = [machine1, machine2, machine3, machine4, machine5]
20.
21. machine_1 = machine(10, 10, 0, 0, 0, 50, 50, 0, 0)
22. machine_2 = machine(10, 10, 0, 0, 0, 0, 0, 50, 0)
23. machine_3 = machine(10, 10, 0, 0, 0, 0, 0, 50, 0)
24. machine_4 = machine(10, 10, 0, 0, 0, 0, 0, 0, 100)
25. machine_5 = machine(10, 10, 0, 0, 0, 0, 0, 0, 0)
26. machine_array_optimal = [machine_1, machine_2, machine_3, machine_4, machine_5]
```

A machinery (or facility) has been defined with a class composed of 4 + *n* variables; where *n* is equal to the number of machineries (or facilities) which constitute the plant. From now on the concepts of machinery and facility can be considered as interchangeable.

The first 2 variables are respectively: the width and the length of the facility. Thanks to this variable we can do unequal area problems. The algorithm is suitable also for equal area problems as they are a particular subcategory of the unequal area problems. In this example, width and length of every machine is set as 10, as it is possible to be seen in lines 14→18 and 21→25.

The second two variables are respectively: the x-position and the y-position of the machineries on the grid. These two variables are set to 0 and are the only two variables of the class that will be changed through the algorithm.

The last *n* variables of the *i-th* machinery are the value of the *i-th* row of the matrix of handling costs. The matrix can be obtained both deterministically or stochastically with a Microsoft Excel's spreadsheet. It depends on the user preference.

In the lines 14→19 are initialized 5 machineries and 1 array containing the 5 machineries. In the lines 21→26 there is a duplicate of the machineries and the array; the need to duplicate the machineries derives from some constraints on how python saves the variables in the memory of the computer.

Python Module: objective_function

```
1. def objective_function(machine_array):
2.     z = 0
3.     cost = [[0, 50, 50, 0, 0], [0, 0, 0, 50, 0], [0, 0, 0, 50, 0], [0, 0, 0, 0,
4. 100], [0, 0, 0, 0, 0]]
5.     for i in range(0, 5):
6.         for j in range(0, 5):
7.             z = z + pow(pow(machine_array[j].x - machine_array[i].x, 2) +
8. pow(machine_array[j].y - machine_array[i].y,2), 0.5) * cost[i][j]
9. return int(z)
```

Objective_function is a python function that needs as input an array in order to be able to run. The array in question is an array of machineries from the class "*machine*" (see line 19 and line 26 of *Python Module: classofmachine*).

The second information needed is the cost matrix. This matrix never changes with this algorithm.

With this kind of information, the function is able to return (line 9) the calculated Euclidean distance (lines 5→8) between all the couples of machineries. It has been chosen to use the Euclidean distance thanks to its popularity in the literature; also rectangular distance gives trustable results.

An assumption made with this Euclidean distance is that the flow departs and arrives at the centroid of the facility. Doing so, this Euclidean distance is calculated translating the centroid to the top left corner as it is the allocation point on the grid for each facility.

Python Code: Algorithm 1

```
1. from classofmachine import machine_array
2. from classofmachine import machine_array_optimal
3. from objective_function import objective_function
4. import random
5.
6. width_max = 40
7. length_max = 20
8. n_iteration = 100
9.
10. numberofmachines = len(machine_array)
11. solution_optimal = 0
12.
13. while solution_optimal == 0:
14.     i = 0
15.     for i in range(0, numberofmachines):
16.         machine_array_optimal[i].x = random.randint(0, width_max -
```

```
machine_array_optimal[i].width)
        machine_array_optimal[i].y = random.randint(0, length_max
machine_array_optimal[i].length)
    i = 0
    j = 0
    z = 0
    for i in range(0, numberofmachines):
        for j in range(0, numberofmachines):
            if max(machine_array_optimal[j].x, machine_array_optimal[i].x) ==
machine_array_optimal[j].x:
                wx = i
            else:
                wx = j
            if max(machine_array_optimal[j].y, machine_array_optimal[i].y) ==
machine_array_optimal[j].y:
                wy = i
            else:
                wy = j
            if j != i \
                    and (max(machine_array_optimal[j].x,
machine_array_optimal[i].x) - min(machine_array_optimal[j].x,
machine_array_optimal[i].x)) < machine_array_optimal[wx].width \
                    and (max(machine_array_optimal[j].y,
machine_array_optimal[i].y) - min(machine_array_optimal[j].y,
machine_array_optimal[i].y)) < machine_array_optimal[wy].length:
                z = z + 1
    if z == 0:
        solution_optimal = machine_array_optimal

n = 0
solution_new = 0

while n <= n_iteration:
    while solution_new == 0:
        i = 0
        for i in range(0, numberofmachines):
            machine_array[i].x = random.randint(0, width_max -
machine_array[i].width)
            machine_array[i].y = random.randint(0, length_max -
machine_array[i].length)
        i = 0
        j = 0
        z = 0
        for i in range(0, numberofmachines):
             for j in range(0, numberofmachines):
                if max(machine_array[j].x, machine_array[i].x) ==
machine_array[j].x:
                    wx = i
                else:
                    wx = j
                if max(machine_array[j].y, machine_array[i].y) ==
machine_array[j].y:
                    wy = i
                else:
                    wy = j
                if j!=i \
                        and (max(machine_array[j].x, machine_array[i].x) -
min(machine_array[j].x, machine_array[i].x)) < machine_array[wx].width \
                        and (max(machine_array[j].y, machine_array[i].y) -
```

```
76. min(machine_array[j].y, machine_array[i].y)) < machine_array[wy].length:
77.                     z = z + 1
78.         if z == 0:
79.             solution_new = machine_array
80.     deltaE = objective_function(solution_new) -
81. objective_function(solution_optimal)
82.     if deltaE < 0:
83.         w = 0
84.         for w in range(0, numberofmachines):
85.             solution_optimal[w].x = machine_array[w].x
86.             solution_optimal[w].y = machine_array[w].y
87.     solution_new = 0
88.     n = n + 1
89.
90. total_cost = objective_function(solution_optimal)
91.
92. print(solution_optimal[0].x, solution_optimal[0].y, "machine 1")
93. print(solution_optimal[1].x, solution_optimal[1].y, "machine 2")
94. print(solution_optimal[2].x, solution_optimal[2].y, "machine 3")
95. print(solution_optimal[3].x, solution_optimal[3].y, "machine 4")
96. print(solution_optimal[4].x, solution_optimal[4].y, "machine 5")
97. print(total_cost, "is total cost")
```

In the lines 1→5 are imported from the modules, the arrays of machines and the objective function explained above, it is also imported from the random library used to generate random numbers.

In the lines 6→8 are initialized the width and length of the plant and the number of iterations that the user wants for that specific run.

From line 13 to line 44 there is the first cycle of solution generation. The aim here is to originate an initial feasible solution. The scheme used in the lines 13→44 will be also re-proposed later inside lines 49→88; the focus will be on these latter.

In line 49 is initialized a while loop that will end when *n* will be equal to *max_iterations*. Inside this loop the algorithm will search in total *max_iteration* (integer number) of feasible solutions. The feasibility is requested from the algorithm to exit from the while condition in line 50.

From line 52 to line 56, thanks to the *random.randint*() function, the algorithm allocates on the grid all the machineries. The allocation is made by assigning randomly to the x-position and y-position (third and fourth property of an element belonging to the class "machine", see *Python Module: classofmachine*) an integer number greater than 0 but minor than, respectively, width max and length max of the plant.

From line 60 to line 77 the algorithm checks the feasibility of the solution. If the solution is feasible, in the lines 78 and 79 the newfound feasible solution is saved in the variable *solution_new* and is met the condition to exit from the while loop of line 50.

In the lines 60→77 the physical feasibility is checked through geometric conditions:

- In the lines 60 and 61 with two for loops is put the base to check every couple of machineries in the subsequent lines.
- In the lines 62→66 a new variable *wx* takes the value of the index of the machinery with the highest value of the x-position.
- In the lines 67→71 a new variable *wy* takes the value of the index of the machinery with the highest value of the y-position.
- In the lines 72→77 for every couple (i, j) of machineries with $i \neq j$ is checked the physical feasibility through the validity of one of the two formulas:

$$machine\ with\ highest\ \ x_{value} - machine\ with\ lowest\ x_{value} \geq width\ machine\ with\ lowest\ x_{value}$$

$$machine\ with\ highest\ \ y_{value} - machine\ with\ lowest\ y_{value} \geq length\ machine\ with\ highest\ y_{value}$$

This geometrical condition can be better comprehended through a visual representation viewable in *figure 38.*

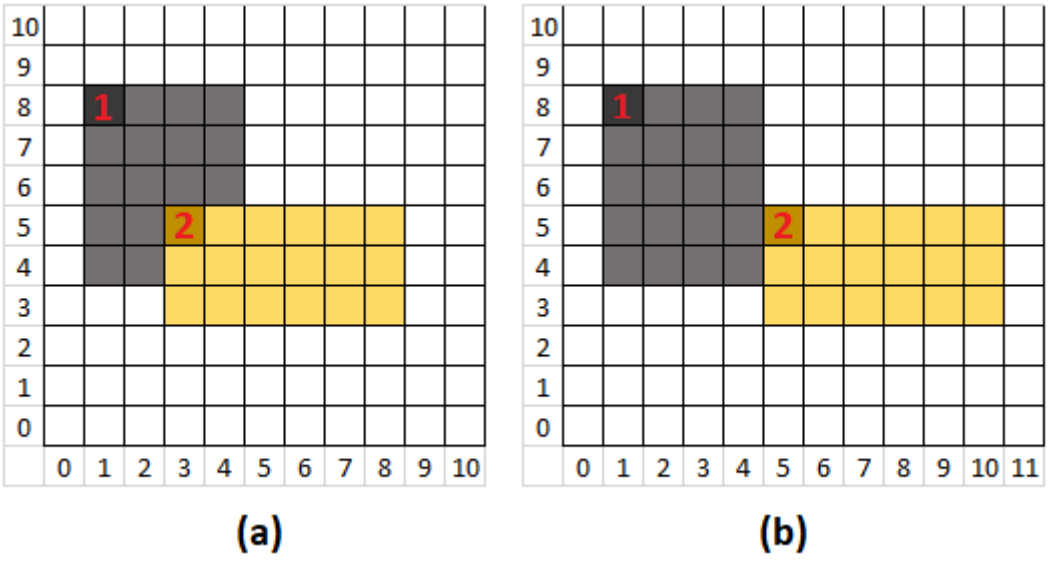

Figure 38. Geometrical Constraints in a Grid System

Note that the grid starts with position (0,0): this serves to align python results with reality. In python the first element of a list/set is in position 0 and not 1.

In *figure 38.a* the geometrical constraints are not respected, in fact, we have an overlap of facility 1 and facility 2, while in *figure 38.b* we do not have the overlapping. Through the formulas:

Figure 38.a:

x_{value} of facility 1 = 1, x_{value} of facility 2 = 3

y_{value} of facility 1 = 8, y_{value} of facility 2 = 5

width of facility with lowest x_{value} (facility 1) = 4

length of facility with highest y_{value} (facility 1) = 5

First formula become: $3 - 1 \geq 4$ → not satisfied

Second formula become: $8 - 5 \geq 5$ → not satisfied

→Overall: NOT FEASIBLE

Figure 38.b:

x_{value} of facility 1 = 1, x_{value} of facility 2 = 5

y_{value} of facility 1 = 8, y_{value} of facility 2 = 5

width of facility with lowest x_{value} (facility 1) = 4

length of facility with highest y_{value} (facility 1) = 5

First formula become: $5 - 1 \geq 4$ → satisfied

Second formula become: $8 - 5 \geq 5$ → not satisfied

→Overall: FEASIBLE

If 1 (or 2) condition is respected for every couple of machineries, the feasibility of the solution is assured.

From line 80 to line 86 if the objective function of the newfound solution is minor than the objective function of the actual optimal solution, the actual optimal solution is overwritten; if not, nothing happens.

When the iterations are over, the solution is then displayed through the lines of code 92→97. The solution displayed comprehends: the total cost of the layout found and the x-position and y-position of every machinery on the grid.

The algorithm was set with a simple set of data. The solution obtained with the algorithm is the following one:

```
22 0 machine 1
21 10 machine 2
11 0 machine 3
11 10 machine 4
1 8 machine 5
3072 is total cost
```

It is known that the total cost of the optimal solution is 3000. The algorithm in about 23 minutes with 100 iterations has obtained a result of 3072. Even if the result is not 3000, we can affirm that the solution found is almost optimal. As is possible to comprehend in *figure 39* the only reason for a not perfect result is given by a non-perfect alinement of facility 1 and facility 5.

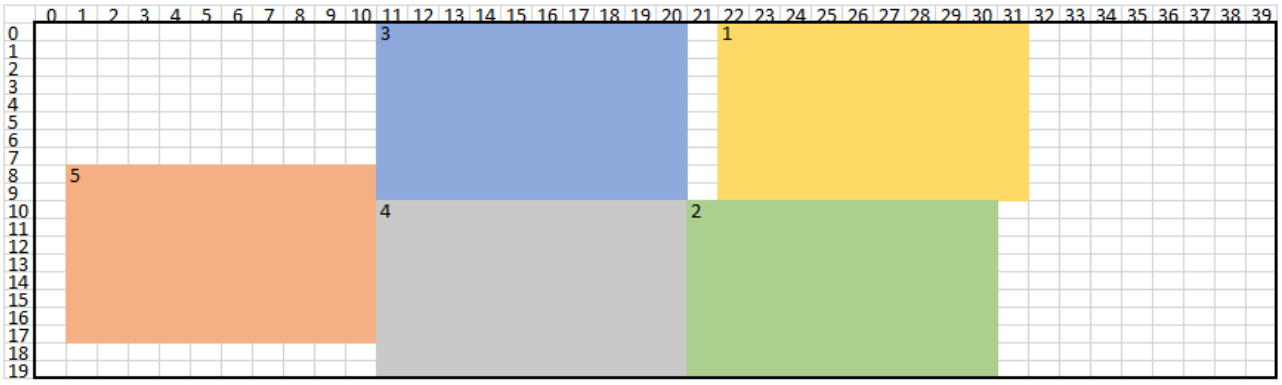

Figure 39. Graphical representation of the solution provided by the algorithm

As the reader can comprehend from the 23 minutes needed to generate 100 feasible solutions in this easy facility layout problem, the algorithm is effective but not efficient. The algorithm proposed has some pros and many cons.

Cons

- Before finding a feasible solution, the algorithm originates too many unfeasible solutions. The calculation time required results to be too time consuming.
- The computation time increases proportionally with the increase of the number of facilities.
- The computation time increases exponentially with the reduction of the percentage of unused plant floor. In our case the percentage of space used was $\frac{5\times(10\times10)}{20\times40} = 62{,}5\%$

Pros

- With a low number of facilities is almost assured the suboptimality of the solution found in a reasonable amount of iterations.
- There is not a limitation in the maximum number of facilities to be taken into consideration
- If the machines have equal areas it is easier to find the optimal solution thanks to the possibility to reduce the problem. Let us take as example the facility layout problem used as a case in the algorithm. Facilities have all the same area and shape (10×10) and the size of the plant is 40×20. We can easily reduce by a factor of 10 the input data and transform the same

problem to equal unitary area 1×1 and a plant dimension 4×2. The algorithm now can find, with this particular data, 1000 feasible solutions in about 1,5 seconds and the solution found is the optimal one (new total cost is 1/10 of the previous one and in fact the solution found has the total cost of 300). The next algorithm tries to optimize this kind of equal area problems.

5.2 SECOND ALGORITHM: FLP WITH EQUAL AREA

The second python code proposed aims at solving grid system equal area facility layout problems finding a suboptimal solution in an acceptable amount of time also for problems with a high number of departments.

The algorithm is created starting from the first algorithm proposed. The structure is the same: at first one solution is originated, then starts a loop of *n* iterations with *n* = *max_iterations* in which *n* feasible solutions are found and compared with the actual optimal solution and replace it in case of lower total handling cost.

Even if the structure is the same, the second algorithm is working in the opposite way of the first algorithm. While the first algorithm assigns to facilities a random location, the second algorithm assigns to locations a random facility.

The assumption made is that the shape of every facility is 1×1 or that the shape of the facilities of the FLP can be approximated to a 1×1 (as seen in the pros of the first algorithm).

Python Code: Algorithm 2

```
1. from classofmachinery import machine_array
2. from classofmachinery import machine_array_optimal
3. from objective_function import objective_function
4. import random
5.
6. width_max = 5
7. length_max = 7
8. n_iteration = 10000
9.
10. numberofmachines = len(machine_array)
11. solution_optimal = 0
12. allocated = []
13.
14. while solution_optimal == 0:
15.     i = 0
16.     j = 0
17.     for i in range(0, width_max):
18.         for j in range(0, length_max):
19.             k = 0
20.             while k == 0:
```

```
                a = random.randint(0, len(machine_array_optimal)-1)
                if allocated.count(a) == 0:
                    allocated.append(a)
                    k = 1
                    machine_array_optimal[a].x = i
                    machine_array_optimal[a].y = j
    solution_optimal = machine_array_optimal

print(objective_function(solution_optimal), "first solution")

n = 0
solution_new = 0

while n <= n_iteration:

    while solution_new == 0:
        allocated = []
        i = 0
        j = 0
        for i in range(0, width_max):
            for j in range(0, length_max):
                k = 0
                while k == 0:
                    a = random.randint(0, len(machine_array)-1)
                    if allocated.count(a) == 0:
                        allocated.append(a)
                        k = 1
                        machine_array[a].x = i
                        machine_array[a].y = j
            solution_new = machine_array

    deltaE = objective_function(solution_new) -
objective_function(solution_optimal)
    print(objective_function(solution_new), " new solution", n)
    if deltaE < 0:
        w = 0
        for w in range(0, numberofmachines):
            solution_optimal[w].x = machine_array[w].x
solution_optimal[w].y = machine_array[w].y

        print(deltaE, "is deltaE")
        print(objective_function(solution_optimal), " new optimal solution")
        print(n)
        print()
    solution_new = 0
    n = n + 1

print(solution_optimal[0].x, solution_optimal[0].y, "machine 1")
print(solution_optimal[1].x, solution_optimal[1].y, "machine 2")
print(solution_optimal[…].x, solution_optimal[…].y, "machine …")
...
print(solution_optimal[34].x, solution_optimal[34].y, "machine 35")

total_cost = objective_function(solution_optimal)
print(total_cost, "is total cost")
```

In the first lines are imported the arrays of elements belonging to the class “machine”. Now the array has a number of elements equal to:

$$|array| = width\_plant \times length\_plant$$

Creating an array of that number of machines will assure that to each location of the grid will be allocated one facility. For how the algorithm is structured it does not take into consideration the possibility that a solution of a FLP has empty space on the grid. If the facilities need to be inserted in the plant are not enough to cover all the plant's grid, the user has to give as input, in addition to all the facilities, a number of "null facilities" proper to fill the grid. For "null facilities" is intended facilities with material handling flow from/to other departments equal to 0. With the insertion of null facilities the objective function is not impacted.

The allocation of facilities to position happens in lines 17→27 (and again in 40→50). In these lines a double cycle for is set (the double cycle for is used to work with matrices; in fact we are representing the grid of the plant as it is a matrix): it allows to allocate to a position a randomly facility, starting from the position (0,0) until the position (width_max,0), and doing that for all the rows of the grid.

In order to check if a facility has already been allocated is exploited an empty list call "*allocated[]"*. A number in range (0, length(array)-1) is created. That number represents the index of the facility in the array. If the index is not present in the list *allocated* it is added to the list and then that number is used to allocate, at the position we are referring to in that moment of the for-loop, the facility corresponding to that number.

Having allocated to each position a different facility it is impossible to originate infeasible solutions. This mechanism results in an immense saving in required time to originate a feasible solution. The algorithm shown has been set and run with 35 different facilities linked to a handling cost matrix originated by Microsoft Excel's random number generator and added into the arrays and the objective function. With this matrix of costs:

- The algorithm has generated in just 92 seconds 10'000 feasible solutions and has chosen the best one within them.

For completeness it should be said that a FLP problem with 35 facilities to be allocated to 35 position has a total number of possible layouts of:

$$35! = 1{,}03 \times 10^{40}$$

The investigated solutions are an infinitesimal fraction of the total solutions.

The results of this algorithm are anyways satisfying. The solution found, even if is not the optimal one, can be considered as the perfect point to adapt neighbour searches in order to find a completely

satisfying solution. Furthermore the solution found is chosen between 10’000 solutions: statistically speaking this solution is still in the 0,001% of the best solutions.

5.3 THIRD ALGORITHM: FLP WITH EQUAL AREA AND NEIGHBOUR SEARCH

Python Code: Algorithm 3

```
from classofmachineries import machine_array
from classofmachineries import machine_array_1
…
from classofmachineries import machine_array_5
from objective_function import objective_function
from neighbour_searcher import neighbour_searcher
import random
import numpy as np

width_max = 5
length_max = 7
n_iteration = 10000
numberofmachines = len(machine_array)

in_solution1 = 0
allocated = []
while in_solution1 == 0:
    i = 0
    j = 0
    for i in range(0, width_max):
        for j in range(0, length_max):
            k = 0
            while k == 0:
                a = random.randint(0, len(machine_array_1)-1)
                if allocated.count(a) == 0:
                    allocated.append(a)
                    k = 1
                    machine_array_1[a].x = i
                    machine_array_1[a].y = j
    in_solution1 = machine_array_1

…

in_solution5 = 0
allocated = []
while in_solution5 == 0: …

array_solution = (in_solution1, in_solution2, in_solution3, in_solution4,
  in_solution5)
a = objective_function(in_solution1)
…
e = objective_function(in_solution5)
array_objfunction = [a, b, c, d, e]

z = np.argmin(array_objfunction)
if z == 0:
    solution1 = in_solution1
```

```
elif z == 1:
    solution1 = in_solution2
elif z == 2:
    solution1 = in_solution3
elif z == 3:
    solution1 = in_solution4
elif z == 4:
    solution1 = in_solution5
array_objfunction[z] = 10000000

print("initial solutions:")
print("SOLUTION 1, ", objective_function(solution1))
…
print("SOLUTION 5, ", objective_function(solution5))

n = 0
solution_new = 0
while n <= n_iteration:
    while solution_new == 0:
        allocated = []
        i = 0
        j = 0
        for i in range(0, width_max):
            for j in range(0, length_max):
                k = 0
                while k == 0:
                    a = random.randint(0, len(machine_array)-1)
                    if allocated.count(a) == 0:
                        allocated.append(a)
                        k = 1
                        machine_array[a].x = i
                        machine_array[a].y = j
            solution_new = machine_array
    if objective_function(solution_new) <= objective_function(solution1):
        for w in range(0, numberofmachines):
            solution5[w].x = solution4[w].x
            solution5[w].y = solution4[w].y
        for w in range(0, numberofmachines):
            solution4[w].x = solution3[w].x
            solution4[w].y = solution3[w].y
        for w in range(0, numberofmachines):
            solution3[w].x = solution2[w].x
            solution3[w].y = solution2[w].y
        for w in range(0, numberofmachines):
            solution2[w].x = solution1[w].x
            solution2[w].y = solution1[w].y
        for w in range(0, numberofmachines):
            solution1[w].x = machine_array[w].x
            solution1[w].y = machine_array[w].y
    elif objective_function(solution_new) <= objective_function(solution2):
    …
    elif objective_function(solution_new) <= objective_function(solution5):
    solution_new = 0
      n = n + 1

print("SOLUTION 1, ", objective_function(solution1))
…
print("SOLUTION 5, ", objective_function(solution5))

```

```
106.solution_1 = neighbour_searcher(solution1)
107.…
108.solution_5 = neighbour_searcher(solution5)
109.
110.print("SOLUTION 1 with neighbour search: ", objective_function(solution_1))
111.for i in range(0, len(machine_array)-1):
112.    print(solution_1[i].x, solution_1[i].y, "machine", i)
113.…
114.print("SOLUTION 5 with neighbour search: ", objective_function(solution_5))
115.for i in range(0, len(machine_array) - 1):
116.    print(solution_5[i].x, solution_5[i].y, "machine", i)
```

The algorithm shown above is the third algorithm proposed. It can be seen as an upgrade of the second algorithm (paragraph 5.2, from now on just called algorithm 2). It solves equal area facility layout problems. In addition to algorithm 2, it takes into consideration not just the best solution found but the first 5 better solutions. After having found the 5 best solutions resulting from *n* predetermined iterations, the algorithm applies a local neighbour search operator that, exchanging location of facilities of the initial layout, searches for eventually other better solution in the neighbourhood of the initial suboptimal solution.

The scope of applying the operator to 5 different solutions is to search more local optima, in fact is not assured that the best solution found will still be the best after the optimization carried by the operator.

A dutiful note should be made before explaining the algorithm. The difficulty to manage arrays composed by arrays of class of elements in python obliged the decision to avoid this kind of coding and so the resulting code does not exploit some shortcuts resulting in a long and repetitive code. In order to reduce the repetitiveness, given by the management of 5 different solutions, the reader will see some dots while reading the code: it simply means that are cut some lines repetition applied on different solutions. The code shown is 60% shorter than the original one.

In line 6 is imported the function of the neighbour search operator. It will be presented later

In line 8 is imported the library Numpy that is a library used to manage arrays and matrices.

From line 15 to line 36 are initialized 5 different solutions. While in algorithm 2 there were just one array of machineries called *solution_optimal*, now are managed 5 arrays that will represent the best 5 solutions at the end of the algorithm. Solutions are originated with the same methodology of algorithm 2.

In the lines 38→55 the 5 just initialized solutions are ordered from the best (solution1) to the worst (solution5).

Lines from 63 to 79 are the same algorithm seen in algorithm 2. These lines define randomly *n* solutions where *n* is the number of iterations chosen by the designer.

For every solution found in the lines 63→79, the algorithm, in the lines from 80 to 98, chooses, with an if-elif condition block, if the new solution is better than the actual optimal 5 solutions. For example, if the new solution is worse than the second best but better than the third best: the new solution becomes the third solution, the third solution becomes the fourth solution, the fourth becomes the fifth and the fifth best solution is discarded.

After having found all the *n* solution and chosen the best 5, the algorithm in lines 106→108 calls the *neighbour_searcher* function. Given as input an array of machineries, the function returns another array of machineries modified by the function.

Python Module: neighbour_searcher

```
from objective_function import objective_function
from classofmachineries import machine_array

def neighbour_searcher(solution_i):
    for w in range(0,len(solution_i)-1):
        machine_array[w].x = solution_i[w].x
        machine_array[w].y = solution_i[w].y

    k = 0
    while k <= (len(solution_i)-1):
        w = k + 1
        while w <= (len(solution_i)-1):
            machine_array[k].x = solution_i[w].x
            machine_array[w].x = solution_i[k].x
            machine_array[k].y = solution_i[w].y
            machine_array[w].y = solution_i[k].y
            if objective_function(machine_array) >= objective_function(solution_i):
                machine_array[k].x = solution_i[k].x
                machine_array[w].x = solution_i[w].x
                machine_array[k].y = solution_i[k].y
                machine_array[w].y = solution_i[w].y
            else:
                solution_i[k].x = machine_array[k].x
                solution_i[w].x = machine_array[w].x
                solution_i[k].y = machine_array[k].y
                solution_i[w].y = machine_array[w].y
            w = w + 1
        k = k + 1
    return (solution_i)
```

Differently from a CRAFT methodology, here has been chosen to let a facility to be exchanged with another one even if they are not close together.

After having copied the inserted solution into a new array (*machine_array*) a while loop starts. The operator is made exchanging two facilities at a time. The algorithm starts with facility 0 and looks if exchanging it with facility 1 the objective function will be better or worse. If the solution originated is better than the actual solution, the solution inserted in the function is updated with the newfound solution. The algorithm continues evaluating the change between facility 0 and all the other ones, then makes the same for facility 1, then facility 2 and so on until all the pairwise exchanges are considered.

There is another difference with CRAFT methodology. In CRAFT, when there is the interest to change location to facility *i* all the possible exchanges are evaluated and only later is chosen the best exchange possible. This reasoning is given by the Heuristic approach according to which choosing the best solution at every iteration will probably lead to the best solution. The choice of not following this path is that this algorithm is able to deal with a high number of facilities for which there is no a-priori information and when applying a metaheuristic algorithm is good to have a randomness variable in order to avoid to be stucked into local optimal solutions. The randomness is given by confirming the first solution that reduces the objective function and continuing from that solution to search eventually for another better solution.

This operator could potentially be applied repeatedly until no new solutions are found.

Back to algorithm 3, in lines 57→60, 102→104, 110→116 the algorithm prints the 5 solutions found; respectively it prints out the first 5 found, the best 5 found after the *n* iterations, the best 5 after the application of the neighbour search operator. The algorithm has been initialized with the same matrix of costs given as input for algorithm 2 (originated through the random number generator of Microsoft Excel). Respectively the solutions found are:

```
Initial solutions:
SOLUTION 1,  8004
SOLUTION 2,  8073
SOLUTION 3,  8111
SOLUTION 4,  8208
SOLUTION 5,  8398

SOLUTION 1,  7732
SOLUTION 2,  7766
SOLUTION 3,  7772
SOLUTION 4,  7776
SOLUTION 5,  7783
```

```
SOLUTION 1 with neighbour search:  7170
SOLUTION 2 with neighbour search:  7318
SOLUTION 3 with neighbour search:  7271
SOLUTION 4 with neighbour search:  7275
SOLUTION 5 with neighbour search:  7783
```

For simplicity are not shown the positions of the 35 facilities characterizing each solution.

Some information to be highlighted on the solutions obtained:

- The average value of the objective function of the initial solutions is 8159
- The average of the objective function of the best 5 solutions (out of 10000 solutions originated) is 7766.
- The average of the objective function of the solutions after the application of the neighbour search operator is 7263. The improvement on the average objective function value has been of 6,5%.
- The best solution found has a value of 7170. With the same data the algorithm 2 has found an optimal solution, a solution with objective function value of 7720. The net decrease is of 7,3%
- The algorithm spent 339 seconds to find 10’000 solutions. Algorithm 2 spent 92 seconds to find 10’00 solutions. In order to obtain a reduction of -7,3%, the time need by the algorithm has increased by +270%
- The reason why we considered the 5 best solutions is that it is not assured a priori that after the local search operator the first solution would remain the best one. In fact, is possible to notice that solution 3 and 4 at the end present better results than solution 2.

CONCLUSIONS

During the thesis drafting, it has been deeply explained what a Facility Layout Problem is, the reasons why this kind of problem is impacting in factory's everyday operations, the methodologies developed in the past years in order to solve FLPs and the upcoming methods to approach this kind of problems.

After that, in order to explain the concept of discrete event simulation, it was deeply analysed what a discrete system is, what Petri Nets are, what a simulation process is and finally all the knowledge to comprehend the concept of discrete event simulation was available.

All this theoretical introduction served to be able to approach in the best possible way a Relayout Facility Problem that a real-world company has been facing during the months of the drafting of this thesis. In order to solve the company's problem concerning the deep comprehension of which is the best possible future layout, a whole process was shown in this thesis. The first phase of the process was the deep comprehension of the actual company's processes, productive areas, layout, operators, volumes, resources and more. Later on, the Petri Nets were drafted. With a complete vision on how the company operates it has been possible to originate new future possible layouts. Knowing which the needs of the company are and will be, the best scenario has been chosen through a hybrid approach to solve facility layout problem.

A good manner while solving a facility layout problem is to check its feasibility and if there are inefficiencies through a simulation model; for this reason, a simulation model has been developed. The model can simulate the company's logistic and productive operations under some assumptions. The results of the simulation run on the proposed model, remarked the fact that the scenario chosen with the hybrid methodology is the best layout for the company.

In order to make a simulation it is important to simulate everything, even if the focus of the simulation will be only on determined characteristics of the company. It is difficult to understand a priori what our variables of interest can influence. In this case, the results of the simulation did not highlight new unthought correlations inside the logistic and/or productive operations of the company, and this confirmed that the chosen scenario is, to all effects, the best one.

There is still room for improvement for what concerns the simulation model proposed. In fact, too many simplifications were made and so the quality of the simulation's output can be improved by increasing the accuracy of the model. Given the fact that the process of moving the areas inside the plant as suggested by the new chosen layout will not start before at least one year from the drafting

of this thesis, the company has time to gather more precise data in order to enrich the simulation model.

REFERENCES

[1] Heragu. S.S. (1997) Facrlrlres Desrgn, PWS Publishing Co., Boston, MA.

[2] M. Enea, G. Galante, E.J.J.o.I.M. Panascia, The facility layout problem approached using a fuzzy model and a genetic search, 16 (2005) 303-316.

[3] Shore, R. H., & Tompkins, J. A. (1980). Flexible facilities design. AIIE Transactions, 12(2), 200–205.

[4] Gupta, R. M. (1986). Flexibility in layouts:Asimulation approach. Material Flow, 3, 243–250.

[5] Bullington, S. F.,&Webster,D.B. (1987).Evaluating the flexibility of facilities layouts using estimated relayout costs. Proceedings of the IXth International Conference on Production Research, 2230–2236.

[6] Savsar, M. (1991). Flexible facility layout by simulation. Computers and Industrial Engineering, 20(1), 155–165.

[7] Rosenblatt, M. J., & Lee, H. L. (1987). A robustness approach to facilities design. International Journal of Production Research, 25(4), 479–486.

[8] Kouvelis, P., & Kiran, A. S. (1990). The plant layout problem in automated manufacturing systems. Annals of Operations Research, 26, 397–412.

[9] Kouvelis, P., & Kiran, A. S. (1991). Single and multiple period layout models for automated manufacturing systems. European Journal of Operational Research, 52, 300–314.

[10] Sadan Kulturel-Konak, Approaches to uncertainties in facility layout problems:Perspectives at the beginning of the 21st Century, J Intell Manuf (2007) 18:273–284

[11] Nicol, L. M., & Hollier, R. H. (1983). Plant layout in practice. Material Flow, 1(3), 177–188.

[12] Koopmans, T., & Beckmann, M. J. (1955). Assignment Problems and the Location of Economic Activities. Cowles Foundation Discussion Papers 4. Cowles Foundation for Research in Economics, Yale University.

[13] Urban, T. L. (1998). Solution procedures for the dynamic facility layout problem. Annals of Operations Research, 76, 323–342.

[14] Montreuil, Benoit. 1991. A modelling framework for integrating layout design and ow network design. Material Handling'90 . Springer, 95{115.

[15] Meller, Russell D, Kai-Yin Gau. 1996. The facility layout problem: recent and emerging trends and perspectives. Journal of manufacturing systems 15(5) 351{366.

[16] Sherali, Hanif D, Barbara MP Fraticelli, Russell D Meller. 2003. Enhanced model formulations for optimal facility layout. Operations Research 51(4) 629{644.

[17] Castillo, Ignacio, Joakim Westerlund, Stefan Emet, Tapio Westerlund. 2005. Optimization of block layout design problems with unequal areas: A comparison of milp and minlp optimization methods. Computers & Chemical Engineering 30(1) 54{69.

[18] Anjos, Miguel F, Anthony Vannelli. 2006. A new mathematical-programming framework for facility-layout design. INFORMS Journal on Computing 18(1) 111{118.

[19] Liu, Qi, Russell D Meller. 2007. A sequence-pair representation and mip-model-based heuristic for the facility layout problem with rectangular departments. IIE transactions 39(4) 377

[20] Afshin OroojlooyJadid, Mohammad Firouz, A Stochastic Unequal Area Facility Layout Problem

[21] Lacksonen, T. A., & Enscore, E. E. (1993). Quadratic assignment algorithms for the dynamic layout problem. International Journal of Production Research, 31(3), 503–517.

[22] Erel, E., Ghosh, J. B., & Simon, J. T. (2003). New heuristic for the dynamic layout problem. Journal of the Operational Research Society, 54(12), 1275–1282.

[23] Kochhar, J. S., & Heragu, S. S. (1999). Facility layout design in a changing environment. International Journal of Production Research, 37(11), 2429–2446.

[24] V.A. Deshpande, "Plant layout optimization using CRAFT and aldep methology," Researchgate publication, vol. 57, no. 0032-9924, pp. 32-42, 2016.

[25] Tate, David M, Alice E Smith. 1995. Unequal-area facility layout by genetic search. IIE transactions 27(4) 465{472.

[26] Tavakkoli-Moghaddain, R, E Shayan. 1998. Facilities layout design by genetic algorithms. Computers & industrial engineering 35(3) 527{530.

[27] Li, Heng, Peter ED Love. 2000. Genetic search for solving construction site-level unequal-area facility layout problems. Automation in Construction 9(2) 217{226.

[28] Logendran, R, T Kriausakul. 2006. A methodology for solving the unequal area facility layout problem using distance and shape-based measures. International journal of production research 44(7) 1243{1272.

[29] Scholz, Daniel, Anita Petrick, Wolfgang Domschke. 2009. Stats: a slicing tree and tabu search based heuristic for the unequal area facility layout problem. European Journal of Operational Research 197(1) 166{178.

[30] Komarudin, Kuan Yew Wong. 2010. Applying ant system for solving unequal area facility layout problems. European Journal of Operational Research 202(3) 730{746.

[31] Scholz, Daniel, Florian Jaehn, Andreas Junker. 2010. Extensions to stats for practical applications of the facility layout problem. European Journal of Operational Research 204(3) 463{472.

[32] Baykasoglu, Adil, Nabil NZ Gindy. 2001. A simulated annealing algorithm for dynamic layout problem. Computers & Operations Research 28(14) 1403{1426.

[33] Balakrishnan, Jaydeep, Chun Hung Cheng, Daniel G Conway, Chun Ming Lau. 2003. A hybrid genetic algorithm for the dynamic plant layout problem. International Journal of Production Economics 86(2) 107{120.

[34] Dunker, Thomas, Gunter Radons, Engelbert Westkamper. 2005. Combining evolutionary computation and dynamic programming for solving a dynamic facility layout problem. European Journal of Operational Research 165(1) 55{69.

[35] Baykasoglu, Adil, Turkay Dereli, Ibrahim Sabuncu. 2006. An ant colony algorithm for solving budget constrained and unconstrained dynamic facility layout problems. Omega 34(4) 385{396.

[36] McKendall, Alan R, Jin Shang. 2006. Hybrid ant systems for the dynamic facility layout problem. Computers & Operations Research 33(3) 790{803.

[37] Balakrishnan, Jaydeep, Chun Hung Cheng. 2006. A note on \a hybrid genetic algorithm for the dynamic plant layout problem". International Journal of Production Economics 103(1) 87{89.

[38] Kulturel-Konak, Sadan. 2007. Approaches to uncertainties in facility layout problems: Perspectives at the beginning of the 21st century. Journal of Intelligent Manufacturing 18(2) 273{284.

[39] Drira, Amine, Henri Pierreval, Sonia Hajri-Gabouj. 2007. Facility layout problems: A survey. Annual Reviews in Control 31(2) 255{267.

[40] Balakrishnan, Jaydeep, Chun Hung Cheng. 2009. The dynamic plant layout problem: Incorporating rolling horizons and forecast uncertainty. Omega 37(1) 165{177.

[41] McKendall, Alan R, Artak Hakobyan. 2010. Heuristics for the dynamic facility layout problem with unequal-area departments. European Journal of Operational Research 201(1) 171{182.

[42] Sahin, Ramazan, Kadir Ertogral, Orhan Turkbey. 2010. A simulated annealing heuristic for the dynamic layout problem with budget constraint. Computers & Industrial Engineering 59(2) 308{313.

[43] Pillai, V Madhusudanan, Irappa Basappa Hunagund, Krishna K Krishnan. 2011. Design of robust layout for dynamic plant layout problems. Computers & Industrial Engineering 61(3) 813{823.

[44] Xu, Jiuping, Zongmin Li. 2012. Multi-objective dynamic construction site layout planning in fuzzy random environment. Automation in Construction 27 155{169.

[45] Chen, Gary Yu-Hsin. 2013. A new data structure of solution representation in hybrid ant colony optimization for large dynamic facility layout problems. International Journal of Production Economics 142(2) 362{371.

[46] Mazinani, Mostafa, Mostafa Abedzadeh, Navid Mohebali. 2013. Dynamic facility layout problem based on flexible bay structure and solving by genetic algorithm. The International Journal of Advanced Manufacturing Technology 65(5-8) 929{943.

[47] Pourvaziri, Hani, B Naderi. 2014. A hybrid multi-population genetic algorithm for the dynamic facility layout problem. Applied Soft Computing 24 457{469.

[48] Ulutas, Berna, A Attila Islier. 2015. Dynamic facility layout problem in footwear industry. Journal of Manufacturing Systems 36(0) 55{61.

[49] Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13, 533–549.

[50] Glover, F., Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers, Boston/ Dordrecht/London

[51] Heragu, S.S., Kusiak, A., 1991. Efficient models for the facility layout problems. European Jorrnal of Operational Research (53), 1–13

[52] Rochat, Y., Taillard, E., 1995. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1, 147–167

[53] JohnMcCall, Genetic algorithms for modelling and optimisation, Journal of Computational and Applied Mathematics, Volume 184, Issue 1, 1 December 2005, Pages 205-222

[54] Nima Moradi, Stochastic Facility Layout Planning Problem: A Metaheuristic and Case Study

[55] Palekar, U. S., Batta, R., Bosch, R. M.,& Elhence, S. (1992). Modeling uncertainties in plant layout problems. European Journal of Operational Research, 63, 347–359.

[56] Hameed, A.S., Aboobaider, B.M., Mutar, M.L., Choon, N.H., A new hybrid approach based on discrete differential evolution algorithm to enhancement solutions of quadratic assignment problem, International Journal of Industrial Engineering Computations, 2020 11(1), pp. 51-72

[57] Christos G. Cassandras, Stéphane Lafortune, Introduction to Discrete Event Systems Second Edition, Springer 2008.