Development of a Methodology to Solve the Line Balancing Problem with Parallel Workstations.

By

Ana María González-Garcés

A thesis submitted in partial fulfillment of the requirements for the degree of
MASTER IN SCIENCE
In
Industrial Engineering
UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
2006

Approved by

_____	_____
María Irizarry, Ph.D. President, Graduate Committee	Date
_____	_____
Pedro Resto, Ph.D. Member, Graduate Committee	Date
_____	_____
Sonia Bartolomei Suarez, Ph.D. Member, Graduate Committee	Date
_____	_____
María Medina, ME. Representative of Graduate Studies	Date
_____	_____
Agustín Rullán, Ph.D. Chairperson of the Department	Date

ABSTRACT

Since 1950 researchers have proposed methodologies to find an optimal allocation of tasks to workstations in an assembly line. However, most of the developed models solve the simple line balancing problem. The main outcomes of this thesis work were: (1) the development of a methodology for the design of a cost-oriented Simulated Annealing-based heuristic for line balancing with parallel stations, stochastic times and mixed products, (2) the design of a Simulated Annealing-based heuristic named ANAMAR06, and (3) the design of a user friendly Matlab-based tool for execution of ANAMAR06.

Results from ANAMAR06, with deterministic processing times, were compared to those obtained from an optimization model. The average percentage between ANAMAR06 and the mixed integer linear model was 1.63%. The same comparison was done between results from a modified Amen's and two Gaithe's heuristic model. The proposed algorithm outperformed all three.

RESUMEN

Desde 1950 los investigadores han propuesto metodologías para realizar una óptima asignación de las tareas a los centros de trabajo de una línea de ensamblaje. No obstante, gran parte de los modelos desarrollados ofrecen solución al problema simple de balanceo de línea.

Las contribuciones mas relevantes presentadas en esta tesis son: (1) el desarrollo de una metodología para el diseño de un heuristico basado en la teoría de Recocido Simulado que permita solucionar el problema de balanceo de línea con estaciones en paralelo, tiempos de procesamiento estocásticos y producto mixto, (2) el diseño de un heurístico llamado ANAMAR06 el cual está basado en Recocido Simulado y (3) el diseño de una herramienta amigable al usuario en Matlab® para la ejecución de ANAMAR06.

Los resultados de ANAMAR06 con tiempos de procesamientos determinísticos fueron comparados con los obtenidos por un modelo de optimización y se obtuvo una diferencia promedio de 1.63% entre heurístico propuesto y programación entera mixta. De manera adicional, se comparó ANAMAR06 con una versión modificada del heurístico de Amen y dos variantes del modelo de Gaither. ANAMAR06 superó los tres heurísticos analizados.

ACKNOWLEDGEMENTS

I would like to thank my parents and my brother for their unconditional love and support. I am immensely thankful with my family, my friends and all the people who believe in me during these last years.

There are no words to express the gratitude to those who support me and encourage me during my studies. I am thankful to those who work with me, those who listened to me and those who laugh with me and laugh at me. They know who they are. They do not need to be mentioned.

I am in debt with Dr. María Irizarry for being a wonderful advisor. Without her continuous guide and support, and her eternal patience it would have been impossible to achieve this goal.

I want to thank my graduate committee, Dr. Sonia Bartolomei and Dr. Pedro Resto for teaching me and giving me direction during my graduate studies. I would like to thank all my professors, especially Dr. David González for sharing his invaluable expertise and knowledge. I also want to thank Dr. Mario Padrón and Joel Rivera for their technical support.

I am indebted with Edwin Garavito for his technical support in the development of the user interface.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

An assembly line consists of a sequence of stations performing repeatedly a specified group of tasks over product units. Each product unit spends approximately the same amount of time at each workstation in the line, based on a cycle time used to determine the allocation of activities along the line.

The name of assembly line has been applied to flow oriented production systems which are typically used in manufacturing high demand products. Assembly lines must guarantee continuous and economical production. Therefore, its design, installation and operation are considered by managers key pieces for the effectiveness of their business.

Assembly lines can be clustered according to the requirements and characteristics of the products. The first attribute to classify an assembly line is the number of products processed. If only one product is assembled and all models are identical, the line is called "single-model line". On the other hand, if several models are manufactured, the line can be classified as "mixed-model" or "multi-model". In mixed-model lines, set ups are usually negligible and the balancing problem depends on the processing times and the tasks required for the different products. In multi-model lines, units must be processed in batches due to significant set up times. Also, product sequencing becomes a major issue.

Assembly lines are also classified according to the layout. Usually, assembly lines are arranged as "serial lines" where workstations are placed one followed by the other in a straight line. In a "U-shaped assembly line" the end and the beginning of the line are at

close proximity. Therefore, an operator can operate at workstations located at the beginning of the line as well as workstations at the end of the line. This helps overcome the line balancing inflexibility problem of serial lines.

Typically, in both, serial and U-shape lines there is only one operator per workstation which limits the minimum cycle time to the maximum of the processing times for the tasks. In case being needed, parallel workstations are used to deal with long duration operations. For instances where parallel stations are allowed, the maximum workload is limited by a multiple of the cycle time. This multiple corresponds to the number of parallels in each workstation. Because of the flexibility given by this condition a better balancing can be achieved.

For any assembly line an important decision is the adequate arrangement of the line. The decision problem of optimally assigning tasks to workstations in order to guarantee continuous product flow is known as the assembly line balancing problem.

The main objective in line balancing is maximizing efficiency which could be understood as making the best use of resources such as time, capital and human talent. Many researchers have focused their efforts in solving the line balancing problem using integer programming models, heuristics methods and other procedures.

As manufacturing processes evolve demanding more flexible production systems, the line balancing problem becomes more complex. Characteristics such as cost factors, product mix, stochastic task times and parallel stations must be integrated to the models in order to create a representation closer to reality. However, research work that incorporates all

these concepts in one single effective methodology has not been found in the literature.

The main purpose of the research described in this document is to offer a solution for the line balancing problem of typical manufacturing systems via the application of a comprehensive heuristic procedure.

2. LITERATURE REVIEW

The literature review has been divided into two basic parts. First, previous research in the area of line balancing and second, previous research and applications of Simulated Annealing heuristic models.

2.1 Line Balancing Problem

Since the assembly line balancing problem (ALB) was first formulated by Helgeson in 1954 many solutions to the assembly line balancing problem have been proposed.

Most of the research widely known in assembly line balancing such as the studies made by Dar-El (Mansoor) [6] and Talbot, et al. [27] has been oriented to the analysis and solution of the simple assembly line balancing problem (SALBP). According to Becker and Scholl [5] classification, the SALBP has the following characteristics and assumptions:

- Mass-production of one homogeneous product
- Paced line with fixed cycle time
- Deterministic operation times
- No assignment restrictions besides the precedence constraints
- Serial line layout with m one-sided stations
- All stations equally equipped with respect to machines and workers

The traditional SALBP admits three variants in the objective function:

– SALBP-1 Problem type 1: The objective is to minimize the number of workstations; where the objective is to determine the minimum number of workstations given a cycle time. This cycle time is calculated taking into consideration the requirements of a production plan or the demand forecasts.

  This problem can be stated as follows: for a given cycle time *c*, each task has to be assigned to one station so that the number *m* of stations is minimized and no precedence constraint is violated.

  A simple theoretical lower bound on the minimal number of stations is given by the equation:

$$LB = \left| \frac{\sum_{i=1}^{n} t_i}{c} \right| \qquad \text{Equation 2.1}$$

  where;

  LB= lower bound for number of stations, and

  t_i= processing times for task *i.*

  The LB is the smallest integer that solves the SALBP-1 problem. Therefore, all solutions for this model must have a number of stations at least equal to the lower bound calculated.

– SALBP-2 Problem type 2: The objective is to minimize the cycle time. This problem is generated when the number of workstations or production employees is fixed and the minimum cycle time has to be calculated. Becker and Scholl [5] defined the

cycle time as maximum or average time available for each cycle. Therefore, the minimal cycle is equivalent to the maximum workload. Being the station time S_k the sum of operation times of all tasks assigned to station *k*, the solution to SALBP-2 is stated as follows:

$$Min\ c = Max\ S_k \qquad \text{Equation 2.2}$$

The loads S_k are obtained creating different SALBP-1 scenarios, thus the problem type 2 is solved by iteratively solving type 1 problems.

– SALBP-E Problem type E: It has the objective of maximizing efficiency: The efficiency can be calculated as the quotient between the sum of all the task processing times and the product $m \cdot c$, where *m* is the total number of workstations and *c* is the line cycle time. This problem is solved by searching in the interval $[c_{\min}, c_{\max}]$ or $[m_{\min}, m_{\max}]$ a solution maximizing line efficiency. The efficiency is calculated using the equation:

$$E = \frac{\sum_{i=1}^{n} t_i}{m \cdot c} \qquad \text{Equation 2.3}$$

For solving SALBP-1, a large number of exact and heuristic procedures are available. Several of them are evaluated and organized in the literature review made by Ghosh and Gagnon [9]. Some effective branch and bound procedures are proposed by Hoffmann [10] and Scholl and Klein [24]. Additionally, a set of different heuristic techniques classified as single-pass, composite, backtracking and optimal decision rules have been exhaustively analyzed and compared by Talbot et al. [27] using a computational experiment.

Solution procedures for SALBP-2 and SALBP-E are usually search methods which iteratively solve several SALBP-1 instances. A modification of SALOME, a bidirectional branch and bound is proposed by Klein and Scholl [24] to solve the SALBP-2. They determined a minimal cycle time and assigned tasks to the m workstations with station times not exceeding the minimal cycle time. This assignment is called feasible for the respective cycle. The problem is solved by iteratively checking for several trial cycle times whether or not a feasible assignment of all tasks to *m* stations exists.

The assumptions of the SALBP are very restrictive and the model obtained might not represent the industrial reality. An extensive term used to classify problems with characteristics different to those of the SALBP is the generalized assembly line balancing problem (GALBP). Parallel workstations, mixed product lines, stochastic processing times and U-layout assembly lines are examples of characteristics considered in the GALBP.

2.1.1 Parallel workstations

The traditional SALBP requires the assignment of each task to a single workstation. Consequently, the production rate is limited by the longest task time. Due to the indivisibility of tasks, the maximal task time t_{max} is a lower bound on the cycle time. If there are one or more tasks with task times greater than the desired cycle time, paralleling of stations can be used to resolve the conflict.

Essentially, when using a parallel station model two or more replicas of a workstation performing the same set of tasks are permitted. Therefore, the option of a significant

increment in additional fixed cost must be considered. Cost oriented models such as the model presented by Pinto et al. [19] propose the minimization of labor costs which consist of fixed costs for duplicating a station, regular wage costs and overtime costs.

Azkin and Zhou [4] formulated a mathematical model and a heuristic procedure to tackle the GALBP in a fast and accurate way. Askin and Zhou based their heuristic procedure on a mixed integer programming model which has the following objective function:

$$\min\left\{\sum_{k=1}^{K} y_k L + \sum_{k=1}^{K} y_k \sum_{l=1}^{L} z_{lk} A_l\right\} \qquad \text{Equation 2.4}$$

where decision variables are:

y_k =Number of stations in parallel at stage k,

x_{ijk}= Proportion of task j of model i assigned to stage k,

$$z_{lk} = \begin{Bmatrix} 1, \text{ if tooling equipment type l is required at stage k} \\ 0, \text{ otherwise} \end{Bmatrix},$$

L =Fixed cost per period to open a station (include cost of labor and overhead),

and,

A_l= Amortized unit cost for equipment/tooling type j.

This mathematical model is a nonlinear integer program difficult to solve using regular computers. Therefore, the authors proposed a heuristic procedure to solve the problem without demanding high computational time.

They considered two situations in which parallel stations should be allowed. First, the case where the processing time for task j is greater than the line cycle time, and second

when there is no task which can fit into the current station, but the station is closed with considerable idle time. The decision to create a parallel workstation is based on the comparison of incremental tooling cost and the penalty cost of unutilized station time. The penalty cost of unutilized station time ΔP_K is calculated as follows:

$$\Delta P_K = L \cdot y_k \cdot T_k^a \qquad \text{Equation 2.5}$$

where;

L = Fixed cost per period to open a station (include labor and overhead costs),

y_k = Number of stations in parallel at station k, and

T_k^a = Available time remaining per cycle for each workstation at stage k.

One of the most interesting contributions of Askin and Zhou is the recursive but simple procedure to generate parallel workstations through the comparison of costs. However, the major limitation of the technique is the generation of only one solution. There is not presented a dynamic search for better alternatives. The model is static with no replications which might lead to balancing away from the optimal point.

McMullen and Frazier [15] developed a Simulated Annealing model considering stochastic processing time and a simple mixed-model environment where it is allowed to generate a parallel station as long as its utilization increases. Also, an additional objective function oriented to the minimization of the smoothness index restricts inflated duplications.

The smoothness index s is computed as:

$$s = \sqrt{\sum_{k=1}^{m} (\dot{w}_k - w_k)^2} \qquad \text{Equation 2.6}$$

where;

m = total number of stations,

$\dot{w}_k$ = number of workers required in workstation k, and

w_k = integer-adjusted workers required in station k.

A computational experiment was conducted to analyze the performance of the model. The production performance measures of interest were average WIP level, average flow-time, system throughput, system utilization, on-time completion, average unit labor cost, and cycle time performance. The model showed excellent results for those cases where cycle time performance was the primary objective. However, when the main concern is minimizing the design cost, some of the traditional line balancing techniques provided better solutions than the proposed algorithm.

Vilarinho and Simaria [30] constructed a two-stage heuristic method with zoning restrictions and allowed the user to control the process of creating parallel stations by defining a limit of parallels or upper bound. The main objective for this model was to minimize the number of workstations given a cycle time and additionally maximize the balance between workstations.

The objective function proposed by the authors to solve the assembly line problem was stated as follows:

$$Min\,Z=\sum_{k=1}^{s}k\cdot x_{NK}+\frac{S'}{S'-1}\sum_{m=1}^{M}q_m\sum_{k=1}^{S'}\left(\frac{s_{km}}{\sum_{l=1}^{S'}s_{lm}}-\frac{1}{S'}\right)^2+\frac{M}{S'(M-1)}\sum_{K=1}^{S}\sum_{m=1}^{M}\left(\frac{q_m s_{km}}{S_k}-\frac{1}{M}\right)^2 \quad \text{Equation 2.7}$$

where decision variables are:

$$x_{ik}=\begin{Bmatrix}1, \text{if task } i \text{ is assigned to workstation } k\\ 0, \text{otherwise}\end{Bmatrix},$$

$$r_k=\begin{Bmatrix}1, \text{if task } k \text{ can be replicated}\\ 0, \text{otherwise}\end{Bmatrix},$$

s_{km} = idle time of station k due to model m,

k= number of workstations, k=1,...,S,

m= type of model, m=1,…,M,

N= total number of tasks,

D_m= forecast demand for model m,

q_m= proportion of the number of units of model m being assembled,

$q_m = D_m / \sum_{p=1}^{M} D_p$, and

S'= the actual number of workstations required to meet the demand in the assembly line (S'=K: x_{NK}=1).

The first term in the objective function minimizes the index of the workstation to which the last task is assigned, thus minimizing the number of workstations. The second term balances the workload between the workstations and the third term balances the workload within each workstation.

In the first stage of the algorithm, an initial solution is obtained using a version of the rank positional weight heuristic and the program begins to look for the solution that

minimizes the number of stations in the assembly line. In the second stage the workload between stations and within the stations are balanced. The heuristic was tested on a set of twenty problems and the efficiency of the balance was calculated. The proposed procedure reached the optimal solution in small-sized problems. In large-sized problems the efficiency was not less than 80%.

One of the most relevant contributions made by the authors is the addition of restrictions to the model in order to obtain a feasible solution when some tasks cannot be performed with others in the same workstation. However, a limitation related to the objective function was identified. In section 2.14 a description of why minimizing the number of workstations cannot be appropriate because it can lead to non optimal solutions for cost oriented environments is presented.

2.1.2 Mixed-Model Assembly Lines

In mixed-model production, products which differ from each other with respect to size, color, material, or equipment are manufactured on the same line. This scenario presents additional challenges since tasks, processing times and precedence constraints vary from model to model. Some of the assumptions made to deal with the problem are: (1) precedence constraints consistent from model to model, and (2) same line balance used for all models.

Several techniques have been proposed to tackle the mixed-model ALBP such as McMullen and Frazier [15] and Merengo [17]. The typical technique is based on calculating a weighted average for each task in the line, considering the contribution of

the model in the product-mix. A new precedence diagram is built by merging each model's precedence diagram into a single one. Weighted averages, $\overline{t}_x$, are calculated as follows:

$$\overline{t}_i = \frac{\sum_{m=1}^{M} t_{m,i} \cdot d_m}{\sum_{m=1}^{M} d_m} \qquad \text{Equation 2.8}$$

where;

i= 1,…,N,

N=total number of tasks in the line,

M= total number of models in the line,

m= model index,

$t_{m,i}$ = Processing time of task x for model n, and

d_m = Number of units of model n to be produced during the planning period

Askin and Zhou [4] constructed a composite task sequence (CTS) which has two basic properties: (1) it contains all required tasks for each model i, and (2) it contains all task precedence relations for each model i. Therefore, the precedence of each model can be identified as a subset of this general sequence.

An example of the CTS construction for products A, B y C is shown in Figure1.

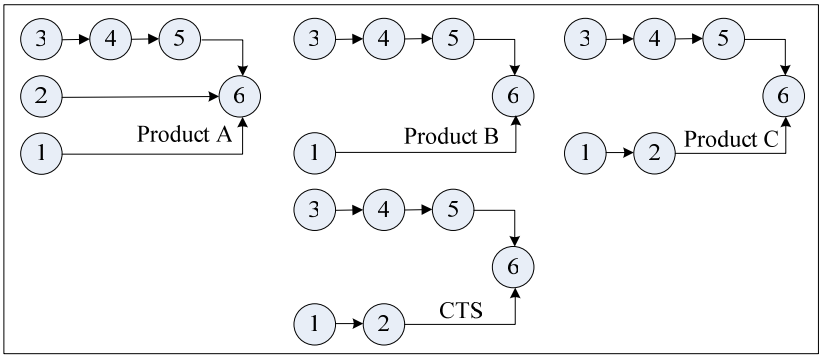

Figure 1 CTS Example

McMullen and Frazier calculated weighted average processing times and constructed the CTS to solve the mix-product assembly line problem as a single product assembly line balancing problem. The objective function for their model integrated three single objectives:

$$Min: E_1 = \sum_{j=1}^{r} (\dot{w}_j L + m_j Q) \qquad \text{Equation 2.9}$$

$$Min: E_2 = \sqrt{\sum_{j=1}^{r} (\dot{w}_j + w_j)^2} \qquad \text{Equation 2.10}$$

$$Min: E_3 = \prod_{j=1}^{r} p_j \qquad \text{Equation 2.11}$$

where;

r = total number of workstations,

$\dot{w}_k$ = number of workers required in workstation k,

w_k = integer-adjusted workers required in station k,

m_j = number of pieces of equipment required in work center j,

L = labor cost per worker in \$/year,

Q = equipment cost per piece in \$/year, and

p_j = probability of lateness in work center.

The first objective function is concerned with minimizing the sum of costs associated with both labor requirement and equipment requirement. The second objective intends to minimize the 'smoothness', it means that the work must be distributed into workstations as evenly as possible. Finally, the third objective function minimizes the probability of lateness across the workstations. This objective function is analyzed in the section 2.1.3.

2.1.3 Stochastic Task Times

Another assumption of SALBP is the deterministic nature of processing times. In highly manual production lines the variations in processing times from item to item is an inevitable event. Procedures including random task times have been stated by Arcus [3] and Suresh and Sahu [26]. Most of the procedures developed for the stochastic version are modified extensions of the procedures for deterministic models.

The majority of researchers in the revised literature assumes normally distributed task times and calculates the estimated standard deviations for the tasks by multiplying the expected task duration by a coefficient of variation (CV) term. Usually, different levels of CV are tested as in Erel and Sabuncuoglu [7].

A time-oriented objective of stochastic models is the minimization of the probability of exceeding the cycle time in any station such as in Reeve and Thomas [21]. This function, denominated the lateness function, can be obtained by multiplying the lateness measure of all workstations involved in the layout of interest, as done by McMullen and Frazier. They calculated the function of lateness as follows:

$$\text{lateness function} = \prod_{j=1}^{r} p_j \qquad \text{Equation 2.12}$$

The probability of lateness p_j is estimated by integrating the normal distribution function and r is the number of workstations.

The stochastic nature of the processing times generates a new risk related to the probability of incomplete jobs. Additional costs are incurred if the product is not completed in time equal or smaller than the cycle time. These incompletion costs are reduced by decreasing the station utilizations. This can be done by increasing the number of stations or the cycle time.

A cost oriented objective which quantifies the risk of incomplete jobs is presented in Sarin and Erel [22]. They developed a cost model for the single-model stochastic assembly line balancing problem with the objective of minimizing the total labor cost and the expected incompletion cost arising from tasks not completed within the given cycle time. The objective function for their model was:

$$Min\ Z = \text{Total labor cost+ Total incompletion cost} \qquad \text{Equation 2.13}$$

The problem of minimizing Z was solved by varying the number of stations K and the allocations of tasks for a given cycle time C. The objective function can be rewritten as follows:

$$Min\ Z = c * K * L + \sum_{i=1}^{N} \left\{ \beta_i \left[IC_i + \sum_{k \in Ai} IC_k \right] - \sum_{j=1}^{fSi} SB_i^j \right\} \qquad \text{Equation 2.14}$$

where;

c= cycle time,

K= number of stations on the line,

L= labor rate,

N = number of tasks,

β_i = probability that task i is not completed within c,

IC_i= incompletion cost of task i, for i= 1,…,N,

A_i= set of tasks following task i on the precedence diagram and in the station which contains i,

fS_i= the number of feasible starting events for task i, and

SB_i^j = Over-counted incompletion costs.

Some of the most relevant assumptions considered by Sarin and Erel in this model are:

- Task performance times are random variables. They are independent of each other and the parameters of the distributions are known.
- The tasks assigned to a station are performed in a given order.
- Incomplete tasks are completed off the line at a cost which is not dependent on the fraction of the task completed on the line.
- No blocking due to incomplete tasks

Sarin and Erel's research work presents an exhaustive deduction of probability for incomplete jobs based on normal probability distribution. However, their deductions can be used only for single product assembly lines. Also, parallel workstations are not allowed.

2.1.4 Cost Oriented Models

Usually, the main objective in the line balancing problem is minimizing the number of workstations for a given cycle time. This type of problem has been called the Time-Oriented Assembly Line Balancing. Since there is an understandable relationship between the number of workstations, the cycle time and the total cost it is evident that the ALBP could be directly stated as a cost minimization problem.

Amen [1] presents a backtracking procedure as an exact method to obtain a solution to the cost oriented ALBP. The objective function of the model is to minimize the total costs per product unit which is computed as the sum of labor and capital cost as follows:

$$Min\,TC = \sum_{m=1}^{M} ck_m^{sw} + Mk^{sc} \qquad \text{Equation 2.15}$$

where;

c = cycle time,

k_m^{sw} = wage rate of station m,

M = total number of stations, and

k^{sc} = cost of capital per station.

An effective station wage rate is defined and the total labor cost per product is calculated as the sum of the wage rates of each station multiplied by the cycle time. It is assumed that the cost of capital depends on the total line length and that all stations have the same dimensions. Amen also proves that the "maximally loaded station rule" used in time-oriented models is not adequate when the objective is to minimize the total cost per unit. This rule consists on assigning tasks to the stations as long as the cycle time is not exceeded.

In 2001, Amen [2] developed station-oriented priority rules and compared them to existing ones using a large set of randomly generated problem instances. Amen classified different heuristics as methods with random choice task assignment-Z, methods with one problem-oriented priority rule-P, methods with several problem-oriented priority rules-H, methods with exact solution of sliding problem windows-F and an exact method-E. He proposes a new *P* rule called "best change of idle cost" and the "exact solution of sliding problem windows" which according to a computer experiment performs better in the cost-oriented problem.

Amen's study is the basis for Scholl and Becker's [23] research work. They started with the exact procedure considering wages and capital costs proposed by Amen, and corrected one of the dominance rules used in the branch and bound model. The objective function proposed by Scholl and Becker minimizes the total cost per product unit which is given by the sum of the station wage rates w_{sk} (per time unit) multiplied by the fixed cycle time *c*. This objective function is given by the following equation:

$$Min\ TC = c \cdot \sum_{m=1}^{\overline{m}} ws_k \qquad \text{Equation 2.16}$$

where;

m = number of workstations, and

$\overline{m}$ =upper bound on m.

Although Amen's work is the most relevant study made in the cost oriented area, relevant complexities as stochastic task times, multiple product and parallel workstations were not considered in his models.

2.2 Simulated Annealing

The existence of a great amount and variety of difficult problems that need to be solved efficiently encouraged the development of high performance procedures to find good solutions to these large problems. A heuristic method is a procedure to solve an NP hard[1] problem through an intuitive approach. Although these methods do not provide the optimal solution to the problem they offer a good approach. Sometimes heuristics algorithms find the optimal solution in a brief period of time.

Simulated Annealing (SA) is a family of heuristic optimization methods, derived by a natural analogy with the statistical physics of random systems described by Kirkpatrick [13] in 1983. Kirkpatrick, applied simulation annealing to solve several problems occurring in the computer's industry but his work concentrates on the routing problem that arises in automatic wiring of integrated circuits and the statement and solution of the traveling salesman problem using the Simulated Annealing theory.

Simulated Annealing is a Monte Carlo technique which implements a global minimization algorithm that works for arbitrary functions. The algorithm is proposed as an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system.

---

[1] NP hard is the complexity class of decision problem that are intrinsically harder than those that can be solved in polynomial time.

SA was first described by Metropolis et al. [18]. Later on, Kirkpatrick [13] applied it to solve combinatorial problems. Simulated Annealing's major advantage over other methods is an ability to avoid becoming trapped at a local optimum. The algorithm employs a random search which not only accepts changes that decrease the objective function, but also some changes that increase it. The latter are accepted with a probability calculated according to certain rule.

In the SA algorithm an initial solution and a control parameter called temperature T are specified. This initial solution is set as the current solution. As the algorithm runs, this temperature is systematically decreased according to a cooling rate, and neighboring solutions to the current solution are found.

For each iteration, the value of the objective function is calculated. If the value is better to that of the current solution, the neighboring solution becomes the new current solution. On the other hand, if the neighboring solution provides an objective function value inferior to that of the current solution, the neighboring solution may still become the current solution according to certain acceptance probability. The acceptance probability p is computed according to the criterion established by Metropolis. Nicholas Metropolis proposed a modification to the simple Monte Carlo simulation to find the best state of a system. The Metropolis procedure establishes that a new point in the search space is sampled by making a slight change to the current point and new configuration points may be accepted although their costs exceed the costs obtained for the best solution. To accept these points Metropolis proposed a criterion based on probability and thermodynamics laws as follows:

$$p = exp\left(-\frac{\partial f}{T}\right)$$ Equation 2.17

where;

T = Temperature, and

∂f = change in the objective function.

A random number between zero and one is generated, if the random number is smaller than p the solution is accepted. This strategy prevents the algorithm from getting trap in a local optimum.

2.2.1 Design of the SA algorithm

The theory of Simulated Annealing is based on the following two principles:

A model of the process based on Markov chains

A Markov chain is a representation of a stochastic process showing the different states of the process with transition probabilities for moving from one state to another. In Simulated Annealing this measure corresponds to the probability that a new solution is generated and accepted. It has been proven by researchers that at each value of the temperature, the SA algorithm performs a number of iterations large enough for the state probability distribution to approach a stationary value. Research on Markov chains in SA leads to identify stationary (equilibrium) states for the process and calculate convergence speed for this meta-heuristic. The main objective of this analysis is to determine the minimum number of transitions required to obtain the optimum (or near to optimum) solution.

The Metropolis algorithm

The SA algorithm contains two nested loops. First, there is an outer loop which controls the decrease in temperature. Second, there is an inner loop where transitions from one state *i* to another state *j* is guided by the Metropolis algorithm.

In this algorithm state E_j is generated by a perturbation mechanism and accepted with the following probability:

$$q_{ij} = \begin{cases} 1, & E_j - E_i \le 0 \\ \exp\left(-\dfrac{f(s_j) - f(s_i)}{k_B \cdot T}\right) & E_j - E_i > 0 \end{cases} , \qquad \text{Equation 2.18}$$

In the process of solidification of materials, if the temperature is low enough, the solid can reach thermal equilibrium. The probability of being at a state *i* with energy E_i is given by the Boltzman distribution:

$$P_T(X = i) = \frac{\exp\left(\dfrac{-E_i}{k_B \cdot T}\right)}{\sum_j \exp\left(\dfrac{-E_j}{k_B \cdot T}\right)} , \qquad \text{Equation 2.19}$$

where; X represents the state with thermal equilibrium.

The Simulated Annealing algorithm can be modeled as a Markov chain with transition probability function given by:

$$\theta_{ij}(T) = \begin{cases} \dfrac{1}{|N(s_i)|}, & if\ f(s_j) \le f(s_i), s_j \in N(s_i) \\ \dfrac{1}{|N(s_i)|} \cdot \exp\left(-\dfrac{f(s_j) - f(s_i)}{T}\right), & if\ f(s_j) > f(s_i), s_j \in N(s_i) \\ 1 - \sum_{k, k \ne i} p_{ik} q_{ik}, & if\ i = j \end{cases} , \qquad \text{Equation 2.20}$$

where;

$\theta_{ij}(T) =$ probability of making a transition from state *i* to state *j*,

$N(s_i) =$ number of possible solutions in the neighborhood of state *i*,

$f(s_j) =$ objective function value at any state *j* in the neighborhood of state *i*,

$p_{ik} =$ the perturbation probability of making a transition from state *i* to state *k*, and

$q_{ik} =$ the probability of accepting the solution obtained at neighborhood point *k*.

When a perturbation was attempted unsuccessfully the state remains the same, therefore *i=j*. For any combination (*i,j*) the perturbation probability is calculated as follows:

$$p_{ij} = \begin{cases} \dfrac{1}{|N(s_i)|}, & if\ s_j \in N(s_i) \\ 0, & otherwise \end{cases} \quad , \qquad \text{Equation 2.21}$$

Additionally, the acceptance probability is calculated using the following expression:

$$q_{ij}(T) = \begin{cases} 1, & if\ f(s_j) \le f(s_i) \\ \exp\left(-\dfrac{f(s_j) - f(s_i)}{T}\right), & otherwise \end{cases} \quad , \qquad \text{Equation 2.22}$$

Under some circumstances of neighborhood structure the Markov chain can be considered as an "ergodic" process. An ergodic process has the property that in the long run it reaches a stationary distribution, irrespective of the initial state.

If $\pi_{Tk}(s_i)$ is the probability that s_i is the current solution after *k* steps of the algorithm at temperature *T*, the state probability vector can be described as:

$$\pi_{Tk} = \left(\pi_{Tk}(s_1), \ldots \pi_{Tk}(s_i), \ldots\right) \quad , \qquad \text{Equation 2.23}$$

For ergodic Markov chains, the state probability vector converges to a limiting probability vector:

$$\lim_{k\to\infty} \pi_{Tk} = \pi_T \quad , \qquad \text{Equation 2.24}$$

It can be proven that for the SA algorithm the state probability vector converges to:

$$\lim_{k\to\infty} \pi_{Tk}(s_i) = \frac{\exp\left(\frac{-f(s_i)}{T}\right)}{\sum_{sj\in S} e\exp\left(\frac{-f(s_j)}{T}\right)} \quad , \qquad \text{Equation 2.25}$$

Considering two states s_i and s_j with $f(s_i) < f(s_j)$ the ratio between the state probabilities can be expressed as:

$$\frac{\pi_{Tk}(s_i)}{\pi_{Tk}(s_j)} \xrightarrow{k\to} \frac{\exp\left(\frac{-f(s_i)}{T}\right)}{\exp\left(\frac{-f(s_j)}{T}\right)} \quad , \qquad \text{Equation 2.26}$$

$$\frac{\pi_{Tk}(s_i)}{\pi_{Tk}(s_j)} = \exp\left(\frac{f(s_j)-f(s_i)}{T}\right) \xrightarrow{T\to 0} \infty$$

This convergence to ∞ is possible only if:

$$\lim_{k\to\infty} \lim_{T\to\infty} \pi_{Tk}(s_j) = 0 \quad , \qquad \text{Equation 2.27}$$

These mathematical deductions prove that if the SA algorithm is run long enough with an infinite number of temperature values and for each temperature value with an infinite number of steps it will reach an optimal solution at the end of the process. However, it is not clear what end means and it is needed infinite number of iterations to guarantee optimality**.**

2.2.2 Simulated Annealing Structure

The basic structure of SA shown in Figure 2 contains the following key elements:

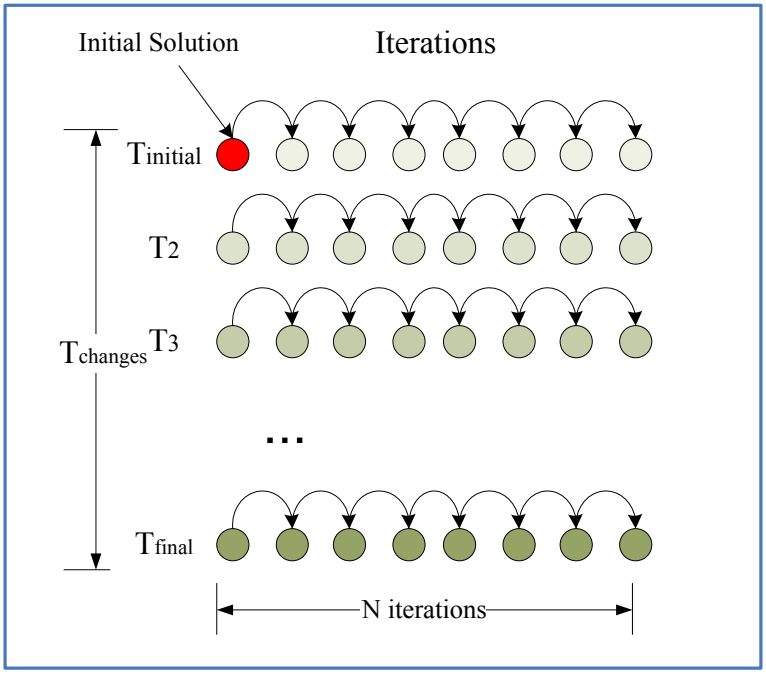

Figure 2 SA Structure

a) A solutions generator: This is the mechanisms used by the algorithm to find new solutions or configurations. Considering SA as a search heuristic, this generator should introduce small random changes, and allow all possible solutions to be reached.

b) Solutions evaluation: The SA algorithm does not require or deduce information. It simply needs to be supplied with an objective function.

c) An annealing schedule: The standard implementation of the SA algorithm is one in which homogeneous Markov chains of finite length are generated at decreasing temperatures. The following parameters are required to define the annealing schedule:

- An initial temperature T_0 ,
- A final temperature T_f or a stopping criterion,
- A length for the Markov chains or number of iterations, and
- A cooling rule.

An initial temperature T_0 is related to the acceptance probability X_0 and the average change in the objective function $\overline{\partial f}$. The initial temperature can be estimated by:

$$T_0 = -\frac{\partial f}{\ln(X_0)} \quad , \qquad \text{Equation 2.28}$$

The final temperature is determined by fixing the number of temperature values to be used, or the total number of solutions to be generated.

Many cooling rules have been proposed but the simplest and most common is the geometric scheme:

$$T_{k+1} = \alpha T_k \quad , \qquad \text{Equation 2.29}$$

where;

α is a constant close to but smaller than 1. Kirkpatrick [13] recommend α=0.95.

The basic structure of the SA algorithm is shown in Figure 3. To initialize the algorithm, the initial temperature, the final temperature and an initial solution must be provided. The cooling rule is specified along with the desired number of iterations for each level of the current temperature. The objective function value for the current solution will be referred to as E_c, and the objective function value for the best solution will be referred to

as E_b. The objective function for the initial solution is evaluated and set as the best value E_b.

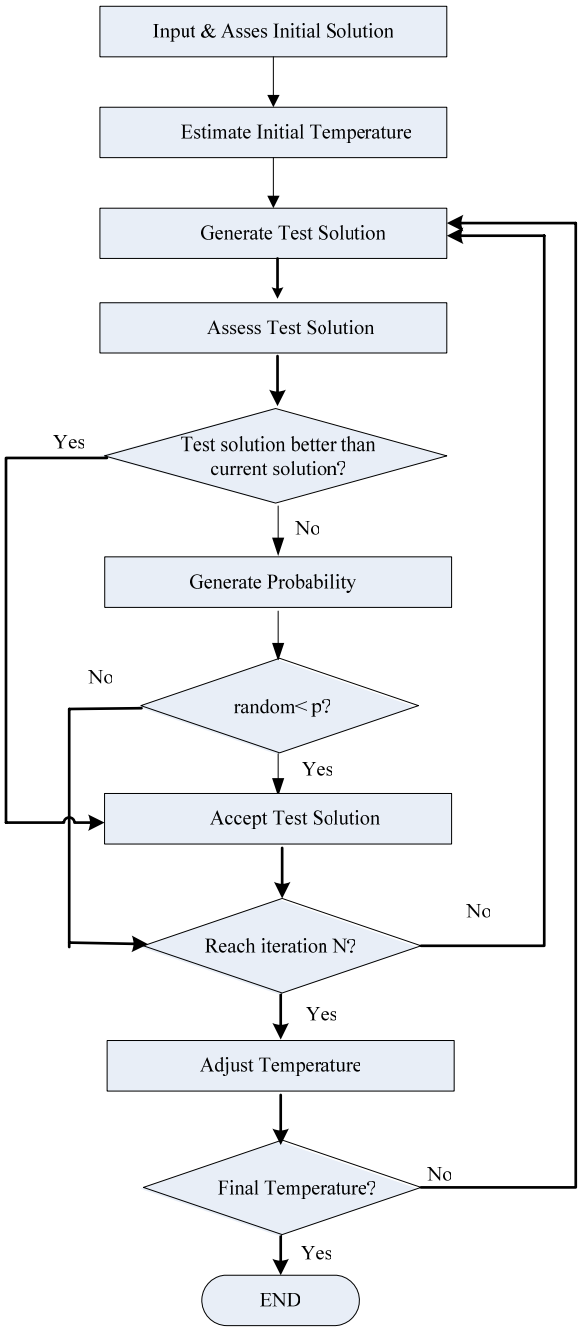

Figure 3 Simulated Annealing Structure

Once the problem has been initialized and the objective function has been evaluated a neighboring solution is generated. This new solution is named the test solution and its objective function value E_t, is calculated and compared with the best value E_b. If Et is better than E_c, the current solution is replaced by the test solution. Otherwise, the Metropolis criterion is evaluated to define if the test solution should be accepted.

If the test solution was accepted, the objective function value for the test function is compared with E_b. If the new value is better than E_b then E_b is replaced by E_t. Whether, the best solution was updated or not, the number of iterations N is increased. If N is less than the maximum number of iterations for each temperature the process is repeated from the step where the new neighboring solution was calculated. Otherwise, the temperature T is adjusted according to the cooling rate and N new iterations are generated. The cycle continues until T reaches the value specified for the final temperature.

2.2.3 Simulated Annealing Applied to Line Balancing

As mentioned before, Simulated Annealing could be efficiently used to solve large combinatorial optimization problems. Some of the problems within industrial engineering solved through SA have been: facility design with multiple floors, facility layout problems in cellular manufacturing systems, product sequencing, mixed-model sequencing with multiple objectives, generation of robotic assembly sequences, balancing of U-type assembly systems, and assembly line balancing with paralleling of workstations.

Kara and Ozcan [12] proposed a SA algorithm to simultaneously solve the balancing and

sequencing problem of mixed-model U lines. The basic assumptions for this algorithm were: products with similar characteristics, deterministic processing times and no parallelism. The single objective function was oriented to the minimization of workstations and workloads deviation.

One of the most interesting approaches used to solve the line balancing problem is presented in McMullen and Frazier. They developed a SA algorithm to solve the type I balancing problem with parallel workstations. In order to make the model closer to reality they assumed multiple products and stochastic processing times. After analyzing the impact of a set of different objectives two of them E_1 and E_3 were selected. These objective functions were evaluated simultaneously in the algorithm: (1) minimizing the total cost and (2) minimizing the deviation from the fixed cycle time represented by the equations 2.9 and 2.11.

McMullen and Frazier suggested a geometric cooling rule and a solutions generator based on two principles: trade and transfer. The algorithm is compared to 23 different heuristic rules and shows excellent results in terms of percentage of units completed within the cycle time and average system utilization. However, the results obtained for unit total cost are relatively poor.

Along this literature review several approaches to tackle different types of the line balancing problem have been presented. Some of the researchers use simple heuristics rules to obtain a good allocation of the tasks in workstations and the others use more composite techniques such as integer programming and meta-heuristics. The limitations related to computational time required to process the mathematical model of the GALBP

has lead to proposed procedures which provide good but not optimal solutions. Meta-heuristics such as Ant Colony, Tabu Search, Genetic Algorithms and Simulated Annealing have proven to provide excellent line balancing configurations. Among this group Simulated Annealing demonstrated to be a fast and accurate algorithm. The major advantage over other methods is its ability to avoid becoming trapped at a local optimum.

3. PROBLEM DEFINITION

The overall objective of the line balancing problem is to determine the assignment of tasks to workstations in order to minimize number of stations, cycle time or cost. The line balancing problem analyzed in this thesis work pursues minimizing the total line operating and investment cost when parallel stations are allowed.

The assignment of tasks to stations is depicted in shown in Figure 4. It involves the allocation of tasks and determination of the number of parallel stations subject to capacity and precedence constraints.

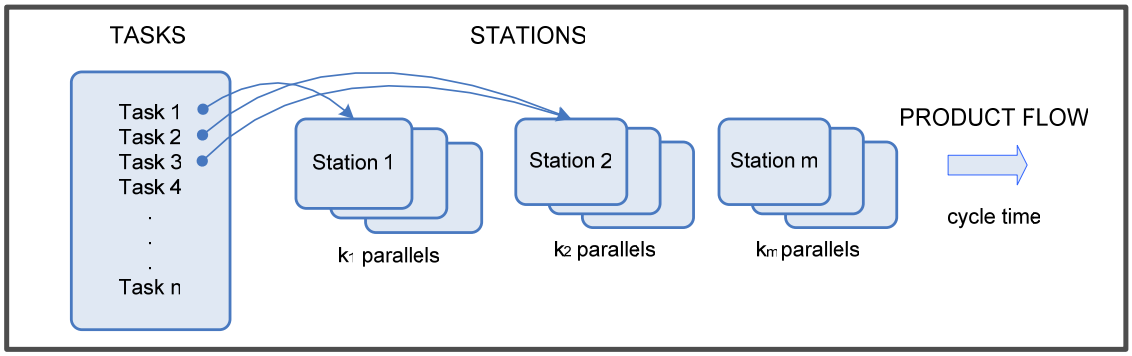

Figure 4 Line Balancing Problem Representation

The line balancing algorithm designed in this thesis applies to mixed-product production lines. Products are assumed to belong to a product family and therefore to have similar characteristics. Set up times to change from one model to another are assumed to be negligible. Product flows between adjacent workstations in quantities defined as unit load sizes.

The objective of the line balancing model is to minimize the total cost associated with operating the production line containing serial and parallel stations. At each work center a

group of manufacturing tasks are performed to satisfy customers' demand. It is assumed that processing times are probabilistic and the user can choose one among a group of probability functions included in the model. Processing times can be the same or differ among models within the product family.

3.1 Development of the Cost Function

The problem analyzed involves a set of costs that determine the quality of the solution generated. A brief description of the costs identified for the problem follows:

- Station Costs (S): These include the workstation capital investment cost, labor cost and overhead as a percentage of labor cost. These costs are the same for all serial positions in the line. It is assumed they include equipment which is the same for all workstations and only one operator at each work center. Station costs increase with the addition of parallel stations. When the cycle time is smaller than the longest processing time, the station costs control the unrestricted creation of parallel workstations.

  The total station costs for the line are calculated using the following expression:

$$\text{Station costs} = \sum_{i=1}^{K} S \cdot p_k \quad , \qquad \text{Equation 3.1}$$

  where; S is the amortized station costs in dollars per unit time, p_k is the total number of parallel stations at serial position k, and K is the total number of serial positions in the line.

- Equipment and tooling cost *(E)*: The amortized costs of tooling and machinery required to perform a task are gathered in this category. Equipment and tooling cost depend on the tasks and vary from one task to another. Clearly if equipment cost for a task is high the algorithm will tend to avoid the addition of parallel stations.

  Total equipment and tooling cost is calculated using the following equation:

$$\text{Equipment and tooling costs} = \sum_{k=1}^{K} \sum_{i \in k} E_i \cdot p_k \quad , \qquad \text{Equation 3.2}$$

  where; *Ei* is the amortized cost for tooling and equipment required to perform task *i* assigned to workstation at serial position *k*.

- Lateness Cost *(L)*: One of the main assumptions in the problem statement is the stochastic behavior of processing times. These can behave according to a normal, uniform or triangular distribution. Because processing times are not deterministic there is a possibility of exceeding the total expected time for the work center. This measurement is called the lateness probability. The probability of performing the tasks assigned to station *k* in a time lower or equal to the line cycle time is calculated as follows:

$$TP_k = \int_{-\infty}^{cycle} F(t)dt \quad , \qquad \text{Equation 3.3}$$

  where; *F(t)* corresponds to the probability distribution function for the work center's processing time. If processing times are normally distributed the probability that the tasks allocated to station *k* are performed on time is obtained using the following equation:

$$TP_k = \frac{1}{\sigma_k \sqrt{2\pi}} \int_{-\infty}^{cycle} exp\left(-\frac{(u - t_k)^2}{2\sigma_k} \right) du \quad , \qquad \text{Equation 3.4}$$

where; t_k is the expected total duration of all tasks assigned station *k* and σ_k is the standard deviation of processing time of the station. Following basic probability principles σ_k is obtained from:

$$\sigma_k = \sqrt{\sum_{\forall i \in k} \sigma_i^2} \quad , \qquad \text{Equation 3.5}$$

Therefore, lateness probability of the line *LP* can be obtained using Equation 3.6.

$$LP = 1 - \prod_{k=1}^{K} TP_k \quad , \qquad \text{Equation 3.6}$$

When product flow takes longer than the line cycle time shop orders are not delivered on time which could lead to cancelled orders or a loss in the market share. The cost associated has been chosen to be a percentage of the product price representing a penalty for all possible lost orders due to lateness in lead times. The mathematical expression for calculating lateness probability and cost is presented in section 3.3.

3.2 Line Balancing Constraints

As mentioned previously, the line balancing problem focuses on the assignment of tasks to workstations subject to a series of constraints. The next section presents these constraints.

3.2.1 Capacity Constraints:

These constraints restrict the total number of tasks that can be assigned to a work center based on a line cycle time. The line cycle time is established by the user based on the production requirements. Frequently, the cycle time is calculated using the Equation 3.7

$$c = \frac{available\ time\ for\ production}{required\ production} \quad , \qquad \text{Equation 3.7}$$

The sum of all tasks processing times grouped in a work center is called "station load". This load cannot exceed the cycle time to satisfy demand requirements. However, there are instances which require completing a product unit in a cycle lower than the longest task, for these particular cases is necessary to duplicate workstations or create a parallel work center. If parallel stations are generated then the station load is calculated as follows:

$$\text{load}_k = \frac{\sum_{i=1,\forall i \in k} t_i}{p_k} \quad , \qquad \text{Equation 3.8}$$

The mathematical expression presented in Equation 2.1 calculates the minimum number of workstations required given a line cycle time. This formula provides the lower bound for the number of workstations without differentiating series from parallel stations.

In this line balancing procedure is proposed the utilization of a unique aggregated cycle time which is obtained employing Equation 3.7. The balance obtained using this single cycle time, and the weighted average time, is equivalent to do different balances for each

model of the product family using individual processing times and individual cycle times.

The individual cycle times can be calculated by determining the fraction of the total available time that the company is willing to assign to the fabrication of each model type. A new weight that includes the participation of the model and the sum of the processing times is calculated using the following expression:

$$w_m = \frac{p_m \cdot \sum_{i=1}^{N} t_{im}}{\sum_{m=1}^{M} p_m \left(\sum_{i=1}^{N} t_{im} \right)} \quad , \qquad \text{Equation 3.9}$$

where;

p_m= participation of model m,

t_{im}= processing time of task i for model m,

N= total number of tasks to balance, and

M= total number of models.

After calculating the weight it is possible to compute the individual cycle time using Equation 3.10.

$$c_m = \frac{available\ time\ for\ production \cdot w_m}{required\ production\ of\ model\ m} \quad , \qquad \text{Equation 3.10}$$

Taking into consideration that the models manufactured in a mix-product line belong to a product family the variations in processing times and precedence restrictions are usually slight. It can be demonstrated that if variability of the individual processing times are small (approximately variation coefficients up to 10%) the individual line balances

obtained are identical to the one generated using weighted average times and aggregated cycle time. Moreover, if the processing times differ significantly among the models, the number of stations obtained for the partial balances is the same for the individual balances and the aggregate line balancing but the specific allocation of the tasks might vary between the balances.

When, the precedence restrictions differ substantially, the application of the proposed methodology for mix-product results in a composite task sequence similar to a serial line causing a reduction in the efficiency of the balance. Therefore, the adequate selection of the products of the family that will be manufactured in the same line is critical to generate an effective design. It can be concluded that products must present similar characteristics, precedence relations and processing times.

3.2.2 Precedence Constraints:

Manufacturing operations must be performed in a certain order defined by the technical characteristics of each product. These are called precedence constraints. The set of all precedence relationships is represented by a network where arcs indicate precedence restrictions and nodes symbolize tasks. An example of a precedence diagram is shown in Figure 5.

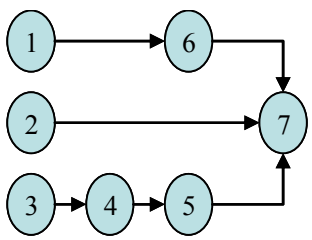

Figure 5 Precedence Diagram

To facilitate mathematical operations, this graphical representation is substituted by a square matrix where every precedence restriction is symbolized with a one and the inexistence of relationship is symbolized by a zero.

The precedence matrix for the diagram illustrated in Figure 5 is shown in Table 1.

Table 1 Precedence Matrix

Tasks	Tasks							
		1	**2**	**3**	**4**	**5**	**6**	**7**
	1	1	1	0	0	0	0	0
	2	0	1	0	0	0	0	0
	3	0	0	1	0	1	1	0
	4	0	0	0	1	0	0	0
	5	0	0	0	0	1	1	0
	6	0	0	0	0	0	1	0
	7	0	0	0	0	0	0	1

3.2.3 Technical Constraints

These include two groups of constraints. First, tasks can only be assigned to one serial position in the line. The second group relates to constraints in the maximum number of parallel stations because of equipment availability or constraints in capital investment.

3.3 Stochastic Processing Times for the Mix-Product Line

The total cost of the line has been defined as the sum of capital investment, operation and lateness cost. Lateness cost is calculated based on the probability that products are not delivered in the expected time. This probability relies on the total workstation load, the unit load size and the probability distribution of the processing times.

Processing times of the mix-model are obtained using the weighted average technique. In case of constant processing times, weighted average times are calculated using

Equation 2.8. However, when times are probabilistic it is necessary to calculate the average according to the rules of operations with random variables. Weighted averages and standard deviation for uniform and triangular data are obtained via simulation. Sets of one thousand random numbers for the each model's process time are combined in order to generate an estimate of the weighted time for the task. A scheme of the procedure applied to obtain the weighted averages and the parameters of the empirical distributions is shown in Figure 6.

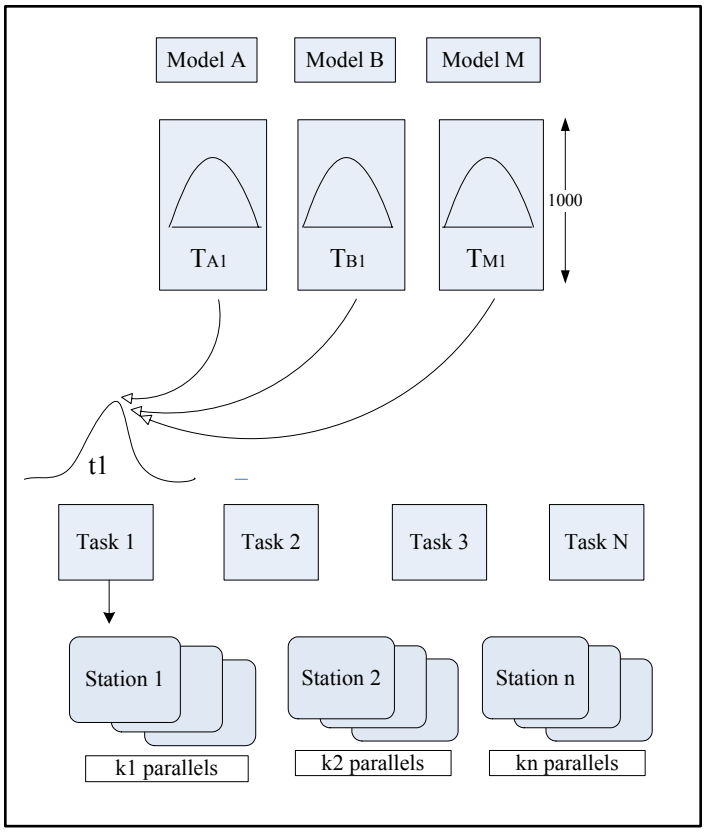

Figure 6. Stochastic Processing Times

Initially, the parameters and probability distribution of the individual processing times are set, and then a vector of one thousand random values is generated for each task and each

model. The arrays of individual times are combined to generate an empirical distribution of the data, to estimate the mean and standard deviation of the processing times and calculate the weighted average for each task. The new array corresponds to the tasks' processing times and their estimated mean and variance which are used in all iterations to calculate the lateness probability of the line.

The lateness probability is evaluated using the general Equation 3.6. The probability of finishing the operation on time in station k, TP_k, is defined according to the following general expression:

$$TP_k = \left(probability\left(\frac{t_k \cdot q}{p_k} \le c \cdot q \right) \right)^{p_k} \quad , \qquad \text{Equation 3.11}$$

Where, q represents the unit load size, c is the line cycle time and p_k. is the number of parallel station at serial position k. Intuitively, it is presumed that when the standard deviation of the processing times increases the lateness probability increases. Additionally, the possibility of making a delayed delivery of a product decreases as the unit load size increases. This is a consequence of adding positive and negative deviations from the expected time. One case study was used to test the impact of variability and unit load size on the lateness probability. The case study was solved at three levels of unit load and variance coefficient.

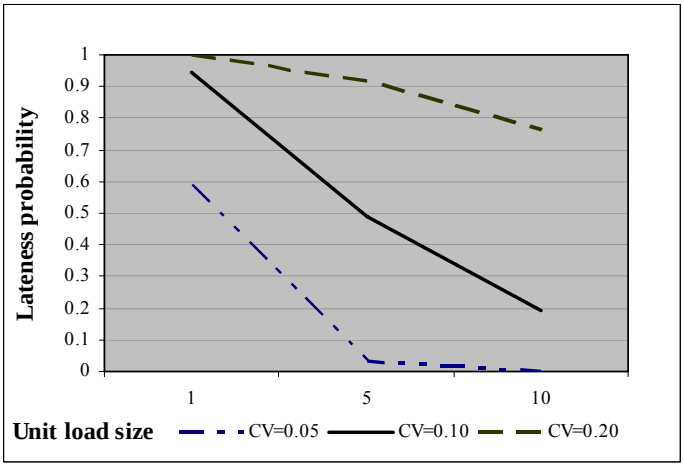

Figure 7 Lateness Probability vs CV

As shown in Figure 7, the lateness probability decreases as the unit load size increases. The lowest values for lateness probability were observed for the lowest variance coefficient tested. These deductions reaffirm the classic trade off between work in process cost and the ability to respond on time to customer's demand.

3.4 Mathematical Formulation of the Problem

The model proposed assigns tasks to workstations and determine the optimal number of parallel workstations in order to obtain a design which minimizes capital investment, operation and lateness cost.

Prior to proposing a mathematical model of the problem it is required to convert the individual precedence sequence into a composite precedence sequence using Askin and Zhou [4] technique. Additionally, individual processing times are integrated into a set of single weighted average times. Weighted average times are computed using Equation 2.8

The notation used for the development of the optimization model follows:

Indexes:

i= task number, i=1,…, n,

k= station number, k=1,…, K, and

p=number of parallels for each station, p=1,…, P.

Parameters:

n =total tasks to be balanced,

c=line cycle time,

q=unit load size,

t_i=weighted average processing time for task i,

E_i=equipment and tooling cost for task i,

S=station costs to open a workstation,

L=lateness cost penalization,

LP_k= lateness probability for station k,

tp_k=parallels workstations at station k,

ld_k=load of station k,

s_i=station number where i is allocated, and

SP=set (ai,bi) of tasks such that task a must precede task b for model i.

Decision variables:

$$x_{ik} = \begin{cases} 1, \textit{if task i is assigned to station k} \\ 0, \textit{otherwise} \end{cases},$$

$$x_{ikp} = \begin{cases} 1, \textit{if task i is assigned to station k with p parallels} \\ 0, \textit{otherwise} \end{cases}, \text{ and}$$

$$pa_{pk} = \begin{cases} 1, \textit{if p parallels are assigned to station k} \\ 0, \textit{otherwise} \end{cases}.$$

One essential restriction considered in this type of problem is related to the assigment of tasks to workstations. If task *i* was assigned to station *k*, that task is not available anymore and cannot be allocated to any other workstation. Adittionally, this model entails a series of escenarios where station *k* is replicated in *p*=1,…,*P* is proposed parallels. Since the binary variable x_{ikp} is only activated when task *i* was assigned to station *k* with *p* parallels, correct allocation and no duplication of tasks is guaranteed with the following expresions:

$$\sum_{p=1}^{P}\sum_{k=1}^{K} x_{ikp} = 1 \quad , \qquad for\ i = 1,...,n\,, \qquad \text{Equation 3.12}$$

The second constraint is used to calculate decision variable x_{ik}. That variable indicates whether task *i* was assigned to station *k* or not. Because x_{ik} is binary, this equation take into consideration that only a number of of *p* parallels can be activated for each work center.

$$\sum_{p=1}^{P} x_{ikp} = x_{ik} \quad , \qquad for\ i = 1,...,n\ and\ k = 1,...,K, \qquad \text{Equation 3.13}$$

The third constraint is added to the model in order to capture the number of the station where each task was assigned

$$\sum_{k=1}^{K} k \cdot x_{ik} = s_i \quad , \qquad for\ i = 1,...,n, \qquad \text{Equation 3.14}$$

The precedence relations are included into the formulation through the fourth constraint. Each precedence constraint must be expressed as an equation where the station number of a succeeding task *bi* (s_{bi}) must be bigger than or equal to the station number of the

precceding task *ai* (s_{ai}). This constraint guarantees that a task is assigned once all its precedents have been allocated.

$$s_{bi} - s_{ai} \geq 0 \quad , \qquad for\ ai, bi \in SP, \qquad \text{Equation 3.15}$$

It is decisive to ensure that all the tasks of a workstation have the same number of parallels; otherwise the solution obtained for the problem will lack of sense. The fifth constraint is used to determine which value of *p* was activated for station k. That decision variable pa_{pk} is restricted to values zero or one; therefore, the sixth constraint ensures that only one possible value of p parallels has been activated for one particular station.

$$10000 \cdot pa_{kp} - \sum_{i=1}^{n} x_{ikp} \geq 0 \quad , \quad for\ k = 1,...,K\ and\ p = 1,...P, \qquad \text{Equation 3.16}$$

$$\sum_{p=1}^{P} pa_{kp} = 1 \quad , \qquad for\ p = 1,...,P, \qquad \text{Equation 3.17}$$

Analogous to the third constraint given by the Equation 3.14 in Equation 3.18 is calculated the number of parallels. This is not a binary variable but a number between 1 and an upper limit of parallel stations given by the user.

$$\sum_{p=1}^{P} p \cdot pa_{kp} = tp_k \quad , \qquad for\ i = 1,...,n, \qquad \text{Equation 3.18}$$

In the total cost function is expected to include station, equipment and lateness cost. Lateness cost depends on the distribution of the processing times and the unit load size. The unit load size is the number of product units to be moved at once between adjacent

workstations. As the unit load size increases the lateness probability decreases as a result of a delay lessening the effect caused by those units in the load which take less than the expected processing time.

If task times for individual products are distributed normally with mean t_i and standard deviation σ_i and the product flows in batches of load size q, then station parameters σ_k and t_k are calculated as follows:

$$t_k = q \cdot \sum_{\forall i \in k} t_i \quad , \qquad \text{Equation 3.19}$$

$$\sigma_k = \sqrt{q^2 \cdot \sum_{\forall i \in k} \sigma_i^{\,2}} \quad , \qquad \text{Equation 3.20}$$

The expected time for manufacturing q units at station k is t_k units of time, with a standard deviation σ_k. If processing times are normally distributed the probability TP_k that tasks allocated to station k with p parallels are completed on time is calculated as follows:

$$TP_k = \left(\frac{1}{\sigma_k \sqrt{2\pi}} \int_{-\infty}^{cycle*q} exp\left(-\frac{\left(u - \frac{t_k}{p} \right)^2}{2\sigma_k} \right) du \right)^p , \qquad \text{Equation 3.21}$$

Then the lateness probability of the line is computed using Equation 3.6 as the complementary probability of finishing all the items on time.

The total work load for the station k is t_k time units. This measure is obtained through the Equation 3.22 and Equation 3.23. In this scenario the limit time to complete the work

in a workstation is defined by $q \cdot c$ as shown in Equation 3.24.

$$q \cdot \sum_{i=1}^{n} \frac{x_{ikp} \cdot t_i}{p} = t_{kp} \qquad for\ k = 1,...,K\ and\ p = 1,...,P, \qquad \text{Equation 3.22}$$

$$\sum_{p=1}^{P} t_{kp} = t_k \qquad for\ k = 1,...,K, \qquad \text{Equation 3.23}$$

$$q \cdot c - t_k \geq 0 \qquad for\ k = 1,...,K, \qquad \text{Equation 3.24}$$

Additional constraints are integrated into the model in order to incorporate the costs associated with delayed shop orders and loss of costumers. Equation 3.25 calculates the standard deviation of the station load as a sum of the standard deviation of the processing times assigned to the workstation.

$$\sigma_k - \sqrt{q^2 \cdot \sum_{i=1}^{n} x_{ik} \cdot \sigma_i^2} = 0 \qquad , \qquad \text{Equation 3.25}$$

Finally, it is calculated the probability of processing in the station *k* one unit of product in a period of time lower or equal than the line cycle. This measure is computed using the cumulative probability of the normal distribution as shown in Equation 3.26

$$TP_k - \left(\frac{1}{\sigma_k \sqrt{2\pi}} \int_{-\infty}^{cycle} exp\left(-\frac{\left(u - \frac{t_k}{tp_k} \right)^2}{2\sigma_k} \right) du \right)^{tp_k} = 0 \qquad , \qquad \text{Equation 3.26}$$

Once all the decision variables have been described and the constraints have been established it is possible to propose the objective function which minimizes the total investment and operational cost.

Making the lateness cost *LC* a percentage of the product price representing a penalty for possible lost orders due to lateness, the total cost of the line balance is expressed as follows:

$$Z = \sum_{k=1}^{K} tp_k \cdot S + \sum_{k=1}^{K} \sum_{i=1}^{n} tp_k \cdot E_i \cdot x_{ik} + LC \cdot LP \quad , \qquad \text{Equation 3.27}$$

The optimization problem is formulated as follows:

$$Min\ Z = \sum_{k=1}^{K} tp_k \cdot S + \sum_{k=1}^{K} \sum_{i=1}^{n} tp_k \cdot E_i \cdot x_{ik} + LC \cdot \left(1 - \prod_{k=1}^{K} TP_k \right) \quad , \quad \text{Equation 3.28}$$

Subject to: Equations 3.12 to 3.18, and Equations 3.22 to 3.26.

The complexity of the proposed model is high and it cannot be solved using traditional methodologies. Additionally, the model is restricted to normally distributed processing times. Considering that it is desirable to build a general model that promptly provide a near to the optimum solution for real cases that involve normal, uniform or triangular processing times it is necessary to employ alternative methodologies to find a solution to the problem. A Simulated Annealing heuristic was designed to tackle the problem. The procedure developed consists of a Simulated Annealing that employs a dynamic cooling schedule where parameters are adjusted depending on the problem complexity. The following sections describe the general SA algorithm and the selection of appropriate parameters in order to achieve a superior performance.

4. METHODOLOGY FOR THE DESIGN OF A SIMULATED ANNEALING-BASED HEURISTIC

The methodology proposed for the design of a Simulated Annealing-based heuristic is presented in Figure 8. It is highly based on the use of design of experiments for the evaluation and selection of heuristic parameters and an initial solution from which the heuristic works to obtain a near to optimum solution.

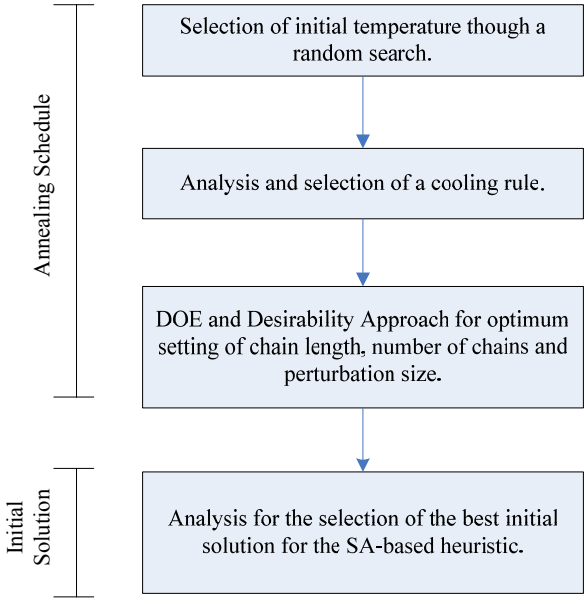

Figure 8 Methodology for the Design of a Simulated Annealing-Based Heuristic

Each one of the steps in the proposed methodology is described next.

4.1 Solution Representation and Generation

Solution representation and neighboring generation are two essential parts to consider in SA design. Solution representation should allow manipulating the current solution

through small perturbations and allow reaching all possible solutions.

4.1.1 Solution Representation

In the SA algorithm a solution is represented through an *n* by *k* matrix, along with a k length vector where the number of parallel workstations is stored. Table 2 and Table 3 show an example of a possible line balance solution with twelve tasks and five workstations in series.

Table 2 Task Assignment

Task Assignment				
St 1	**St 2**	**St 3**	**St 4**	**St 5**
1	0	0	0	0
0	**2**	0	0	0
3	0	0	0	0
0	**4**	0	0	0
0	**5**	0	0	0
0	0	**6**	0	0
0	0	**7**	0	0
0	**8**	0	0	0
0	0	0	**9**	0
0	0	0	**10**	0
0	0	0	0	**11**
0	0	0	**12**	0

In this example tasks 1 and 3 were assigned to serial position 1, tasks 2, 4, 5 and 8 were assigned to serial position 2 and so on.

Table 3 Parallel Workstations

Station Number	Number of parallels
St 1	1
St 2	2
St 3	1
St 4	3
St 5	1

4.1.2 Solution Generation

Often the solution space of an optimization problem has many local minima. A simple local search algorithm proceeds by evaluating initial solution and generating a new solution from the neighborhood.

The matrix of a current solution is modified through small random changes in order to obtain neighboring solutions. The mechanisms used to generate new feasible designs were called trade and transfer. Selection of any of these methods is made through a random number generation. These mechanisms are explained next.

Trade: The first step is to choose randomly one station *x* and one of both adjacent workstations *y*. A trade is made between the last task in station *x* and first task in station *y* if *y* follows *x* or between first task in station *x* and last task in station *y* if *y* proceeds *x*. Trade is performed strictly if the precedence constraints are not violated.

An example for this procedure is shown in Figure 9 and Figure 10. Tasks 9 and 11 within workstations 3 and 4 have been chosen through random selection. If the precedence constraints are no violated those tasks are exchanged, otherwise the algorithm attempts another perturbation mechanism. Once the trade has been made, the number of parallels and work loads are recalculated.

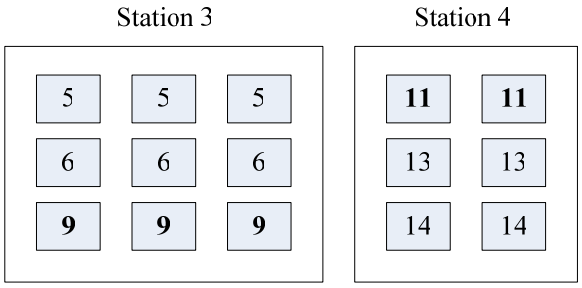

Figure 9 Trade Between Adjacent Workstations

As a result of this trading process, stations can become smaller or bigger especially with the cancellation or addiction of parallel workstations. This phenomenon is shown in Figure 10.

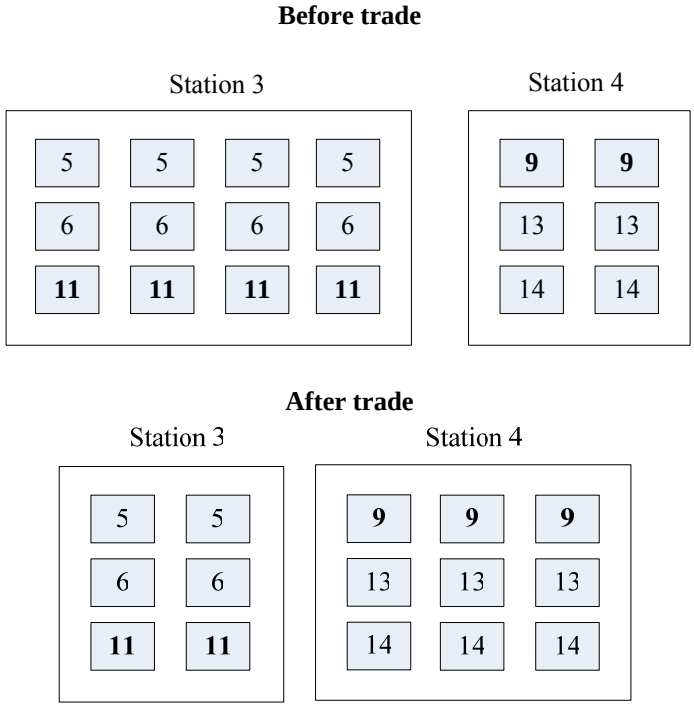

Figure 10 Change in Workstation Size After Trading

Transfer: As in the trade mechanism, the first step is the random selection of workstation *x*. Next a task in *x* is randomly chosen. Then the target task is transferred to an adjacent workstation *y*. Choices regarding to direction and task to transfer are not

deliberate but random decisions. As shown in Figure 11, if station 7 was randomly chosen for a transfer, any task in 7 can be transferred to either station 6 or 8.

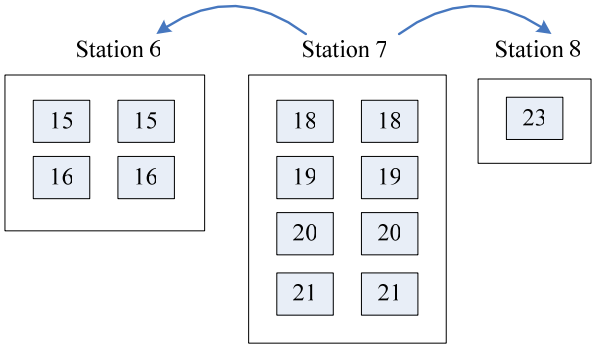

Figure 11 Station 7 Before Transfer

Figure 12 shows an example of forward transference. Task 19 is randomly selected and then transferred to adjacent station 8. As part of the trade process the number of parallel workstations is recalculated because of the resulting mutation in the current solution matrix. Then the total line balancing cost and SA statistics are updated.

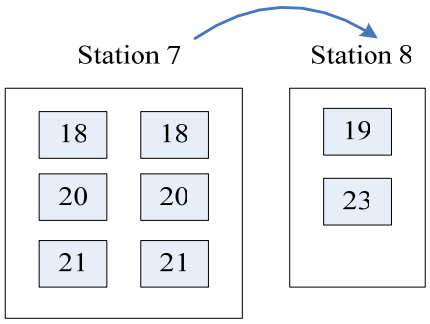

Figure 12 Forward Transfer Example

4.2 Selection of the Annealing Schedule

In each iteration, the SA algorithm replaces the current solution by a random solution from the neighborhood which is chosen with a probability that depends on the difference between the corresponding function values and the temperature T_k. The control

parameter T_k has the same function for the procedure as the temperature of the Metropolis algorithm. Therefore, as the temperature decreases the probability of accepting worse configurations decrease.

The impact of the temperature on the performance of the algorithm is such that the current solution changes almost randomly when the temperature is too large but increasingly finds better solutions as the temperature tends to zero.

Although Simulated Annealing has been widely used during the last two decades, there is still a lack of practical information to help the user in designing an appropriate annealing schedule that assures a good performance of the algorithm. Most of the concerns lie on the selection of the initial temperature and a cooling rule in order to perform a fast and accurate search over the solution's space.

4.2.1 Initial Temperature

According to Triki and Collette [28] the initial temperature should allow the SA to perform a random walk over the landscape. This suggests that the initial temperature should be high enough to assure a complete walk. However, it is not desirable to perform unnecessary iterations which consume excessive computational time. Also, it has been proven that the quality of solutions at high temperatures is relative poor.

To solve this tradeoff several rules to calculate the initial temperature have been stated. The initial temperature is obtained with the Van Laarhoven equation as follows:

$$T_0 = -\frac{\overline{\Delta f}^{(+)}}{\ln(\chi_0)} \quad , \qquad \text{Equation 4.1}$$

This equation is based on an initial value χ_0 which is defined as the ratio between the number of the bad transitions the user is willing to accept and the total number of bad transitions. The initial acceptance ratio is also defined as the average increase in acceptance probability.

The numerator in the equation, $\overline{\Delta f}^{(+)}$ is the average change in the objective function value and is estimated by conducting an initial random search of *n* number of steps. The average change is considered as an approximation of the depth of the deepest local minimum.

It is extremely important not to underestimate the value of $\overline{\Delta f}^{(+)}$ since that would result in a low value for the initial temperature T_0. As a consequence, the SA algorithm might not perform a complete walk over the landscape and could get trapped in a local minimum.

Figure 13 shows two different landscapes for which the initial temperature computed using Laarhoven's formula was the same. Although the objective function "a" has four local minima of depth 1 and objective function "b" has a minimum depth of 4, they cannot be distinguished by only counting the number of moves with higher cost. In fact, if the landscape presents a deep minima, then $\overline{\Delta f}^{(+)}$ may underestimate the depth of these minima, and the computed initial temperature may be too low.

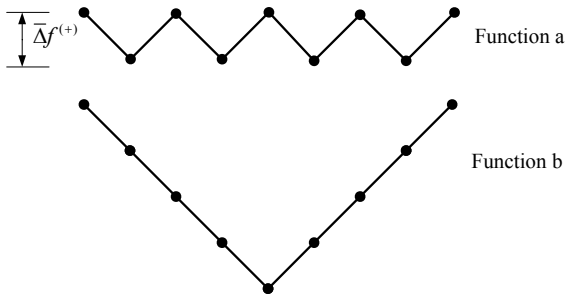

Figure 13 Different Landscapes with the Same $\overline{\Delta}f^{(+)}$

A simulation was conducted in order to test the robustness of the Van Laarhoven equation in finding the initial temperature. A case study was used to perform 30 random walks for each of the following walk lengths: *n*=20, 50,100 and 200 steps. The average change in the objective function $\overline{\Delta}f^{(+)}$ was estimated for each random walk resulting in an estimated value for T_0. It was expected to obtain approximately the same value for T_0 regardless of the walk length. Results are summarized graphically in Figure 14. The graph shows that short random walks resulted in higher estimates for the initial temperature.

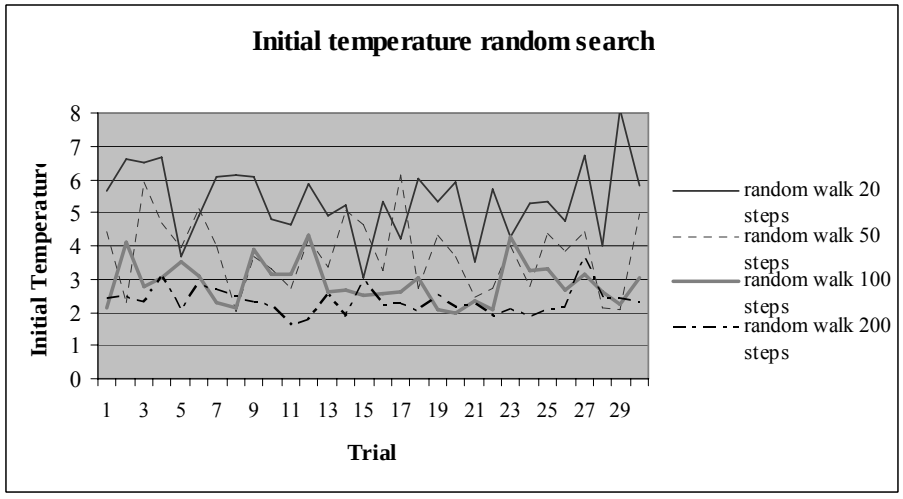

Figure 14 Initial Temperature Search

An ANOVA was performed and results are summarized in Table 4. These show that at a 95% confidence level, the walk length has a significant impact on the resulting estimated value for T_0.

Table 4 ANOVA for Random Walk Lengths

Source	DF	SS	MS	F	P Value
Walk length	3	56.49	28.24	27.35	0.000
Error	87	89.85	1.03		
Total	89	1.46.34			

A boxplot shown in Figure 15 shows the tendency to decrease the estimated initial temperature value as the number of steps is increased. Intuitively, this inverse relationship can be explained as a consequence of weighing deep changes in the cost function between a larger set of elements. Apparently, small changes occur with higher frequency than drastic increments in the objective function.

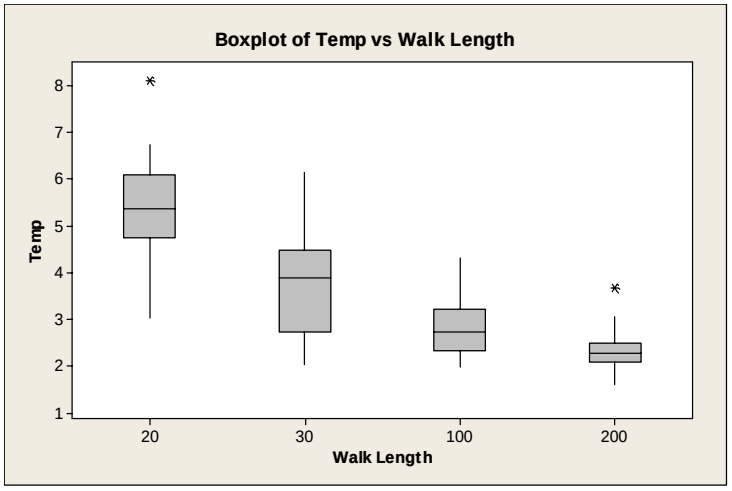

Figure 15 Boxplot Temperature vs Walk Length

Based on results it is concluded that the walk length has a significant impact on the estimated value for T_0 and since the temperature should be high enough to assure a

search over the solution landscape, short random walks are appropriate to estimate the initial temperature for the SA algorithm.

4.2.2 Cooling Rule

Almost all the Simulated Annealing algorithms documented in the literature employ simple stepwise reduction rules to update the temperature. Among these rules the most frequently used is the geometric cooling rule given by:

$$T_{k+1} = \alpha \cdot T_k \quad , \qquad \text{Equation 4.2}$$

Where, α is a reduction factor and $0<\alpha<1$.

Another simple strategy to decrease the temperature is the linear cooling rule stated as follows:

$$T_{k+1} = T_k - \Delta T \quad , \qquad \text{Equation 4.3}$$

Where ΔT is the station decrement step taken at each one of the L trials. These two schemes were analyzed by Randelman and Grest [20]. They found that reductions achieved using the two schemes to be comparable, and also noted that the final value of objective function was, in general, improved with slower cooling rates, at the expense, of course, of greater computational effort. Finally, they observed that the algorithm performance depended more on the quotient $\Delta T / L$ than on the individual values of ΔT and L.

Other researchers such as Van Laarhoven [29] and Huang [11] have analyzed approaches using the standard deviation of the distribution of the objective function to determine the next temperature decrement. The advantage of this scheme is that the temperature

decrement is controlled dynamically. Therefore, those approaches can be applied to all types of problems. Those schemes called "adaptive" are based on the idea of quasi-equilibrium. To maintain quasi-equilibrium, the expected decrement in the average objective function value must be less than the standard deviation of the distribution of the function value.

The schedule proposed by Van Laarhoven is based on the following adaptive rule to update temperatures:

$$T_{k+1} = T_k \cdot \frac{1}{1 + \frac{\ln(1+\partial)}{3\sigma_{(T_k)}} T_k} \quad , \qquad \text{Equation 4.4}$$

where; ∂ is a "small" real number and $\sigma_{(T_k)}$ is the standard deviation of the cost function evaluated for solutions collected up to temperature T_k.

Another typical adaptive rule was proposed by Huang (1986). In his scheme λ is a constant parameter ($0 < \lambda \leq 1$) that has to be determined by the user. A typical value of λ is 0.7. The updated temperature is determined as follows:

$$T_{k+1} = T_k \cdot \exp\left(-\frac{\lambda T_k}{\sigma_{(T_k)}} \right) \quad , \qquad \text{Equation 4.5}$$

This schedule has been widely used and is known to provide an efficient general cooling schedule.

Triki and Collette [28] verified that the adaptive decrement rules proposed by Van Laarhoven and Huang are equivalent, but they differ significantly from the geometric cooling schedules. They also noticed that adaptive rules have a tendency to degenerate at

low temperatures. Certainly, when the temperature becomes sufficiently low, the SA algorithm will become trapped into a local minimum. Because of the good quality of current solutions most of the trials will lead to worse solutions and the probability of accepting such transitions will be very low due to the low temperature. Therefore, most of the transitions will be rejected.

According to the literature, the cooling rule must be an asymptotic function that does not decrease drastically. A simulation of three decrement rules at different levels was run in to find the scheme with the best performance for this particular algorithm. The results of the simulation are plotted in Figure 16.

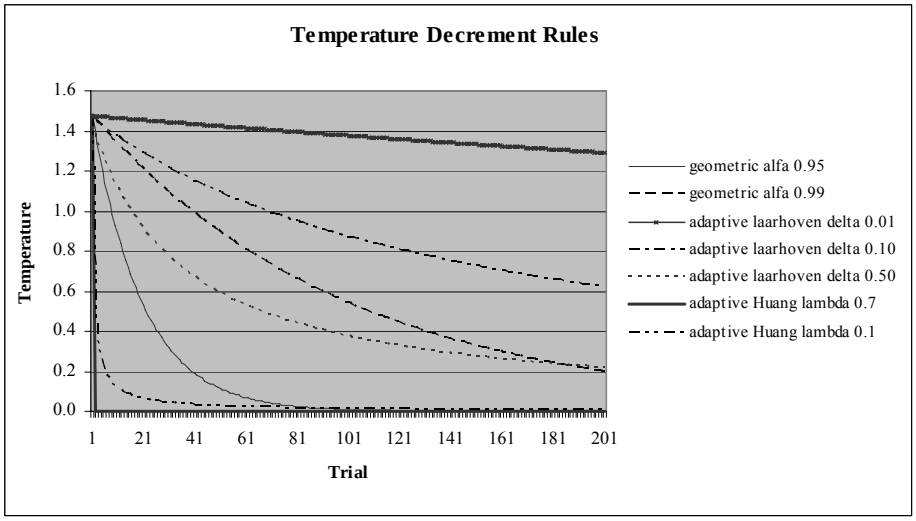

Figure 16 Temperature Decrement Rules Plot

The geometric cooling rule with $\alpha = 0.95$ presents a satisfactory performance. However, popular adaptive rules like Laarhoven and Huang at 0.7 do not show evidence of adequate behavior. In one hand Laarhoven's rule do not decline consistently, and in the other hand Huang shows a rapid descend in the first ten trials.

Because of the significant difference observed between the behavior described in the literature and the one obtained via simulation it was necessary to perform additional tests to determine the best cooling rule. The capability of the algorithm for finding a minimal solution was measured through an experiment where seven cooling rules were used for two different case studies. The case-studies were analyzed as blocks and the interest of the experiment was to determine if the cooling rule affected the performance of the algorithm. The performance measure used was the magnitude of the best cost found.

An analysis of variance was performed to identify the impact of the cooling rule on the performance of the SA algorithm. The results are summarized in Table 5. These show there is not statistical evidence to support the hypothesis that the cooling rules analyzed affect the results of the algorithm at a 95% confidence level. However, following the guidelines given in the literature the geometric rule with $\alpha = 0.95$ and Huang with $\lambda = 0.1$ are preferred due to their asymptotic shape.

Table 5 ANOVA Results Case vs Cooling Rule

Source	DF	SS	MS	F	P
Rule	7	0.00078	0.00011	0.44	0.861
Case	1	1.87487	1.87487	7469.74	0.000
Interaction	7	0.00077	0.00011	0.44	0.863
Error	16	0.00402	0.00025		
Total	31	1.88044			
S	0.01584	R-Sqr= 99.79%		R-Sqr= 99.59%	

4.2.3 Chain Length

The number of iterations known as the chain length at any given temperature should be large enough to guarantee that the thermal equilibrium is reached. The chain length can be fixed, be tied to the achievement of equilibrium or it can depend on a minimum

acceptance criterion. For example, each chain length terminates when the number of accepted solutions reaches a given bound L_m. Some researches have proposed empirical equations to express the fixed chain length as a function of the problem size.

According to Huang's procedure to establish the number of iterations, the length of the Markov chain should be tied to achieve a specific state. However, it is typical to use other methods as the minimum acceptance criterion L_m or the minimum chain length $L_{\min}$ to guarantee the validity of the statistics. Usually the minimum chain length criterion depends on the size of the solution space. For line balancing problems the size of the solution space may be expressed as a function of different variables: number of tasks, relation between the longest processing time and the cycle time, and finally the density of the precedence matrix. The density of the precedence matrix is a measure of the network complexity. This characteristic was called by Dar-El (Mansoor) the F-Ratio and is defined as:

$$D = \frac{2 \cdot d}{N \cdot (N-1)} \quad , \qquad \text{Equation 4.6}$$

Where, d is the number of precedence relations which is equal to the number of ones in the triangular precedence matrix and $N \cdot (N-1)/2$ is the total number of cells in the partial (triangular) matrix. For balancing cases with few precedence relations D approaches a value of zero. On the other hand, when the precedence constraints are as in a serial assembly line, D reaches values near one. The relationship between the dimension of the solution space and the precedence density is inversely proportional. Notice that when D assumes a value close to zero there are numerous alternatives for

allocating the tasks via the permutation of tasks and the generation of parallel stations.

When the ratio between the longest processing time and cycle times is greater than one the heuristic is forced to generate at least one parallel workstation. This increment in the complexity of the feasible solutions may be represented as an expansion of the solution space.

Considering the facts mentioned before, a rational rule to define the minimum chain length might be:

$$L_{\min} = \frac{N \cdot t_{\max}}{c \cdot D} \quad , \qquad \text{Equation 4.7}$$

where;

N = total number of tasks to balance,

$t_{\max}$ = maximum duration time,

c = line cycle time, and

D = assembly network density.

Some facts are known about the evolution of the SA: first, when T_k approaches zero, transitions are accepted with decreasing probability. Therefore, the number of trials required to achieve the minimum number of transitions $L_{\min}$ must decrease. Additionally, at the beginning of the search, when the temperature is still close to the initial temperature, most transitions are accepted the variance is relatively high and therefore long chains are required to explore numerous alternatives for better solutions.

An improved minimum length chain rule which adapts as annealing proceeds is calculated according toEquation 4.8.

$$L_{\min} = \text{int}\left(m \cdot \frac{N \cdot t_{\max} \cdot \sqrt{T_k}}{c \cdot D} \right) \quad , \qquad \text{Equation 4.8}$$

where;

m = adjustment multiplier, and

T_k = temperature for iteration k.

This scheme generates long chains at high temperatures and short chains when a temperature is close to zero, in this manner the computational time reduces. Variable m in this equation corresponds to a multiplier. The value of m is obtained through an experimental process.

4.2.4 Final Temperature

In some simple implementations of Simulated Annealing the final temperature is determined by fixing the number of temperature values to be used, or the total number of solutions to be generated. Alternatively, the search can be stopped if it is identified lack of progress. This lack of progress can be defined in a number of ways, but a useful basic definition is: no improvement (i.e. no new best solution) being found in a fixed number of entire Markov chains.

For the line balancing problem the final temperature may be the temperature reached after a fixed number of Markov chains. Deliberately this number of chains depends of the balancing problem complexity. The deduction made previously for line balancing seems appropriate to define the number of temperature chains.

$$T_{changes} = \frac{M \cdot N \cdot t_{\max}}{c \cdot D} \quad , \qquad \text{Equation 4.9}$$

A stopping rule is also established to prevent expending computational resources unnecessarily. It is suggested to stop the algorithm when after Y consecutive sequences the objective function has not shown improvement, where Y is calculated as $0.5 \cdot (T_{changes})$. This measure is considered conservative since half of the total Markov chains require considerable computational time. However, it is attempted to explore the solution space efficiently without eliminating the probability for finding an enhanced solution in final temperatures.

4.2.5 System Perturbation

Although the size of the perturbation is not consider relevant part of the annealing schedule, after performing test runs of the algorithm, the impact of this characteristic was identified. Some of the important considerations about the size of the perturbation are: it is desirable a neighboring generator able to escape from local optima and the perturbation should shake enough the system to avoid trapping the algorithm. Trade and transfer cause a single and small shake which might not alter the solution in the magnitude required to escape from local optima. However, a step should not be so large that the optimal point is constantly exceeded.

Two different perturbation magnitudes were tested in order to analyze the cost function response. In Figure 17 shows a plot of the cost function response when only one trade or transfer movement is performed at each trial. The plot shows how at lower temperatures, changes in the cost function is barely perceptible. Significant changes occur at the

beginning of the process because new designs are soft modifications of last solutions.

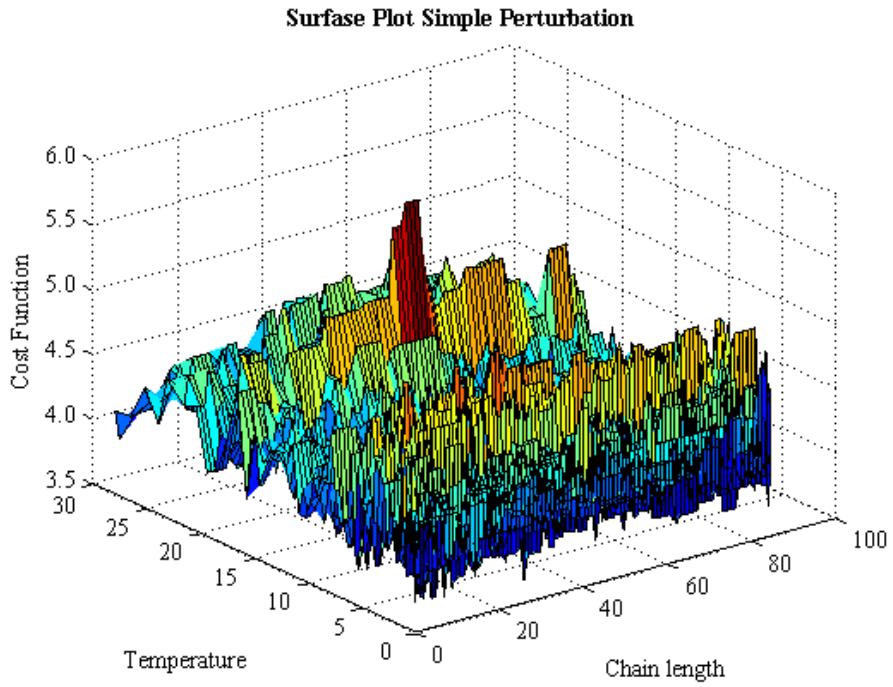

Figure 17 Cost Function for a Perturbation Magnitude of 1

By the other hand, in Figure 18 solution present significant fluctuations during different simulation states which demonstrates the ability of the algorithm to perform an exhaustive search in a bigger zone of the solution space.

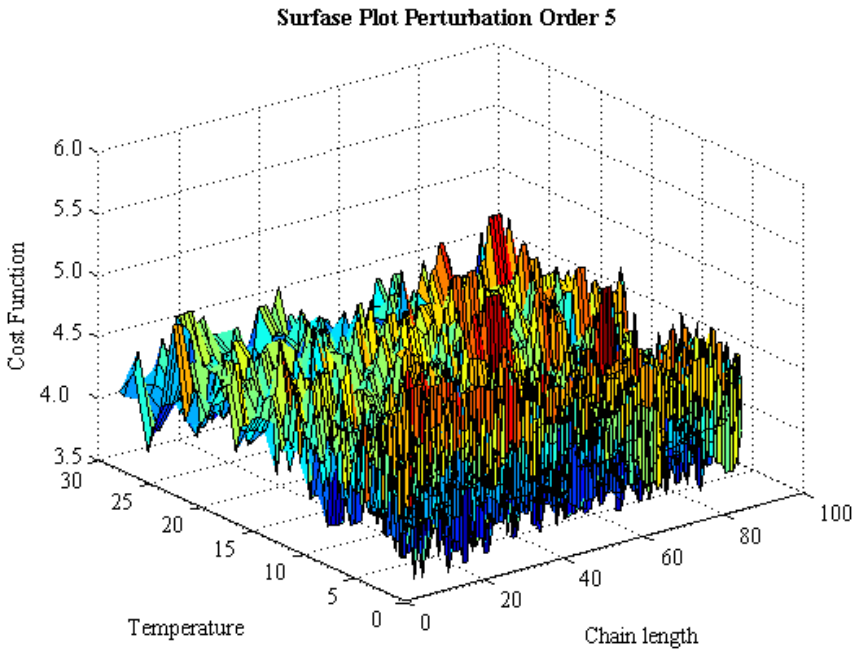

Figure 18 Cost Function for a Perturbation Magnitude of 5

4.3 Optimal SA parameter setting through DOE

In previous sections design factors for the SA algorithm were described and analyzed. However, it has not been identified the effect of these factors over the solution quality. To determine the adequate setting of these design parameters a factorial experiment was designed and run. The influence of the cooling rule, Markov chains length, number of Markov chains, and the perturbation size on the total cost was investigated using a central composite experiment. The response variables were defined as the difference in percentage between the minimal total cost and the total cost observed at each experimental condition, and computational time required to finalize the simulation.

Initially, an exploratory experiment for two case studies with 18 and 23 tasks respectively at two different cooling rules was run. The factors analyzed were the cooling rule,

geometric rule with $\alpha = 0.95$ and the adaptive Huang with λ= 0.1, the number and the length of Markov chains and the perturbation magnitude. The results obtained in this preliminary phase indicate that responses variables were not susceptible to changes in the case study; also more accurate results were obtained using the geometric rule although computational time spent in calculations was considerably higher. In order to obtain the appropriate setup to generate precise solutions in a short period of time a central composite design was used for a case study of 23 tasks with two replications. When the response was "computational time" the assumption of normality on residuals was accomplished employing a logarithm transformation. The factors studied and the un-coded levels are presented in Table 6. The levels for factors A and B correspond to the multipliers in Equation 4.8 and Equation 4.9 respectively.

Table 6 Factors and Levels of the CCD

Levels	**Factors**		
	A. Chain length	B. Number of chains	C. Perturbation magnitude
Low	m=0.25	M=1	Perturbation= 1
High	m=2.75	M=5	Perturbation=5

Table 7 presents the ANOVA results on the percentage difference from the minimum cost observed. At a 95% confidence level results indicate that the length of Markov chains and number of chains have a significant impact on the response quality. Additionally, the interaction between the number of Markov chains and the perturbation magnitude result in a p-value smaller than 0.05 which proves some relevance in the model.

Table 7 ANOVA for Cost Difference

Estimated Regression Coefficients for cost difference				
Term	**Coef**	**SE Coef**	**T**	**P**
Constant	2.7118	0.2826	9.597	0.000
Length of chains	-1.4741	0.2599	-5.671	0.000
Number of chains	-0.6303	0.2599	-2.425	0.022
Perturbation magnitude	-0.2464	0.2599	-0.948	0.351
length*length	0.8767	0.4956	1.769	0.087
number*number	0.3454	0.4956	0.697	0.491
perturb*perturb	0.4520	0.4956	0.912	0.369
length*number	0.0895	0.2906	0.308	0.760
length*perturb	0.2246	0.2906	0.773	0.446
number*perturb	-0.6156	0.2906	-2.118	0.043
R-Sq=0.666		R-Sq(adj)=0.566		

An analysis of variance was also performed to evaluate the variable "computational time". Results summarized in Table 8 reveal a strong influence of all factors over the CPU time. The response variable increments as other factors increase.

Table 8 ANOVA for log Computational Time

Estimated Regression Coefficients for log (computational time)				
Term	**Coef**	**SE Coef**	**T**	**P**
Constant	1.6723	0.010232	163.449	0.000
length	0.5726	0.009412	60.842	0.000
number	0.0497	0.009412	5.277	0.000
perturb	0.0589	0.009412	6.258	0.000
length*length -	0.2412	0.017947	-13.441	0.000
number*number -	0.0445	0.017947	-2.481	0.019
perturb*perturb	0.0058	0.017947	0.325	0.747
length*number	0.0254	0.010523	2.413	0.022
length*perturb	0.0075	0.010523	0.709	0.484
number*perturb	0.0011	0.010523	0.102	0.920
R-Sq=0.993		R-Sq(adj)=0.991		

The ANOVA results reflect a logical trade off between the quality of the response and the time required to achieve a good solution. To deal with this concern the desirability approach was used. The desirability approach is a methodology used for the optimization processes with multiple responses. The methodology assigns a "score" to a set of response variables and chooses factor settings that maximize that score.

In this particular case, the main purpose of this optimization process is minimizing both, the deviation from optima and the time required to perform the algorithm, simultaneously. The "Response Optimizer" in Minitab® was used to identify the combination of values for chain length, number of chains and perturbation size which jointly minimize the percentage cost difference and the CPU time. The limits established for the evaluation of the desirability function were: an upper bound of 2% on the percentage cost difference and 1.8 (i.e. 60 seconds) for the logarithm of the computational time.

As shown in Figure 19, the length of the Markov chains had a significant impact on both responses. The number of chains and perturbation size had an impact only on the percentage cost difference. The optimum coded settings for the length, number and perturbation size are 0.0462, 1, and 0.8035, respectively.

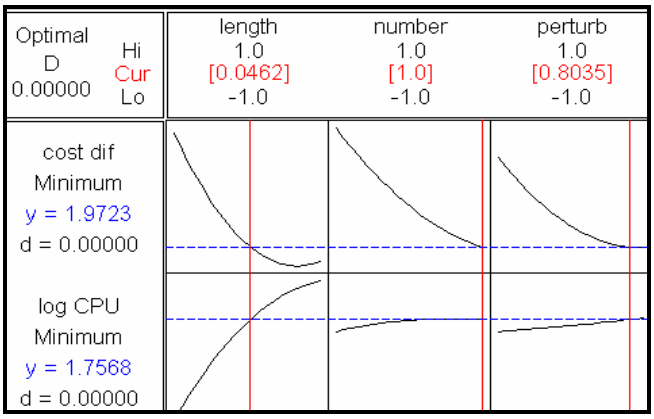

Figure 19 Response Optimizer for Percentage Cost Difference and CPU Time

The uncoded values of the optimal setting are shown in Table 9.

Table 9 Results Response Optimizer

Factor	Setting
Multiplier length of Markov chains	1.819
Multiplier number of Markov chains	3
Perturbation size	4.607

After analyzing all the parameters and defining the most favorable set up for relevant factors it is possible to design a scheme which optimizes resources and achieves high-quality results.

- **Initial Temperature:** The initial Temperature is set using the formula presented below, for a random walk of twenty steps and probability of 0.8 of accepting bad transitions.

$$T_0 = -\frac{\overline{\Delta f}^{(+)}}{\ln(0.8)} \quad , \qquad \text{Equation 4.10}$$

- **Cooling Rule:** After comparing different traditional and adaptive cooling rules, the geometric scheme with $\alpha = 0.95$ was chosen. The cooling rule is as follows:

$$T_{k+1} = 0.95 \cdot T_k \quad , \qquad \text{Equation 4.11}$$

- **Final Temperature:** Simulated Annealing algorithm stops the iterative process after $T_{changes}$ number of Markov chains each one of $L_{\min}$ length are completed. Equations to calculate those parameters has been rewritten as follows:

$$T_{changes} = \text{int}\left(3 \cdot \frac{N \cdot t_{\max}}{c \cdot D}\right) \quad , \qquad \text{Equation 4.12}$$

$$L_{\min} = \text{int}\left(1.819 \cdot \frac{N \cdot t_{\max} \cdot \sqrt{T}}{c \cdot D}\right) \quad , \qquad \text{Equation 4.13}$$

- **System Perturbation:** Neighboring solutions are produced in each trial by making *x* trade or transfer changes, where x is calculated as follows:

$$x = \text{int}\left(4.607 \cdot rand\right) \quad , \qquad \text{Equation 4.14}$$

Rand represents a number between 0 and 1.

4.4 Evaluation and Selection of an Initial Solution

Simulated Annealing is characterized for providing good solutions for combinatorial problems. However, choosing the appropriate parameters for the algorithm is key for success. Several research works confirm that the quality of the initial solution used in the algorithm affects the performance of the SA. Which criteria the user should use for selecting an initial solution and which solution is best are two common concerns on this subject.

Intuitively the user could assume that a good initial solution is one generated using techniques or methodologies validated for the particular problem. In line balancing several decision rules varying from simple rules to complex heuristics have been used to assign tasks to workstations.

Brian Talbot et al. [27] defined four categories to group the decision rules. In the first category they clustered "Single Pass Decision Rules" which consist of simple attribute, priority rules such as the maximum ranked positional weight, maximum number of immediate followers, maximum task time, etc. Those rules consist of a list processing procedures that assigns tasks to work center according to a priority attribute. The second category group, "Composite Decision Rules", is a combination of single pass decision rules. Occasionally, when a tie between two o more tasks occur, a single rule is not able to discriminate among tasks on an available list. Therefore, a combination of rules is used to break the ties. Other complex heuristic rules which require programming an algorithm are clustered in the category of "Backtracking Decision Rules". Finally, optimization methods like branch and bound, integer programming and dynamic programming are clustered in the category for "Optimal-Seeking Decision Rules"

Taking into consideration the importance of the initial solution in the design of an appropriate algorithm, some of the decision rules classified as single pass decision rules along with a heuristic procedure were chosen to generate an initial feasible solution. A brief description of the rules selected is presented below.

4.4.1 Heuristic Rules and Procedures

Single Pass Decision Rules

- Maximum rank positional weight (MaxRPW1): The positional weight of a task is its processing time plus the task time of all following tasks as shown in Equation 4.15.

$$RPW = t_i + \sum_{\forall j \in Si} t_j \qquad \text{Equation 4.15}$$

- Maximum total number of followers (MaxTFOL2): The priority is based on the total number of follower tasks, with a higher priority given to tasks with higher values of MaxTFOL2.

- Maximum task duration (MaxTD3): Tasks are ordered in descending order of task duration time. High duration tasks have a greater priority.

- Minimum task slack (MinSLACK4): The slack of a task is the difference between the upper bound and the lower bound, where upper bound and lower bound are given by:

  where;

$$Slack_i = UB_i - LB_i \quad , \qquad \text{Equation 4.16}$$

$$UB_i = N + 1 - \text{int}_+ \left[\frac{t_i + \sum_{j \in Pi} t_j}{c} \right] \quad , \text{ and} \qquad \text{Equation 4.17}$$

$$LB_i = \text{int}_+ \left[\frac{t_i + \sum_{j \in Si} t_j}{c} \right] \quad , \qquad \text{Equation 4.18}$$

A higher priority is given to tasks with small slack values.

- Random task assignment (Rand5): In this approach, task priority is assigned at random.

The single pass decision rules mentioned above have been included into a task assignment heuristic model designed to balance lines controlling the number of parallel stations generated. The heuristic model assigns tasks to workstations and controls the creation of parallel stations by comparing the current utilization and the utilization of the station with parallels. Parallel workstations are created only when utilization increases with the replication.

First, all the tasks are positioned in a list using a single pass decision rule. A list of assignable tasks is created based on precedence constraints. Then, task *j* is selected from the assignable list, if the processing time of task *j* is smaller than the available time in the station. The task is allocated to the station and the statistics are updated. Otherwise, if the processing time exceeds the available time a new search is performed. The heuristic creates a fitable list picking up from the assignable list all the tasks for which processing times are smaller than the available station time. If the fitable list is not empty, the heuristic changes task *j* with the first in this list and it assigns the new task to the station. On the other hand, when the fitable list is empty, the algorithm evaluates and compares current utilization of the station with utilization of the station with task *j* and parallel

workstations. If the utilization with parallels is greater than a minimum permissible utilization defined by the user, task *j* is allocated into the station and statistics are updated. Otherwise, the station is closed and a new empty station is created.

A flowchart describing this heuristic is presented in Figure 20.

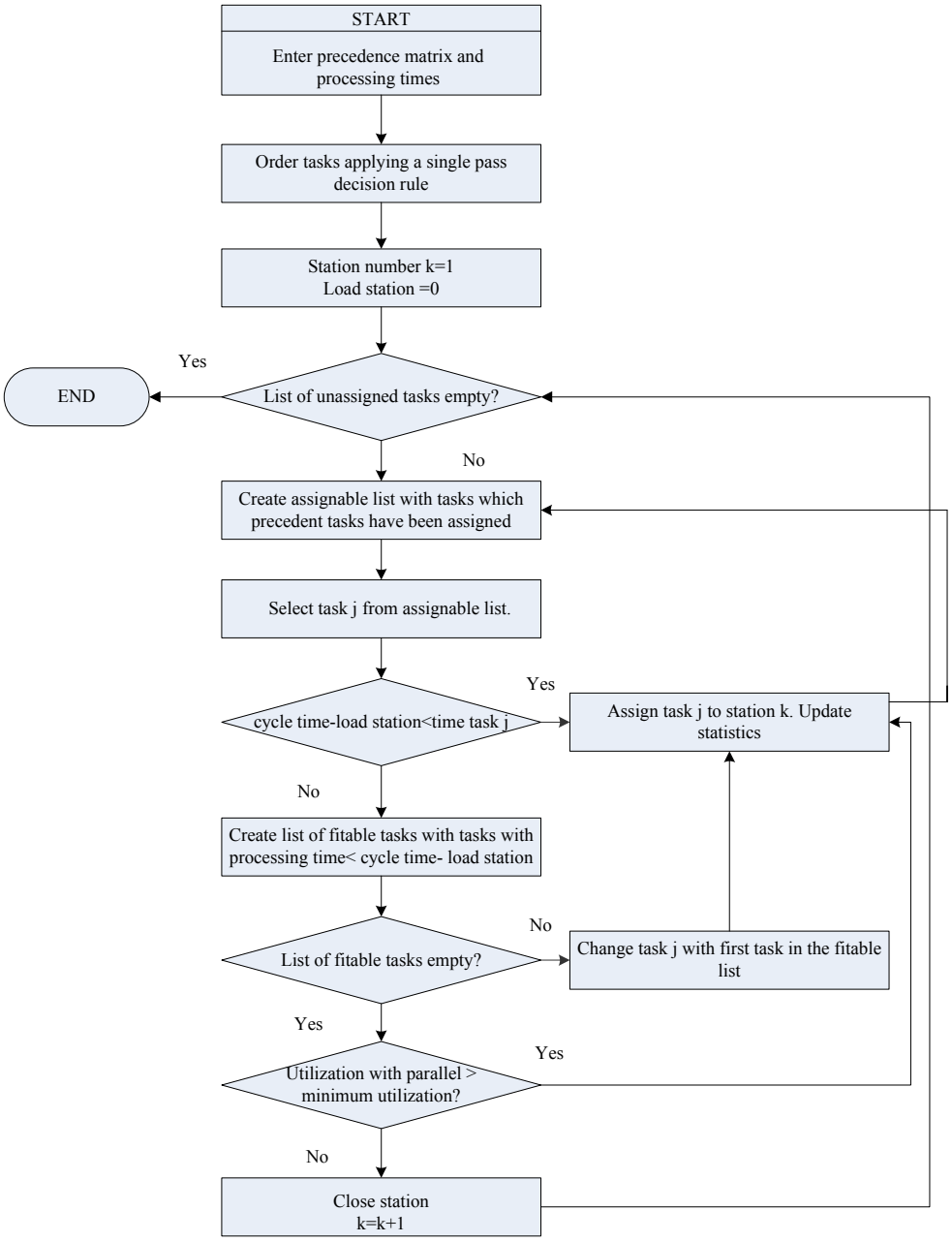

Figure 20 Heuristic to Assign Tasks to Workstations

Heuristic Procedure

The quality of the initial solution provided by a single pass decision rule was also compared with the composite heuristic proposed by Mejía, 2005 [16]. Mejia's heuristic assigns tasks to workstations based on the cost caused by unbalance between stations. The algorithm evaluates costs under different scenarios with or without parallel stations and assigns tasks only if the unbalance cost is reduced.

4.4.2 Comparison of Results

Four different case studies were analyzed in order to evaluate the impact of the initial solution in the SA performance. The five single pass decision rules and Mejía's heuristic model were used to solve the different case studies. Three replications of each experimental condition were run. The results of an analysis of variance, summarized in Table 10, indicate that the initial solution provided to the algorithm impacts the performance of the SA. This conclusion reiterates the findings made by different researchers about the SA performance.

Table 10 ANOVA Results for Initial Solution

Source	DF	SS	MS	F	P
Heuristic Method	5	15.329	3.066	2.920	0.022
Case Studies	3	69.815	23.272	22.130	0.000
Interaction	15	20.885	1.392	1.320	0.225
Error	48	50.472	1.052		
Total	71	156.501			

The results obtained were plotted in the boxplot shown in Figure 21. The results lead to conclude that lower costs are achieved when the initial solution is obtained using maximum task duration priority rule.

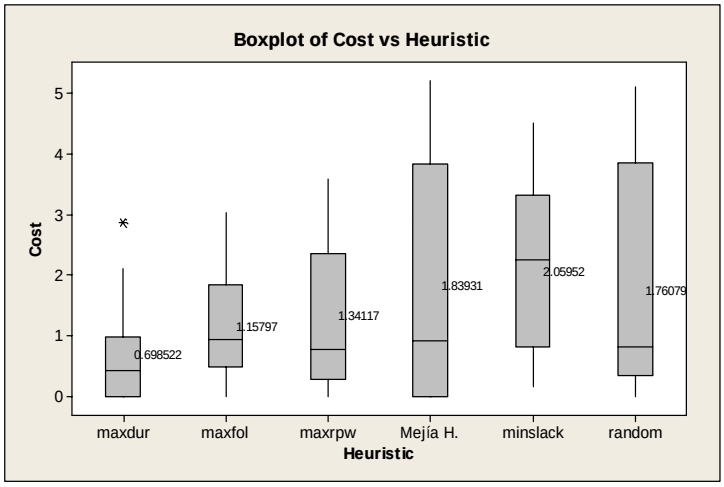

Figure 21 Initial Solution Boxplot

4.5 Final Design Called ANAMAR06

In this chapter a description of the general operation of the proposed algorithm called ANAMAR06 along with the development of the different parameters of this heuristic were presented. The specifications of the algorithm include the selection of an appropriate value for the components of the annealing schedule, the definition of the mechanism to generate a solution from the neighborhood and the size of the perturbation. Additionally, a single pass decision rule combined with a heuristic procedure to assign tasks and generate parallel workstations is employed to generate a good initial solution.

The initial temperature was set using the Van Laarhoven equation. The number of steps in the random walk was fixed to twenty in order to obtain an estimate of the local minima depth.

The size of the algorithm and the computational time spent in finding the final solution rely on the size and complexity of the problem. The schedule is dynamic, adaptive and

depends on factors such as network density, ratio between maximum processing time and cycle time, number of tasks to balance and current temperature. The number and the length of the Markov chains were determined using results from a factorial experiment.

A descriptive flowchart showing the steps of the ANAMAR06 is presented in Figure 22.

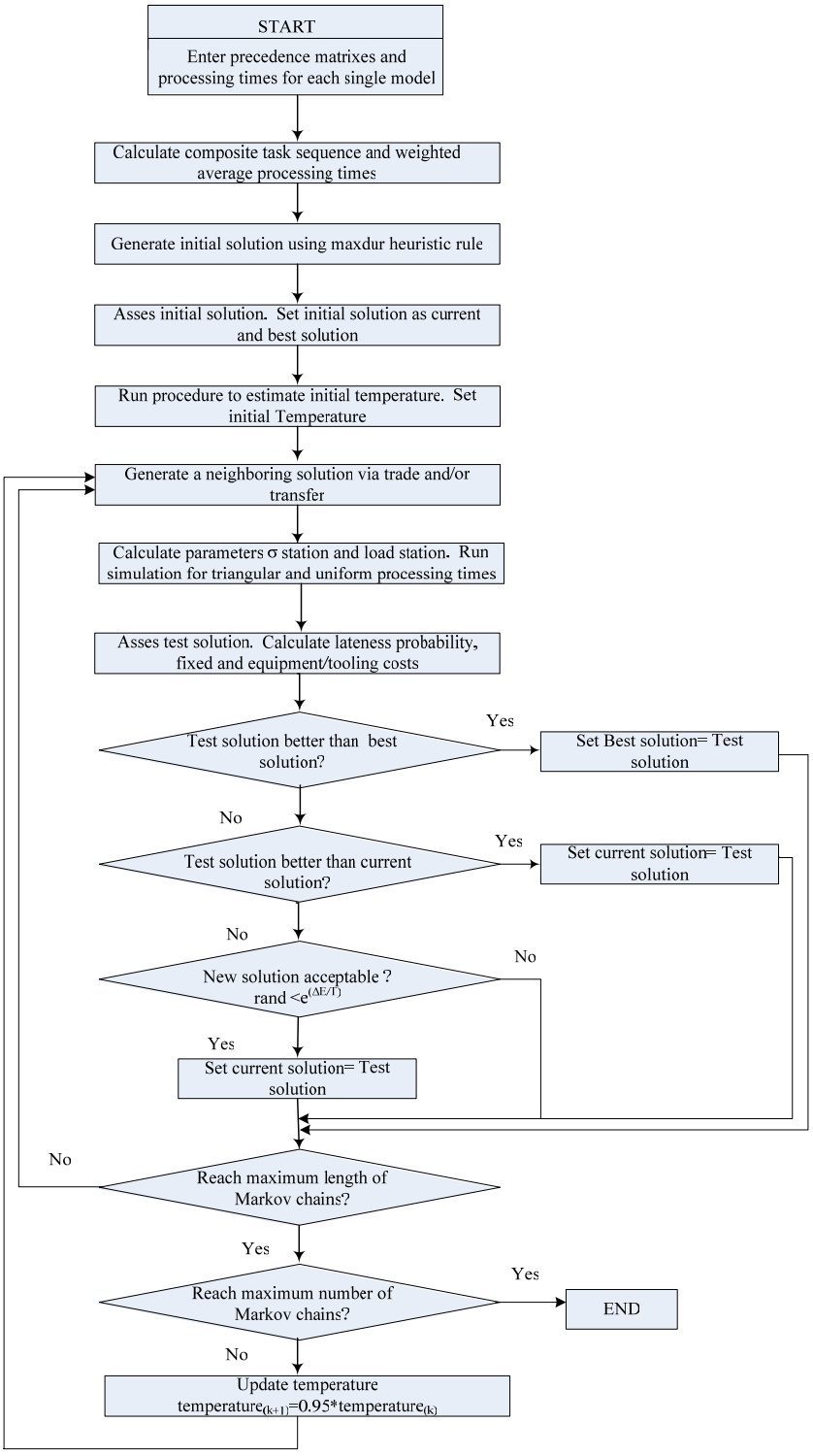

Figure 22 Flowchart Final SA algorithm

5. RESULTS

In this chapter, are presented the computational results from experiments used to evaluate the performance of ANAMAR06. The main objective of this experimental phase is to compare the procedure developed in this thesis work against the mixed integer linear programming model proposed in section 3.3.1 and other widely known line balancing procedures. Moreover, in this chapter an analysis of the impact of diverse variables over the quality of the solution is presented. The variables considered were: total number of tasks, density of the precedence network and the ratio between line cycle and maximum processing time.

The ANAMAR06 algorithm for which results are being compared to those from an optimization model does not include the lateness cost due to the complexity of the non-linearity of the objective function and the absence of equivalent procedures in the literature.

The experiments are performed using a group of problems from the benchmark data sets for ALBP. This data set has been used for testing and comparing solution procedures in several relevant studies during the last two decades. The precedence and the processing times can been downloaded from Scholl and Klein's web page[2]. The specifications of the problem set used in the experiments are summarized in Table 11.

[2] URL address: http://www.assembly-line-balancing.de/

Table 11 Experimental Data Set

Name	Number of tasks	Sum of tasks times	Max task time	Min task time	Max task time / Min task time	Network Density
Mansoor	11	185	45	2	22.5	0.6000
Mitchell	21	105	13	1	6.5	0.7095
Buxey	29	324	25	1	12.5	0.5074
Kilbridge	45	544	55	1	27.5	0.4455
Hahn	53	14056	1775	40	887.5	0.8382
Tonge	70	3510	156	1	78	0.5942

5.1 Performance Evaluation of ANAMAR06

A computational study was completed with the purpose of evaluating the performance of ANAMAR06 based on the quality of the solution. The computational speed of the algorithm is not considered a significant variable in this part of the process. The computational speed is inversely proportional to the quality of the solution. Therefore, the parameters of the algorithm were set up to values resulting in CPU times within a reasonable range of values without compromising the quality of the solution.

All the cases chosen for experimentation were solved using three approaches: (1) Gaither's heuristic [8], (2) a modified version of Amen's [2] single pass decision rule, and (3) the SA algorithm designed in this thesis work a brief explanation of each approach is presented next.

5.1.1 Heuristic Procedures

Gaither's heuristic places a task into a workstation, using different selection rules, only if the utilization of the station increases. The motivation for paralleling is to increase utilization as much as possible. The first relevant step of this heuristic is to create a list of all tasks that are ready for immediate assignment. The tasks placed on this list are

tasks which have their entire predecessors already allocated to workstations. After generating this list of the assignable tasks a second list is created. This second list includes all assignable tasks which increase the utilization of the current work center. If the list of eligible tasks is empty the station is closed, otherwise a task from the second list is chosen and parallel stations are created when needed. Seven task decision rules have been proposed to choose tasks from the eligible list. For experimental purposes only, two decision rules were used: a task that maximizes utilization and a task with the maximum processing time.

Amen's main objective contrasts with Gaither's purpose of maximizing the utilization of the workstation. Amen's single pass decision rule pursues the minimization of the unitary cost per product by minimizing the idle cost. Idle cost is caused by idle time and wage rate differences of the tasks assigned to the same station. Amen assumes that tasks differ in their level of difficulty. Hence there could be differences in the corresponding wage rates of the tasks.

The wage rate at any given station is the maximum of the wage rates for all tasks assigned to the station. The idle cost of the station is therefore calculated by multiplying the maximum wage rate by the station's idle time. Amen's single pass decision rule is called the "Best change in idle cost". In contrast with Gaither's heuristic, Amen's procedure does not generate maximally loaded station, but station resulting in the minimum cost possible.

The best change of idle cost, Δk_i, is calculated as follows:

$$\Delta k_i = \begin{cases} -t_i w_i & if\ w_i \le w_k \\ (w_i - w_k)c - t_i w_i & if\ w_i > w_k \end{cases}, \qquad \text{Equation 5.1}$$

where;

i = task to be assigned to station k,

t_i = processing time for task i,

c= line cycle time,

w_k = wage rate of task i, and

w_i = wage rate of task i**.**

If the wage rate of task i is smaller than the wage rate of the workstation, by assigning task i to station k, there is a reduction in the idle cost which he calculates as $-t_i w_i$. If the wage rate of task i is higher than the workstation wage rate, then w_k becomes w_i. This means that tasks previously assigned, and other which could also be assigned to the workstation, will all be paid at this new higher rate. The idle time is reduced at the expense of increasing the cost of previously assigned tasks and future task assignments. A graphical representation of the priority rule is presented in Figure 23

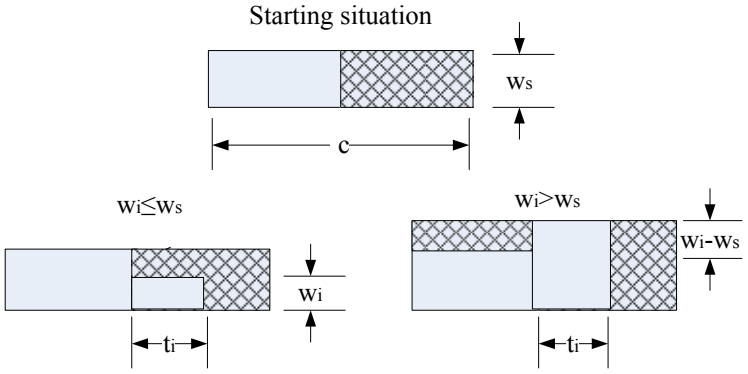

Figure 23 Best Change in Idle Cost

Differences in wage rates, due to differences in level of difficulty of the tasks were not considered in the development of the proposed SA algorithm. However, one major contribution of the proposed algorithm is considering differences in investment costs due to task specific equipment and tooling requirements. Therefore, a change to the rule is proposed to benefit from its merits. The best change in idle cost is calculated as follows:

Case A: No paralleling required

$$\Delta k_i = \begin{cases} -t_i E_k & if\ E_i = 0 \\ E_i c - t_i (E_k + E_i) & if\ E_i > 0 \end{cases} , \qquad \text{Equation 5.2}$$

where;

E_k = Equipment and tooling cost of station k prior to assigning task *i*, and

E_i = Equipment and tooling cost of task *i*.

Case B: With parallel stations

$$\Delta k_i = \begin{cases} -(d_k - d_i) \cdot E_k p & if\ E_i = 0 \\ \left(E_i c - (d_k - d_i) \cdot (E_k - E_i)\right) \cdot p & if\ E_i > 0 \end{cases} , \qquad \text{Equation 5.3}$$

where;

d_k = Station idle time prior to assigning task *i*,

p = Number of parallel stations at serial position *k*,

l_k = Workload at station *k* prior to assigning task *i*, and

d_i = Station idle time after assigning task *i*, per each *c* units of time=

$$d_i = c - \frac{(l_k + t_i)}{p} , \qquad \text{Equation 5.4}$$

An example of each case with positive equipment and tooling cost is presented in Figure 24.

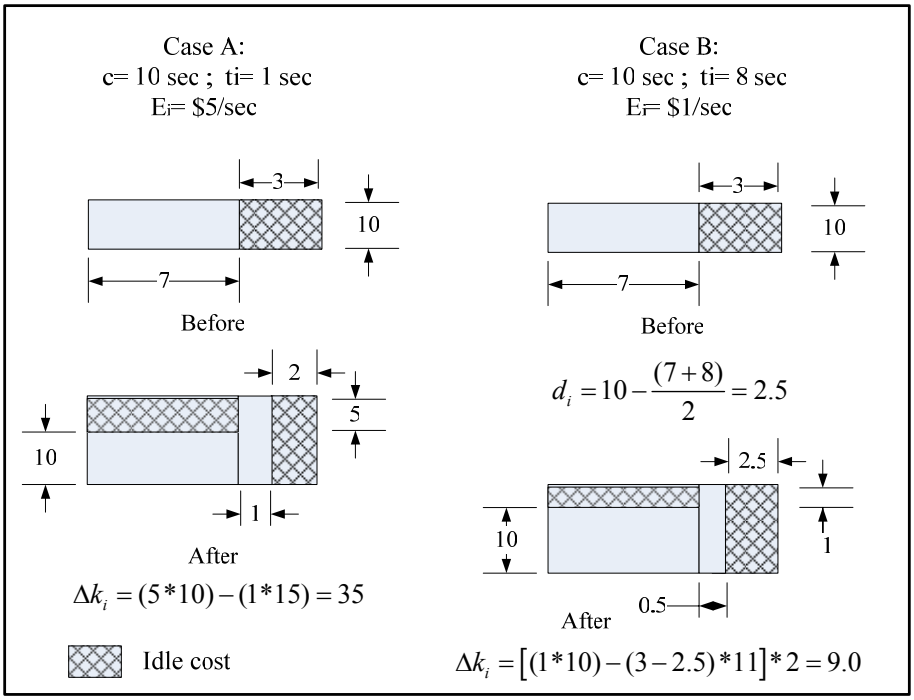

Figure 24 Example "Best Change in Idle Cost"

5.1.2 Optimization Model

Results from both procedures, Gaither's and Amen's, along with results from the SA algorithm were compared with mixed integer linear programming (MILP) model. Reaffirming that the mathematical model proposed in this research cannot be solved using traditional optimization techniques a simplified MILP model is proposed.

The model is based on the following assumptions:

- Processing times are known, deterministic, and independent. Processing times do not fluctuate due to workers expertise.
- Precedence relations between tasks are known.

- Work in process inventory between stations is not allowed.

This model differs from the one explained in section 3.4.2 because it assumes that processing times are deterministic and the lateness cost is excluded from the objective function.

The main purpose of the function presented below is minizing station costs related to the capital investment required to create and operate any workstation and the cost of the tooling and machinery necessary to perform a particular task in a workcenter.

$$Min\ Z = \sum_{k=1}^{K} tp_k \cdot S + \sum_{k=1}^{K}\sum_{i=1}^{n} tp_k \cdot E_i \cdot x_{ik}, \qquad , \qquad \text{Equation 5.5}$$

The mathematical formulation of the problem is summarized as follows:

Minimize Total Cost

$$Min\ Z = \sum_{k=1}^{K} tp_k \cdot S + \sum_{k=1}^{K}\sum_{i=1}^{n} tp_k \cdot E_i \cdot x_{ik}$$

Subject to: Equation 3.12 to Equation 3.18, and Equation 5.6 to Equation 5.6.

$$\sum_{i=1}^{n} \frac{x_{ikp} \cdot t_i}{p} = ld_{pk} \qquad for\ k = 1,...,K\ and\ p = 1,...,P, \qquad \text{Equation 5.6}$$

$$\sum_{p=1}^{P} ld_{pk} = ld_k \qquad for\ k = 1,...,K, \qquad \text{Equation 5.7}$$

$$c - ld_k \geq 0 \qquad for\ k = 1,...,K, \qquad \text{Equation 5.8}$$

The Equations 3.12 to 3.18 are explained in Section 3.4. Capacity limitations are included in Equation 5.6, Equation 5.7, and Equation 5.8. The Equation 5.6 computes

the load of the station k with p parallels, next the final load of the station is stored in variable ld_k which cannot exceed the line cycle time.

The MILP model is solved using branch and bound. Branch and bound technique entails the definition of a group of subproblems where the range of the integer variables is restricted. Upper and lower bounds of the cost function are calculated for nodes or subregions of the problem. The core of the approach is the rule applied to prone nodes and bound the problem. If the lower bound for a subregion X from the search tree is greater than the upper bound for any other subregion Y then A can be discarded from the search.

Obtaining the optimal solution for this model is complex and demands excessive computational time even for moderate sized instances of the problem. The time required finding the optimal solution increases exponentially as the number of tasks and maximum parallels allowed are incremented. The simplest problem with eleven tasks requires 148 restrictions and 260 decision variables for a maximum number of three parallels and a maximum number of workstations equal to the theoretical lower bound calculated using Equation 2.1. Empirically was found that a computational time smaller than 24 hours is only possible when the number of tasks is about twenty or less, and more than three parallels per station are not allowed. This conclusion is drawn after running different case studies in typical computer with a processor of 1 Giga-hertz and 528 Mega bytes RAM memory.

Due to the magnitude and complexity of the optimization problems, it was required to make use of an advanced optimization tool. The set of MILP problems were submitted to

NEOS server[3] to be solved using the SCIP solver. All problems were interrupted after reaching ten hours of computational time, even if an optimum solution was not reached.

In order to restrict the size of the problems the maximum number of parallels for each station was set to three. This additional constraint could cause the generation of sub-optimal solution. Hence, it is possible to find cases where the ANAMAR06 results are better than the ones obtained from the optimization model. Because of the size of the problems, optimal solutions were found only for Mansoor and Mitchell's case studies. Results from MILP are presented in Table 12. No results are presented for Tongue case study at a ratio of 0.5, since after ten hours of computational time MILP did not find any feasible solution.

Table 12 Optimization Results

Case	Number of tasks	Ratio Cycle/Max task Time	Solution $/sec	Optimal solution
Mansoor	11	0.5	0.06337	Yes
		1.0	0.03533	Yes
		1.5	0.02148	Yes
Mitchell	21	0.5	0.11960	Yes
		1.0	0.06333	Yes
		1.5	0.04255	Yes
Buxey	29	0.5	0.22884	No
		1.0	0.11517	No
		1.5	0.07661	No
Kildbridge	45	0.5	0.08944	No
		1.0	0.04486	No
		1.5	0.03162	No
Hahn	53	0.5	0.09769	No
		1.0	0.04725	No
		1.5	0.03542	No
Tongue	70	0.5	NA	No solution
		1.0	0.19454	No
		1.5	0.14354	No

[3] URL address: http://neos.mcs.anl.gov/

ANAMAR06 found good solutions in computational time of less than five minutes. The CPU times for ANAMAR06 are plotted in Figure 25. Those varied from 12 up 260 seconds for the largest case study. This shows that in terms of computational time ANAMAR06 outperformed the optimization model which was not capable of finding an optimal solution in 600 minutes limit for 66% of the case studies.

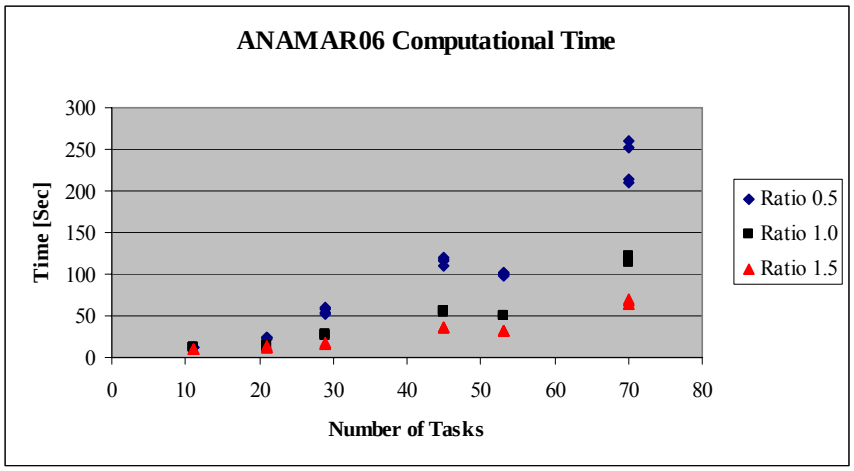

Figure 25 ANAMAR06 Computational Time

Higher CPU times were observed for the cases where parallel stations were required, those with a ratio of 0.5. In general the shortest CPU times were observed for cases not requiring parallel stations, those with a ratio of 1.5.

Figure 26 summarizes the results obtained from ANAMAR06 algorithm. It shows the average percentage difference between results from the optimization model and the SA algorithm. Each case study was run ten times for each of the three levels of ratio between the cycle time and the longest processing time. In all the cases the percentage cost difference do not exceeded 12%. The ANAMAR06 solutions for the case studies with 11 and 21 tasks were at most 1.35% above the optimum solution.

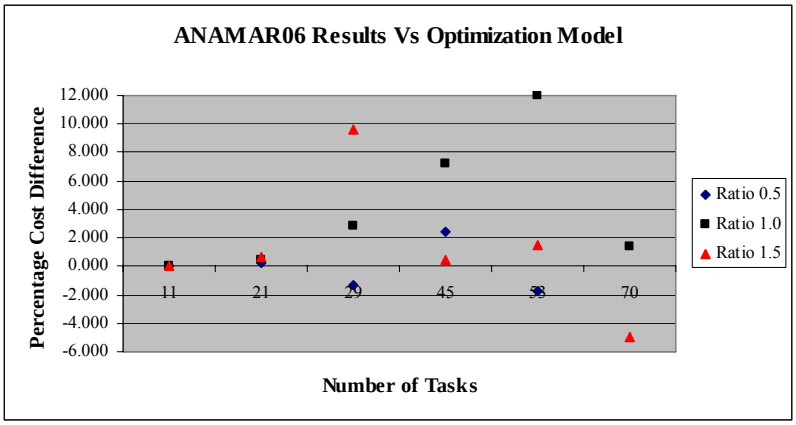

Figure 26 ANAMAR06 Results

The annealing schedule for one of the problems analyzed is plotted in Figure 27. The objective function fluctuates drastically at the beginning of the simulation. The variance of the cost settles down in final stages of the algorithm since the probability of accepting bad transitions decreases significantly as temperatures approaches zero.

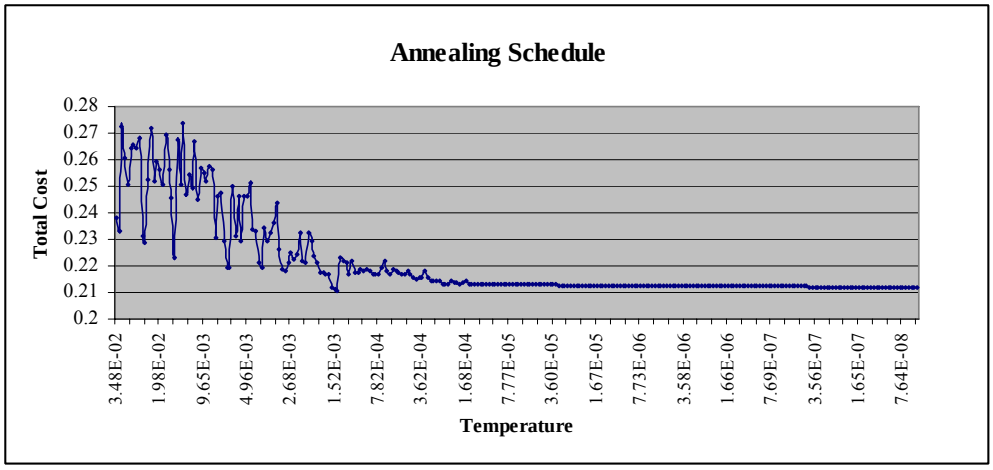

Figure 27 Typical Annealing Schedule

Results for Amen's, Gaither's and the SA algorithm are summarized in Table 13 and are presented graphically in Figure 28 . Higher percentage differences between the heuristic and MILP were obtained in no iterative procedures of Amen and Gaither.

Table 13 Heuristics Performance

Case	Number of tasks	Ratio Cycle/ Max task t	ANAMAR06	Modified Amen	Gaither Max Uti	Gaither Max Dur
Mansoor	11	0.50	0.00000	8.278365	0.80790	0.57279
		1.00	0.00000	1.330314	3.62170	1.33491
		1.50	0.00000	1.596834	32.23483	34.46931
Mitchell	21	0.50	0.28974	7.525084	2.93695	1.28140
		1.00	0.44291	0.484762	0.83860	1.76703
		1.50	0.69695	0.430082	1.16133	1.39635
Buxey	29	0.50	-1.35917	83.51687	20.04930	20.09300
		1.00	2.79455	9.854997	16.40428	78.93055
		1.50	9.55830	18.5511	38.36002	35.82777
Kildbridge	45	0.50	2.46775	5.636181	5.92266	6.38780
		1.00	7.26275	10.23406	10.95232	11.66781
		1.50	0.49146	13.96268	1.19922	4.39016
Hahn	53	0.50	-1.69855	0.475995	-1.02148	-0.87919
		1.00	-1.69855	11.28042	10.44089	19.14654
		1.50	11.98645	4.322417	7.29329	2.57055
Tongue	70	1.00	1.40434	4.055721	5.69250	0.60860
		1.50	-4.91687	41.02689	-3.10834	1.14129

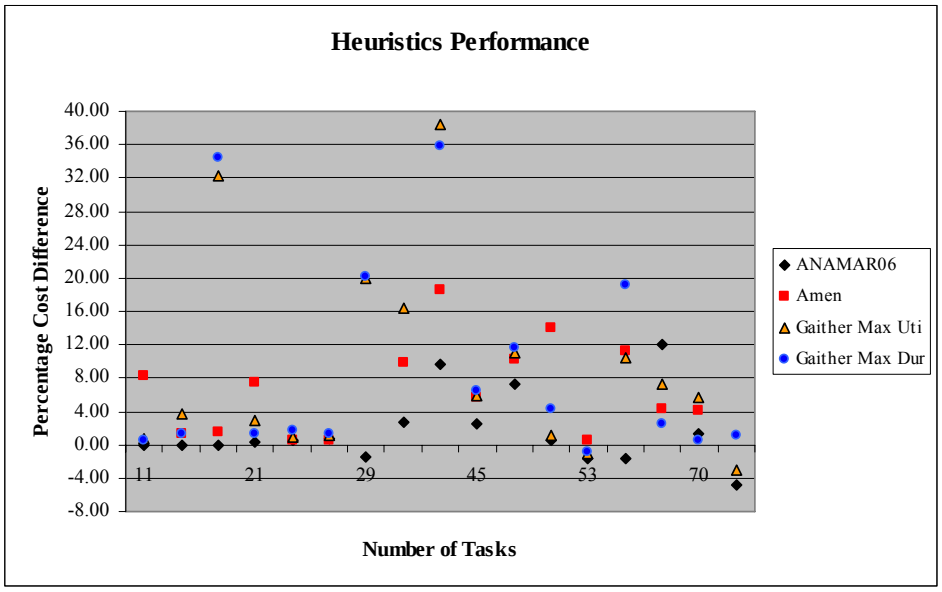

Figure 28 Heuristics Performance

As shown in Table 13, in 88% of the cases the ANAMAR06 outperformed Amen's and 97% of the cases it outperformed Gaither's heuristic. The statistics summarized in Table

14 lead to conclude that ANAMAR06 achieves higher quality solutions with lower variability among the complete data set. In 75% of the cases ANAMAR06 achieved a percentage difference less than 2.5% and in 50% of the cases the percentage difference was less than 0.44%. The highest difference was 12%. The percentage difference representing the 75th percentile in Amen's results is more than 4 times higher than the corresponding value for ANAMAR06.

Table 14 Statistics Summary

Statistics	ANAMAR06	Modified Amen	Gaither Max Uti	Gaither Max Dur
Mean	1.630709	13.091928	9.046234	12.982746
Standard deviation	4.266504	20.630264	11.663771	20.647076
90th Percentile	8.180968	27.541418	24.923514	35.012696
75th Percentile	2.467751	11.280423	10.952318	19.146544
50th Percentile	0.442911	7.525084	5.692496	2.570552

The results obtained reveal that the meta-heuristic proposed in this research work performs better than other procedures when the main objective is minimizing the total cost of the line. Additionally, the computational time and capacity requirements of the algorithm are minimal, allowing a typical user to perform the algorithm in a traditional personal computer.

The results obtained from ANAMAR06, Gaither and the modified Amen correspond to a problem without upper bound on the number of parallel workstations. It is indispensable to perform and additional analysis in order to confirm if there is evidence that leads to conclude that under the same circumstances ANAMAR06 performs favorably compared with optimization.

A set of supplementary tests with a maximum number of three parallels was run. The results on the percentage cost difference are summarized in Table 15. The proposed algorithm outperformed optimization in 18% of the cases, in 90% ANAMAR06 achieve percentage cost differences smaller than 10%.

Table 15 Performance of ANAMAR06 with Upper Bound on Parallel Workstations

Case	Number of tasks	Ratio	SA % cost difference
Buxey	29	0.5	-0.44106
		1.0	2.32567
		1.5	6.97997
Kildbridge	45	0.5	5.19867
		1.0	9.84008
		1.5	0.20304
Hahn	53	0.5	0.07284
		1.0	11.88633
		1.5	3.42985
Tongue	70	1.0	4.55646
		1.5	-5.00814

The statistical results from the bounded ANAMAR06 are presented in Table 16. The variability of the bounded ANAMAR06 was 15% greater to the one obtained in the unrestricted problem. In general, the average difference of the percentage cost was 3.5 for case studies from 29 up to 70 tasks. The 95% confidence interval for the mean of the differences indicates that the solution obtained by ANAMAR06 is 2.6176% to 4.4813% above the cost obtained by optimization. Taking into account the reduced computational time required to perform the Simulated Annealing-based algorithm the results are considered satisfactory.

Table 16 Statistical Summary of ANAMAR06 with Upper Bound on Parallel Workstations

Statistics	Bounded ANAMAR06	Unbounded ANAMAR06
Mean	3.549429	1.630709
Standard deviation	4.931251	4.266504
90^{th} Percentile	10.175719	8.180968
75^{th} Percentile	6.603342	2.467751
50^{th} Percentile	3.362586	0.442911

It is important to emphasize that one of the main characteristics of ANAMAR06 is its capability of solving large and unbounded problems. When the upper bound of three parallels was not included into the algorithm it was able to explore a bigger proportion of the solution space. Hence, better results were obtained from ANAMAR06 without constraints on the number of parallels stations allowed.

5.2 Analysis of Heuristic Robustness Performance

A concern in the development of any heuristic is the robustness under different circumstances. The main interest is determining if the characteristics of the line balancing problem affect the performance of ANAMAR06 and quality of results. The variables that could increase the complexity of the problem are intuitively selected to perform a factorial experiment. It is assumed that the line balancing problem enlarges when the number of tasks is high, the network density is low, meaning that there is a lot of flexibility for assigning tasks to stations, and the ratio is lower than one. As the number of tasks increase, the size of the solution matrix increases enlarging the solution space of the problem. On the other hand, when the network density decreases the number of alternatives to allocate the tasks rises significantly. Moreover the role performed by the ratio between the cycle time and the maximum task is critical in defining the

difficulty level of the balance. If the ratio is lower than one the solution formed by a set of stations without parallels is not feasible and the algorithm is forced to generate parallel workstations and therefore it is necessary to explore a wider solution space.

The list of factors chosen and their experimentation levels are shown in Table 17. The case studies chosen exhibited similar processing times with same station and equipment costs. The response variable was defined as the percentage cost difference between the cost obtained in each experimental condition and the case with the lowest overall resulting cost. In some instances the heuristic results were better than the optimization outcomes due to the limit imposed on the time and the upper bound of three on the number of parallel stations.

Table 17 Factorial Experiment Factors and Levels

Levels	**Factors**		
	A. Number of Tasks	B. Network Density	C. Ratio
Low	21	0.40	0.5
High	45	0.80	1.5

Eight replicates were performed at each combination of factor settings. A logarithmic transformation was employed to validate normality of the data. The results summarized in Table 18. These reveal that the algorithm is sensitive to all the factors analyzed. The conjectures made previously about the variables "number of tasks" and "network density" appears to be correct. The performance of the algorithm decreases when the number of tasks increases and the network density decreases. However, definitive conclusions cannot be drawn before a careful analysis of the main factor interactions is performed.

Table 18 ANOVA for Factorial Experiment

Estimated Effects and Coefficients for Log Percentage Cost Difference					
Term	**Effect**	**Coef**	**SE Coef**	**T**	**P value**
Constant		0.2332	0.01237	18.85	0.00
Number tasks	0.3453	0.1726	0.01237	13.95	0.00
Network density	-0.1948	-0.0974	0.01237	-7.87	0.00
Ratio	0.1411	0.0706	0.01237	5.70	0.00
Number tasks*Network density	-0.2119	-0.1059	0.01237	-8.56	0.00
Number tasks*Ratio	0.1709	0.0854	0.01237	6.91	0.00
Network density*Ratio	-0.2252	-0.1126	0.01237	-9.10	0.00
Number tasks*Network density*Ratio	-0.2091	-0.1045	0.01237	-8.45	0.00
S = 0.0989663		R-Sq = 90.97%		Sq(adj) =89.85%	

Figure 29 shows the relations between the main factors. The slope of the lines confirms that the second order interaction is significant for all the cases although the relation is stronger between "ratio" and "network density". In general, the smallest percentage differences were achieved for the smaller number of tasks, a high network density, and a smaller ratio forcing parallel stations.

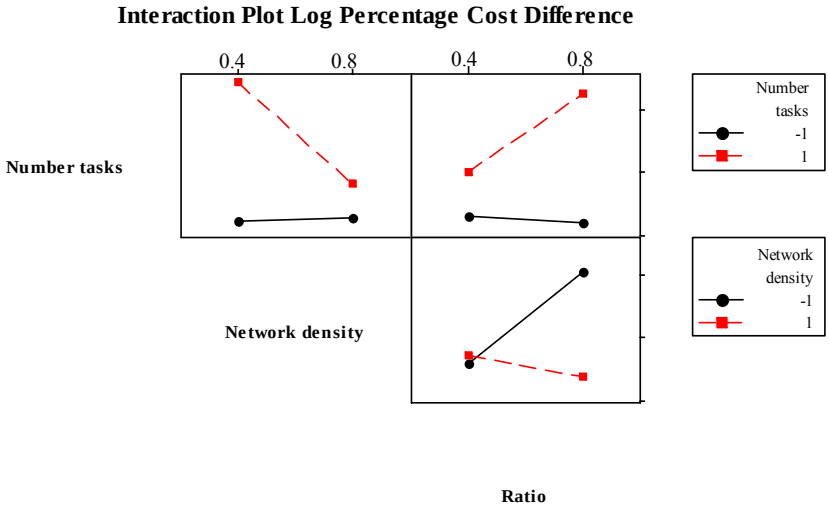

Figure 29 Interaction Plot

With a power of 95% the ANOVA was capable of detecting differences between means as small as 0.23%.

6. SUMMARY AND CONCLUSIONS

6.1 Summary

There is a significant volume of literature on the line balancing problem. However, most of the articles focus on obtaining solutions considering some individual aspects of the general problem. In this research work a comprehensive algorithm was developed to tackle the GALBP for a mixed product line with stochastic processing times and parallel workstations.

A Simulated Annealing-based algorithm named ANAMAR06 was designed in order to provide an accurate solution in a practical computational time to the problem of optimal allocation of tasks minimizing the total cost of the line. The total cost of the line was defined as the sum of a station cost, the equipment cost that relays on the specific task requirements and finally a lateness cost which is a function of variability in task times and the workstations' load. It is considered that all the tasks require different tools and equipment. Therefore, the cost of paralleling depends not only in the cost of the space, table and labor but in the increment of tools and equipment needed to perform any specific task in a paralleled workstation.

Precedence constraints and processing times of the individual models within a family are integrated using composite task sequences and weighted averages. The algorithm allows normal, triangular or uniformly distributed task times and calculates the workstation loads using average times obtained via simulation. To calculate the lateness probability

when task times are either triangular or uniform, an empirical distribution developed forehand is used in conjunction with the chosen unit load size.

The parameters of the annealing structure were defined using results from a set of experiments. A geometric cooling rule and an adaptive cooling schedule that depends on the size of the problem were employed in the final design of the algorithm.

Five single pass decision rules and the Mejía's Heristic were employed in order to analyze the impact of the initial solution on the performance of ANAMAR06 algorithm. This experimental process leaded to the selection of an adequate procedure to provide the initial balance to ANAMAR06.

The performance of the algorithm was evaluated based on the cost percentage difference from the optimal solution and the computational time required. The MILP was solved using branch and bound with a specified computational time limit and an upper bound on the number of parallels. For 66% of the cases the MILP did not find the optimal solution in the time limit.

A factorial experiment was conducted in order to prove the hypothesis problem characteristics such as the number of tasks, network density and cycle time ratio had an impact on the performance of ANAMAR06.

6.2 Conclusions

The problem analyzed in this research deals with modeling and offering a solution to a real concern of the modern manufacturing industry. The mathematical formulation of this

problem becomes extremely complex and cannot be solved with traditional optimization.

A large volume of literature on Simulated Annealing corroborates the capability of the meta-heuristic to solve models when the complexity of the problem makes difficult the use of traditional optimization methodologies. Nevertheless, there are no specific guidelines on the selection of the design parameters. The correct selection of the cooling schedule and the solution generation mechanism is critical to guarantee the success of the algorithm since those elements define the size and direction of the search performed by the SA.

By using experimentation through the entire design process it was assured settings of each parameter which contributes to maximize the performance of the proposed algorithm. Completing a small random walk to avoid underestimating the initial temperature, employing a geometric cooling rule that do not decrease drastically the probability of accepting bad solutions and identifying a number of chains dependent on the size of the problem were some of the issues addressed in the design stage.

Furthermore, it was analyzed the role of the mechanism to generate neighboring solutions and the size of the perturbation. The mechanism must assure that all feasible changes in the solution matrix are possible and the perturbation must be large enough to help preventing the algorithm to get trapped in local optima.

A tool was developed in Matlab® to facilitate obtaining an initial line balance with heuristic rules found in the literature review. It was demonstrated through experimentation that the initial solution from the different rules had a significant effect on

the algorithm performance. The maximum duration rule generated more accurate results.

The main contribution of this thesis work is the development of an algorithm that solves the cost oriented GLBP, achieving satisfactory solutions in reduced computational time. The algorithm designed performs better than others non iterative widely known procedures. In 88% of the cases ANAMAR06 outperformed Amen's modified heuristic and in 97% of the times outperformed Gaither's heuristic. In 75% of the cases the percentage difference from the optimum was smaller than 2.46% for the unbounded ANAMAR06 on the number of parallel workstations.

6.3 Future Work

This research addresses the balancing problem for a serial line where mix product is manufactured. It is assumed that the models manufactured in the line do not exhibit significant differences in processing times or precedence restrictions. Therefore they are grouped into a product family. The composite task sequence and weighted average time are employed to integrate the individual precedence constraints and the processing times. However, in this thesis is not presented a study of the adequacy of this methodology. It is proposed to perform a further research to analyze the limit conditions to use the composite task sequence for an efficient mix-product line balance. This analysis may conclude in the definition of acceptable time differences, acceptable precedence differences or basic rules for the appropriate use of CTS and weighted average times.

When stations may work at two segments of the line, the line becomes a U-shaped assembly line. In a U-shape line the precedence constraints are not as restrictive as in a

serial line. Therefore, the U-shape exhibits higher flexibility. It is proposed to offer an approach to solve the U-shape line balancing problem for a production scenario characterized by stochastic processing times, mix product and parallel workstations.

Another possible extension of the research work presented in this document is the utilization of SA to solve the balancing problem for a multi product line. In a multi product line the set up times are not negligible and the batch size performs an important role into the optimization model. Therefore, the problem is extended from the optimal assignment of the resources to the optimal use of them. This model entails allocating tasks into workstations, defining the batch size and the process sequence of the models.

REFERENCES

[1] Amen, M., An exact method for cost- oriented assembly line balancing. International Journal of Production Economics. Vol 64. P. 187-195. 2000.

[2] Amen, M., Heuristic methods for cost-oriented assembly line balancing: A comparison on solution quality and computing time. International Journal of Production Economics. Vol 69. P. 255-264. 2001.

[3] Arcus, A.L. COMSOAL: A computer method of sequencing operations for assembly lines. International Journal of Production Research. Vol 4. P. 259-277. 1966.

[4] Askin R., and Zhou, M., A parallel station heuristic for the mixed-model production line balancing problem. International Journal of Production Research. Vol 35(11). P. 3095-3105. 1997.

[5] Becker, C., and Scholl, A., A survey on problems and methods in generalized assembly line balancing. Invited review for the special issue "Balancing of automated assembly and transfer lines" of the European Journal of Operational Research. 2003.

[6] Dar-El (Mansoor), Solving large single model assembly line balancing problems- a comparative study. AIIE Transactions. Vol 7. P. 302-315. 1974.

[7] Erel, E., Sabuncuoglu, I., and Sekerci, H., Balancing of U-type assembly systems using Simulated Annealing. International Journal of Production Research. Vol 39. P. 3003-3015. 2001.

[8] Gaither, N., Production and Operations Management, 7th ed. Duxbury Press, Boston. 1996.

[9] Ghosh, S., and Gagnon, R.J. A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. International Journal of Production Research. Vol. 27(4). P. 637-670. 1989.

[10] Hoffmann, T., Eureka., A hybrid system for assembly line balancing. Management Science. Vol. 38(1). P. 39-47.1992.

[11] Huang, M., Romeo, F. and Sangiovanni-Vincentelli, A., An efficient general cooling schedule for Simulated Annealing. Proceedings of the IEEE International Conference on Computer-Aided Design Santa Clara. P. 381-384. 1986.

[12] Kara, Y., and Ozcan, U., A Simulated Annealing approach for balancing and sequencing of mixed-model U-lines. 2004.

[13] Kirkpatrick, S., Gelatt C., and Vecchi, M., Optimization by Simulated Annealing, Science. Vol 220, No 4598. P. 671-680. 1983.

[14] Klein, R., and Scholl, A., Maximizing the production rate in simple assembly line balancing – A branch and bound procedure. European Journal Operational Research. Vol 91. P. 367-385. 1996.

[15] McMullen, P., and Frazier, G., Using Simulated Annealing to solve a multiobjective assembly line balancing problem with parallel workstations. International Journal of Production Research. Vol 36. P. 2717-2741. 1998.

[16] Mejia, H., Minimización de los costos totales en el problema de balanceo de línea con ciclo variable y estaciones en paralelo. 2005.

[17] Merengo, C., Nava, F., and Pozzettis, A., Balancing and sequencing manual mixed-model assembly lines. International Journal of Production Research. Vol 37(12). P. 2835-2860. 1999.

[18] Metropolis, N., Rosenbluth, A., Rosenbluth, M., and Teller, A., Equation of state calculations by fast Computing machines. Journal of Chemical Physics. Vol 21. P. 1087-1092. 1953.

[19] Pinto, P.A., Dannenbring, D.G., and Khumawala, B.M., Branch and bound and heuristic procedures for assembly line balancing with parallel of stations. International Journal of Production Research. Vol 19. P. 565-576. 1981.

[20] Randelman, R.E., and Grest, G.S., N-City Traveling Salesman Problem-Optimization by Simulated Annealings. Journal of Statistical Physics. Vol 45. P. 885-890. 1986.

[21] Reeve, R., and Thomas, W., Balancing stochastic assembly lines. AIIE Transactions. Vol 5(3). P. 223-229. 1973.

[22] Sarin, S., and Erel, E., Development of cost model for the single model stochastic assembly line balancing problem. International Journal of Production Research. Vol 28(7). P. 1305-1316. 1990.

[23] Scholl, A., and Becker, C., A note on "An exact method for cost-oriented assembly

line balancing. Jenaer Schriften Zur Wirtschaftswissenschaft ,FSU Jena. 2003b.

[24] Scholl, A., and Klein, R., SALOME: A bidirectional branch and- bound procedure for assembly line balancing. INFORMS Journal on Computing. Vol 9. P. 319-334. 1997.

[25] Scholl, A., and Klein R., Balancing assembly lines effectively. A computational comparison. European Journal of Operational Research. Vol 114. P. 50-58. 1999.

[26] Suresh, G., and Sahu, S., Stochastic assembly line balancing using Simulated Annealing. International Journal of Production Economics. Vol 32(8). P. 1801-1910. 1994.

[27] Talbot , F., Patterson, J. and Gehrlein, W., A comparative evaluation of heuristic line balancing techniques. Management Science. Vol 32. P. 430-436. 1986.

[28] Triki, E., Collette, Y., and Siarry, W., A theoretical study on the behavior on Simulated Annealing leading to a new cooling schedule. European Journal of Operational Research. Vol 166. P. 77-92. 2005.

[29] Van Laarhoven, P., and Aarts, E., Simulated Annealing: Theory and Applications. Kluwer Academic Publishers. 1997.

[30] Vilarinho, P., and Simaria, S., A two-stage heuristic method for balancing mixed-model assembly lines with parallel workstations. International Journal of Production Research. Vol 40 (6). P. 1405-1420. 2002.

APPENDIXES

Appendix A: Simulated Annealing Main Algorithm

This appendix contains the code of the main routine used to solve the line balancing problems. The code has been constructed in Matlab® 7.0. The algorithm invokes other routines that perform specific process.

```
SA ALGORITHM FOR LINE BALANCING WITH PARALLEL WORKSTATIONS%

%-------------------------------------------------------------------------

%RUN SOLINICIALMIXT SUBROUTINE TO FIND AN INITIAL SOLUTION
solinitialmixt;

% SOLUTION STORED IN VARIABLES “stations” and “parallels”
%-------------------------------------------------------------------------

% COMPUTE OBJECTIVE FUNCTION FOR INITIAL SOLUTION%

loads=zeros(size(parallels,1),1);
costA=zeros(size(parallels,1),1);
for i=1:size(stations,2)
    for j=1:n
        if stations(j,i)>0;
            costA(i)=costA(i)+costequipment(j,1);%*60;
            loads(i)=loads(i)+TI2(stations(j,i));
        end
    end
end

%-------------------------------------------------------------------------
latenessmixt; %RUNS LATENESS SUBROUTINE TO FIND LATENESS COST

% LATENESS PROBABILITY STORED IN VARIABLE “probability”
%-------------------------------------------------------------------------

coststot=0;
for i=1:size(stations,2)
      coststot=coststot+parallels(i)*(costL+costA(i)+costlab)+
      costlateness*probatility
end
%-------------------------------------------------------------------------

% PERFORMS ROUTINE TO FIND INITIAL TEMPERATURE
```

initemp

```
%INITIAL TEMPERATURE STORED IN VARIABLE “Temperaturei”
%-------------------------------------------------------------------------
Temperature=Temperaturei;

%-------------------------------------------------------------------------
% INITIALIZE SIMULATED ANNEALING STATISTICS%

Eb=coststot;
Ec=coststot;
Sb=stations;
stationst=stations;
loadt=loads;
parallelst=parallels;
tri=1;%trials counter%
trial=1;
failedchain=0;
failed=0;

%-------------------------------------------------------------------------

density=(size(find(triu(PRE2,1)==1),1))/(n*(n-1)*0.5);

% TOTAL MARKOV CHAINS%
numbertri=ceil(3*n*max(TI2)/(density*cycletime));

while tri<=numbertri
    % MAXIMUM TEMPERATURE CHANGES%
      t=1;
% MARKOV CHAINS LENGTH
   length=ceil(1.819*sqrt(Temperature)*n*max(TI2)/(density*cycletime));

    while t<=length && failed<(0.5*length);
        failed=0;
      %MAXIMUM NUMBER OF ITERATIONS FOR EACH TEMPERATURE%
        shakes=ceil(rand*4.607);
        totshakes=0;

        while totshakes<shakes;
            mutation=ceil(rand*2);
            c=1;

            while c==1;
                random0=ceil(rand*size(stationst,2));
                %STATION TO MUTATE%
                random1=ceil(rand*size(stationst,1));
                 %TASK TO TRANSFER
                if stationst(random1,random0)>0;
                    break;
                end
            end

%-------------------------------------------------------------------------

      %FIND NEIGHBORING SOLUTION VIA TRADE OR TRANSFERING
```

```
            if mutation==2;

      %PERFORMS ROUTINE TO FIND A NEIGHBORING SOLUTION USING TRADE
            trade;

            else;

      %PERFORMS ROUTINE TO FIND A NEIGHBORING SOLUTION USING TRANSFER
            transfer;

      % NEIGHBOURING SOLUTION STORED IN VARIABLES "stationst" AND
       "parallelst"

            end
            totshakes=totshakes+1;
        end
%-------------------------------------------------------------------------

      %RUNS SUBROUTINE TO CALCULATE COSTS
        fcostosmixt

      % NEW COST FUNCTION VALUE STORED IN VARIABLE "costtest"
%-------------------------------------------------------------------------

        %SIMULATED ANNEALING UPDATE
         Et=costtest;
        %UPDATE SA STATISTICS
        if Et<Eb;
            Eb=Et;
            Sb=stationst;
            Ec=Et;
            Sc=stationst;
            parallelsb=parallelst;
            loadb=loadt;
            failed=0;
            failedchain=;
        else;
            failed=failed+1;
        end
        if Et<=Ec;
            Ec=Et;
            Sc=stationst;
            loadc=loadt;
            parallelsc=parallelst;

        else Et>Ec;

%-------------------------------------------------------------------------
            %METROPOLIS CRITERION EVALUATION%
            f=exp(-(Et-Ec)/Temperature);

            if rand<f;
            %STORE STATISTICS
                Ec=Et;
                Sc=stationst;
                loadc=loadt;
```

```
                parallelsc=parallelst;
            end
%-------------------------------------------------------------------------
        end
        stationst=Sc;
        t=t+1;
        trial=trial+1;
    end
    Temperature=Temperature*0.95;
    tri=tri+1;
end

end
```

Appendix B: Experimental Data Set

This appendix contains the data set used to evaluate the results of the algorithm. The processing times and precedence constraints are taken from Scholl and Klein web site http://www.wiwi.uni-jena.de. Equipment, labor and space costs are taken from Puerto Rican case studies.

Mansoor Case Study

Task	Process time (sec)
1	4
2	38
3	45
4	12
5	10
6	8
7	12
8	10
9	2
10	10
11	34

Precedence relations	
a	b
1	4
2	4
2	5
3	11
4	6
5	7
6	8
7	9
8	10
9	10
10	11

Task	Equipment Cost $/sec
1	0.0000000
2	0.0001139
3	0.0000000
4	0.0000000
5	0.0000813
6	0.0001437
7	0.0000054
8	0.0000000
9	0.0000000
10	0.0001355
11	0.0002169

Space $/sec	Station $/sec	L Cost $/ sec	Labor Cost $/sec
0.0010542	0.0000081	0.0010623	0.0058650

Mitchell Case Study

Task	Process time (sec)
1	4
2	3
3	9
4	5
5	9
6	4
7	8
8	7
9	5
10	1
11	3
12	1
13	5
14	3
15	5
16	3
17	13
18	5
19	2
20	3
21	7

Precedence relations	
a	b
1	2
1	3
2	21
3	4
4	5
4	21
5	6
5	7
6	8
7	8
7	14
8	9
9	10
9	11
9	12
9	13
10	15
11	15
12	15
13	17
13	18
14	19
15	16
15	18
16	17
17	20
18	19

Task	Equipment Cost/sec	Task	Equipment Cost/sec
1	0.0002711	12	0.0000813
2	0.0000000	13	0.0000000
3	0.0000000	14	0.0000054
4	0.0000000	15	0.0000136
5	0.0000949	16	0.0000244
6	0.0000271	17	0.0001220
7	0.0001355	18	0.0000217
8	0.0000271	19	0.0000000
9	0.0000000	20	0.0000000
10	0.0000000	21	0.0000271
11	0.0001355		

Space $/sec	Station $/sec	L Cost $/ sec	Labor Cost $/sec
0.0010542	0.0000081	0.0010623	0.0058650

Buxey Case Study

Task	Process time (sec)
1	7
2	19
3	15
4	5
5	12
6	10
7	8
8	16
9	2
10	6
11	21
12	10
13	9
14	4
15	14
16	7
17	14
18	17
19	10
20	16
21	1
22	9
23	25
24	14
25	14
26	2
27	10
28	7
29	20

Precedence relations	
a	b
1	3
1	25
2	6
2	26
3	4
4	5
5	8
5	13
6	8
7	9
7	12
7	25
8	11
8	16
9	10
10	14
10	15
11	17
12	15
13	17
14	16
15	19
16	18
17	20
18	22
19	21
20	23
21	22
22	23
23	24
23	28
24	29
25	29
26	27
27	29
28	29

Task	Equipment Cost $/sec
1	0.0001241
2	0.0002607
3	0.0000000
4	0.0000000
5	0.0001862
6	0.0003290
7	0.0000124
8	0.0000000
9	0.0000000
10	0.0003103
11	0.0004966
12	0.0006207
13	0.0000000
14	0.0000000
15	0.0000000
16	0.0018621
17	0.0006207
18	0.0018621
19	0.0006207
20	0.0000000
21	0.0000000
22	0.0003103
23	0.0001862
24	0.0000000
25	0.0000124
26	0.0000310
27	0.0003103
28	0.0002793
29	0.0001862

Space $/sec	Station $/sec	L Cost $/sec	Labor Cost $/sec
0.0024138	0.0000186	0.00510000	0.0024324

Kildbridge Case Study

Task	Process time (sec)
1	1
2	9
3	10
4	10
5	17
6	17
7	13
8	13
9	20
10	20
11	10
12	11
13	6
14	22
15	11
16	19
17	12
18	3
19	7
20	4
21	55
22	14
23	27
24	29
25	26
26	6
27	5
28	24
29	4
30	5
31	7
32	4
33	15
34	3
35	7
36	9
37	4
38	7
39	5
40	4
41	21
42	12
43	6
44	5
45	5

Precedence relations			
a	b	a	b
1	3	17	27
1	7	18	19
2	4	19	20
2	8	19	33
3	5	20	21
4	6	21	22
5	9	22	28
6	10	23	33
7	9	24	33
7	14	25	26
8	10	26	38
8	14	27	28
9	41	27	33
10	41	28	38
11	13	29	41
12	13	30	41
12	37	31	41
13	14	32	41
13	15	33	34
14	17	33	35
14	25	33	36
14	29	34	38
14	30	35	40
14	31	36	38
14	32	37	43
15	16	38	40
15	18	39	41
15	23	40	41
15	24	41	42
16	19	42	44
17	26	42	45

Task	Equipment Cost $/sec
1	0.0000000
2	0.0000000
3	0.0000419
4	0.0000000
5	0.0000000
6	0.0000000
7	0.0000734
8	0.0000000
9	0.0000000
10	0.0000000
11	0.0001048
12	0.0000000
13	0.0000000
14	0.0000881
15	0.0000105
16	0.0000315
17	0.0000000
18	0.0000189
19	0.0000000
20	0.0000419
21	0.0000000
22	0.0000000
23	0.0000000
24	0.0000210
25	0.0000000
26	0.0000000
27	0.0000000
28	0.0000000
29	0.0001048
30	0.0000000
31	0.0000629
32	0.0000000
33	0.0000000
34	0.0000210
35	0.0000000
36	0.0000000
37	0.0000315
38	0.0000000
39	0.0000419
40	0.0000000
41	0.0000189
42	0.0000000
43	0.0000000
44	0.0000063
45	0.0000000

Space $/sec	Station $/sec	L Cost $/sec	Labor Cost $/sec
0.00093196	0.00000629	0.00093826	0.00347625

Hahn Case Study

Task	Process time (sec)	Task	Process time (sec)
1	971	28	69
2	142	29	99
3	142	30	70
4	142	31	70
5	103	32	158
6	96	33	191
7	99	34	70
8	1207	35	53
9	160	36	50
10	180	37	125
11	82	38	353
12	60	39	70
13	112	40	128
14	420	41	65
15	1556	42	1775
16	236	43	91
17	259	44	91
18	125	45	113
19	601	46	487
20	80	47	138
21	80	48	80
22	70	49	80
23	89	50	65
24	89	51	40
25	105	52	742
26	330	53	1085
27	132		

Precedence relationships			
a	b	a	b
1	2	28	29
1	3	28	30
1	4	28	31
1	5	28	32
1	6	28	33
1	7	28	34
2	36	29	35
3	36	30	35
4	9	31	35
5	9	32	35
6	9	33	35
7	9	34	35
8	9	35	36
9	10	36	37
10	11	37	38
11	12	37	39
12	13	37	40
12	14	38	41
12	15	39	41
13	16	40	41
13	17	41	42
13	18	42	43
14	22	42	44
15	16	42	45
15	17	42	46
16	29	42	47
16	30	43	48
16	31	44	49
17	19	45	51
18	42	45	52
19	20	46	50
19	21	47	51
19	22	47	52
20	23	48	51
21	24	48	52
22	25	49	51
23	36	49	52
24	36	50	51
25	26	50	52
26	27	51	53
27	28	52	53

Task	Equipment Cost $/sec	Task	Equipment Cost $/sec
1	0.0000000	28	0.0002711
2	0.0000000	29	0.0000000
3	0.0000000	30	0.0000000
4	0.0000000	31	0.0001355
5	0.0000000	32	0.0000813
6	0.0000000	33	0.0000000
7	0.0000407	34	0.0000054
8	0.0002169	35	0.0000136
9	0.0000244	36	0.0001355
10	0.0000000	37	0.0001220
11	0.0000244	38	0.0000813
12	0.0000000	39	0.0000000
13	0.0001627	40	0.0000000
14	0.0000000	41	0.0000271
15	0.0000000	42	0.0000244
16	0.0000000	43	0.0000000
17	0.0000000	44	0.0000244
18	0.0000000	45	0.0000000
19	0.0000000	46	0.0001627
20	0.0000000	47	0.0000000
21	0.0002711	48	0.0000000
22	0.0000000	49	0.0000000
23	0.0000000	50	0.0000000
24	0.0000000	51	0.0000000
25	0.0008133	52	0.0000000
26	0.0000271	53	0.0000000
27	0.0001355		

Space $/sec	Station $/sec	L Cost $/sec	Labor Cost $/sec
0.00105422	0.00000813	0.00106235	0.00437500

Tongue Case Study

Task	Process time (sec)	Task	Process time (sec)
1	17	36	40
2	66	37	2
3	54	38	1
4	52	39	3
5	6	40	13
6	88	41	16
7	21	42	25
8	128	43	21
9	68	44	43
10	70	45	30
11	85	46	83
12	21	47	89
13	134	48	56
14	135	49	59
15	94	50	43
16	90	51	11
17	50	52	26
18	143	53	44
19	19	54	121
20	54	55	38
21	50	56	68
22	40	57	22
23	73	58	7
24	12	59	16
25	152	60	32
26	42	61	25
27	45	62	27
28	74	63	156
29	26	64	28
30	11	65	15
31	31	66	26
32	50	67	18
33	102	68	72
34	46	69	23
35	35	70	27

Precedence relationships			
a	b	a	b
1	2	28	35
1	41	29	35
1	69	30	31
1	70	31	32
2	3	32	35
3	4	33	34
3	68	34	35
4	6	35	36
4	7	35	44
5	6	35	48
5	24	35	51
5	30	35	53
6	8	35	56
7	8	35	60
8	12	35	61
9	10	35	62
10	11	36	37
11	12	37	38
12	13	38	39
12	14	39	40
13	23	40	42
14	23	41	42
15	16	42	43
16	17	43	50
16	18	44	45
17	19	45	46
18	19	46	47
19	20	47	50
19	22	48	49
19	57	49	50
20	21	51	52
21	23	52	54
22	23	53	54
23	25	54	55
23	31	57	58
23	33	58	59
24	25	59	60
25	26	61	65
25	27	62	63
25	28	63	64
25	29	64	65
26	35	64	66
27	35	64	67

Task	Equipment Cost $/sec	Task	Equipment Cost $/sec
1	0.00000000	36	0.00013554
2	0.00000000	37	0.00008133
3	0.00000000	38	0.00000000
4	0.00004066	39	0.00000000
5	0.00000000	40	0.00000000
6	0.00000000	41	0.00000000
7	0.00000000	42	0.00002711
8	0.00000000	43	0.00000000
9	0.00000000	44	0.00000000
10	0.00005422	45	0.00000000
11	0.00000000	46	0.00000000
12	0.00000000	47	0.00013554
13	0.00000000	48	0.00000000
14	0.00000000	49	0.00008133
15	0.00000000	50	0.00000000
16	0.00000000	51	0.00000000
17	0.00000000	52	0.00002711
18	0.00000000	53	0.00000000
19	0.00000000	54	0.00000000
20	0.00000000	55	0.00004066
21	0.00000542	56	0.00000000
22	0.00001355	57	0.00005422
23	0.00000000	58	0.00000000
24	0.00000000	59	0.00002440
25	0.00000000	60	0.00000000
26	0.00027108	61	0.00000000
27	0.00000000	62	0.00000813
28	0.00000000	63	0.00000000
29	0.00000000	64	0.00001355
30	0.00009488	65	0.00000542
31	0.00002711	66	0.00000000
32	0.00013554	67	0.00000000
33	0.00002711	68	0.00002440
34	0.00000000	69	0.00009488
35	0.00000000	70	0.00005422

Space $/sec	Station $/sec	L Cost $/sec	Labor Cost $/sec
0.00105422	0.00000813	0.00106235	0.00729167

Appendix C: User Manual

USER MANUAL

The SA for line balancing interface was developed in Matlab® 7.0. This graphical user interface provides a windows environment with four option menus.

1. Load
   - Load precedence matrix
   - Load processing times
   - Load models participation
2. Calculate
   - Calculate general variance
   - Calculate composite task sequence
3. Analysis
4. Run SA

Getting Started

To perform a line balance is required to perform some sequential steps. The option menus will activate as the user complete the stages of the process.

The first step is to open Matlab ® and type MIXTFORM. The user interface will open in the Matlab desktop.

Upload data:

- Choose **Load** ► Load precedence matrix
- Select and open the excel file that contains the precedence matrixes for the models. See Figure

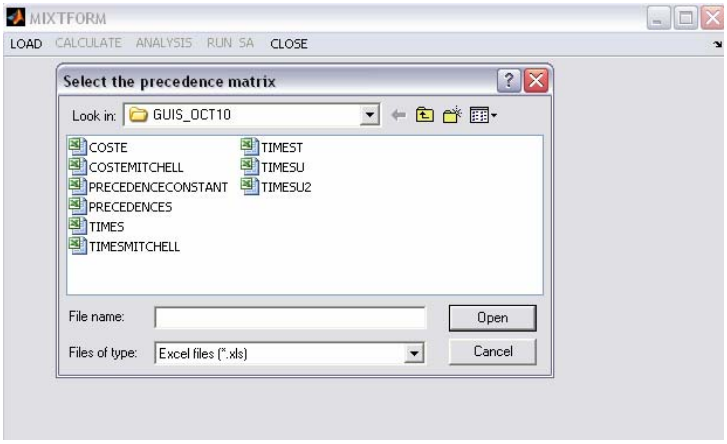

Figure 1

The precedence relationships are represented using a square matrix with zeros and ones. If, in the cell 3,5 (row, column) exist a 1 it means that is require to complete task 3 before performing task 5. The precedence file must have as active worksheets as number of models to balance. Each active worksheet must contain the precedence matrix of the model. See Figure 2.

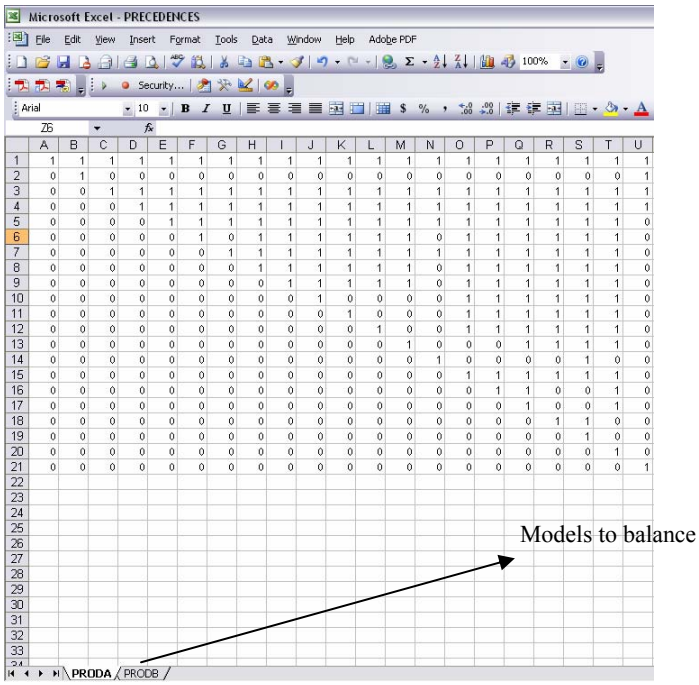

Figure 2

After uploading the precedence matrix the user can upload the processing times

- Choose **Load** ► Load processing times
- Select and open the excel file that contains the processing times
- Choose **Load** ► Load models participation

As the precedence file, this file the number of active worksheets must be equal to the number of models. Each worksheet contains the processing times and the parameters of the probability distribution in case they distribute according to the normal, uniform or triangular distribution. For each case is defined a specific format:

- Normal times → Column 1: Mean times, Column 2: Variation coefficient.
- Uniform times → Column 1: parameter "a", Column 2: parameter "b".
- Triangular times: Column 1: lower endpoint, Column 2: mode, Column 3: upper endpoint.

Calculate CTS and weighted average times

- Choose **Calculate** ► Calculate weighted times and variance
- Select times distribution. See Figure .
- Choose **Calculate** ►Calculate Composite Sequence

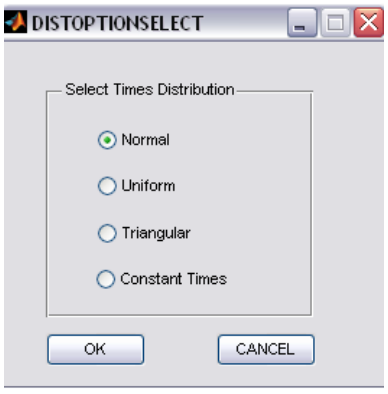

Figure 3

After calculating the composite sequence the system displays the resulting composite

task sequence and weighted average processing times as shown in Figure .

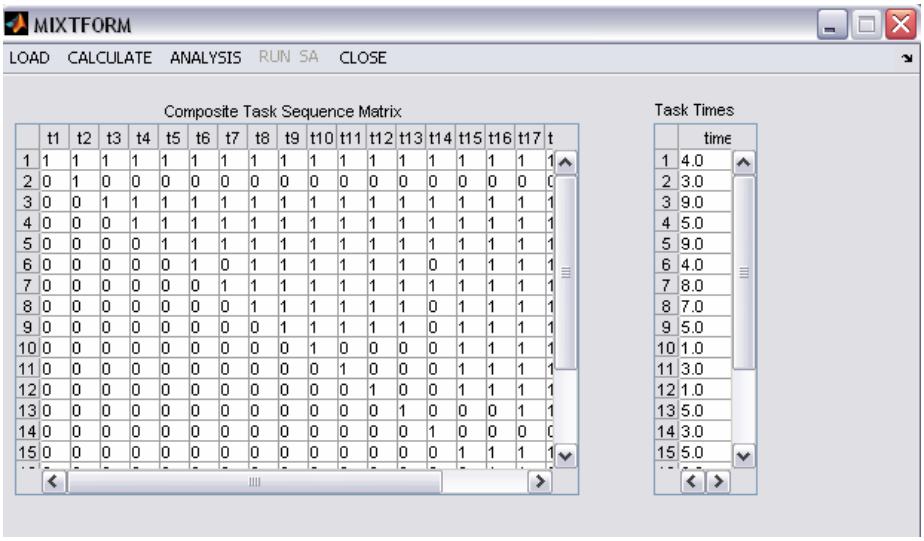

Figure 4

Input costs

- Choose **Analysis** ►Input Costs
- Click "Upload Equipment and Tooling Cost"
- Select and open the excel file that contains the equipment costs
- Input Station, Labor and Lateness cost
- Click "Save Costs"

Input problem parameters

- Choose **Analysis** ►Input Parameters
- Input Cycle time, Load quantity, Minimum utilization
- Click "Save Parameters"

Once all input data has been uploaded the user can select the option "**Run SA**" to solve the line balancing problem. The interface automatically displays the results for the balance and the discriminated cost as shown in Figure.

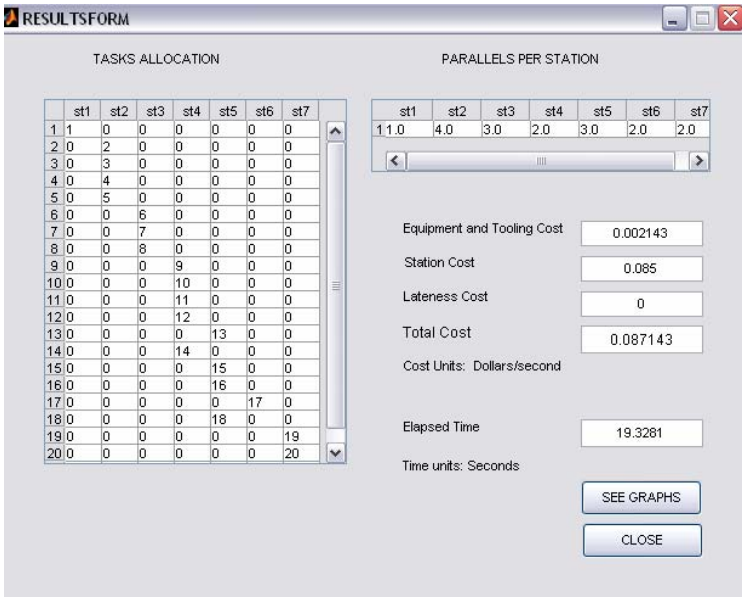

Figure 5

Additionally, it is available the plot for Cost function vs. Iteration. This plot is accessed by clicking "See Graphs" in the Results panel. See Figure .

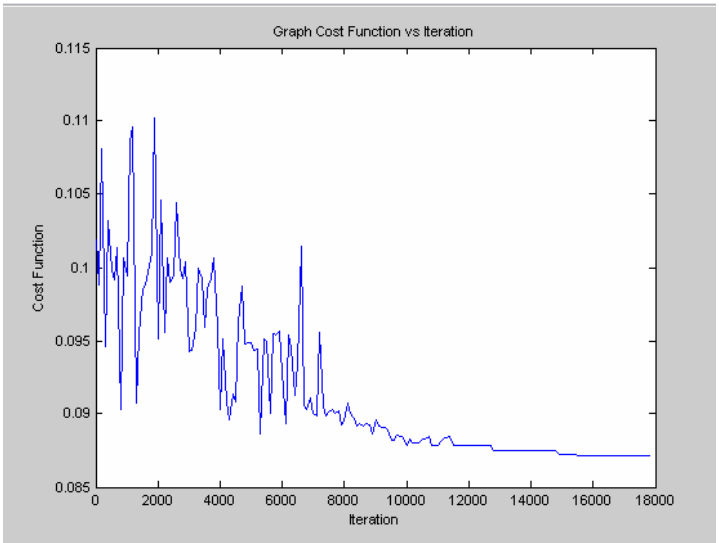

Figure 6