INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

DIVISIÓN DE INGENIERÍA Y ARQUITECTURA
PROGRAMA DE GRADUADOS EN CIENCIAS DE LA INGENIERÍA

OPTIMAL WORKERS ALLOCATION FOR THE CROSSDOCKING - JUST IN TIME SCHEDULING PROBLEM

THESIS

PRESENTED AS A REQUIREMENT
TO OBTAIN THE DEGREE OF:

DOCTOR OF PHILOSOPHY IN ENGINEERING
MAJOR IN INDUSTRIAL ENGINEERING

GUILLERMO ARTURO ÁLVAREZ PÉREZ

MAY OF 2007

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY
CAMPUS MONTERREY

DIVISIÓN DE INGENIERÍA Y ARQUITECTURA
PROGRAMA DE GRADUADOS EN CIENCIAS DE LA INGENIERÍA

The members of this committee recommend that the thesis of Guillermo Arturo Álvarez Pérez be accepted as a partial requirement to obtain the degree of:

Doctor of Philosophy in Engineering
Major in Industrial Engineering

Thesis committee:

_____
José Luis González Velarde, Ph. D.
Advisor

_____
John Welsh Fowler, Ph. D.
Co-advisor

_____
Jorge Limón Robles, Ph. D.
Synodal

_____
Neale Ricardo Smith Cornejo, Ph. D.
Synodal

_____
Francisco Román Ángel Bello Acosta, Ph. D.
Synodal

APPROVED

_____
Francisco Román Ángel Bello Acosta, Ph. D.
Director of the Graduate Program in Sciences of Engineering

Dedicatory

To **God**

To my wife **Azalea**

To my daughter **Andrea**

To my parents **Alfonso** and **Consuelo**

To my siblings **Alfonso**, **Érika** and **Diana**

To all the people I love

To the President of México, **Felipe de Jesús Calderón Hinojosa**

Acknowledgements

I am deeply grateful to my thesis advisor, **Dr. José Luis González Velarde**, for his help and guidance during all these years.

I am also grateful to the members of my thesis committee, **Dr. John Welsh Fowler**, **Dr. Jorge Limón Robles**, **Dr. Neale Ricardo Smith Cornejo**, and **Dr. Francisco Román Ángel Bello Acosta**, for their comments and suggestions.

I thank all of the faculty members and classmates that are part of the Doctoral Program in Industrial Engineering at Tecnológico de Monterrey.

Finally, I want to acknowledge the financial support received from Tecnológico de Monterrey through grant number CAT025 to carry out this work.

Abstract

In this work, a warehouse is allowed to function as a crossdock to minimize costs for a scheduling problem. These costs are due to two factors: the number of teams of workers hired to do the job, and the transit storage time for cargo. Each team of workers has a fixed cost per working day, and the cargo can incur early and tardy delivery costs. Then, the transit storage time for cargo is minimized according to Just in Time (JIT) scheduling. The goal is to obtain both: the optimal number of teams of workers in the crossdock and a schedule that minimizes the transit storage time for cargo. An integrated model to obtain both the optimal number of teams of workers and the schedule for the problem is written. The model uses the machine scheduling notation to describe it. Since the problem is known as NP-hard, a solution approach based on a combination of two metaheuristics, Reactive GRASP embedded in a Local Search algorithm and Tabu Search (RGLSTS), is provided. The results obtained from the exact method that uses the ILOG CPLEX 9.1 solver for 14 problem instances and the results obtained from the RGLSTS metaheuristic algorithm for the same problem instances are discussed.

This research has an important academic contribution because it involves the development of a metaheuristic algorithm not previously applied to a relevant problem that has not received attention. Besides, the source codes of the programs that solve the problem are available for the reader and they can be modified according to the user needs.

In the industry field, the algorithm mentioned above can be easily adapted in order to be applied to a real problem (i.e., large transshipments in companies like Wal-Mart, HEB, among others).

Obtaining optimal or near optimal solutions for the problem of this work represents an improvement in the movement or distribution of the workforce and products, reducing this way, hiring costs, transportation costs and inventory costs.

Key words: Workers allocation, Just in Time scheduling, machine scheduling, crossdocking, metaheuristics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Logistics, and particularly, inventory, transportation, scheduling, and workforce allocation are important activities within factories and they play a central role in the operations research field. Their study has led to the development of many models and algorithms which have also been applied to other scientific, academic, and industrial fields. These topics are complex and they involve many variables, uncertainties, and costs. Most of the time, the objective is to optimize results, i.e. to maximize profits or minimize costs, taking into consideration the available resources assigned to it. Very often, to do this, it is necessary to use sophisticated models and optimization techniques as well as powerful information technology [Crainic and Laporte (1998)].

The scheduling activity is strongly related to manufacturing, inventory, and transportation. Scheduling can help the manufacturing industry to reduce production and inventory costs. Also, it can help the transportation industry to reduce transportation costs. Obtaining optimal or near optimal solutions to problems related to these areas represents an improvement in the production as well as in the movement or distribution of the products. This is one of the reasons why the scheduling area is so important nowadays [Rosas (1991)]. The study of scheduling problems is not new. History, examples, notation, and references can be found in Pinedo (2002).

In a simple model, scheduling involves the assignment of jobs to a single machine in an optimal sequence (assignment-sequencing problem). Of course, this problem can be as complex as needed. Some of the performance measures in which the scheduling models have focused are: the maximum completion time or makespan

(C_{max}), the total weighted completion time ($\sum w_j C_j$), the maximum lateness (L_{max}), the number of tardy jobs ($\sum U_j$), the total tardiness ($\sum T_j$), and the total weighted tardiness ($\sum w_j T_j$), among others. All these performance outputs are regular measures, this is, the scheduling is non-decreasing in the completion times [Pinedo (2002)]. In other words, if all completion times were reduced or stayed the same, the performance measure would decrease or stay the same.

In particular, tardiness is a due date related regular measure and it has to do with customer satisfaction and costs associated with the delivery time. There is another due date related performance measure, called earliness, which is not a regular measure and it also has to do with the delivery time, but in the opposite way of the tardiness. Tardiness implies costs for jobs being completed after their due date, leads to unsatisfied customers, and perhaps even a loss of sales because of the late delivery. On the other hand, earliness implies additional inventory costs. This situation has changed because of the appearance of the Just in Time (JIT) philosophy, developed by Toyota Motor Company Ltd., which indicates that earliness and tardiness must be considered together when measuring the performance of a schedule [Rivera (1996)]. As mentioned before, earliness is not a regular measure, so, an earliness-tardiness performance measure is not a regular one. Earliness, tardiness, and earliness-tardiness functions can be seen in Figure 1.1.

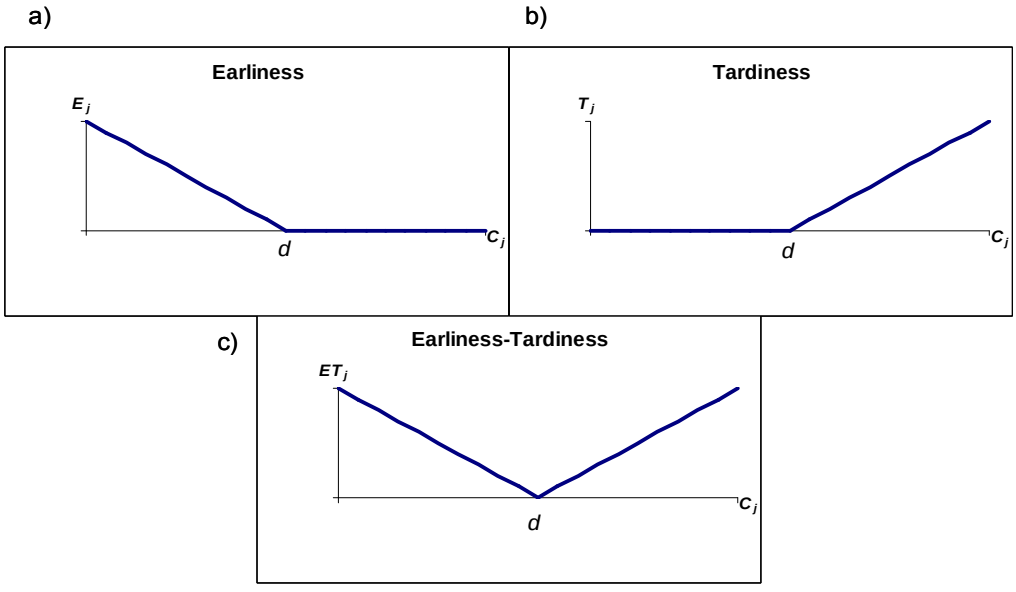


Figure 1.1 a) Earliness function; b) Tardiness function; c) Earliness-tardiness function. These three functions have a common due date *d*

Scheduling to minimize both earliness and tardiness costs has been strongly motivated by the adoption of the JIT concept in the manufacturing industry, which aims to complete the jobs exactly at their due date, not earlier and not later. Some other examples that include the concept of earliness-tardiness minimization are: the harvest of crop products which should be conducted around the time of the crop, and the production of perishable goods which should not be finished too early to avoid their possible decay, and should not be finished too late to avoid missing the delivery [Leung (2004)]. Under a JIT philosophy, it is highly desirable to have the jobs finished by the exact time requested by the customer. Otherwise, the jobs that are finished earlier or later than their due date will incur penalties. The objective of JIT scheduling is then to obtain a schedule that minimizes those penalties and part of this thesis deals with that objective.

This project is closely related to the work done by Li et al. (2004) but considering now the workforce allocation task. It also has a relationship with the work done by Rosas (1991).

In this thesis problem, **when the workforce is known and fixed** (purely *crossdocking - JIT scheduling problem*), the results obtained by our algorithm which uses an integer programming model [Nemhauser & Wolsey (1999)] taken from Li et al. (2004) are compared to the results obtained by their algorithm for the 16 problem instances presented in their work. Obviously, the objective values shown by both versions must be the same when both algorithms reach the optimal value (sometimes this is not possible due to computer memory limits). On the other hand, **when the workforce is unknown and variable** (*optimal workers allocation for the crossdocking - JIT scheduling problem*), a similar but extended integer programming model is presented.

An important section of the project has to do with the development of a metaheuristic algorithm [Díaz et al. (1996)] to find *good* solutions for big problem instances. This

metaheuristic algorithm is also used for problem instances with known optimal value to determine how close its solutions are from the optimal ones.

1.1 Thesis Structure.

In chapter 2, the problem faced in this thesis is formally described. This chapter also includes a literature review about some previous works related to this problem.

Chapter 3 talks about heuristic and metaheuristic methods in general and about the GRASP and Tabu Search methods, the metaheuristics applied to the problem of this thesis, in particular.

Chapter 4 includes a description of the approaches applied to find a solution for the crossdocking - JIT scheduling problem (exact method and metaheuristic method). The results obtained from the different solution approaches are discussed. Analysis and comparisons are made.

Chapter 5 is very similar to Chapter 4, but applied to the bigger problem known as the optimal workers allocation for the crossdocking - JIT scheduling problem.

In Chapter 6, conclusions and future work related to this project are mentioned.

Finally, some Linear Programming theory, an example of an MIP, the source codes (written in C language) for all of the algorithms used in this work, the problem instances, and the solutions obtained by the metaheuristic algorithms are shown in the appendixes section. Some appendixes are not printed and they only appear in electronic format in the CD at the end of this thesis.

1.2 Methodology.

In order to create this thesis, the following methodology was used:

1. Bibliographic research about workforce allocation, crossdocking - JIT scheduling and related works.
2. Bibliographic research about problems complexity, heuristics, and metaheuristics.
3. Bibliographic and technical research about ILOG CPLEX 9.1, which was the software library used to run the problem instances for the exact model.
4. Design, implementation, and execution of the algorithm that generates problem instances that can be used as inputs for the exact model and for the metaheuristic algorithm.
5. Design, implementation, and execution of the exact model which finds an optimal (when possible) feasible solution for a particular problem instance.
6. Design, implementation, and execution of the metaheuristic algorithm which finds a feasible solution for a particular problem instance.
7. Design, implementation, and execution of the algorithm that converts a solution from the metaheuristic into an initial solution for the exact method.
8. Analysis of the obtained results and conclusions.

Chapter 2

Problem Description

Typically, storage and order picking are the main operations of the handling activity in a warehouse. These operations are labor intensive and are expensive. Handling costs and space utilization need to be considered when working with a warehouse. Even more, warehouses need to be configured to handle equipment and an inventory management system is required to have everything under control. Crossdocking tries to reduce or eliminate these issues by reducing warehouses to purely transshipment centers where receiving and shipping are its only functions. Shipments need to expend very little time at crossdocks before being moved into the next level in the supply chain. At crossdocks, inbound trucks arrive with cargo that is sorted, consolidated, and loaded onto outbound trucks sent to manufacturing sites, retailers or even another warehouse or crossdock. In a crossdock, the customer is predetermined and there is no need for storage.

The crossdock can be divided into an import area where breakdown occurs and an export area where buildup occurs. In the import area, incoming containers are broken down, and in the export area, containers are built up after consolidation, if necessary. Since incoming containers come from a number of suppliers, incoming cargo will reach the crossdock at different times. Items, including breakdowns, are then either sent away directly or sent to the export area to be loaded into outgoing containers. Outbound cargo is shipped away by vehicles with scheduled departure times. So, in this context, each incoming container has a release time and a due date and each outgoing container has a due date. This situation is shown in Figure 2.1.

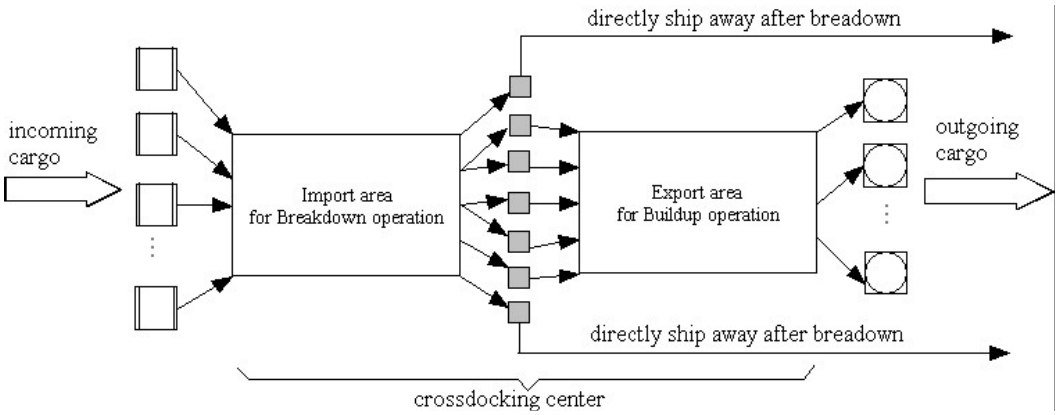


Figure 2.1 A crossdock flow - taken from Li et al. (2004)

Each incoming (outgoing) container is processed by a breakdown (buildup) team of workers in the import (export) area. Since such teams are limited in number, scheduling teams to jobs has to be precise. Timing is extremely important for crossdocking. The idea is then to obtain a schedule to specify when to start breakdown and when to complete buildup of all cargo where the goal is to complete processing each container exactly at its due date. This is true for the purely crossdocking - JIT scheduling problem where the number of teams of workers in each side of the crossdock is a given parameter.

For the optimal workers allocation for the crossdocking - JIT scheduling problem, in which the number of teams of workers in each side of the crossdock is an unknown variable that has to be determined, the costs are due to two factors: the number of teams of workers hired to do the job, and the transit storage time for cargo. Each team of workers has a fixed cost per working day, and the cargo can incur, as mentioned before, early and tardy delivery costs. The cost per working day of each team of workers is the same for both sides of the crossdock, and, as it is known, the transit storage time for cargo is minimized according to JIT scheduling. Then, the complete goal is to obtain both: the optimal number of teams of workers in each side of the crossdock and a schedule that specifies when to start breakdown and when to complete buildup of all cargo.

2.1 Literature Review.

The first study related to JIT scheduling appeared with the work done by Sidney (1977) who analyzed a scheduling problem for a single machine with earliness and tardiness penalties, considering intervals for processing the jobs, and idle times. In that paper, an earliness penalty occurs when a job starts before its target start time. Tardiness penalty occurs when a job finishes after its target due date. A job *j* incurs no penalty if it is processed entirely in the target interval [$start\ time_j$, $due\ date_j$]. The author presented a polynomial time algorithm with $O(n^2)$ order for solving this problem, where *n* is the number of jobs in the problem instance. Later, Lakshminarayan et al. (1978) developed a polynomial time algorithm with *O(nlogn)* order for solving this same problem.

The following variations of JIT machine scheduling problems were studied for a single machine with a common due date with some differences:

- Large common due date (d) - the problem is called unrestricted because the scheduling decision will not be affected by the value of the due date. Bagchi et al. (1986) showed that this problem can be solved in polynomial time
- Not large common due date (d) - Hall et al. (1991) showed that this problem is NP-complete in the ordinary sense, even for not weighted unit earliness and tardiness penalties of jobs ($\alpha = \beta = 1$). In the same work, the authors showed that if unit earliness and tardiness penalties are job-dependent (α_j, β_j), the problem is NP-hard, even for a large common due date (d)

Liaw (1999) proposed a branch and bound algorithm with dominance rules to minimize the sum of weighted earliness (α) and weighted tardiness (β) for a single machine scheduling problem where no machine idle time is allowed.

An excellent survey of JIT scheduling for single machine can be found in Baker and Scudder (1990). They start the review with a basic model that contains symmetric

penalties and a common due date, and then they add some features to this basic model to form a framework for models classification with the following characteristics:

- Linear and quadratic objective functions
- Symmetric unit earliness and tardiness penalties ($\alpha = \beta$), different unit earliness and tardiness penalties ($\alpha \neq \beta$), and job-dependent unit earliness and tardiness penalties (α_j, β_j)
- Common due date (d) and job-dependent due dates (d_j)

Job-dependent due dates (d_j) complicate the problem because most of the properties of optimal schedules do not hold any longer. Garey et al. (1988) showed that the problem of finding minimal cost schedules with job-dependent due dates (d_j) is NP-complete.

In many studies, release times are not considered by researchers because jobs are assumed to be ready at time 0. Mazzini and Armentano (2001) developed a constructive heuristic and an adjacent pairwise interchange heuristic to solve the problem where each job has its own release time (r_j) and its own due date (d_j).

There is little research on JIT scheduling for parallel machines and most research done on this problem deals with a common due date (d). Few authors study this problem with job-dependent due dates (d_j). Laguna and González-Velarde (1991) proposed a search heuristic for the uncommon weighted earliness penalty (α_j) problem with job-dependent deadlines (mandatory d_j) in parallel identical machines. Sivrikaya-Serifoglu and Ulusoy (1999) employed two genetic algorithm approaches to heuristically solve a parallel machine scheduling problem with common and unequal weighted earliness and tardiness penalties (α, β) where the due dates of the jobs are distinct (d_j) and each job has its own arrival time (r_j). Heady and Zhu (1998) provided a heuristic algorithm for the uncommon weighted (α_j, β_j) JIT scheduling problem with job-dependent due dates (d_j) in a multi-machine system where processing times depend on the job-machine combination. Radhakrishnan and Ventura (2000) provided local search heuristics in the framework of the Simulated Annealing

technique for the parallel machine earliness-tardiness uncommon due date (d_j) sequence-dependent set-up time scheduling problem.

No previous research has been done on the JIT machine scheduling characterization of the crossdocking problem, except for the one discussed in Li et al. (2004). They proposed two algorithms to solve this problem: SWOGA and LPGA. The first one uses Squeaky Wheel Optimization embedded in a Genetic Algorithm and the second one uses Linear Programming within a Genetic Algorithm. However, the work presented in this thesis considers a different metaheuristic approach applied to this problem. Besides, the source codes of our work are available for the reader.

The crossdocking - JIT scheduling problem is NP-hard because if the due dates of the outgoing jobs are very large, the two phases of the problem could be processed independently (the second phase of the problem would not depend on the results of the first phase), and each phase could be reduced to the JIT scheduling problem for parallel machine with job-dependent due dates which it is known to be NP-hard, since the case for single machine is already NP-hard [Garey et al. (1988)].

Rosas (1991), Rivera (1996), and Li et al. (2004) contain a section in their works that include an excellent literature review related to JIT scheduling.

On the other hand, some works related to JIT scheduling but now considering a variable number of teams of workers are referred below. This bigger problem includes the workforce allocation task whose study is varied.

Abernathy et al. (1973) presented a hierarchical scheme of three phases: planning, scheduling, and allocation, to solve a nurse-staffing problem in a hospital. They formulated the planning and scheduling stages as a stochastic programming model, suggested an iterative solution procedure using random loss functions, and developed a non iterative solution procedure for a chance-constrained formulation that considers alternative operating procedures and service criteria. They made the assignment of tasks to multi-functional workers during the allocation phase. Siferd and Benton (1992) extended this hospital nurse staffing and scheduling study.

Baker (1976) studied the basic mathematical models for workforce scheduling with cyclic demand for staff.

Lewis et al. (1998) studied how the tasks in a fixed size office should be organized to maximize throughput when short-term reassignment of workers is difficult, costly, or restricted. Heymann et al. (2000) studied how many workers should be allocated for executing a distributed application and how to assign tasks to workers in order to maximize resource efficiency and minimize application execution time. They proposed an effective scheduling strategy that dynamically measures the execution times of tasks and uses this information to dynamically adjust the number of workers to achieve a desirable efficiency, minimizing the impact of loss of speedup. Brennan and Orwig (2000) examined conflicting approaches to work allocation in an engineering consulting firm. They proposed an analytical framework to determine whether a leveraged approach is superior to a cascaded bin packing approach for the organization's performance.

Iima and Sannomiya (2001) proposed a module type genetic algorithm to solve a modified job-shop scheduling problem with a workers allocation constraint. Campbell and Diaby (2002) used mathematical programming to model a multi-department and labor-intensive service environment problem. They proposed a heuristic based on a linear assignment approximation for allocating cross-trained workers to multiple departments at the beginning of a shift. They considered the re-assignment of tasks to workers within the shifts. Gomar et al. (2002) developed a linear programming model to help optimize the multi-skilled workforce assignment and allocation process in a construction project.

Tharmmaphornphilas and Norman (2004) proposed a quantitative method based on mathematical programming to obtain a proper job rotation interval length in a work setting in order to reduce worker fatigue and injuries and improve the quality of the job. Corominas et al. (2004) solved a problem of allocating types of tasks to the multi-functional workers of a service center over a time horizon assuming equal efficiency for all of the members of the staff.

No previous research has been done on the machine characterization of the optimal workers allocation for the crossdocking - JIT scheduling problem.

It was previously shown that the purely crossdocking - JIT scheduling problem is NP-hard. Therefore, the optimal workers allocation for the crossdocking - JIT scheduling problem is NP-hard as well.

2.2 Relationship of the Problem with Machine Scheduling.

The crossdocking - JIT scheduling problem described before can be modeled naturally as a machine scheduling problem as follows: each incoming container can be thought of as a job which has a release time after which it can be processed, a due date, and a processing time. Each outgoing container can be thought of as a job which has a number of source containers or predecessors which feed it, a due date, and a processing time. These incoming/outgoing jobs are processed by teams of workers which can be thought of as machines. These machines handling incoming/outgoing cargo are parallel because they are able to operate simultaneously.

The parameters used in this model are:

r_i - release time after which incoming container i can be broken down

d_i - due date for incoming container i

p_i - processing time required to break down incoming container i

S_{ij} - the ith source of outgoing container j. Outgoing container j is built from K_j different incoming containers

D_j - due date of outgoing container j

P_j - processing time required to build up outgoing container j

n - number of incoming containers

m - number of breakdown teams

N - number of outgoing containers

M - number of buildup teams

α - penalty for unit time earliness

β - penalty for unit time tardiness

For the purposes of this project, it is assumed that $n > m$, and $N > M$ (only for the purely crossdocking - JIT scheduling problem where m and M are given parameters). The number of jobs n and N do not have to be equal, and the number of teams m and M do not have to be equal either. Another note related to the problem is the representation of the S_{ij} matrix given by the n incoming containers and the N outgoing containers. The cargo of one incoming container might be loaded in zero or more outgoing containers, and the cargo built up in one outgoing container might come for one or more incoming containers. If the cargo of one incoming container is loaded in zero outgoing containers it means that the cargo is directly shipped away after breakdown.

Other assumptions to be considered for this project are:

- Teams are identical
- Teams are available at time 0
- Teams are 100% reliable (machines do not get out of order)
- A team can not process more than one job at the same time
- There are no preemptions in the scheduling, this is, once a team starts to process a job, this job has to be finished before the team can start processing another job
- Containers and teams have infinite capacity (the number of handled items can be any number)
- All the cargo arriving to the crossdock leaves the crossdock

- Distribution times for the jobs inside the crossdock are already included in their corresponding processing times
- The horizon of the process is one working day

In the problem context, suppliers do not want to expend too much time in the crossdock because it is very likely that they have to deliver more cargo to some other customers and they do not want to be late. On the other hand, suppliers should not be early because the crossdock authorities do not want to have their cargo too much time inside the crossdock to avoid inventory and it is very likely that they need that space for some other suppliers. This agrees with the JIT scheduling philosophy.

Incoming jobs are described by $J_I = \{J_{I1}, J_{I2}, \ldots, J_{In}\}$ and outgoing jobs are described by $J_O = \{J_{O1}, J_{O2}, \ldots, J_{ON}\}$. Breakdown teams are described by $M_I = \{M_{I1}, M_{I2}, \ldots, M_{Im}\}$ and buildup teams are described by $M_O = \{M_{O1}, M_{O2}, \ldots, M_{OM}\}$. Jobs in J_I are processed only by teams in M_I, and jobs in J_O are processed only by teams in M_O. A job $J_{Ii} \in J_I$ is described by $\{r_i, p_i, d_i\}$, where r_i, p_i, d_i denote its release time, processing time, and due date, respectively. A job $J_{Oj} \in J_O$ is described by $\{pred_j, P_j, D_j\}$, where P_j and D_j denote its processing time and due date, respectively, and $pred_j$ describes its predecessors, all belonging to J_I. Actually, $pred_j$ represents a column of the S_{ij} matrix. A representation of the sources or predecessors of outgoing containers and the S_{ij} matrix is shown in Figure 2.2.

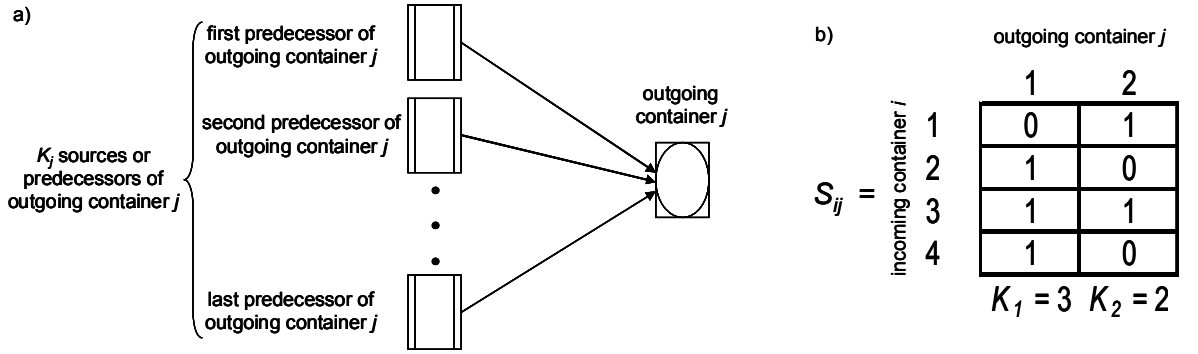


Figure 2.2 a) A representation of the sources or predecessors of an outgoing container *j*; b) An example of an S_{ij} matrix with 4 incoming containers and 2 outgoing containers

Earliness and tardiness penalties of a job J_{Ii} are defined by $e_i = \max\{0, d_i - c_i\}$ and $t_i = \max\{0, c_i - d_i\}$, respectively, where c_i represents the incoming job's finish time. Earliness and tardiness penalties of a job J_{Oj} are defined by $E_j = \max\{0, D_j - C_j\}$ and $T_j = \max\{0, C_j - D_j\}$, respectively, where C_j represents the outgoing job's finish time. The objective of the problem is to find a schedule that minimizes the total penalty. Figure 2.3 shows a more detailed view of Figure 2.1 and it represents a summary of the whole situation.

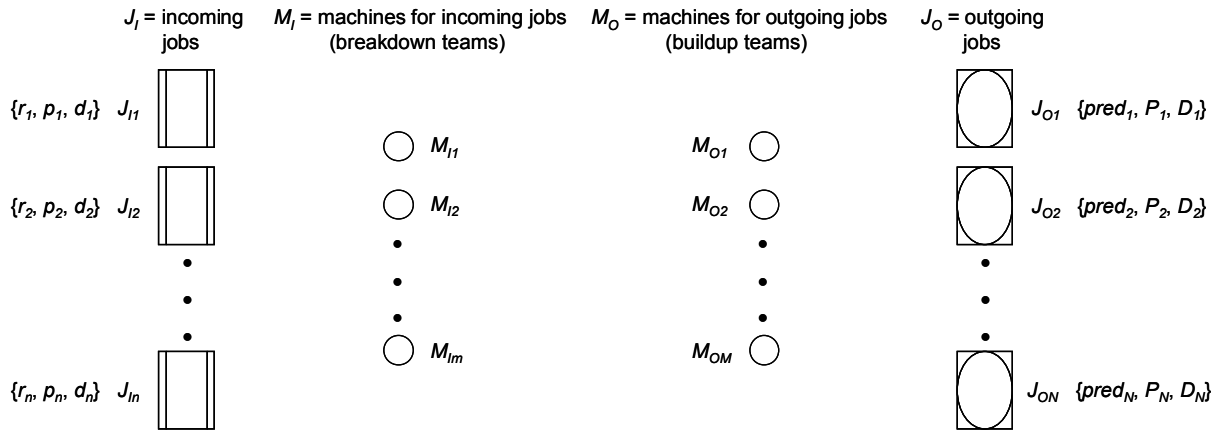


Figure 2.3 A more detailed flow for the crossdocking - JIT scheduling problem

In Figure 2.3, each incoming job represents a container coming from a company like Pepsico, the Coca-Cola Company, Bimbo, Cuauhtémoc-Moctezuma Beer Company, Kraft, Kimberly-Clark, among many others. Each one of these companies' trucks contains several products that are going to be spread out through several locations like, i.e. in the Monterrey city area, Wal-Mart Las Torres, Wal-Mart Valle Oriente, Wal-Mart Lincoln, etc. The Wal-Mart example is used because crossdocking has received much attention as a result of the commercial success of large transshipments in this company [Gue (2001)]. In the same Figure 2.3, $pred_j$ means that job J_{Oj} contains cargo from K_j different incoming containers or jobs.

As mentioned before, cargo is processed in two phases: breakdown and buildup. Precedence relationships exist between these phases in the following sense: each incoming container *i* must be broken down before an outgoing container *j* can

commence to be built up if cargo items in container *j* come from container *i*. In other words, a container can start buildup only if all its source containers have been broken down in order to have all its items correctly loaded. Buildup should not start before complete breakdown because it could be possible to have the heaviest items loaded in the top of the container. This situation is part of another problem known as the Bin Packing Problem [Coffman (1976), Baase (1991), Hochbaum (1997), Horowitz et al. (1998), Cormen et al. (2001)] whose study is beyond the scope of this work.

It is known that there are no release times for buildup. It is also known that buildup can not commence in outgoing container *j* until all its cargo predecessors have been broken down. So, it is possible to define the state variable R_j as follows: the completion time of last incoming container *i* broken down which contains cargo for outgoing container *j*, or $R_j = \max\{c_i\}$, *i* = first predecessor of outgoing container *j*, …, last predecessor of outgoing container *j*. R_j can be seen as the release time of the outgoing container *j*, however, its value depends on the current schedule.

In summary, the crossdocking - JIT scheduling problem can be viewed as a two-phase parallel machine scheduling problem. Also, each phase of the problem can be seen as an assignment - scheduling problem (the sequencing activity is implicitly included in the scheduling activity), according to Figure 2.4.

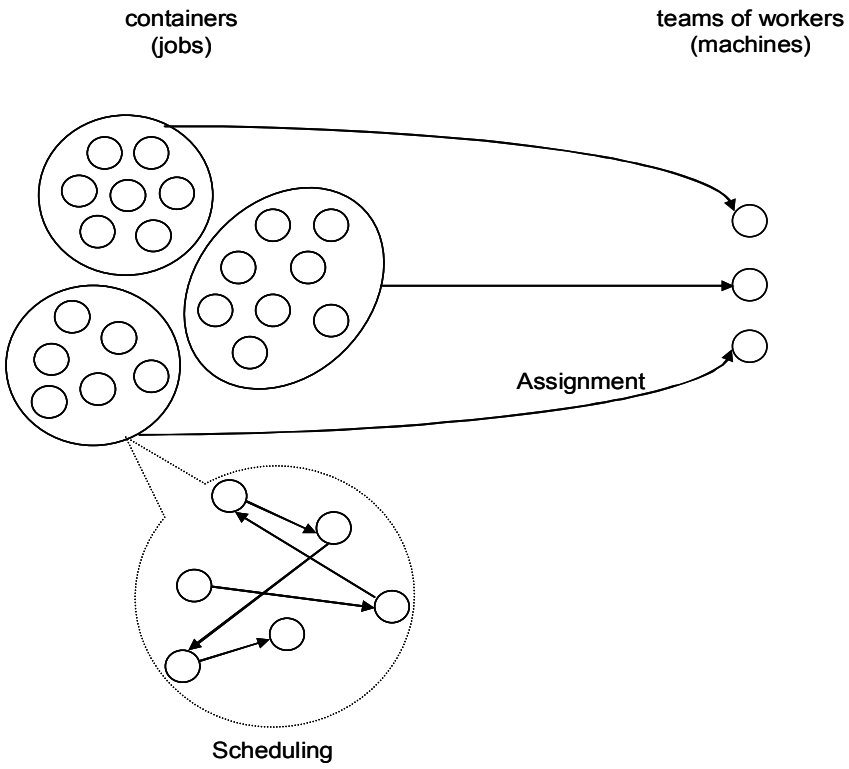


Figure 2.4 Inbound or outbound area of the crossdocking - JIT scheduling problem seen as an assignment - scheduling problem

The sequencing part of the problem shown in Figure 2.4 can also be seen as a Traveling Salesman Problem (TSP) [Lawler et al. (1985)]. So, the whole picture of the Figure 2.4 can be viewed as a Vehicle Routing Problem (VRP) [Crainic and Laporte (1998)].

The optimal workers allocation for the crossdocking - JIT scheduling problem can be modeled as a machine scheduling problem just exactly in the same way as it is done for the purely crossdocking - JIT scheduling problem. However, in this bigger problem the incoming and outgoing jobs (n and N, respectively) are processed by an unknown number of teams of workers (m and M) which can also be thought of as machines. The assumptions to be considered and most of the parameters used for the model for this bigger problem are the same mentioned for its sub-problem, but the number of breakdown teams (m) and the number of buildup teams (M) are not

considered parameters any more, and the cost of a team hired (h) is a new parameter that has to be considered.

As mentioned before, the number of teams hired to do the breakdown (m) and the buildup (M) is unknown. Obviously, in both cases the minimum number of teams hired is 1 and the maximum number of teams hired for the breakdown is the total number of incoming jobs (n) and for the buildup is the total number of outgoing jobs (N). As said earlier, the number of jobs n and N do not have to be equal, and the number of teams m and M do not have to be equal either.

2.3 Model for the Problem.

Crossdocking - JIT scheduling problem

As the crossdocking - JIT scheduling problem described above can be seen as a machine scheduling problem, it is possible to formulate it with the following integer programming model taken from Li et al. (2004):

Decision variables

y_{ik} = 1 if incoming container i is processed by breakdown team k and 0 otherwise, for i = 1, …, n, k = 1, …, m

Y_{jk} = 1 if outgoing container j is processed by buildup team k and 0 otherwise, for j = 1, …, N, k = 1, …, M

I_{ijk} = 1 if incoming containers i and j are both processed by breakdown team k and i precedes (not necessarily immediately) j, and 0 otherwise, for i, j = 1, …, n, $i \neq j$, k = 1, …, m

J_{ijk} = 1 if outgoing containers i and j are both processed by buildup team k and i precedes (not necessarily immediately) j, and 0 otherwise, for i, j = 1, …, N, $i \neq j$, k = 1, …, M

c_i - completion time of incoming container i, i = 1, …, n

C_j - completion time of outgoing container j, j = 1, …, N

Variables y_{ik} and Y_{jk} represent assignment variables, I_{ijk} and J_{ijk} represent sequencing variables, and c_i and C_j represent scheduling variables. The values assigned to the assignment variables and to the scheduling variables represent a specific solution for the problem.

State variables: their values depend on the current built schedule

e_i - earliness of incoming container i, i = 1, …, n

E_j - earliness of outgoing container j, j = 1, …, N

t_i - tardiness of incoming container i, i = 1, …, n

T_j - tardiness of outgoing container j, j = 1, …, N

Objective function

$$\text{Minimize } \sum_{i=1}^{n} \left(\alpha e_i + \beta t_i\right) + \sum_{j=1}^{N} \left(\alpha E_j + \beta T_j\right)$$

Constraints

For job to team uniqueness - each job must be processed by exactly one team:

$$(1) \qquad \sum_{k=1}^{m} y_{ik} = 1,\ i = 1, \ldots, n \text{ (breakdown area)}$$

$$(2) \qquad \sum_{k=1}^{M} Y_{jk} = 1,\ j = 1, \ldots, N \text{ (buildup area)}$$

For job precedence relationships:

$$(3) \qquad y_{ik} + y_{jk} - (I_{ijk} + I_{jik}) \leq 1 \text{ (breakdown area)}$$

$$(4) \quad 2(I_{ijk} + I_{jik}) - y_{ik} - y_{jk} \leq 0 \text{ (breakdown area)}$$

$$i, j = 1, \ldots, n, i < j, k = 1, \ldots, m$$

These two previous groups of constraints come from a transformation of the following relationships:

$$y_{ik} + y_{jk} = 2 \Leftrightarrow I_{ijk} + I_{jik} = 1 \text{ (jobs } i \text{ and } j \text{ are processed by the same team)}$$

$$y_{ik} + y_{jk} \leq 1 \Leftrightarrow I_{ijk} + I_{jik} = 0 \text{ (jobs } i \text{ and } j \text{ are not processed by the same team)}$$

$$i, j = 1, \ldots, n, i < j, k = 1, \ldots, m$$

A similar reasoning is used for the outgoing containers, obtaining:

$$(5) \quad Y_{ik} + Y_{jk} - (J_{ijk} + J_{jik}) \leq 1 \text{ (buildup area)}$$

$$(6) \quad 2(J_{ijk} + J_{jik}) - Y_{ik} - Y_{jk} \leq 0 \text{ (buildup area)}$$

$$i, j = 1, \ldots, N, i < j, k = 1, \ldots, M$$

For sufficient time between jobs on the same team - if job i precedes job j, there must be enough time between them for job j to be completed:

$$(7) \quad c_i \leq (c_j - p_j) + G(1 - I_{ijk}), i, j = 1, \ldots, n, i \neq j, k = 1, \ldots, m \text{ (breakdown area)}$$

$$(8) \quad C_i \leq (C_j - P_j) + G(1 - J_{ijk}), i, j = 1, \ldots, N, i \neq j, k = 1, \ldots, M \text{ (buildup area)}$$

where G is a nonzero real number such that $G \geq \max\{f(x) \mid x \in D\}$ and $f : D \rightarrow \mathrm{R}$ for some D, $\delta \in \{0, 1\}$. Then, for each $\delta \in \{0, 1\}$ and $x \in D$, the following are equivalent:

- $\delta = 0 \rightarrow f(x) \leq 0$
- $f(x) - G \cdot \delta \leq 0$

This binary variables reduction lemma is better explained in Sierksma (2001).

Since it is not easy to know the exact value that satisfies the conditions mentioned above, Li et al. (2004) recommend, for practical purposes, to use a big value for G. They used $G = 10{,}000$ for their runs.

The rest of the groups of constraints that complete the model are:

$$(9) \quad c_i - r_i \geq p_i, i = 1, \ldots, n \text{ (breakdown area)}$$

(10) $C_j - c_i \geq P_j$, $j = 1, \ldots, N$, i = first predecessor of outgoing container j, …, last predecessor of outgoing container j (buildup area)

(11) $c_i - d_i = t_i - e_i$, $i = 1, \ldots, n$ (breakdown area)

(12) $C_j - D_j = T_j - E_j$, $j = 1, \ldots, N$ (buildup area)

$y_{ik} \in \{0, 1\}$, $i = 1, \ldots, n$, $k = 1, \ldots, m$

$Y_{jk} \in \{0, 1\}$, $j = 1, \ldots, N$, $k = 1, \ldots, M$

$I_{ijk} \in \{0, 1\}$, $i, j = 1, \ldots, n$, $i \neq j$, $k = 1, \ldots, m$

$J_{ijk} \in \{0, 1\}$, $i, j = 1, \ldots, N$, $i \neq j$, $k = 1, \ldots, M$

$c_i, e_i, t_i \in Z^+$ (nonnegative integer numbers), $i = 1, \ldots, n$

$C_j, E_j, T_j \in Z^+$ (nonnegative integer numbers), $j = 1, \ldots, N$

The group of constraints (9) enforces release times in the breakdown area. The group of constraints (10) specifies that an outgoing container can start buildup only if all its source containers have been broken down. The groups of constraints (11) and (12) specify each job's earliness and tardiness in the breakdown area and the buildup area, respectively.

This model contains:

- A total of variables (including state variables) equal to $n^2m + N^2M + 3(n+N)$, from which:
  - $n^2m + N^2M$ are binary variables and $3(n+N)$ are integer variables
  - $n^2m + N^2M + n + N$ are decision variables and $2(n+N)$ are state variables
- A total of constraints equal to $3n + 2N + 2(n^2 - n)m + 2(N^2 - N)M + \sum_{j=1}^{N} K_j$, where

$$N \leq \sum_{j=1}^{N} K_j \leq nN$$

- A total of non-empty cells in the technological coefficients matrix (see Appendix 1) equal to $4n + 3N + nm + NM + 7(n^2 - n)m + 7(N^2 - N)M + 2\sum_{j=1}^{N} K_j$

The technological coefficients matrix is sparse because the total of non-empty cells is too small with respect to the total of cells in that matrix. This total of cells is computed multiplying the total of variables times the total of constraints. This situation can be noticed in Table 2.1

n	*m*	*N*	*M*	Total of variables	Total of constraints		Total of non-empty cells		Total of cells		Percentage of occupation	
					Min case	Max case	Min case	Max case	Min case	Max case	Min case	Max case
4	2	3	2	71	93	102	297	315	6603	7242	4.5%	4.3%
5	3	4	3	150	219	235	739	771	32850	35250	2.2%	2.2%
10	2	11	3	626	1083	1182	3718	3916	677958	739932	0.5%	0.5%
15	3	14	2	1154	2075	2271	7161	7553	2394550	2620734	0.3%	0.3%
20	8	18	7	5582	10478	10820	36730	37414	58488196	60397240	0.1%	0.1%
24	11	25	12	13983	26691	27266	93689	94839	373220253	381260478	0.0%	0.0%

Table 2.1 Percentage of occupation of the technological coefficients matrix for different crossdocking - JIT scheduling problem instances

It can be seen in Table 2.1 that the percentage of occupation of the technological coefficients matrix tends to zero as the problem instance grows.

Optimal Workers Allocation for the crossdocking - JIT scheduling problem

The integer programming model for the optimal workers allocation for the crossdocking - JIT scheduling problem presented below is very similar to the one shown for its sub-problem, the purely crossdocking - JIT scheduling problem. It is possible to formulate it using the machine scheduling notation as follows:

Decision variables

y_{ik} = 1 if incoming container *i* is processed by breakdown team *k* and 0 otherwise, for *i* = 1, …, *n*, *k* = 1, …, *n*

Y_{jk} = 1 if outgoing container j is processed by buildup team k and 0 otherwise, for j = 1, …, N, k = 1, …, N

I_{ijk} = 1 if incoming containers i and j are both processed by breakdown team k and i precedes (not necessarily immediately) j, and 0 otherwise, for i, j = 1, …, n, $i \neq j$, k = 1, …, n

J_{ijk} = 1 if outgoing containers i and j are both processed by buildup team k and i precedes (not necessarily immediately) j, and 0 otherwise, for i, j = 1, …, N, $i \neq j$, k = 1, …, N

c_i - completion time of incoming container i, i = 1, …, n

C_j - completion time of outgoing container j, j = 1, …, N

m_k = 1 if breakdown team k is hired and 0 otherwise, for k = 1, …, n

M_k = 1 if buildup team k is hired and 0 otherwise, for k = 1, …, N

Variables y_{ik} and Y_{jk} represent assignment variables, I_{ijk} and J_{ijk} represent sequencing variables, c_i and C_j represent scheduling variables, and m_k and M_k represent machines variables. The values assigned to the assignment variables, scheduling variables and machines variables represent a specific solution for the problem.

State variables: their values depend on the current built schedule

e_i - earliness of incoming container i, i = 1, …, n

E_j - earliness of outgoing container j, j = 1, …, N

t_i - tardiness of incoming container i, i = 1, …, n

T_j - tardiness of outgoing container j, j = 1, …, N

Objective function

$$\text{Minimize } \sum_{i=1}^{n} \left(\alpha e_i + \beta t_i\right) + \sum_{j=1}^{N} \left(\alpha E_j + \beta T_j\right) + h\sum_{k=1}^{n} m_k + h\sum_{k=1}^{N} M_k$$

Constraints

$$(1) \quad \sum_{k=1}^{n} y_{ik} = 1,\ i = 1, \ldots, n \text{ (breakdown area)}$$

$$(2) \quad \sum_{k=1}^{N} Y_{jk} = 1,\ j = 1, \ldots, N \text{ (buildup area)}$$

$$(3) \quad y_{ik} + y_{jk} - (I_{ijk} + I_{jik}) \leq 1 \text{ (breakdown area)}$$

$$(4) \quad 2(I_{ijk} + I_{jik}) - y_{ik} - y_{jk} \leq 0 \text{ (breakdown area)}$$

$$i, j = 1, \ldots, n,\ i < j,\ k = 1, \ldots, n$$

$$(5) \quad Y_{ik} + Y_{jk} - (J_{ijk} + J_{jik}) \leq 1 \text{ (buildup area)}$$

$$(6) \quad 2(J_{ijk} + J_{jik}) - Y_{ik} - Y_{jk} \leq 0 \text{ (buildup area)}$$

$$i, j = 1, \ldots, N,\ i < j,\ k = 1, \ldots, N$$

$$(7) \quad c_i \leq (c_j - p_j) + G(1 - I_{ijk}),\ i, j = 1, \ldots, n,\ i \neq j,\ k = 1, \ldots, n \text{ (breakdown area)}$$

$$(8) \quad C_i \leq (C_j - P_j) + G(1 - J_{ijk}),\ i, j = 1, \ldots, N,\ i \neq j,\ k = 1, \ldots, N \text{ (buildup area)}$$

$$(9) \quad c_i - r_i \geq p_i,\ i = 1, \ldots, n \text{ (breakdown area)}$$

(10) $C_j - c_i \geq P_j$, $j = 1, \ldots, N$, i = first predecessor of outgoing container j, …, last predecessor of outgoing container j (buildup area)

$$(11) \quad c_i - d_i = t_i - e_i,\ i = 1, \ldots, n \text{ (breakdown area)}$$

$$(12) \quad C_j - D_j = T_j - E_j,\ j = 1, \ldots, N \text{ (buildup area)}$$

$$(13) \quad m_k - y_{ik} \geq 0,\ i = 1, \ldots, n,\ k = 1, \ldots, n \text{ (breakdown area)}$$

$$(14) \quad M_k - Y_{jk} \geq 0,\ j = 1, \ldots, N,\ k = 1, \ldots, N \text{ (buildup area)}$$

$$y_{ik} \in \{0, 1\},\ i = 1, \ldots, n,\ k = 1, \ldots, n$$

$$Y_{jk} \in \{0, 1\},\ j = 1, \ldots, N,\ k = 1, \ldots, N$$

$$I_{ijk} \in \{0, 1\},\ i, j = 1, \ldots, n,\ i \neq j,\ k = 1, \ldots, n$$

$$J_{ijk} \in \{0, 1\},\ \ i, j = 1, \ldots, N,\ i \neq j,\ k = 1, \ldots, N$$

$$m_k \in \{0, 1\},\ k = 1, \ldots, n$$

$$M_k \in \{0, 1\},\ k = 1, \ldots, N$$

$c_i, e_i, t_i \in Z^+$ (nonnegative integer numbers), $i = 1, \ldots, n$

$C_j, E_j, T_j \in Z^+$ (nonnegative integer numbers), $j = 1, \ldots, N$

All groups of constraints were previously explained, except for the new groups of constraints (13) and (14) which specify that a job can only be assigned to a team of workers that has been hired. Again, for the group of constraints (7) and (8), it is recommended, for practical purposes, to use a big value for *G* since it is not easy to know the exact value for *G* that satisfies them. We used *G* = 100,000 for the experiments. This value for *G* is bigger than the one mentioned for the previous model because the model of this bigger problem considers the costs due to the number of teams of workers hired in each side of the crossdock. For the cost of a team hired we used a value of *h* = 1,000.

This model contains:

- A total of variables (including state variables) equal to $n^3 + N^3 + 4(n+N)$, from which:
    - $n^3 + N^3 + n + N$ are binary variables and $3(n+N)$ are integer variables
    - $n^3 + N^3 + 2(n+N)$ are decision variables and $2(n+N)$ are state variables
- A total of constraints equal to $3n + 2N - (n^2 + N^2) + 2(n^3 + N^3) + \sum_{j=1}^{N} K_j$, where $N \leq \sum_{j=1}^{N} K_j \leq nN$

- A total of non-empty cells in the technological coefficients matrix equal to $4n + 3N - 4(n^2 + N^2) + 7(n^3 + N^3) + 2\sum_{j=1}^{N} K_j$

Chapter 3

Heuristics and Metaheuristics

One of the goals of this project is to create an algorithm able to solve the optimal workers allocation for the crossdocking - JIT scheduling problem described in Chapter 2. As mentioned before, this problem is NP-hard, therefore, the use of heuristics and metaheuristics to obtain a good feasible solution for big instances is an important option to consider.

3.1 Heuristics.

Given the difficulty to obtain an optimal solution by an exact method, i.e. using the simplex method [Murty (1983), Bazaraa et al. (1990), Dantzig and Thapa (2003)] or a branch and bound algorithm [Horowitz et al. (1998), Neapolitan and Naimipour (1998)], for a group of important combinatorial optimization problems when dealing with big instances, some series of algorithms that provided near optimal feasible solutions in a reasonable processing time started to appear in the last decades. These kinds of algorithms were denominated *heuristics*. In this context, “near optimal” and “reasonable” can be considered as subjective terms.

The word "*heuristic*" derives from the Greek "*heuriskein*," which means "to discover", however, this meaning might be changed for the meaning “to search” because that is what heuristics actually do in practice.

Polya (1957) was one of the first authors in mentioning the word heuristic. He claimed: “*heuristics are methods of solution that aims at generality, at the study of procedures which are independent of the subject-matter and apply to all sorts of*

problems". Zanakis and Evans (1981) defined the heuristics as "*simple procedures, often guided by common sense, that are meant to provide good but not necessarily optimal solutions to difficult problems, easily and quickly*". Another definition of heuristic is given by Adam and Ebert (1991) as a "*set of methods and principles whose result is a satisfactory solution of the problem obtained by using simple criteria that allow correctly identifying good decisions*". Their lack of mathematical rigor and the easiness of their designs have made the heuristics gain acceptance by many practitioners who are interested in a useful tool to obtain a quick solution for complex problems in a way that they can understand. On the other hand, one of the major disadvantages of heuristics is that, generally, the quality of their solutions can not be known. Even though there are many advantages when using a heuristic, if an optimal algorithm can be used effectively to solve a problem, this last action must be done. Zanakis and Evans (1981) explained why and when the use of heuristics is desirable and advantageous.

3.2 Metaheuristics.

In their original definition, "*metaheuristics are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space*". Over time, these methods have also included any procedures that employ strategies for overcoming the trap of local optimality in complex solution spaces, specially those procedures that use one or more neighborhood structures as a mean of defining admissible moves to transition from one solution to another, or to build or destroy solutions in constructive and destructive processes [Glover and Kochenberger (2003)]. Gendreau (2002) defined a metaheuristic as a "*general strategy for guiding and controlling inner heuristics*". Metaheuristics provide general frames that allow the creation of new hybrids by combining different concepts derived from classic heuristics, artificial intelligence,

biological evolution, neural systems, and statistical mechanics, among others.

A number of tools and mechanisms that have emerged from the creation of metaheuristic methods have proved to be so effective that metaheuristics have lately become the preferred method used for solving many types of complex problems, especially combinatorial problems [Glover and Kochenberger (2003)].

Metaheuristics can be classified according to their use of memory: metaheuristics with memory and metaheuristics without memory. Unlike the metaheuristics without memory, the metaheuristics with memory contain structures that retain information about decisions previously taken, allowing that way a kind of learning. Commonly, Tabu Search and Scatter Search are classified as metaheuristics with memory while Simulated Annealing and GRASP are considered metaheuristics without memory [De Alba (2004)]. Of course, some metaheuristics that usually do not make use of memory to solve problems can be adapted to make use of it for a specific purpose.

Heuristics and metaheuristics are important approaches used in the operations research field and, in particular, in the combinatorial optimization field, which includes most of the interesting scheduling problems. Over the last years, these approaches have been used to solve complex problems in several applications, including NP-hard scheduling applications [Crainic and Laporte (1998)].

Very general methods having a wide range of applicability are typically weak with respect to their performance. Genetic Algorithms and Neural Networks tend to belong to this category. Problem specific methods achieve a highly efficient performance but with little use in other problem domains. Tabu Search and Simulated Annealing can be counted as examples of this category. Regardless of the category, heuristics and metaheuristics can be viewed as tools for searching a space of feasible alternatives in order to find a good solution within reasonable time limitations, but without any guarantee of optimality [Blazewicz et al. (2001)].

An excellent description and classification of heuristics and metaheuristics are given in Díaz et al. (1996). Glover and Kochenberger (2003) present research done by several renowned authors in the field of the metaheuristics.

To solve the problem described in Chapter 2 an approach based on a combination of two metaheuristics, GRASP and Tabu Search, is proposed. Each one of these metaheuristics is explained next.

3.2.1 GRASP.

Greedy Randomized Adaptive Search Procedure (GRASP) was developed by Feo and Resende (1989) to study a complex set covering problem. In its basic version, GRASP is a multi-start or iterative method that consists of two phases at each iteration: a constructive phase whose result is a feasible and good but not necessarily optimal solution, and a local search procedure, during which, neighborhoods of the solution are examined until a local optimum is attained. The construction phase is based on the idea that a variety of good solutions can be generated by an "intelligent randomization" of the selection step of a greedy heuristic. These solutions are then passed to an exchange procedure that searches for local improvements. The iterations proceed, keeping the best solution found, until a stopping criterion is reached [Laguna and González-Velarde (1991)]. Then, GRASP has two main parameters: one related to the stopping criterion and another related to the amount of "randomization" allowed in the selection step of a greedy heuristic. This last parameter is often called α. The case $\alpha = 0$ corresponds to a pure greedy algorithm while $\alpha = 1$ is equivalent to a completely random algorithm.

Figure 3.1 shows a basic GRASP pseudo-code taken from Resende and Ribeiro (2001). For this particular case, the stopping criterion of the procedure is the maximum number of iterations while the parameter α is not mentioned.

```
procedure GRASP( Max_Iterations )
    Best_solution ← ∞ or -∞; // minimization or maximization problem
    for i = 1, …, Max_Iterations do
        Solution ← Greedy_Randomized_Construction();
        Solution ← Local_Search( Solution );
        Update_Solution( Solution , Best_Solution );
    end for;
    return Best_solution;
end GRASP.
```

Figure 3.1 Pseudo-code for a basic GRASP procedure

Other pseudo-codes for a basic GRASP are shown in Díaz et al. (1996) and in Resende and González-Velarde (2003).

Unlike the rest of the metaheuristics, which operate over previously obtained solutions, GRASP is a constructive method that focus on building high-quality solutions for further processing in order to get better results. At each step of the construction phase, a substructure is added to a partial solution, initially empty, until a complete solution is found.

Each one of the words that form the acronym GRASP characterizes one of the components of this metaheuristic. At each iteration of the construction phase, GRASP maintains a set of candidate elements that can be feasibly incorporated to the partial solution under construction. All candidate elements are evaluated according to a greedy function in order to select the next element to be added to the construction. This greedy function usually represents the marginal increase in the cost function from adding the element to the partial solution. The evaluation of the elements is used to create a restricted candidate list (RCL) which consists of the best elements, i.e. those whose incorporation to the current partial solution results in the smallest incremental costs (for a minimization problem) -*this is the greedy aspect of the method*-. The element to be added into the partial solution is randomly selected from those in the RCL -*this is the random aspect of the metaheuristic*-. Once the selected element is added to the partial solution, the RCL is updated and the incremental costs are recalculated -*this is the adaptive aspect of the metaheuristic*-. The previous high-level description of the components of the GRASP technique was taken from Resende and Ribeiro (2001) and from Laguna and Martí (2003).

The immediate GRASP strategy predecessor is the semi-greedy heuristic proposed by Hart and Shogan (1987), which is also a multi-start approach based on greedy randomized constructions, but without local search. An older background of the GRASP technique can be found in Lin and Kernighan (1973).

The solutions generated by a greedy randomized construction are not necessarily optimal, even with respect to simple neighborhoods. The local search phase usually improves the constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution found in the neighborhood of the current solution. This procedure can be done during the construction phase or at the end of it. Local search is very important for the GRASP technique because it is useful when searching locally optimal solutions in promising regions of the solutions space.

GRASP is based on the premise that good and diverse initial solutions play an important role in the success of local search methods. The effectiveness of a local search procedure depends on several aspects, such as the neighborhood structure, the neighborhood search techniques, the speed of evaluation of the objective function of neighbor solutions, and the initial solution. The construction phase plays a critical role with respect to providing high-quality starting solutions for the local search. Simple neighborhoods structures are usually used. The neighborhood search may be implemented using either a best-improving or a first-improving strategy. In the case of the best-improving strategy, all neighbors are examined and the current solution is replaced by the best neighbor. In the case of the first-improving strategy, the current solution moves to the first neighbor whose cost function value is strictly less than that of the current solution (for a minimization problem).

The first phase (construction) of the GRASP metaheuristic constitutes the core of this technique. The way in which the second phase (local search) of this procedure is done varies from methods that explore simple neighborhoods through more sophisticated procedures. Very often, a local search ends in a locally optimal solution. To escape from these local optima, several strategies have been suggested, i.e. the use of other metaheuristics as an improvement procedure. Actually, GRASP hybridizations with other metaheuristics that use GRASP results as initial solutions are common. Actually, for the local improvement phase of the work under study in this thesis, the Tabu Search algorithm with the best-improving strategy is used.

3.2.2 Tabu Search.

According to Glover and Laguna (1997), "*Tabu Search (TS) is a metaheuristic procedure that guides a local heuristic search algorithm to explore the solution space beyond local optimality*". The local procedure is a search that uses an operation called "move" to define the neighborhood of any given solution.

TS is based on the premise that problem solving, in order to qualify as intelligent, must incorporate "adaptive memory" and "responsive exploration". The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, TS contrasts with memory-less designs that heavily rely on random processes that implement a form of sampling, i.e. GRASP. Memory-based strategies are then the hallmark of TS approaches. Actually, TS is perhaps the metaheuristic procedure that employs memory in the most strategic and direct way.

On the other hand, the emphasis on responsive exploration in TS derives from the supposition that a bad strategic choice can yield more information than a good random choice. In a system that uses memory, a bad choice based on strategy can provide useful clues about how the strategy may be profitably changed.

TS is concerned with finding new and more effective ways of taking advantages of the mechanisms associated with both elements: adaptive memory and responsive exploration. These two elements of the TS procedure have several important characteristics which are summarized in Table 4.2 of Díaz et al. (1996). This previous high-level description of TS was taken from Glover and Laguna (1997) and from Laguna and Martí (2003).

TS was formally proposed by Glover (1986), but its basic form is founded on some previous ideas proposed by himself [Glover (1977)], including elements like short

term memory to prevent the reversal of recent moves, and longer term frequency memory to reinforce attractive components. The basic principle of TS is to pursue local search whenever it encounters a local optimum by allowing non-improving moves. Cycling back to previously visited solutions is prevented by the use of "memories", called "tabu lists", which record the recent history of the search.

Gendreau (2002) considered TS as an extension of classical local search procedures. In fact, he mentioned that TS can be seen as simply the combination of local search with short-term memories. According to him, the two first basic elements of any TS heuristic are the definition of its "search space" and its "neighborhood structure". The search space of a TS heuristic is simply the space of all possible solutions that can be considered (visited) during the search. The neighborhood of the current solution *S*, denoted by *N(S)*, is a subset of the search space defined by the solutions obtained by applying a single local transformation to *S*. In general, for any specific problem, there are many more possible (an even attractive) neighborhood structures than search space definitions. This follows from the fact that there may be several feasible neighborhood structures for a given definition of the search space. Choosing a search space and a neighborhood structure is by far the most critical step in the design of any TS heuristic.

Figure 3.2 shows basic TS pseudo-code taken from Pinedo (2002). In this particular case, the pseudo-code is applied to a scheduling problem and the stopping criterion of the procedure is the maximum number of iterations allowed.

```
procedure Tabu-Search( S1 , Max_Iterations )
Step 1:
    Set k = 1
    Set S0 = S1
Step 2:
    Select a candidate schedule Sc from the neighborhood of Sk
    If the move Sk → Sc is prohibited by a mutation on the tabu-list then
        Set Sk+1 = Sk
        Go to Step 3
    If the move Sk → Sc is not prohibited by any mutation on the tabu-list then
        Set Sk+1 = Sc
        Enter reverse mutation at the top of the tabu-list
        Push all other entries in the tabu-list one position down
        Delete the entry at the bottom of the tabu-list
    If Value(Sc) < Value(S0) then
        Set S0 = Sc
Step 3:
    Increment k by 1
    If k = Max_Iterations then
        Stop
    Otherwise
        Go to Step 2
```

Figure 3.2 Pseudo-code for a basic Tabu Search procedure

It is interesting to note that in the same year that TS appeared, a similar approach named steepest ascent / mildest descent was proposed by Hansen (1986). However, in the traditional steepest ascent / mildest descent optimization method, the search stops when the value of the objective function evaluated in a solution *S* is not better that the obtained value in the previous iteration, this is, when a local optimum has been found. To avoid this, TS keeps exploring solutions, even non-improving ones.

GRASP and TS metaheuristics are also mentioned in the next chapters, which explain the approach used to solve the problem under study.

Chapter 4

Solution Approach for the Crossdocking - Just in Time Scheduling Problem

This chapter deals with the solution approach for the problem described in Chapter 2 when the number of teams of workers in each side of the crossdock is fixed and known (*m* and *M*). This is called the crossdocking - JIT scheduling problem and it represents a sub-problem (or a particular case) of the problem under study in this thesis work. This sub-problem is, as mentioned before, NP-hard.

4.1 Solution Form.

To solve this problem it is necessary, for the inbound area, to have each one of the *n* incoming jobs assigned in a position in one of the *m* breakdown machines and a completion time. Two notes can be cited with regard to this statement:

- The use of a machine has no fixed cost; then, the model will make use of all of the available machines because that way it is easier to accommodate the jobs in order to obtain better results. It is convenient to remember that $n > m$ and $N > M$
- Once the completion time for a job is obtained, its earliness and tardiness penalties are directly obtained

Using a notation similar to the one used in Laguna and González-Velarde (1991), the incoming schedule, S_I, has the form:

$$S_I = \{\pi_I, c_I\}$$

where:

- $\pi_I = \{\pi_{I1}, \pi_{I2}, \ldots, \pi_{Im}\}$ is the assignment of the n incoming jobs to the m breakdown machines
- c_I is the set of completion times for the n incoming jobs

where π_{Ik} represents the sequence in which the n_k incoming jobs assigned to machine k will be processed. This π_{Ik} sequence has the following form:

$$\pi_{Ik} = \{\pi_{Ik}(1), \pi_{Ik}(2), \ldots, \pi_{Ik}(n_k)\}$$

where $\pi_{Ik}(i)$ is the index of the incoming job in position i on machine k.

A similar reasoning and representation is used for the outbound area and its outgoing schedule, S_O.

As mentioned in Chapter 1, the development of a computer program that solves the integer programming model developed by Li et al. (2004) and the development of a metaheuristic algorithm to find *good* solutions for different problem instances are two important tasks to consider in the project. These tasks are mentioned in the following two sections of this chapter.

4.2 Exact Method of Solution.

Our solution of the integer programming model for the crossdocking - JIT scheduling problem developed by Li et al. (2004) is obtained through a computer program that makes use of the ILOG CPLEX 9.1 library. The results obtained by this program are compared to the results obtained by their program (which also uses the ILOG CPLEX library) for the 16 problem instances presented in their work. Obviously, the objective values shown by both versions must be the same when both algorithms reach the optimal value. The code of the computer program is easily done once the integer model is obtained. It is necessary just to follow the coding conventions mentioned in the ILOG CPLEX 9.1 User's Manual.

ILOG CPLEX search for solutions in nodes trees created according to the model

defined. So, different orders in the definitions of variables and/or constraints might create different search trees (and very likely different solutions if the computer runs out of memory before reaching the optimal solution).

Figure 4.1 shows the output given by the exact method algorithm for the following small problem instance. This problem instance creates an MIP with 71 variables and 99 constraints which can be seen in Appendix 2.

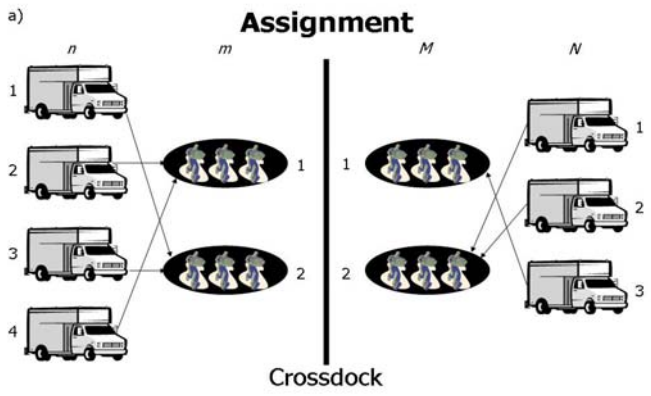
n =	4
m =	2
N =	3
M =	2
α =	1
$ß$ =	100
G =	10000

	r	p	d
1	5	6	14
2	6	12	19
3	8	6	16
4	7	1	11

	P	D
1	2	22
2	5	28
3	19	39

S	1	2	3
1	0	1	1
2	1	1	1
3	1	0	1
4	1	1	0
	K_1=3	K_2=3	K_3=3

Table 4.1 Input data for a crossdocking - JIT scheduling problem instance with 4 incoming jobs and 2 breakdown machines, and 3 outgoing jobs and 2 buildup machines



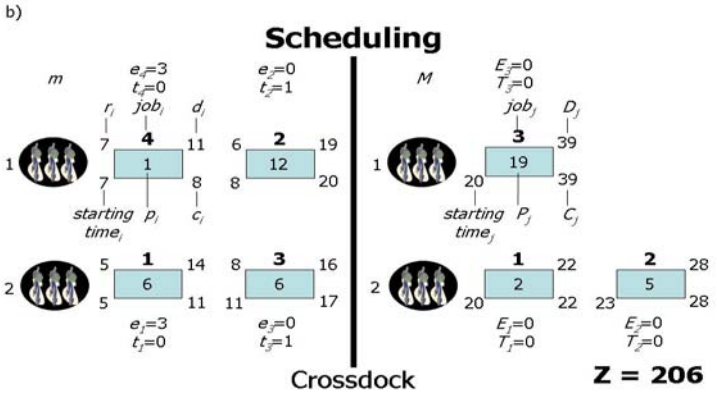


Figure 4.1: Solution for a crossdocking - JIT scheduling problem instance: a) Assignment; b) Scheduling

For the particular crossdocking - JIT scheduling problem instance shown in Table 4.1, Figure 4.1 presents an optimal solution with an objective value of 206 (t_2 = 1, e_3 = 3, e_4 = 3).

The computer program that solves the crossdocking - JIT scheduling problem using the exact model is shown in Appendix 3 (only in electronic format in the software and data CD).

4.3 Alternative Method of Solution.

The problem under study is NP-hard. The experiments show that the integer programming solver of ILOG CPLEX 9.1 takes a long time to reach an optimal solution (when possible) for large problem instances using the formulation of Chapter 2. To obtain high quality faster solutions for the problem an approach based on a combination of two metaheuristics: Reactive GRASP (RG), and Tabu Search (TS), is proposed. The whole algorithm is abbreviated as RGTS. The RG procedure is used to construct initial feasible solutions which, in turn, are used by the TS procedure in order to try to improve those solutions. RG algorithm is embedded in RGTS, so RGTS will offer equal or better objective values than just RG.

This combination of procedures: GRASP and Tabu Search, was used to solve a

similar problem in Laguna and González-Velarde (1991) with good results and is referred by the authors as GTS.

4.3.1 Solutions Construction.

As mentioned before, a Reactive GRASP (RG) metaheuristic algorithm is used to construct initial feasible solutions for the crossdocking - JIT scheduling problem. The following figure describes this RG algorithm:

```
1   for each RG iteration
2       for each section of the crossdock
3           while there are jobs to be scheduled in the section
4               select the greedy function to be used
5               adapt the restricted candidate list (RCL) of admissible jobs according to the selected greedy function
6               select job to be scheduled
7               adapt the RCL of machines according to the time horizon
8               select machine to host the selected job
9               insert the selected job in the selected machine
10              update the schedule
11              mark the job as scheduled
12      if the schedule is good enough and different enough
13          include the schedule in the list of good schedules
```

Figure 4.2: Pseudo-code for the RG algorithm

For the RG algorithm shown above, there are several notes to comment:

For line 1, we defined 3000 iterations.

For line 4, we defined 4 greedy functions (*gf*) that can be used:

- $gf_1(j)$ = *ideal starting time*$_j$ + *potential slack time*$_j$ = (*due date*$_j$ - *processing time*$_j$) + (*due date*$_j$ - *processing time*$_j$) - *release time*$_j$
- $gf_2(j)$ = *ideal starting time*$_j$ = *due date*$_j$ - *processing time*$_j$
- $gf_3(j)$ = *release time*$_j$
- $gf_4(j)$ = *due date*$_j$

These greedy functions are used in both sides of the crossdock and as each job *j* can

only start after its release time, the release time for outgoing containers is defined by their source containers as described in Chapter 2.

The values for the selected greedy function at any given iteration are obtained for each non scheduled job *j* at that iteration. To select the greedy function, a Reactive strategy is used. In GRASP context, using a Reactive strategy means that a parameter value is not fixed, but instead is chosen by the algorithm from a discrete set of possible values. The selection of this parameter value is guided by self-constructed probabilities obtained along previous iterations of the algorithm. This Reactive strategy of GRASP changes the probabilities for the parameter values of being selected in order to favor those values that historically have produced good solutions. In our RG algorithm, for the first 100 iterations, we simply use a uniform distribution to choose the greedy function. In other words, the 4 different greedy functions mentioned above are equally likely to be chosen for the first 100 iterations. For the rest of the iterations, we use the Reactive strategy previously mentioned. The Reactive strategy in the context of GRASP was proposed by Prais and Ribeiro (2000).

The Reactive strategy has been widely used to obtain the value of the usually single parameter α of GRASP but it had not been used to select the greedy function to be applied in an iteration of the algorithm. Usually, a single greedy function is used for all of the iterations of the GRASP algorithm. However, the experiments showed that when we used a single greedy function in our algorithm, a particular greedy function showed better results than the others for some problem instances (see Table 4.6). On the other hand, the Reactive strategy that we applied showed good results for all of the problem instances. So, we decided to apply that strategy in our algorithm to the different greedy functions mentioned above in order to obtain more robustness. In other words, using the Reactive strategy of GRASP, the algorithm is not restricted to one problem instance. This is an important contribution of this work to the GRASP methodology and it has a relationship with the hyper-heuristics field where the algorithms choose a heuristic among a set of different heuristics, depending on the properties of the problem, in order to solve an optimization problem. Most of the time,

each heuristic is selected according to its performance which can be increased or decreased depending on a learning mechanism. This places hyper-heuristics in a higher level of abstraction than most heuristics and allows the user to operate efficiently and effectively within a more general framework [Burke et al. (2003)]. In our case, instead of selecting a heuristic to solve the problem, we select a greedy function per iteration of the GRASP algorithm to do it. By doing this, we avoid the time consuming fine-tuning task, which usually performs well just for some problem instances.

For line 5, we defined that a job *j* belongs to the RCL of jobs if:

$gf_h(j) \leq minValueJ + \alpha_{jobs} * (maxValueJ - minValueJ)$, for each non scheduled job *j*

where *minValueJ* and *maxValueJ* are the minimum and maximum values of all of the $gf_h(j)$ values that correspond to non scheduled jobs, respectively, and α_{jobs} ($0 \leq \alpha_{jobs} \leq 1$) is a parameter that controls the amount of randomization allowed for jobs selection. The parameter value for α_{jobs} is selected using the Reactive strategy from the following discrete set of 7 possible values: 0, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30.

For line 6, the job is simply randomly selected from the RCL of jobs.

For line 7, we defined that a machine *k* belongs to the RCL of machines if:

$machine\ horizon_k \leq minValueM + \alpha_{machines} * (maxValueM - minValueM)$, for each machine *k*

where *minValueM* and *maxValueM* are the minimum and maximum values of all of the $machine\ horizon_k$ values, respectively, and $\alpha_{machines}$ ($0 \leq \alpha_{machines} \leq 1$) is a parameter that controls the amount of randomization allowed for machines selection. The $machine\ horizon_k$ value is defined as the completion time of the job assigned in the last position of machine *k*. If machine *k* has no jobs assigned to it then the $machine\ horizon_k$ value is equal to 0. Again, the parameter value for $\alpha_{machines}$ is selected using the Reactive strategy from the following discrete set of 7 possible values: 0, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30.

For line 8, the machine is simply randomly selected from the RCL of machines.

For line 9, the algorithm applies insertion of jobs in a way similar to the one done in Mazzini and Armentano (2001), but adapting that procedure to the parallel machines problem's structure of this work because they just consider single machine scheduling. When a job *j* is going to be inserted in a machine schedule, the procedure tries to put the job in a place where the cost is minimal. If there is no overlapping between the job *j* and any other already scheduled job in the machine, the procedure starts the insertion of another job. Otherwise, it is necessary to eliminate the overlapping between jobs in such a way the cost increase be minimal. The algorithm considers four possible moves in order to eliminate the overlapping between the inserted job and the already scheduled jobs. These moves are deeply explained in Mazzini and Armentano (2001).

For line 10, the algorithm updates the idle times over a partial feasible schedule in order to decrease its cost. This *update* procedure consists of two phases: in the first, the jobs are shifted to the left, and in the second, the jobs are shifted to the right. These phases are also deeply explained in Mazzini and Armentano (2001). Again, we adapted their procedure to our parallel machines problem's structure. An important characteristic of this *update* procedure is that idle times are inserted during the construction stage. This allows offering a better performance than most of the approaches found in the literature which insert the idle times over complete schedules.

In the same line 10, our implementation includes an ejection chain process in the schedule (after the insertion of idle times) if it is *convenient* and possible, i.e., a movement to the left of the $job_{[j]}$ in a machine (job in position *j* of the machine) might move to the left the $job_{[j-1]}$ in the same machine and this last movement might cause the $job_{[j-2]}$ to be moved to the left in the same machine, and so on. Sometimes, a job might be moved during the process to some other machine (when working with more than one machine in the section of the crossdock that corresponds to that job) if necessary. This ejection chain process is considered only when working with the right section of the crossdock due to the flexibility in the outgoing releases times (R_j's) and it is triggered when in any iteration of the algorithm the insertion of a job

causes that same job or another job to be tardy. This process might affect just the right section of the crossdock, or both (when the movement to the left of a job in the right section of the crossdock affects its predecessors or source containers). An example of the chain ejection process can be seen in Figure 4.3.

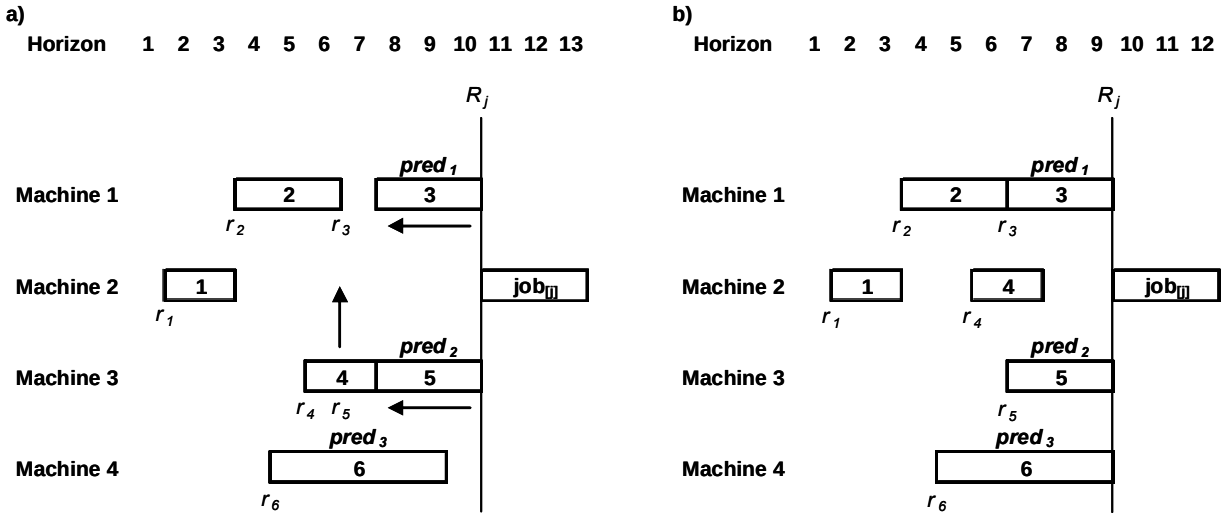


Figure 4.3: An example of the ejection chain process: a) Before the movement of the $job_{[j]}$; b) After the movement of the $job_{[j]}$

It can be seen in Figure 4.3a that the movement to the left of the $job_{[j]}$ in the outbound area affects its predecessors 1 and 2 ($job_{[3]}$ and $job_{[5]}$, respectively) because their completion times are equal to R_j, but it does not affect its predecessor 3 ($job_{[6]}$) because its completion time is lower than R_j. It can be seen in Figure 4.3b that, in the inbound area, the movement to the left of the $job_{[j]}$'s predecessor 1 does not affect the job at its left ($job_{[2]}$), but the movement to the left of the $job_{[j]}$'s predecessor 2 does affect the job at its left ($job_{[4]}$) which is moved to another machine. The R_j decreases by one unit.

In the context of the problem, *convenient* means to have a schedule with a lower cost. We decided to apply this ejection chain process due to the Proximate Optimality Principle (POP) which stipulates that "good solutions at one level are likely to be found close to good solutions at an adjacent level" [Glover and Laguna (1997), Fleurent and Glover (1999)]. In our case, *level* refers to a stage of the construction

phase, and we have defined mechanisms for moving across these levels so that the principle applies.

For lines 12 and 13, we decided to keep a set with the 5 best diverse schedules called *elite solutions* or set *S*. The set *S* is used to guide the procedure as follows: when a schedule *s* with *cost(s)* < *cost(worst(S))* is generated, it is a candidate to be added to *S* replacing *worst(S)* if *s* is different enough from all of the schedules in *S*. In our case, *worst(S)* is the schedule in the fifth position in the ordered by objective value set *S*. To measure how different is *s* from a schedules $s' \in S$, we count the number of identical positions of the jobs in the machines for both schedules *s* and *s'*. If the number of identical positions is greater than 50% of the number of jobs in the complete schedule, the solution *s* is discarded unless an aspiration criteria is satisfied, i.e. *cost(s)* < *cost(best(S))*. A deeper explanation about the Diversification strategy can be found in Fleurent and Glover (1999). Appendix 4 shows the set *S*, output of the RG algorithm for a crossdocking - JIT scheduling problem instance with 20 incoming jobs and 3 breakdown machines, and 21 outgoing jobs and 3 buildup machines. The input data for this problem instance is also shown in Appendix 4. It can be seen in the same appendix that the *elite solutions* in the set *S* are shown in ascending order by objective value. Sometimes, some different solutions have the same objective value.

Figure 4.4 shows a graphical sequence of the output given by an iteration of the RG algorithm just for the left section of the crossdock for the following problem instance:

α =	1
β =	100
n =	10
m =	2

i	r_i	p_i	d_i
1	21	2	24
2	14	2	16
3	6	5	13
4	3	2	6
5	8	5	16
6	1	2	3
7	28	6	34
8	9	6	16
9	4	3	10
10	0	3	4

Table 4.2 Input parameters {r_i, p_i, d_i} for a crossdocking - JIT scheduling problem instance with 10 incoming jobs and 2 breakdown machines

It can be seen in Figure 4.4 that at each step of an iteration of the RG algorithm a job is inserted into the schedule (GRASP is a constructive method). All schedules (partial or complete) are feasible. After that insertion, the already mentioned *update* procedure is done.

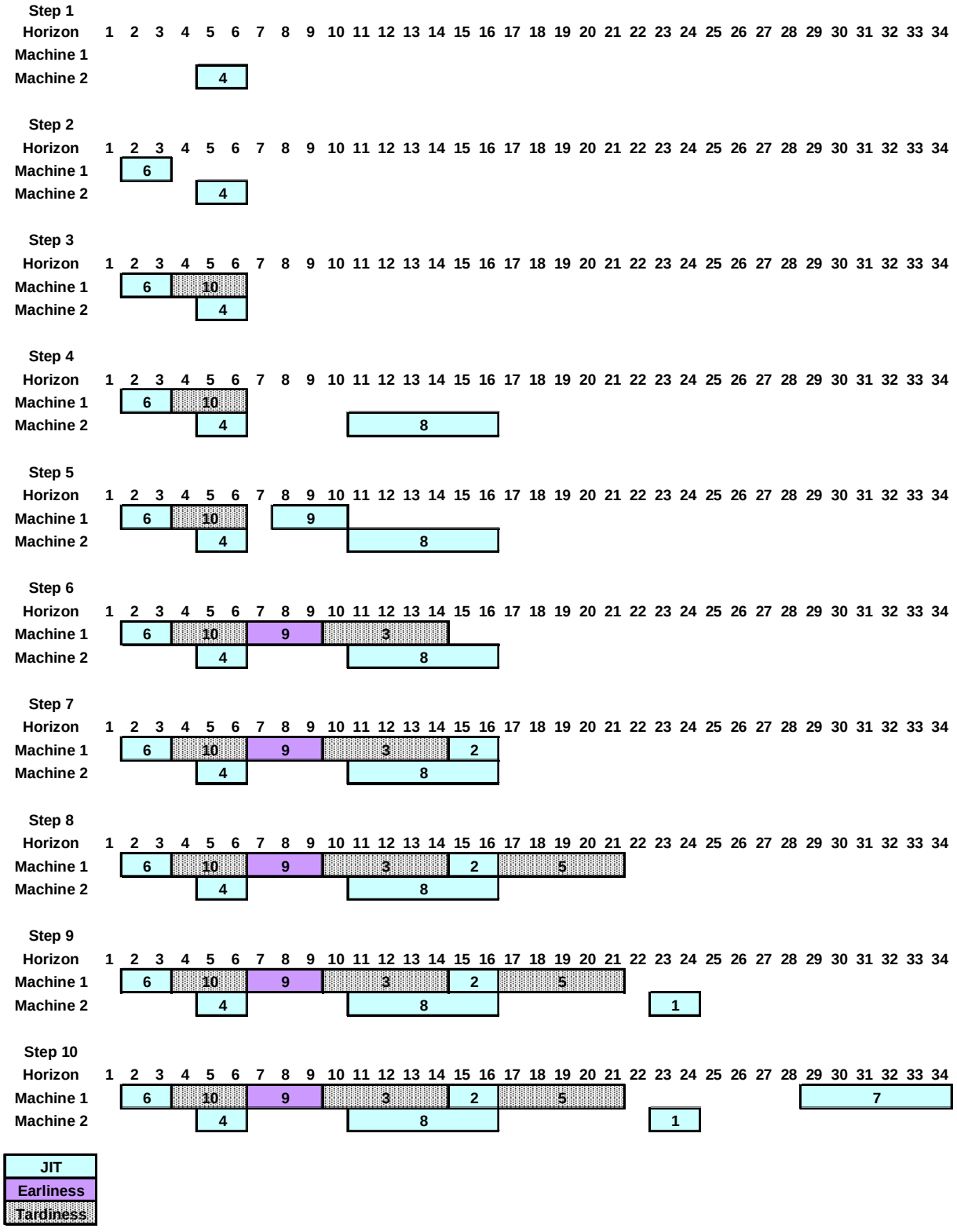


Figure 4.4: Graphical sequence of the output of an iteration of the RG algorithm for the inbound area of the crossdock

4.3.2 Solutions Improvement.

The solutions generated by the RG algorithm and kept in the set *S* previously discussed are used as starting points for the TS algorithm. In our implementation, we make use of a traditional “fixed” size short-term memory in the following context: the memory size is related to the size of the problem instance and it is calculated as the number of jobs divided by the number of machines in each section of the crossdock. So, we have one memory size per section: $memory\,size_{inbound} = \lceil n/m \rceil$ and $memory\,size_{outbound} = \lceil N/M \rceil$. Once the memory size per section is computed it remains the same for all of the 100 iterations that we decided for the TS algorithm.

Given an initial solution, TS tries to improve it by making a succession of moves. A move can improve, deteriorate or leave the solution with the same objective value. Of course, even if the solution remains with the same objective value, the schedule changes from iteration to iteration. Regardless of the move applied, the solution remains feasible.

The moves considered in our algorithm are:

- *InsertLeftSameMachine*($\pi_k(i)$) which consists of transferring the job currently in position *i* of machine *k* one position immediately before it in the same machine. Just after the move is made, the completion time of job *i* ($c_{[i]}$) in machine *k* is equal to the starting time of job *i*-1 ($s_{[i-1]}$) in the same machine *k*. This move might cause infeasibility due to two reasons: the release time constraint of job *i* ($r_{[i]}$) is broken, or there is overlapping between jobs *i* and *i*-2. In any case, the infeasibility is eliminated by shifting to the right the necessary jobs in machine *k* as many unit times as needed
- *InsertRightSameMachine*($\pi_k(i)$) which consists of transferring the job currently in position *i* of machine *k* one position immediately after it in the same machine. Just after the move is made, the starting time of job *i* ($s_{[i]}$) in machine *k* is equal to the completion time of job *i*+1 ($c_{[i+1]}$) in the same machine *k*. This move might cause

infeasibility due to overlapping between jobs *i* and *i*+2. This infeasibility is eliminated by shifting to the right the necessary jobs in machine *k* as many unit times as needed

- *InsertDifferentMachine*($\pi_k(i)$, π_q) which consists of transferring the job currently in position *i* of machine *k* into machine *q*. This move implies removing job *i* from machine *k* and putting it into machine *q* according to the *insertion* procedure mentioned in line 9 of Figure 4.2. This move does not cause infeasibility
- *ExchangeSameMachine*($\pi_k(i)$) which consists of allowing jobs in positions *i* and *i*-1 of machine *k* to exchange positions (also known as Adjacent Pair-wise Interchange or API). Just after the move is made, the starting time of job *i* ($s_{[i]}$) is equal to the starting time of job *i*-1 ($s_{[i-1]}$) and the completion time of job *i*-1 ($c_{[i-1]}$) is equal to the completion time of job *i* ($c_{[i]}$). This move causes infeasibility if the release time constraint of job *i* ($r_{[i]}$) is broken. This infeasibility is eliminated by shifting to the right the necessary jobs in machine *k* as many unit times as needed. We do not consider an exchange between jobs in positions *i* and *i*+1 in the same machine because the exchange of these two jobs is analyzed when *i* is increased to *i*+1
- *ExchangeDifferentMachine*($\pi_k(i)$, $\pi_q(j)$) which consists of allowing jobs in position *i* and *j* of machines *k* and *q*, respectively, to exchange machines assignments. This move implies removing job *i* from machine *k* and putting it into machine *q* and removing job *j* from machine *q* and putting it into machine *k* simultaneously according to the *insertion* procedure mentioned in line 9 of Figure 4.2. This move does not cause infeasibility. A threshold value for each section of the crossdock called $maxDistance_{inbound}$ and $maxDistance_{outbound}$ are used for this movement to detect and eliminate from consideration unreasonable moves. These threshold values are calculated as the 25% of the difference between the maximum completion time and the minimum completion time of all of the jobs in the corresponding section of the crossdock. If the absolute value of the completion

time of $\pi_k(i)$ minus the completion time of $\pi_q(j)$ is greater than the corresponding threshold value, the move is not considered

Figure 4.5 shows the 5 movements considered.

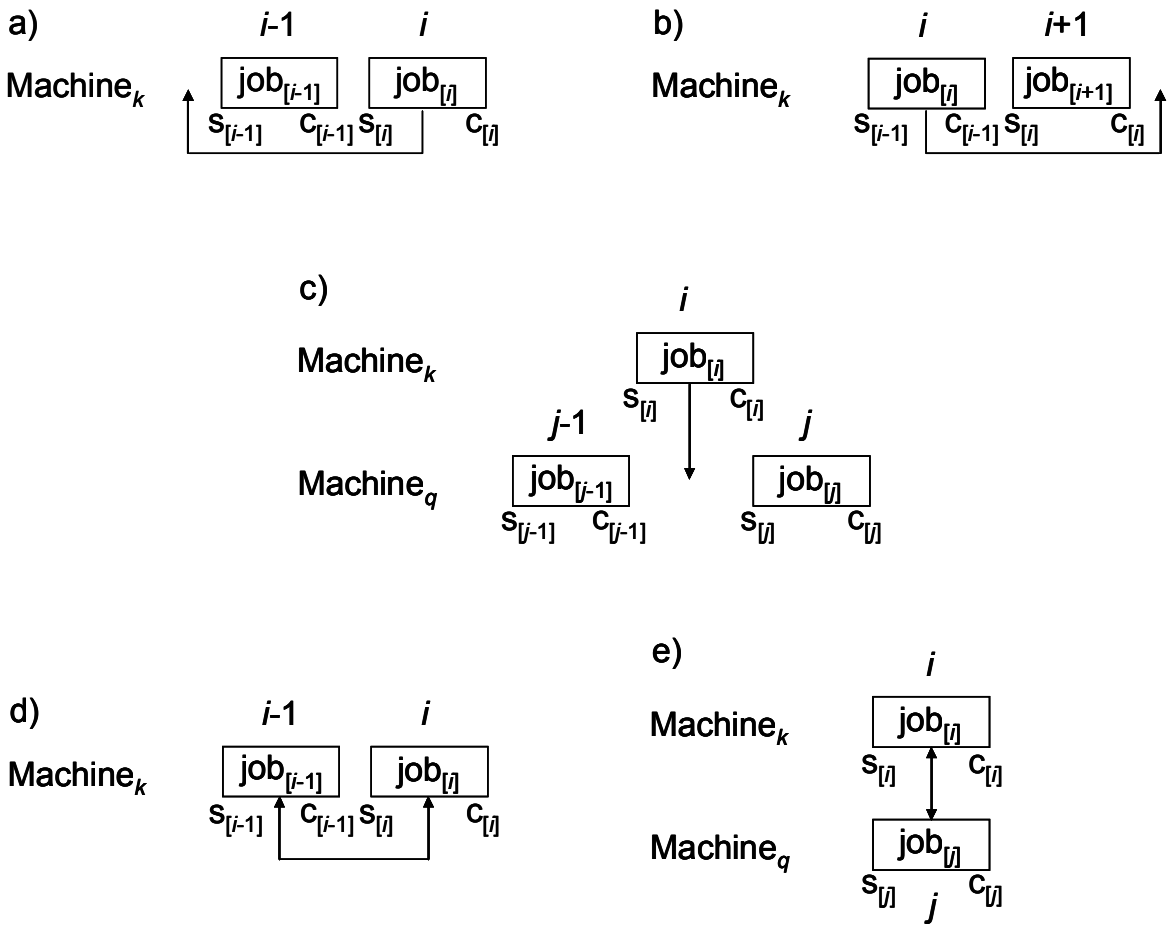


Figure 4.5: Movements of the algorithm: a) *InsertLeftSameMachine*($\pi_k(i)$); b) *InsertRightSameMachine*($\pi_k(i)$); c) *InsertDifferentMachine*($\pi_k(i)$, π_q); d) *ExchangeSameMachine*($\pi_k(i)$); e) *ExchangeDifferentMachine*($\pi_k(i)$, $\pi_q(j)$)

All moves make use of the *update* procedure mentioned in line 10 of Figure 4.2 after the move is made to try to improve the solution. If the move involves two machines, the *update* procedure is applied over both machines.

The moves are applied over all jobs in each section of the crossdock when possible, i.e. the move *InsertLeftSameMachine*($\pi_k(i)$) cannot be applied to the first job in a machine and the move *InsertRightSameMachine*($\pi_k(i)$) cannot be applied to the last

job in a machine. This creates a neighborhood for each one of the two sections of the crossdock. A list of moves and their associated move values is made at every step of the procedure for these neighborhoods. We simply select the move with the best objective value of both neighborhoods to be applied to the current schedule. In case of ties, we select the move randomly. Obviously, the selected move is admissible according to the tabu restrictions being imposed.

The following figure describes the TS algorithm we use to solve the problem:

```
1   for each RG solution s_0
2       bestSolution = s = s_0
3       bestValue = F(s) = F(s_0)
4       for each TS iteration
5           for each section of the crossdock
6               for each job in the section
7                   for each neighbor of the job
8                       save the move and its objective value
9           select the best move that is not tabu
10          apply the move to s
11          update the schedule s
12          update the corresponding tabu structure of the selected move
13          if F(s) < F(s_0)
14              bestSolution = s
15              bestValue = F(s)
```

Figure 4.6: Pseudo-code for the TS algorithm

Appendix 5 shows the set *S*, output of the RGTS algorithm for the same problem instance shown in Appendix 4. It can be seen in Appendix 5 that sometimes the best RGTS solution (Solution 2 in Appendix 5) does not come from the best RG solution (Solution 1 in Appendix 4).

Figure 4.7 shows the output given by the RGTS algorithm for the following problem instance:

α =	1
β =	100
n =	10
m =	2
N =	11
M =	3

i	r_i	p_i	d_i
1	21	2	24
2	14	2	16
3	6	5	13
4	3	2	6
5	8	5	16
6	1	2	3
7	28	6	34
8	9	6	16
9	4	3	10
10	0	3	4

j	$preds_j$	R_j	P_j	D_j
1	6, 7	34	2	58
2	1, 6, 9	24	3	54
3	3, 6, 9	11	3	26
4	3, 7	34	2	46
5	3	11	1	15
6	3, 6, 7, 8	34	4	68
7	5, 6, 8	16	3	29
8	1, 5, 6, 7, 10	34	5	77
9	3, 9	11	2	32
10	1, 5, 7, 8, 10	34	5	68
11	1, 8	24	2	27

Table 4.3 Input data for a crossdocking - JIT scheduling problem instance with 10 incoming jobs and 2 breakdown machines, and 11 outgoing jobs and 3 buildup machines

a)

i	s_i	c_i	e_i	t_i	Assigned machine
1	22	24	0	0	2
2	15	17	0	1	1
3	6	11	2	0	2
4	4	6	0	0	1
5	11	16	0	0	2
6	1	3	0	0	1
7	28	34	0	0	1
8	9	15	1	0	1
9	6	9	1	0	1
10	1	4	0	0	2

j	S_i	C_i	E_i	T_i	Assigned machine
1	56	58	0	0	2
2	51	54	0	0	1
3	23	26	0	0	1
4	44	46	0	0	3
5	14	15	0	0	3
6	64	68	0	0	1
7	26	29	0	0	2
8	72	77	0	0	2
9	30	32	0	0	1
10	63	68	0	0	3
11	25	27	0	0	3

b)

Inbound area

i (s_i, c_i, e_i, t_i)

Machine 1: 6 (1, 3, 0, 0) 4 (4, 6, 0, 0) 9 (6, 9, 1, 0) 8 (9, 15, 1, 0) 2 (15, 17, 0, 1) 7 (28, 34, 0, 0)
Machine 2: 10 (1, 4, 0, 0) 3 (6, 11, 2, 0) 5 (11, 16, 0, 0) 1 (22, 24, 0, 0)

Outbound area

j (S_j, C_j, E_j, T_j)

Machine 1: 3 (23, 26, 0, 0) 9 (30, 32, 0, 0) 2 (51, 54, 0, 0) 6 (64, 68, 0, 0)
Machine 2: 7 (26, 29, 0, 0) 1 (56, 58, 0, 0) 8 (72, 77, 0, 0)
Machine 3: 5 (14, 15, 0, 0) 11 (25, 27, 0, 0) 4 (44, 46, 0, 0) 10 (63, 68, 0, 0)

c)

Inbound area

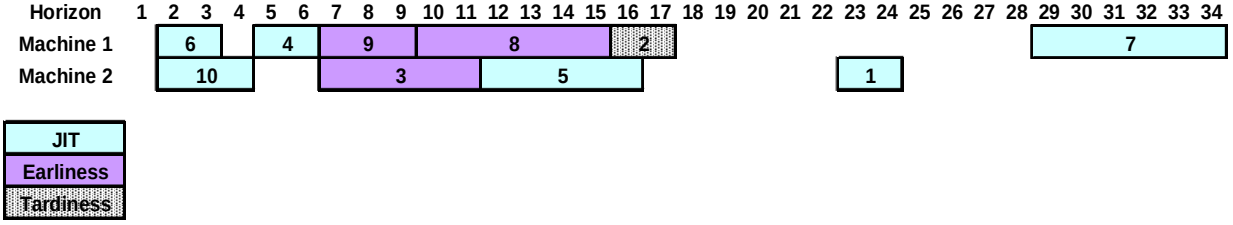


Figure 4.7: RGTS solution: a) Tabular form; b) Machine-Job form; c) Graphical form (only inbound area)

The outputs s_i and S_j in Figure 4.7 are not variables of the model described in Chapter 2; however, they are very useful when coding the RGTS algorithm and they represent the starting times for incoming jobs and outgoing jobs, respectively.

For the particular crossdocking - JIT scheduling problem instance shown in Table 4.3, Figure 4.7 presents an optimal solution with an objective value of 104 (t_2 = 1, e_3 = 2, e_8 = 1, e_9 = 1). However, as mentioned in Chapter 3, heuristics and metaheuristics can not guarantee optimality.

The computer program that solves the crossdocking - JIT scheduling problem using the RGTS algorithm is shown in Appendix 6 (only in electronic format in the software and data CD).

4.4 Computational Experiments.

The test data that we used for the experiments for the problem are the same 16 problem instances shown in Li et al. (2004). They provided us with this data. All data are integer values. The way this data was generated is specified in their work. Although we used these 16 crossdocking - JIT scheduling problem instances to make valid and real comparisons between the behavior of our RGTS metaheuristic and their metaheuristic with best results which they called LPGA, a computer program was coded in C language which can create new problem instances (for possible future research) under the conditions mentioned in their work. This computer program is shown in Appendix 7 (only in electronic format in the software and data CD) and it uses the following parameters:

- r_i ~ round(UNIF(0, 10))
- p_i ~ ceil(EXP(5)) (p_i = 0 does not make any sense)

- d_i ~ round(UNIF(0, 5)) + r_i + p_i
- K_j ~ discrete UNIF(0, n)
- $preds_j$ or column j of S_{ij} matrix ~ random sample without replacement
- P_j ~ ceil(EXP(5)) (P_j = 0 does not make any sense)
- D_j ~ round(UNIF(0, 5)) + max{d_i} that belongs to $preds_j$ + P_j

$$i = 1, ..., n; j = 1, ..., N$$

where n is the number of incoming jobs and N is the number of outgoing jobs.

All parameters are integer values and, obviously, these values can be easily changed in the computer program. All these parameters were mentioned in Chapter 2 and their values are independent of the number of breakdown machines (m) and the number of buildup machines (M).

This problem instance generator program creates as output a text file which is used as input for the exact model algorithm shown in Appendix 3 and for the RGTS algorithm shown in Appendix 6 (both appendixes are only shown in electronic format in the software and data CD). This text file structure is the following:

1 row	{ n m N M *timeHorizon*
n rows	{ r_i p_i d_i
N rows	{ S_{ji}
N rows	{ P_j D_j

Figure 4.8: Input file structure for the exact model algorithm and for the RGTS algorithm

The crossdocking - JIT scheduling problem instances provided by Li et al. (2004) are shown in Appendix 8 (only in electronic format in the software and data CD) and their structure is the same shown in Figure 4.8.

The penalty for one job to be early one unit time (α) is 1 and the penalty for one job to be tardy one unit time (β) is 100 just as in Li et al. (2004).

A personal compatible computer with a processor Pentium IV with a speed of 3 GHz

and a RAM of 1 GB was used to run all of our experiments. The source codes of all of the programs mentioned in this thesis were written in the C programming language [Gottfried (1997)]. For the algorithm that solves the crossdocking - JIT scheduling problem using the integer programming model described in Chapter 2, the ILOG CPLEX 9.1 library was used.

4.4.1 Testing and Comparison of Results.

We show in Table 4.4 the results found from our exact model algorithm and from the RG and the TS algorithms described before. The column *data set* specifies the following parameters: *number of incoming containers* (*n*), *number of machines in the import area* (*m*), *number of outgoing containers* (*N*), *number of machines in the export area* (*M*), *time horizon* (*th*). The parameter *time horizon* is used to create the release times of each incoming container and it represents a time window for their arrivals.

ID	Data set (*n-m-N-M-th*)	CPLEX	Time CPLEX (seconds)	RG	Time RG (seconds)	RGTS	Time TS (seconds)	Time RGTS (seconds)
1	10-2-11-3-30	**104**	~0	104	~0	104	1	1
2	15-3-14-2-35	**8**	41	8	~0	8	1	1
3	20-3-21-3-40	*615*	8221	744	1	616	4	5
4	32-3-34-4-50	*926*	12684	724	1	423	14	15
5	30-4-29-5-46	*211*	12845	211	1	211	13	14
6	32-4-33-5-50	*5*	67704	5	1	5	14	15
7	30-5-30-5-90	**1**	2	1	1	1	12	13
8	40-5-38-5-60	*9*	21503	9	2	9	27	29
9	42-5-43-5-55	*112*	21000	114	2	111	30	32
10	32-5-35-6-54	**3**	12	3	1	3	19	20
11	40-5-43-6-56	*4*	21356	10	2	4	32	34
12	56-5-57-6-62	*21281*	20733	1671	3	858	90	93
13	34-6-32-6-60	*7*	45003	7	1	7	21	22
14	50-7-60-8-70	*10*	34934	10	3	10	88	91
15	90-8-89-9-70	*2351*	48065	14	5	13	450	455
16	93-9-94-9-75	*4526*	36067	48	7	33	480	487

Table 4.4: Experimental results of our solution approach for the crossdocking - JIT scheduling problem

Italicized results from column *CPLEX* in Table 4.4 indicate that it cannot be guaranteed that an optimal solution has been reached because the memory limits

were exceeded. In those cases, the best known integer value is reported (instances 3, 4, 5, 6, 8, 9, 11, 12, 13, 14, 15, and 16). Bold results from the same column *CPLEX* show the instances in which the optimal solutions were found (instances 1, 2, 7, and 10).

As it can be seen in Table 4.4, the RG algorithm is effective and fast. However, it can be improved in many cases by the TS algorithm which is much slower.

The results of the RG and RGTS algorithms for the 16 crossdocking - JIT scheduling problem instances already mentioned are shown in Appendix 9.

Table 4.5 shows a comparison between Li et al. (2004) results and our results. For the column *CPLEX* of this table we used a merge of our results and their results when using the ILOG CPLEX library. The value shown in the column *CPLEX* is the best objective value of both results. In case of tie in the objective value of both results, we report for the column *Time CPLEX* the result of the algorithm that took the lowest time to solve the problem. The differences in the objective values from column *CPLEX* of both works are not due to the memory limits because both groups of authors used a computer with 1 MB of RAM. It is very likely that the differences are due to the way the variables and/or constraints were defined in the computer programs or the parameters used to run the ILOG CPLEX library. We used the default parameters in our experiments when using the ILOG CPLEX library 9.1. The order in which the variables and constraints were defined in our computer program is the same order in which the integer programming model was written in Chapter 2. We do not know the way their computer program was coded and we do not know how they set the ILOG CPLEX parameters either. Besides, they do not specify the ILOG CPLEX version used in their work.

ID	Data set (*n-m-N-M-th*)	CPLEX	Time CPLEX (seconds)	SWOGA	LPGA	Time LPGA (seconds)	RG	Time RG (seconds)	RGTS	Time RGTS (seconds)
1	10-2-11-3-30	**104**	~0	104	104	2	104	~0	104	1
2	15-3-14-2-35	**8**	32	8	8	2	8	~0	8	1
3	20-3-21-3-40	*615*	8221	1305	715	15	744	1	616	5
4	32-3-34-4-50	*926*	12684	924	530	34	724	1	423	15
5	30-4-29-5-46	*211*	12845	409	312	42	211	1	211	14
6	32-4-33-5-50	*5*	51362	5	4	44	5	1	5	15
7	30-5-30-5-90	**1**	2	1	1	43	1	1	1	13
8	40-5-38-5-60	*9*	21503	107	27	40	9	2	9	29
9	42-5-43-5-55	*112*	21000	210	111	45	114	2	111	32
10	32-5-35-6-54	**3**	12	3	3	25	3	1	3	20
11	40-5-43-6-56	**4**	45230	200	4	35	10	2	4	34
12	56-5-57-6-62	*6569*	23374	2463	1384	123	1671	3	858	93
13	34-6-32-6-60	*7*	45003	7	6	38	7	1	7	22
14	50-7-60-8-70	*10*	34934	110	12	46	10	3	10	91
15	90-8-89-9-70	*1147*	41514	113	15	57	14	5	13	455
16	93-9-94-9-75	*3500*	38193	458	149	131	48	7	33	487

Table 4.5: Comparison of results of our solution approach with other authors' solution approach for the crossdocking - JIT scheduling problem

As it can be seen in Table 4.5, RG objective values are always better or equal than SWOGA (one of their two metaheuristics) objective values. Besides RG is always faster than SWOGA; therefore, we could say that RG algorithm outperforms SWOGA algorithm in terms of both, objective value and time.

On the other hand, RG objective values are better than LPGA (their metaheuristic with best results) objective values for instances 5, 8, 14, 15 and 16 (most cases are large problem instances, where RG has a very good behavior in terms of objective value and time). Both algorithms found the same objective value for instances 1, 2, 7, and 10. For the other 7 instances (3, 4, 6, 9, 11, 12, and 13) LPGA obtained better objective values than RG. In all cases, RG is faster than LPGA.

According to Li et al. (2004), LPGA offers better objective values than SWOGA. On the other hand, it is known than RG algorithm is embedded in RGTS, so it is expected that RGTS offers equal or better results than just RG. Then, LPGA and RGTS are compared. Both algorithms found the optimal value for the 5 out of the 16 cases when the exact model algorithm also found the optimal value (instances 1, 2, 7, 10, and 11). For the other 11 problem instances, RGTS found better results than LPGA in 8 occasions (instances 3, 4, 5, 8, 12, 14, 15, and 16), LPGA found better results than RGTS in 2 occasions (instances 6 and 13), and they found the same

objective value in 1 occasion (instance 9). For most cases, RGTS takes less computational effort than LPGA except for very large problem instances. This is due to the TS part of RGTS.

As mentioned before, four greedy functions were used in the Reactive strategy of the GRASP algorithm. It was also mentioned that a particular greedy function showed better results than the others for some problem instances, but another greedy function showed better results than the others for some other different problem instances as it can be seen in Table 4.6. Results from each column *gf* represent the average objective value of 5 runs per instance just for the RG algorithm.

ID	Data set (*n*-*m*-*N*-*M*-*th*)	gf_1	gf_2	gf_3	gf_4
1	10-2-11-3-30	104.0	104.0	104.0	104.0
2	15-3-14-2-35	9.2	10.0	112.0	211.8
3	20-3-21-3-40	837.2	657.4	674.6	701.0
4	32-3-34-4-50	972.8	683.4	1172.2	1189.6
5	30-4-29-5-46	221.6	211.0	398.8	218.2
6	32-4-33-5-50	9.0	5.0	30.8	6.0
7	30-5-30-5-90	1.0	1.0	1.4	1.0
8	40-5-38-5-60	78.6	9.0	36.6	12.0
9	42-5-43-5-55	409.4	113.8	307.0	217.8
10	32-5-35-6-54	7.2	3.0	9.2	3.6
11	40-5-43-6-56	112.6	8.2	153.8	106.2
12	56-5-57-6-62	2916.2	1834.0	3121.2	2135.4
13	34-6-32-6-60	14.2	7.2	13.8	8.0
14	50-7-60-8-70	31.0	10.0	79.4	12.2
15	90-8-89-9-70	254.2	14.0	1509.2	20.8
16	93-9-94-9-75	310.6	44.4	434.4	252.2

Table 4.6: Average objective values for the RG algorithm when using the 4 different greedy functions separately

Chapter 5

Solution Approach for the Optimal Workers Allocation for the Crossdocking - Just in Time Scheduling Problem

This chapter deals with the problem described in Chapter 2 when the number of teams of workers in each side of the crossdock is not a given parameter but an unknown variable which has to be determined. This problem is bigger than the one studied in Chapter 4 (actually, it is a super problem of it) and it is known as the optimal workers allocation for the crossdocking - JIT scheduling problem which is, obviously, NP-hard. The representation of the solution of this problem is the same used for the sub-problem shown in Section 4.1.

5.1 Exact Method of Solution.

Our solution of the optimal workers allocation for the crossdocking - JIT scheduling problem is obtained through a computer program that makes use of the ILOG CPLEX 9.1 library. The code that solves this problem using the integer programing model described in Chapter 2 is shown in Appendix 10 (only in electronic format in the software and data CD). The order in which the variables and constraints were defined in our computer program is the same order in which the integer programming model was written.

Unfortunately, our computer resources are limited in speed and memory and the integer programming model contains too many variables and constraints. So, the

results obtained by the exact method when using the ILOG CPLEX 9.1 library (with default parameters) are very poor in terms of objective value and time as it can be seen in Table 5.1 compared to the results obtained by the alternative approach shown in Table 5.2.

ID	Data set (*n*-*N*-*th*)	Number of breakdown machines (*m*)	Number of buildup machines (*M*)	Machines cost	Schedule cost	Total cost	Time CPLEX (seconds)
1	10-11-30	2	1	3000	111	**3111**	2842
2	15-14-35	3	1	4000	144	*4144*	9469
3	20-21-40	4	2	6000	237	*6237*	19926
4	32-34-50	5	4	9000	7	*9007*	29500
5	30-29-46	5	4	9000	326	*9326*	36602
6	32-33-50	5	4	9000	14	*9014*	28687
7	30-30-90	3	3	6000	107	*6107*	38101
8	40-38-60	6	4	10000	921	*10921*	55770
9	42-43-55	7	6	13000	2	*13002*	44632
10	32-35-54	5	4	9000	23	*9023*	39935
11	40-43-56	6	4	10000	10	*10010*	40176
12	56-57-62	10	9	19000	1	*19001*	62781
13	34-32-60	6	3	9000	128	*9128*	46381
14	50-60-70	7	10	17000	9	*17009*	68780

Table 5.1: Experimental results for the complete version of the optimal workers allocation for the crossdocking - JIT scheduling problem when using the integer programming solver ILOG CPLEX 9.1 with default parameters

Italicized results from column *Total Cost* in Table 5.1 indicate that it can not be guaranteed that an optimal solution has been reached because the memory limits were exceeded. In those cases, the best known integer value is reported. Bold results from the same column *Total Cost* show the instances in which the optimal solutions were found. It can be seen in Table 5.1 that only for the instance 1 an optimal solution was found.

The 14 optimal workers allocation for the crossdocking - JIT scheduling problem instances mentioned in Table 5.1 are the same first 14 out of the 16 problem instances provided by Li et al. (2004) which are shown in Appendix 8 (only in electronic format in the software and data CD). The structure of this 14 problem instances is the same shown in Figure 4.8, but without using the values of *m* and *M* because the number of breakdown machines and the number of buildup machines are not parameters any more. The last 2 problem instances provided by Li et al.

(2004) are not used in this problem because they are too big for our computer resources.

5.2 Alternative Method of Solution.

To obtain high quality faster solutions for the optimal workers allocation for the crossdocking - JIT scheduling problem, a very similar approach to the one described in Chapter 4 is proposed: a combination of the metaheuristics Reactive GRASP but now embedded in a Local Search algorithm (RGLS), and Tabu Search (TS). The whole algorithm is abbreviated as RGLSTS. The RGLS procedure is used to find the number of teams of workers hired in each side of the crossdock and to construct initial schedule solutions which, in turn, are used by the TS procedure in order to try to improve those schedule solutions. TS procedure does not change the number of teams of workers obtained by the RGLS procedure. RGLS algorithm is part of the RGLSTS, so RGLSTS will offer equal or better objective values than just RGLS.

Figure 5.1 describes the RGLS algorithm we used to solve the problem. As mentioned before, the Reactive GRASP (RG) is embedded in a Local Search (LS) algorithm. The RG algorithm is used to construct initial schedule solutions depending on the current number of teams of workers hired in each side of the crossdock (*m* and *M*). The LS algorithm is used to select the number of teams of workers to be hired.

```
1  assign an initial value for the number of machines to be rented in each side of the crossdock
2  while the best value has not been found
3      make a search in the neighborhood
4          for each RG iteration
5              for each section of the crossdock
6                  while there are jobs to be scheduled in the section
7                      select the greedy function to be used
8                      adapt the restricted candidate list (RCL) of admissible jobs according to the selected greedy function
9                      select job to be scheduled
10                     adapt the RCL of machines according to the time horizon
11                     select machine to host the selected job
12                     insert the selected job in the selected machine
13                     update the schedule
14                     mark the job as scheduled
15             if the schedule is good enough and different enough
16                 include the schedule in the list of good schedules
17     update the value for the number of machines to be rented in each side of the crossdock
```

Figure 5.1: Pseudo-code for the RGLS algorithm

For the RGLS algorithm shown above, there are some notes to comment:

For line 1, we defined the initial number of machines in each side of the crossdock as the center point of the *n* x *N* dimension mesh given by the number of incoming and outgoing jobs, respectively, as it can be seen in Figure 5.2. In other words, the initial number of machines for the inbound and outbound areas is $\lceil n/2 \rceil$ and $\lceil N/2 \rceil$, respectively.

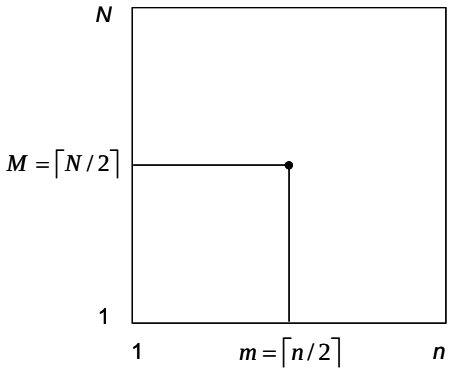


Figure 5.2: Assignment of the initial values for the number of breakdown and buildup machines for the RGLS algorithm

Lines 2, 3 and 17 represent the Local Search (LS) algorithm that embeds the Reactive GRASP (RG) metaheuristic used to construct initial schedule solutions. The LS algorithm creates the path along the point representing the current number of machines (*m*, *M*) moves to reach the best overall value. At each iteration of the LS algorithm, a neighborhood of points around the point that represents the current number of machines (*m*, *M*) is explored as shown in Figure 5.3. The iterations proceed, keeping the best solutions found, until no one of the points around the point (*m*, *M*) shows better results than this point. It can be mention that, during iterations, if the cost of a neighbor point due to the number of machines rented is greater than the best solution found so far, that neighbor point is not explored. This is done to make the LS algorithm faster.

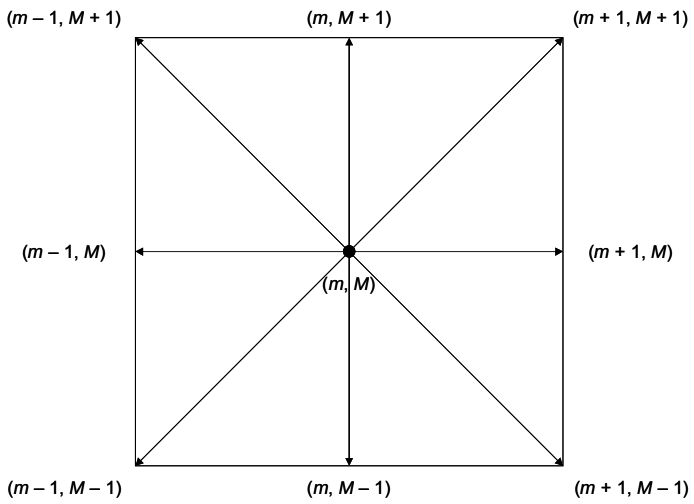


Figure 5.3: Neighborhood of the point (*m*, *M*). This point represents the current number of machines rented in the inbound and outbound areas of the crossdock, respectively

The path from the initial values to the final values for the number of breakdown and buildup machines (*m* and *M*) for the RGLS algorithm for the 14 problem instances already mentioned are included in the results of the RGLSTS algorithm shown in Appendix 11.

Lines 4 - 16 of Figure 5.1 are exactly the same whole lines shown in Figure 4.2.

The TS part of the RGLSTS algorithm is exactly the same TS part of the RGTS algorithm shown in Figure 4.6.

The results obtained by the RGLSTS algorithm are summarized in Table 5.2.

ID	Data set (*n*-*N*-*th*)	Number of breakdown machines (*m*)	Number of buildup machines (*M*)	RGLS Machines cost	RGLS Schedule cost	Time RGLS (seconds)	RGLSTS Schedule cost	Time TS (seconds)	RGLSTS Total cost	Time RGLSTS (seconds)
1	10-11-30	2	1	3000	111	5	111	~0	**3111**	5
2	15-14-35	2	1	3000	1936	8	1148	1	4148	9
3	20-21-40	3	2	5000	922	22	629	5	5629	27
4	32-34-50	3	3	6000	958	63	434	24	6434	87
5	30-29-46	4	3	7000	438	42	338	19	7338	61
6	32-33-50	3	3	6000	256	58	238	21	6238	79
7	30-30-90	2	2	4000	535	43	529	12	4529	55
8	40-38-60	4	3	7000	272	94	63	50	7063	144
9	42-43-55	4	3	7000	1482	145	656	64	7656	209
10	32-35-54	4	3	7000	931	70	51	29	7051	99
11	40-43-56	4	3	7000	540	142	231	53	7231	195
12	56-57-62	6	5	11000	2409	336	298	251	11298	587
13	34-32-60	4	2	6000	1290	59	580	25	6580	84
14	50-60-70	5	4	9000	1359	299	268	83	9268	382

Table 5.2: Experimental results of the optimal workers allocation for the crossdocking - JIT scheduling problem

It can be seen in Table 5.2 that for the instance 1 an optimal solution was found, according to Table 5.1. However, as mentioned in Chapter 3, heuristics and metaheuristics can not guarantee optimality.

As mentioned before, RGLSTS algorithm shows a better behavior in terms of objective value and time than the exact method when using the ILOG CPLEX 9.1 library (with default parameters).

The computer program that solves the optimal workers allocation for the crossdocking - JIT scheduling problem using the RGLSTS algorithm is shown in Appendix 12 (only in electronic format in the software and data CD).

5.3 Model for the Problem (reduced version).

As the results obtained by the exact method and shown in Section 5.1 for the optimal workers allocation for the crossdocking - JIT scheduling problem when using the ILOG CPLEX 9.1 library (with default parameters) for the model described in Chapter 2 are poor in terms of objective value and time compared to the results obtained by the alternative approach named RGLSTS, it has been decided to reuse the model of that chapter with the following changes:

- We used the sum of the teams of workers obtained by the RGLSTS algorithm (that we called $m' + M'$) as an upper bound for the total number of teams of workers that can be hired. This implied the following:
    - We could reduce the number of variables and constraints in the new version of the exact model depending on the values of m' and M' compared to the values of n and N, respectively ($m' \leq n$, and $M' \leq N$). This can be seen in the indexes used in most of the decision variables, in the objective function and in most of the groups of constraints of the new model
    - We created the single constraint (15) which can be seen at the end of the

new model

- We used the solutions obtained by the RGLSTS algorithm as initial solutions in the program in C language that solves the optimal workers allocation for the crossdocking - JIT scheduling problem using the ILOG CPLEX 9.1 library. It was important to consider the following at this point:
    - To set the parameter *CPX_PARAM_MIPSTART* on (by default this parameter is off) in order to accept initial solutions
    - The structures of both types of solutions are different. So, it was necessary to transform the solutions obtained by the RGLSTS algorithm into solutions that could be read by the program that solves the optimal workers allocation for the crossdocking - JIT scheduling problem using the new exact model. The computer program used to do that is shown in Appendix 13 (only in electronic format in the software and data CD)
- To avoid exceeding memory limits, we decided to use the ILOG CPLEX node files storage feature. This required to set the following ILOG CPLEX parameters:
    - *CPX_PARAM_WORKMEM*. We used the default value for this parameter which is 128 MB. This parameter means that once the tree (ILOG CPLEX keeps the information of the problem solution in a tree) storage size exceeds this limit, what happens next is defined by the parameter *CPX_PARAM_NODEFILEIND*. ILOG CPLEX uses node file storage most effectively when the amount of working memory is reasonably large so that it does not have to create node files too frequently. A reasonable amount is to use approximately half the RAM of the computer used to run the problem, but no more than 128 MB
    - *CPX_PARAM_NODEFILEIND*. It indicates the type of storage for the node files. We used the value of 3 for this parameter which means that we wanted to write the node files to disk in a compressed way (by default the value of this parameter is 1 which means that the node files are not written

to disk but they are compressed in RAM)

  - *CPX_PARAM_TRELIM*. This parameter is used to limit the size of the tree kept in the node files so that it does not exceed the amount of disk space chosen. We used a value of 1×10^4 MB (10 GB) for this parameter (the default value of this parameter is 1×10^{75} MB)
  - *CPX_PARAM_WORKDIR*. We used the default value for this parameter which is "." and it means that we wanted to save the node files in the current working directory

- We set the value of the parameter *CPX_PARAM_TILIM* in 90000 seconds which specifies that the limit runtime for a problem instance is 25 hours
- We decided to use the ILOG CPLEX strong branching feature by setting the value of *CPX_VARSEL_STRONG* for the parameter *CPX_PARAM_VARSEL*. With this feature, the program invests considerable effort in analyzing potential branches in the hope of drastically reducing the number of nodes that will be explored in the tree. This is recommended when working with very big problems like the one presented in this chapter

It can be mentioned that, for the new model, the parameters m and M (these are parameters again in the new model) are equal to $(m' + M') - 1$, where $(m' + M')$ is the sum of the teams of workers hired obtained by the RGLSTS algorithm. This value of $(m' + M')$ is used as the maximum number of teams of workers that can be hired in total for both sides of the crossdock. We used the equality $m = M = (m' + M') - 1$ because we need at least one team of workers hired in each side of the crossdock. This modification with respect to the model described in Chapter 2 allowed us to drastically reduce the number of variables and constraints in the new model. Even though, the problem is still big.

The formulation of the optimal workers allocation for the crossdocking - JIT scheduling problem (reduced version) using the machine scheduling notation is the following:

Decision variables

y_{ik} = 1 if incoming container i is processed on breakdown machine k and 0 otherwise, for i = 1, …, n, k = 1, …, m

Y_{jk} = 1 if outgoing container j is processed on buildup machine k and 0 otherwise, for j = 1, …, N, k = 1, …, M

I_{ijk} = 1 if incoming containers i and j are both processed by breakdown machine k and i precedes (not necessarily immediately) j, and 0 otherwise, for i, j = 1, …, n, $i \neq j$, k = 1, …, m

J_{ijk} = 1 if outgoing containers i and j are both processed by buildup machine k and i precedes (not necessarily immediately) j, and 0 otherwise, for i, j = 1, …, N, $i \neq j$, k = 1, …, M

c_i - completion time of incoming container i, i = 1, …, n

C_j - completion time of outgoing container j, j = 1, …, N

m_k = 1 if breakdown machine k is used and 0 otherwise, for k = 1, …, m

M_k = 1 if buildup machine k is used and 0 otherwise, for k = 1, …, M

Variables y_{ik} and Y_{jk} represent assignment variables, I_{ijk} and J_{ijk} represent sequencing variables, c_i and C_j represent scheduling variables, and m_k and M_k represent machines variables. The values assigned to the assignment variables, scheduling variables and machines variables represent a specific solution for the problem.

State variables: their values depend on the current built schedule

e_i - earliness of incoming container i, i = 1, …, n

E_j - earliness of outgoing container j, j = 1, …, N

t_i - tardiness of incoming container i, i = 1, …, n

T_j - tardiness of outgoing container j, j = 1, …, N

Objective function

$$\text{Minimize } \sum_{i=1}^{n} \left(\alpha e_i + \beta t_i\right) + \sum_{j=1}^{N} \left(\alpha E_j + \beta T_j\right) + h\sum_{k=1}^{m} m_k + h\sum_{k=1}^{M} M_k$$

Constraints

(1) $\sum_{k=1}^{m} y_{ik} = 1$, $i = 1, \ldots, n$ (breakdown area)

(2) $\sum_{k=1}^{M} Y_{jk} = 1$, $j = 1, \ldots, N$ (buildup area)

(3) $y_{ik} + y_{jk} - (I_{ijk} + I_{jik}) \leq 1$ (breakdown area)

(4) $2(I_{ijk} + I_{jik}) - y_{ik} - y_{jk} \leq 0$ (breakdown area)

$i, j = 1, \ldots, n, i < j, k = 1, \ldots, m$

(5) $Y_{ik} + Y_{jk} - (J_{ijk} + J_{jik}) \leq 1$ (buildup area)

(6) $2(J_{ijk} + J_{jik}) - Y_{ik} - Y_{jk} \leq 0$ (buildup area)

$i, j = 1, \ldots, N, i < j, k = 1, \ldots, M$

(7) $c_i \leq (c_j - p_j) + G(1 - I_{ijk})$, $i, j = 1, \ldots, n, i \neq j, k = 1, \ldots, m$ (breakdown area)

(8) $C_i \leq (C_j - P_j) + G(1 - J_{ijk})$, $i, j = 1, \ldots, N, i \neq j, k = 1, \ldots, M$ (buildup area)

(9) $c_i - r_i \geq p_i$, $i = 1, \ldots, n$ (breakdown area)

(10) $C_j - c_i \geq P_j$, $j = 1, \ldots, N$, i = first predecessor of outgoing container j, …, last predecessor of outgoing container j (buildup area)

(11) $c_i - d_i = t_i - e_i$, $i = 1, \ldots, n$ (breakdown area)

(12) $C_j - D_j = T_j - E_j$, $j = 1, \ldots, N$ (buildup area)

(13) $m_k - y_{ik} \geq 0$, $i = 1, \ldots, n, k = 1, \ldots, m$ (breakdown area)

(14) $M_k - Y_{jk} \geq 0$, $j = 1, \ldots, N$, $k = 1, \ldots, M$ (buildup area)

(15) $\sum_{k=1}^{m} m_k + \sum_{k=1}^{M} M_k \leq (m' + M')$ (both areas)

$y_{ik} \in \{0, 1\}$, $i = 1, \ldots, n$, $k = 1, \ldots, m$

$Y_{jk} \in \{0, 1\}$, $j = 1, \ldots, N$, $k = 1, \ldots, M$

$I_{ijk} \in \{0, 1\}$, $i, j = 1, \ldots, n$, $i \neq j$, $k = 1, \ldots, m$

$J_{ijk} \in \{0, 1\}$, $i, j = 1, \ldots, N$, $i \neq j$, $k = 1, \ldots, M$

$m_k \in \{0, 1\}$, $k = 1, \ldots, m$

$M_k \in \{0, 1\}$, $k = 1, \ldots, M$

$c_i, e_i, t_i \in Z^+$ (nonnegative integer numbers), $i = 1, \ldots, n$

$C_j, E_j, T_j \in Z^+$ (nonnegative integer numbers), $j = 1, \ldots, N$

This model is very similar to the one shown in Chapter 2, but with the changes previously mentioned in this section. Again, we used G = 100,000 and h = 1,000 for the experiments. The new single constraint (15) is used to reduce the solution search space.

This model contains:

- A total of variables (including state variables) equal to $(n^2+1)m + (N^2+1)M + 3(n+N)$, from which:
  - $(n^2+1)m + (N^2+1)M$ are binary variables and $3(n+N)$ are integer variables
  - $(n^2+1)m + (N^2+1)M + n + N$ are decision variables and $2(n+N)$ are state variables
- A total of constraints equal to $3n + 2N + 2(n^2 - n)m + 2(N^2 - N)M + nm + NM + \sum_{j=1}^{N} K_j$, where $N \leq \sum_{j=1}^{N} K_j \leq nN$

- A total of non-empty cells in the technological coefficients matrix equal to

$$4n + 3N + 3nm + 3NM + 7(n^2 - n)m + 7(N^2 - N)M + 2\sum_{j=1}^{N} K_j$$

The computer program that solves the optimal workers allocation for the crossdocking - JIT scheduling problem using the previous exact model (reduced version) is shown in Appendix 14 (only in electronic format in the software and data CD).

The results obtained by the reduced version of the problem are shown in Table 5.3. It can be seen that few problem instances (the smallest ones) improved their objective values compared to the solutions provided by the RGLSTS algorithm as initial solutions for the exact method when using the ILOG CPLEX 9.1 library with the specific parameters mentioned above. However, the optimal value can not be guaranteed for most of the problems instances when running the reduced version of the optimal workers allocation for the crossdocking - JIT scheduling problem for up to 25 hours (an arbitray but reasonable timeout).

ID	Data set (*n*-*m*-*N*-*M*-*th*)	Number of breakdown machines (*m*)	Number of buildup machines (*M*)	Machines cost	Schedule cost	Total cost	Time CPLEX (seconds)
1	10-2-11-2-30	2	1	3000	111	**3111**	3
2	15-2-14-2-35	2	1	3000	651	**3651**	1129
3	20-4-21-4-40	3	2	5000	566	*5566*	90000
4	32-5-34-5-50	3	3	6000	434	*6434*	90000
5	30-6-29-6-46	4	3	7000	338	*7338*	90000
6	32-5-33-5-50	3	3	6000	238	*6238*	90000
7	30-3-30-3-90	2	2	4000	529	*4529*	90000
8	40-6-38-6-60	4	3	7000	63	*7063*	90000
9	42-6-43-6-55	4	3	7000	656	*7656*	90000
10	32-6-35-6-54	4	3	7000	51	*7051*	90000
11	40-6-43-6-56	4	3	7000	231	*7231*	90000
12	56-10-57-10-62	6	5	11000	298	*11298*	90000
13	34-5-32-5-60	4	2	6000	580	*6580*	90000
14	50-8-60-8-70	5	4	9000	268	*9268*	90000

Table 5.3: Experimental results for the reduced version of the optimal workers allocation for the crossdocking - JIT scheduling problem when using the integer programming solver ILOG CPLEX 9.1 with specific parameters

Italicized results from column *Total Cost* in Table 5.3 indicate that it can not be guaranteed that an optimal solution has been found because the time limit was

reached. In those cases, the best known integer value is reported. Bold results from the same column *Total Cost* show the instances in which the optimal solutions were found. It can be seen in Table 5.3 that only for the instances 1 and 2 an optimal solution was found and it was improved the objective value for instance 3 with respect to its initial solution provided by the RGLSTS algorithm.

The 14 optimal workers allocation for the crossdocking - JIT scheduling problem instances mentioned in Table 5.3 refer to the first 14 out of the 16 problem instances provided by Li et al. (2004) which are shown in Appendix 8 (only in electronic format in the software and data CD). However, now each problem instance requires two input files: one contains the data provided by Li et al. (2004) -slightly modified- and the other contains the initial solution obtained from the RGLSTS algorithm. The structure of the input files related to the data for these 14 problem instances is the same shown in Figure 4.8, but adding the value of *m'* + *M'* at the end of the first line. The 28 input files corresponding to these 14 problem instances are shown in Appendix 15 (only in electronic format in the software and data CD).

Chapter 6

Conclusions

The crossdocking - JIT scheduling problem is a sub-problem of the optimal workers allocation for the crossdocking - JIT scheduling problem. The number of teams of workers in each side of the crossdock is a fixed and known parameter in the sub-problem.

The crossdocking - JIT scheduling problem is a relevant NP-hard problem that has not received much attention by researchers. The work presented in this thesis has an important academic contribution because it involves the development of a metaheuristic algorithm not previously applied to that problem.

To obtain optimal or near optimal solutions for the crossdocking - JIT scheduling problem represents an improvement in the movement or distribution of the products, reducing in this way, transportation costs and inventory costs.

Our solution approach to solve the NP-hard crossdocking - JIT scheduling problem is based on a combination of two metaheuristics, Reactive GRASP (RG) and Tabu Search (TS), abbreviated as RGTS. It is efficient and it offers good results with modest computational effort. It represents an excellent alternative to the approach studied in Li et al. (2004) for the same problem. The combination of these two metaheuristics, GRASP and Tabu Search, has been applied to other problems with also good results.

Experiments showed that RG offers good solutions in very short times, but it can be improved in many cases by TS which is slower.

On the other hand, the optimal workers allocation for a crossdocking - JIT scheduling problem is, obviously, harder to solve than its sub-problem, but it is more realistic

and interesting, as well. As this bigger problem is more complex than its sub-problem, two versions of it were analyzed: a complete version and a reduced version. The reduced version obtains better results faster than the complete version; however, it is necessary to know the results from the alternative algorithm first in order to be able to use it.

The alternative algorithm used to solve the optimal workers allocation for a crossdocking - JIT scheduling problem is very similar to the approach used for its sub-problem: a combination of the metaheuristics Reactive GRASP but now embedded in a Local Search algorithm (RGLS), and Tabu Search (TS). The whole algorithm is abbreviated as RGLSTS.

Again, experiments showed that RGLS offers good solutions which can be improved in many cases by TS. These solutions are obtained with modest computational effort compared to the exact model.

6.1 Future Work.

Due to time limitations, only one fixed cost for teams of workers was used to make tests for the optimal workers allocation for the crossdocking - JIT scheduling problem (the cost per working day of each team of workers was the same for both sides of the crossdock). Some other tests might be done using different fixed costs for these teams in order to make an analysis of the impact of the workforce costs in the schedule of the jobs.

Some other extensions that can be applied in the context of the present work are the following:

- To have teams with different speeds
- To have stochastic sick days for some members of the teams (equivalent to have stochastic failures for the machines)

- To have workers with different salaries (equivalent to have machines with different costs)
- To have stochastic arrival times and/or processing times for the jobs
- To have job-dependent early and tardy costs

This research work can also be extended to some other relevant problems related to the transportation industry that make use of crossdocking.

On the other hand, the codes of the algorithms developed in this project can be used for further researches that keep a relation with GRASP and Tabu Search methods due to their clarity in the writing, detailed design, and scalability.

Appendix 1

Linear and Integer Programming

The Optimization area is part of the Mathematical Programming that deals with the designed methods to obtain the best result (maximum or minimum) of an objective function through the appropriate selection of the decision variables, in a limited acting environment, and subject to operational and/or design constraints.

A common formulation for an optimization problem can be seen as follows:

$$\min/\max \; \boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x})$$

subject to:

$$\boldsymbol{h_i}(\boldsymbol{x}) \leq \boldsymbol{b_i},\ i = 1, \ldots, m_1$$

$$\boldsymbol{h_i}(\boldsymbol{x}) \geq \boldsymbol{b_i},\ i = m_1+1, \ldots, m_1+m_2$$

$$\boldsymbol{h_i}(\boldsymbol{x}) = \boldsymbol{b_i},\ i = m_1+m_2+1, \ldots, m_1+m_2+m_3$$

$$\boldsymbol{l_j} \leq \boldsymbol{x_j} \leq \boldsymbol{u_j},\ j = 1, \ldots, n$$

In this formulation:

- $z = f(x)$ is the objective function
- $h_i(x) \sim b_i,\ i = 1, \ldots, m = m_1+m_2+m_3$ are the functional constraints (the sign "~" can be "≤", "≥" or "=")
- $l_j \leq x_j \leq u_j,\ j = 1, \ldots, n$ are the state constraints
- $b_i,\ i = 1, \ldots, m = m_1+m_2+m_3$ are known parameters
- $x_j,\ j = 1, \ldots, n$ are the unknown decision variables, so x is a vector of size n
- l_j and u_j are the known lower and upper bounds of the decision variable j, respectively

If the objective function and the functional constraints are linear equations, then the problem is a *Linear Programming* (LP) problem.

If all of the decision variables are non-negative and there are no inequality constraints in an LP problem it is said that the problem is in its standard form. In other words, the problem is a standard LP problem.

It is possible to represent a standard LP problem using a matrix formulation as follows:

$$\min/\max \boldsymbol{z} = \boldsymbol{c}^t\boldsymbol{x}$$

$$\text{subject to } \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

where:

- $\boldsymbol{A}$ is an m x n given matrix known as the *technological coefficients matrix* (m is the total number of constraints and n is the total number of variables)
- $\boldsymbol{b}$ is a given vector of size m known as the *capacities vector* (some authors call it the *right hand side vector* or simply the *rhs vector*)
- $\boldsymbol{c}$ is a given vector of size n known as the *costs vector*
- $\boldsymbol{x}$ is the vector to find of size n known as the *solution vector*, $\boldsymbol{x} \in \mathbf{R}^+$ (nonnegative real numbers)

It is always possible to represent any non-standard LP problem in its standard form by adding slack variables to it and using some other transformations.

A deeper explanation about Linear Programming can be found in Murty (1983).

In the same context of optimization problems, there exists an interesting group of them called *combinatorial optimization problems*. In this kind of problems, some or all of the decision variables are integers and, generally, the solutions space is formed by subsets of integer numbers [Díaz et al. (1996)]. In other words, integer and combinatorial optimization deals with problems of maximizing or minimizing a function of many variables subject to integrality restrictions on some or all of the decision variables [Nemhauser & Wolsey (1999)]. If some decision variables of the

problem are continuous and some of them are integer then it is said that the problem is a *Mixed-Integer Problem* (MIP). A special case of the MIP is the *Pure Integer Problem* (PIP), where there are no continuous decision variables.

A representation of an MIP problem using a matrix formulation would be as follows:

$$\min/\max \boldsymbol{z = c^t x + h^t y}$$

$$\text{subject to } \boldsymbol{Ax + Gy = b}$$

where:

- $\boldsymbol{A}$ is an m x n given matrix
- $\boldsymbol{G}$ is an m x p given matrix
- $\boldsymbol{b}$ is a given vector of size m
- $\boldsymbol{c}$ is a given vector of size n
- $\boldsymbol{h}$ is a given vector of size p
- $\boldsymbol{x}$ is a vector to find of size n, $\boldsymbol{x} \in \mathbf{Z^+}$ (nonnegative integer numbers)
- $\boldsymbol{y}$ is a vector to find of size p, $\boldsymbol{y} \in \mathbf{R^+}$ (nonnegative real numbers)

A representation of a PIP problem using a matrix formulation would be as follows:

$$\min/\max \boldsymbol{z = c^t x}$$

$$\text{subject to } \boldsymbol{Ax = b}$$

where:

- $\boldsymbol{A}$ is an m x n given matrix
- $\boldsymbol{b}$ is a given vector of size m
- $\boldsymbol{c}$ is a given vector of size n
- $\boldsymbol{x}$ is a vector to find of size n, $\boldsymbol{x} \in \mathbf{Z^+}$ (nonnegative integer numbers)

Sometimes, the integer variables are used to represent logical or belonging relationships and therefore are constrained equal to 0 or 1. In that case, the 0-1 MIP

(or 0-1 PIP) is obtained and $\boldsymbol{x} \in \mathbf{Z}^{+}$ is replaced by $\boldsymbol{x} \in \mathbf{B}$ (binary numbers).

Most of the combinatorial optimization problems are NP-complete, which means that their complexity grows exponentially according to the problem instance. Then, it is possible to say that there is not an exact algorithm that solves them in a “reasonable” amount of time if the problem instance is “big”.

There are many important real applications that can be stated as combinatorial optimization problems. Some of the most famous problems of this kind in the literature are: the Traveling Salesman Problem or TSP [Lawler et al. (1985)], the Knapsack Problem and the Bin Packing Problem [Coffman (1976), Baase (1991), Hochbaum (1997), Horowitz et al. (1998), Cormen et al. (2001)], among others.

Appendix 2

Integer Programming Model for the Crossdocking - JIT Scheduling Problem Instance shown in Table 4.1

```
min
+ e1 + e2 + e3 + e4 + aE1 + aE2 + aE3 + 100 t1 + 100 t2 + 100 t3 + 100 t4 + 100 aT1 + 100 aT2 +
100 aT3

Subject to
+ y11 + y12 = 1
+ y21 + y22 = 1
+ y31 + y32 = 1
+ y41 + y42 = 1
+ aY11 + aY12 = 1
+ aY21 + aY22 = 1
+ aY31 + aY32 = 1
+ y11 + y21 - I121 - I211 <= 1
+ y12 + y22 - I122 - I212 <= 1
+ y11 + y31 - I131 - I311 <= 1
+ y12 + y32 - I132 - I312 <= 1
+ y11 + y41 - I141 - I411 <= 1
+ y12 + y42 - I142 - I412 <= 1
+ y21 + y31 - I231 - I321 <= 1
+ y22 + y32 - I232 - I322 <= 1
+ y21 + y41 - I241 - I421 <= 1
+ y22 + y42 - I242 - I422 <= 1
+ y31 + y41 - I341 - I431 <= 1
+ y32 + y42 - I342 - I432 <= 1
- y11 - y21 + 2 I121 + 2 I211 <= 0
- y12 - y22 + 2 I122 + 2 I212 <= 0
- y11 - y31 + 2 I131 + 2 I311 <= 0
- y12 - y32 + 2 I132 + 2 I312 <= 0
- y11 - y41 + 2 I141 + 2 I411 <= 0
- y12 - y42 + 2 I142 + 2 I412 <= 0
- y21 - y31 + 2 I231 + 2 I321 <= 0
- y22 - y32 + 2 I232 + 2 I322 <= 0
- y21 - y41 + 2 I241 + 2 I421 <= 0
- y22 - y42 + 2 I242 + 2 I422 <= 0
- y31 - y41 + 2 I341 + 2 I431 <= 0
- y32 - y42 + 2 I342 + 2 I432 <= 0
+ aY11 + aY21 - J121 - J211 <= 1
+ aY12 + aY22 - J122 - J212 <= 1
+ aY11 + aY31 - J131 - J311 <= 1
+ aY12 + aY32 - J132 - J312 <= 1
+ aY21 + aY31 - J231 - J321 <= 1
+ aY22 + aY32 - J232 - J322 <= 1
- aY11 - aY21 + 2 J121 + 2 J211 <= 0
- aY12 - aY22 + 2 J122 + 2 J212 <= 0
- aY11 - aY31 + 2 J131 + 2 J311 <= 0
- aY12 - aY32 + 2 J132 + 2 J312 <= 0
- aY21 - aY31 + 2 J231 + 2 J321 <= 0
- aY22 - aY32 + 2 J232 + 2 J322 <= 0
+ 10000 I121 + c1 - c2 <= 9988
+ 10000 I122 + c1 - c2 <= 9988
+ 10000 I131 + c1 - c3 <= 9994
+ 10000 I132 + c1 - c3 <= 9994
+ 10000 I141 + c1 - c4 <= 9999
+ 10000 I142 + c1 - c4 <= 9999
+ 10000 I211 - c1 + c2 <= 9994
+ 10000 I212 - c1 + c2 <= 9994
```

```
+ 10000 I231 + c2 - c3 <= 9994
+ 10000 I232 + c2 - c3 <= 9994
+ 10000 I241 + c2 - c4 <= 9999
+ 10000 I242 + c2 - c4 <= 9999
+ 10000 I311 - c1 + c3 <= 9994
+ 10000 I312 - c1 + c3 <= 9994
+ 10000 I321 - c2 + c3 <= 9988
+ 10000 I322 - c2 + c3 <= 9988
+ 10000 I341 + c3 - c4 <= 9999
+ 10000 I342 + c3 - c4 <= 9999
+ 10000 I411 - c1 + c4 <= 9994
+ 10000 I412 - c1 + c4 <= 9994
+ 10000 I421 - c2 + c4 <= 9988
+ 10000 I422 - c2 + c4 <= 9988
+ 10000 I431 - c3 + c4 <= 9994
+ 10000 I432 - c3 + c4 <= 9994
+ 10000 J121 + aC1 - aC2 <= 9995
+ 10000 J122 + aC1 - aC2 <= 9995
+ 10000 J131 + aC1 - aC3 <= 9981
+ 10000 J132 + aC1 - aC3 <= 9981
+ 10000 J211 - aC1 + aC2 <= 9998
+ 10000 J212 - aC1 + aC2 <= 9998
+ 10000 J231 + aC2 - aC3 <= 9981
+ 10000 J232 + aC2 - aC3 <= 9981
+ 10000 J311 - aC1 + aC3 <= 9998
+ 10000 J312 - aC1 + aC3 <= 9998
+ 10000 J321 - aC2 + aC3 <= 9995
+ 10000 J322 - aC2 + aC3 <= 9995
+ c1 >= 11
+ c2 >= 18
+ c3 >= 14
+ c4 >= 8
- c2 + aC1 >= 2
- c3 + aC1 >= 2
- c4 + aC1 >= 2
- c1 + aC2 >= 5
- c2 + aC2 >= 5
- c4 + aC2 >= 5
- c1 + aC3 >= 19
- c2 + aC3 >= 19
- c3 + aC3 >= 19
+ c1 + e1 - t1 = 14
+ c2 + e2 - t2 = 19
+ c3 + e3 - t3 = 16
+ c4 + e4 - t4 = 11
+ aC1 + aE1 - aT1 = 22
+ aC2 + aE2 - aT2 = 28
+ aC3 + aE3 - aT3 = 39

INT y11
INT y12
INT y21
INT y22
INT y31
INT y32
INT y41
INT y42
INT aY11
INT aY12
INT aY21
INT aY22
INT aY31
INT aY32
INT I121
INT I122
INT I131
INT I132
INT I141
INT I142
INT I211
INT I212
INT I231
INT I232
INT I241
INT I242
```

```
INT I311
INT I312
INT I321
INT I322
INT I341
INT I342
INT I411
INT I412
INT I421
INT I422
INT I431
INT I432
INT J121
INT J122
INT J131
INT J132
INT J211
INT J212
INT J231
INT J232
INT J311
INT J312
INT J321
INT J322
```

This MIP uses the letter "*a*" as a differentiator between variables related to the inbound area (lower-case letters) and variables related to the outbound area (upper-case letters), except for the sequencing variables for the inbound and outbound area, I_{ijk} and J_{ijk}, respectively, which remained without changes.

Appendix 4

Output of the RG Algorithm for the following Crossdocking - JIT Scheduling Problem Instance

n **=**	20
m **=**	3
N **=**	21
M **=**	3
α **=**	1
ß **=**	100
G **=**	10000

	r	***p***	***d***
1	13	5	19
2	16	2	21
3	23	2	26
4	11	4	15
5	27	2	29
6	10	4	17
7	36	3	41
8	32	2	37
9	10	2	12
10	37	3	43
11	7	4	13
12	10	3	15
13	30	4	35
14	30	6	36
15	15	2	19
16	17	4	24
17	9	2	12
18	31	6	40
19	12	6	20
20	15	4	21

	P	***D***
1	5	50
2	2	53
3	2	47
4	2	50
5	4	28
6	3	21
7	5	78
8	5	65
9	3	24
10	4	53
11	5	45
12	2	58
13	4	58
14	5	60
15	3	52
16	4	56
17	4	68
18	4	72
19	4	71
20	1	36
21	1	16

S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0
2	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	1	0	1	1	0
3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
4	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0
6	1	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0
7	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0
8	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0
10	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0
13	1	0	0	1	0	0	0	1	0	0	0	1	1	0	0	1	1	0	0	0	0
14	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0
16	1	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0
17	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
19	0	1	1	0	1	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0
20	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	K_1=5	K_2=2	K_3=2	K_4=2	K_5=4	K_6=3	K_7=5	K_8=5	K_9=3	K_{10}=4	K_{11}=5	K_{12}=2	K_{13}=4	K_{14}=5	K_{15}=3	K_{16}=4	K_{17}=4	K_{18}=4	K_{19}=4	K_{20}=1	K_{21}=1

Solution 1

Inbound area

***i* (s_i, c_i, e_i, t_i)**

Machine 1: 11 (7, 11, 2, 0) 4 (11, 15, 0, 0) 15 (15, 17, 2, 0) 2 (17, 19, 2, 0) 16 (19, 23, 1, 0) 13 (30, 34, 1, 0) 18 (34, 40, 0, 0)
Machine 2: 17 (9, 11, 1, 0) 12 (11, 14, 1, 0) 1 (14, 19, 0, 0) 20 (19, 23, 0, 2) 5 (27, 29, 0, 0) 8 (35, 37, 0, 0) 10 (37, 40, 3, 0)
Machine 3: 9 (10, 12, 0, 0) 6 (12, 16, 1, 0) 19 (16, 22, 0, 2) 3 (24, 26, 0, 0) 14 (30, 36, 0, 0) 7 (38, 41, 0, 0)

Outbound area

***j* (S_j, C_j, E_j, T_j)**

Machine 1: 5 (24, 28, 0, 0) 11 (40, 45, 0, 0) 4 (48, 50, 0, 0) 2 (51, 53, 0, 0) 12 (56, 58, 0, 0) 8 (60, 65, 0, 0) 18 (68, 72, 0, 0)
Machine 2: 6 (19, 22, 0, 1) 20 (35, 36, 0, 0) 15 (49, 52, 0, 0) 13 (54, 58, 0, 0) 17 (64, 68, 0, 0) 7 (73, 78, 0, 0)
Machine 3: 21 (15, 16, 0, 0) 9 (23, 26, 0, 2) 3 (40, 42, 5, 0) 1 (42, 47, 3, 0) 10 (47, 51, 2, 0) 16 (51, 55, 1, 0) 14 (55, 60, 0, 0) 19 (67, 71, 0, 0)

Total penalty for crossdock: 725

Solution 2

Inbound area

Machine 1: 11 (7, 11, 2, 0) 12 (11, 14, 1, 0) 1 (14, 19, 0, 0) 20 (19, 23, 0, 2) 3 (24, 26, 0, 0) 13 (31, 35, 0, 0) 7 (38, 41, 0, 0)
Machine 2: 9 (10, 12, 0, 0) 6 (12, 16, 1, 0) 2 (16, 18, 3, 0) 16 (18, 22, 2, 0) 14 (30, 36, 0, 0) 10 (37, 40, 3, 0)
Machine 3: 17 (9, 11, 1, 0) 4 (11, 15, 0, 0) 15 (15, 17, 2, 0) 19 (17, 23, 0, 3) 5 (27, 29, 0, 0) 8 (32, 34, 3, 0) 18 (34, 40, 0, 0)

Outbound area

Machine 1: 6 (18, 21, 0, 0) 9 (22, 25, 0, 1) 5 (25, 29, 0, 1) 1 (41, 46, 4, 0) 10 (46, 50, 3, 0) 2 (50, 52, 1, 0) 16 (52, 56, 0, 0) 8 (60, 65, 0, 0) 19 (67, 71, 0, 0)
Machine 2: 20 (35, 36, 0, 0) 11 (40, 45, 0, 0) 17 (64, 68, 0, 0) 7 (73, 78, 0, 0)
Machine 3: 21 (15, 16, 0, 0) 3 (42, 44, 3, 0) 4 (44, 46, 4, 0) 15 (46, 49, 3, 0) 13 (49, 53, 5, 0) 12 (53, 55, 3, 0) 14 (55, 60, 0, 0) 18 (68, 72, 0, 0)

Total penalty for crossdock: 744

Solution 3

Inbound area

Machine 1: 9 (10, 12, 0, 0) 6 (12, 16, 1, 0) 15 (16, 18, 1, 0) 2 (18, 20, 1, 0) 20 (20, 24, 0, 3) 5 (27, 29, 0, 0) 18 (34, 40, 0, 0)
Machine 2: 11 (7, 11, 2, 0) 12 (11, 14, 1, 0) 1 (14, 19, 0, 0) 16 (19, 23, 1, 0) 14 (30, 36, 0, 0) 7 (38, 41, 0, 0)
Machine 3: 17 (9, 11, 1, 0) 4 (11, 15, 0, 0) 19 (15, 21, 0, 1) 3 (24, 26, 0, 0) 13 (31, 35, 0, 0) 8 (35, 37, 0, 0) 10 (37, 40, 3, 0)

Outbound area

Machine 1: 9 (23, 26, 0, 2) 11 (40, 45, 0, 0) 10 (49, 53, 0, 0) 13 (54, 58, 0, 0) 8 (60, 65, 0, 0) 18 (68, 72, 0, 0)
Machine 2: 6 (20, 23, 0, 2) 20 (35, 36, 0, 0) 1 (43, 48, 2, 0) 4 (48, 50, 0, 0) 2 (51, 53, 0, 0) 14 (55, 60, 0, 0) 19 (67, 71, 0, 0)
Machine 3: 21 (15, 16, 0, 0) 5 (24, 28, 0, 0) 3 (45, 47, 0, 0) 15 (49, 52, 0, 0) 16 (52, 56, 0, 0) 12 (56, 58, 0, 0) 17 (64, 68, 0, 0) 7 (73, 78, 0, 0)

Total penalty for crossdock: 813

Solution 4

Inbound area

Machine 1: 11 (7, 11, 2, 0) 12 (11, 14, 1, 0) 1 (14, 19, 0, 0) 16 (19, 23, 1, 0) 14 (30, 36, 0, 0) 7 (38, 41, 0, 0)
Machine 2: 17 (9, 11, 1, 0) 4 (11, 15, 0, 0) 19 (15, 21, 0, 1) 3 (24, 26, 0, 0) 13 (31, 35, 0, 0) 8 (35, 37, 0, 0) 10 (37, 40, 3, 0)
Machine 3: 9 (10, 12, 0, 0) 6 (12, 16, 1, 0) 15 (16, 18, 1, 0) 2 (18, 20, 1, 0) 20 (20, 24, 0, 3) 5 (27, 29, 0, 0) 18 (34, 40, 0, 0)

Outbound area

Machine 1: 21 (15, 16, 0, 0) 5 (24, 28, 0, 0) 1 (43, 48, 2, 0) 4 (48, 50, 0, 0) 16 (52, 56, 0, 0) 12 (56, 58, 0, 0) 8 (60, 65, 0, 0) 18 (68, 72, 0, 0)
Machine 2: 6 (20, 23, 0, 2) 20 (35, 36, 0, 0) 3 (45, 47, 0, 0) 10 (49, 53, 0, 0) 13 (54, 58, 0, 0) 17 (64, 68, 0, 0) 7 (73, 78, 0, 0)
Machine 3: 9 (23, 26, 0, 2) 11 (40, 45, 0, 0) 15 (48, 51, 1, 0) 2 (51, 53, 0, 0) 14 (55, 60, 0, 0) 19 (67, 71, 0, 0)

Total penalty for crossdock: 814

Solution 5

Inbound area

Machine 1: 17 (9, 11, 1, 0) 4 (11, 15, 0, 0) 1 (15, 20, 0, 1) 20 (20, 24, 0, 3) 5 (27, 29, 0, 0) 18 (34, 40, 0, 0)
Machine 2: 11 (7, 11, 2, 0) 6 (11, 15, 2, 0) 15 (15, 17, 2, 0) 2 (17, 19, 2, 0) 16 (19, 23, 1, 0) 13 (31, 35, 0, 0) 8 (35, 37, 0, 0) 10 (37, 40, 3, 0)
Machine 3: 9 (10, 12, 0, 0) 12 (12, 15, 0, 0) 19 (15, 21, 0, 1) 3 (24, 26, 0, 0) 14 (30, 36, 0, 0) 7 (38, 41, 0, 0)

Outbound area

Machine 1: 6 (19, 22, 0, 1) 20 (35, 36, 0, 0) 1 (45, 50, 0, 0) 16 (52, 56, 0, 0) 12 (56, 58, 0, 0) 8 (60, 65, 0, 0) 18 (68, 72, 0, 0)
Machine 2: 21 (15, 16, 0, 0) 9 (23, 26, 0, 2) 11 (40, 45, 0, 0) 15 (48, 51, 1, 0) 2 (51, 53, 0, 0) 14 (55, 60, 0, 0) 19 (67, 71, 0, 0)
Machine 3: 5 (24, 28, 0, 0) 3 (45, 47, 0, 0) 4 (47, 49, 1, 0) 10 (49, 53, 0, 0) 13 (54, 58, 0, 0) 17 (64, 68, 0, 0) 7 (73, 78, 0, 0)

Total penalty for crossdock: 815

Appendix 5

Output of the RGTS Algorithm for the Crossdocking - JIT Scheduling Problem Instance shown in Appendix 4

Solution 1

Inbound area

i (s_i, c_i, e_i, t_i)

Machine 1: 11 (7, 11, 2, 0) 4 (11, 15, 0, 0) 15 (15, 17, 2, 0) 2 (17, 19, 2, 0) 16 (19, 23, 1, 0) 13 (30, 34, 1, 0) 18 (34, 40, 0, 0)
Machine 2: 17 (9, 11, 1, 0) 12 (11, 14, 1, 0) 1 (14, 19, 0, 0) 20 (19, 23, 0, 2) 5 (27, 29, 0, 0) 8 (35, 37, 0, 0) 10 (37, 40, 3, 0)
Machine 3: 9 (10, 12, 0, 0) 6 (12, 16, 1, 0) 19 (16, 22, 0, 2) 3 (24, 26, 0, 0) 14 (30, 36, 0, 0) 7 (38, 41, 0, 0)

Outbound area

j (S_j, C_j, E_j, T_j)

Machine 1: 5 (24, 28, 0, 0) 11 (40, 45, 0, 0) 4 (47, 49, 1, 0) 15 (49, 52, 0, 0) 16 (52, 56, 0, 0) 12 (56, 58, 0, 0) 18 (68, 72, 0, 0)
Machine 2: 6 (19, 22, 0, 1) 20 (35, 36, 0, 0) 1 (45, 50, 0, 0) 2 (51, 53, 0, 0) 13 (54, 58, 0, 0) 17 (64, 68, 0, 0) 7 (73, 78, 0, 0)
Machine 3: 21 (15, 16, 0, 0) 9 (23, 26, 0, 2) 3 (45, 47, 0, 0) 10 (49, 53, 0, 0) 14 (55, 60, 0, 0) 8 (60, 65, 0, 0) 19 (67, 71, 0, 0)

Total penalty for crossdock: 715

Solution 2

Inbound area

Machine 1: 11 (7, 11, 2, 0) 12 (11, 14, 1, 0) 1 (14, 19, 0, 0) 20 (19, 23, 0, 2) 13 (31, 35, 0, 0) 8 (35, 37, 0, 0) 7 (38, 41, 0, 0)
Machine 2: 9 (10, 12, 0, 0) 6 (12, 16, 1, 0) 2 (16, 18, 3, 0) 16 (18, 22, 2, 0) 14 (30, 36, 0, 0) 10 (37, 40, 3, 0)
Machine 3: 17 (9, 11, 1, 0) 4 (11, 15, 0, 0) 15 (15, 17, 2, 0) 19 (17, 23, 0, 3) 3 (24, 26, 0, 0) 5 (27, 29, 0, 0) 18 (34, 40, 0, 0)

Outbound area

Machine 1: 6 (18, 21, 0, 0) 9 (22, 25, 0, 1) 20 (35, 36, 0, 0) 3 (45, 47, 0, 0) 10 (49, 53, 0, 0) 13 (54, 58, 0, 0) 8 (60, 65, 0, 0) 19 (67, 71, 0, 0)
Machine 2: 11 (40, 45, 0, 0) 1 (45, 50, 0, 0) 2 (51, 53, 0, 0) 14 (55, 60, 0, 0) 7 (73, 78, 0, 0)
Machine 3: 21 (15, 16, 0, 0) 5 (24, 28, 0, 0) 4 (47, 49, 1, 0) 15 (49, 52, 0, 0) 16 (52, 56, 0, 0) 12 (56, 58, 0, 0) 17 (64, 68, 0, 0) 18 (68, 72, 0, 0)

Total penalty for crossdock: 616

Solution 3

Inbound area

Machine 1: 9 (10, 12, 0, 0) 6 (12, 16, 1, 0) 15 (16, 18, 1, 0) 2 (18, 20, 1, 0) 16 (20, 24, 0, 0) 5 (27, 29, 0, 0) 18 (34, 40, 0, 0)
Machine 2: 11 (7, 11, 2, 0) 12 (11, 14, 1, 0) 1 (14, 19, 0, 0) 20 (19, 23, 0, 2) 14 (30, 36, 0, 0) 7 (38, 41, 0, 0)
Machine 3: 17 (9, 11, 1, 0) 4 (11, 15, 0, 0) 19 (15, 21, 0, 1) 3 (24, 26, 0, 0) 13 (31, 35, 0, 0) 8 (35, 37, 0, 0) 10 (37, 40, 3, 0)

Outbound area

Machine 1: 9 (24, 27, 0, 3) 11 (40, 45, 0, 0) 10 (49, 53, 0, 0) 13 (54, 58, 0, 0) 8 (60, 65, 0, 0) 18 (68, 72, 0, 0)
Machine 2: 6 (20, 23, 0, 2) 20 (35, 36, 0, 0) 3 (45, 47, 0, 0) 4 (48, 50, 0, 0) 2 (51, 53, 0, 0) 14 (55, 60, 0, 0) 19 (67, 71, 0, 0)
Machine 3: 21 (15, 16, 0, 0) 5 (24, 28, 0, 0) 1 (44, 49, 1, 0) 15 (49, 52, 0, 0) 16 (52, 56, 0, 0) 12 (56, 58, 0, 0) 17 (64, 68, 0, 0) 7 (73, 78, 0, 0)

Total penalty for crossdock: 811

Solution 4

Inbound area

Machine 1: 11 (7, 11, 2, 0) 12 (11, 14, 1, 0) 1 (14, 19, 0, 0) 20 (19, 23, 0, 2) 14 (30, 36, 0, 0) 7 (38, 41, 0, 0)
Machine 2: 17 (9, 11, 1, 0) 4 (11, 15, 0, 0) 19 (15, 21, 0, 1) 3 (24, 26, 0, 0) 13 (31, 35, 0, 0) 8 (35, 37, 0, 0) 10 (37, 40, 3, 0)
Machine 3: 9 (10, 12, 0, 0) 6 (12, 16, 1, 0) 15 (16, 18, 1, 0) 2 (18, 20, 1, 0) 16 (20, 24, 0, 0) 5 (27, 29, 0, 0) 18 (34, 40, 0, 0)

Outbound area

Machine 1: 21 (15, 16, 0, 0) 5 (24, 28, 0, 0) 4 (47, 49, 1, 0) 15 (49, 52, 0, 0) 16 (52, 56, 0, 0) 12 (56, 58, 0, 0) 8 (60, 65, 0, 0) 18 (68, 72, 0, 0)
Machine 2: 6 (20, 23, 0, 2) 20 (35, 36, 0, 0) 3 (45, 47, 0, 0) 10 (49, 53, 0, 0) 13 (54, 58, 0, 0) 17 (64, 68, 0, 0) 7 (73, 78, 0, 0)
Machine 3: 9 (24, 27, 0, 3) 11 (40, 45, 0, 0) 1 (45, 50, 0, 0) 2 (51, 53, 0, 0) 14 (55, 60, 0, 0) 19 (67, 71, 0, 0)

Total penalty for crossdock: 811

Solution 5

Inbound area

Machine 1: 17 (9, 11, 1, 0) 4 (11, 15, 0, 0) 1 (15, 20, 0, 1) 20 (20, 24, 0, 3) 5 (27, 29, 0, 0) 18 (34, 40, 0, 0)
Machine 2: 11 (7, 11, 2, 0) 6 (11, 15, 2, 0) 15 (15, 17, 2, 0) 2 (17, 19, 2, 0) 16 (19, 23, 1, 0) 13 (31, 35, 0, 0) 8 (35, 37, 0, 0) 10 (37, 40, 3, 0)
Machine 3: 9 (10, 12, 0, 0) 12 (12, 15, 0, 0) 19 (15, 21, 0, 1) 3 (24, 26, 0, 0) 14 (30, 36, 0, 0) 7 (38, 41, 0, 0)

Outbound area

Machine 1: 6 (19, 22, 0, 1) 20 (35, 36, 0, 0) 15 (49, 52, 0, 0) 16 (52, 56, 0, 0) 12 (56, 58, 0, 0) 8 (60, 65, 0, 0) 18 (68, 72, 0, 0)
Machine 2: 21 (15, 16, 0, 0) 9 (23, 26, 0, 2) 11 (40, 45, 0, 0) 1 (45, 50, 0, 0) 2 (51, 53, 0, 0) 14 (55, 60, 0, 0) 19 (67, 71, 0, 0)
Machine 3: 5 (24, 28, 0, 0) 3 (45, 47, 0, 0) 4 (47, 49, 1, 0) 10 (49, 53, 0, 0) 13 (54, 58, 0, 0) 17 (64, 68, 0, 0) 7 (73, 78, 0, 0)

Total penalty for crossdock: 814

Appendix 9

Results of the RG and RGTS Algorithms for the 16 Crossdocking - JIT Scheduling Problem Instances shown in Appendix 8

The RG algorithm solution shown corresponds to its best RGTS algorithm solution for each problem instance.

Instance 1

RG algorithm

Inbound area

***i* (s_i,c_i,e_i,t_i)**
Machine 1: 6 (1, 3, 0, 0) 4 (4, 6, 0, 0) 9 (6, 9, 1, 0) 8 (9, 15, 1, 0) 2 (15, 17, 0, 1) 7 (28, 34, 0, 0)
Machine 2: 10 (1, 4, 0, 0) 3 (6, 11, 2, 0) 5 (11, 16, 0, 0) 1 (22, 24, 0, 0)

Outbound area

***j* (S_j, C_j, E_j, T_j)**
Machine 1: 3 (23, 26, 0, 0) 9 (30, 32, 0, 0) 2 (51, 54, 0, 0) 6 (64, 68, 0, 0)
Machine 2: 7 (26, 29, 0, 0) 1 (56, 58, 0, 0) 8 (72, 77, 0, 0)
Machine 3: 5 (14, 15, 0, 0) 11 (25, 27, 0, 0) 4 (44, 46, 0, 0) 10 (63, 68, 0, 0)

Total penalty for crossdock: 104

RGTS algorithm

Inbound area

Machine 1: 6 (1, 3, 0, 0) 4 (4, 6, 0, 0) 9 (6, 9, 1, 0) 8 (9, 15, 1, 0) 2 (15, 17, 0, 1) 7 (28, 34, 0, 0)
Machine 2: 10 (1, 4, 0, 0) 3 (6, 11, 2, 0) 5 (11, 16, 0, 0) 1 (22, 24, 0, 0)

Outbound area

Machine 1: 3 (23, 26, 0, 0) 9 (30, 32, 0, 0) 2 (51, 54, 0, 0) 6 (64, 68, 0, 0)
Machine 2: 7 (26, 29, 0, 0) 1 (56, 58, 0, 0) 8 (72, 77, 0, 0)
Machine 3: 5 (14, 15, 0, 0) 11 (25, 27, 0, 0) 4 (44, 46, 0, 0) 10 (63, 68, 0, 0)

Total penalty for crossdock: 104

Instance 2:

RG algorithm

Inbound area

Machine 1: 2 (2, 6, 0, 0) 3 (16, 19, 0, 0) 10 (19, 25, 0, 0) 9 (28, 33, 0, 0) 11 (33, 39, 0, 0)
Machine 2: 7 (5, 11, 0, 0) 5 (16, 22, 0, 0) 4 (24, 29, 0, 0) 6 (33, 35, 0, 0) 12 (36, 38, 0, 0)
Machine 3: 1 (16, 18, 1, 0) 8 (18, 24, 0, 0) 15 (24, 26, 0, 0) 13 (26, 28, 0, 0) 14 (33, 39, 0, 0)

Outbound area

Machine 1: 4 (36, 39, 1, 0) 6 (39, 40, 0, 0) 5 (40, 42, 2, 0) 2 (42, 44, 0, 0) 12 (53, 54, 0, 0) 9 (57, 58, 0, 0) 7 (60, 65, 0, 0) 8 (65, 69, 0, 0) 14 (76, 81, 0, 0)
Machine 2: 10 (38, 43, 0, 0) 11 (50, 55, 0, 0) 3 (65, 69, 0, 0) 1 (73, 76, 4, 0) 13 (76, 81, 0, 0)

Total penalty for crossdock: 8

RGTS algorithm

Inbound area

Machine 1: 2 (2, 6, 0, 0) 3 (16, 19, 0, 0) 10 (19, 25, 0, 0) 9 (28, 33, 0, 0) 11 (33, 39, 0, 0)
Machine 2: 7 (5, 11, 0, 0) 5 (16, 22, 0, 0) 4 (24, 29, 0, 0) 6 (33, 35, 0, 0) 12 (36, 38, 0, 0)
Machine 3: 1 (16, 18, 1, 0) 8 (18, 24, 0, 0) 15 (24, 26, 0, 0) 13 (26, 28, 0, 0) 14 (33, 39, 0, 0)

Outbound area

Machine 1: 4 (36, 39, 1, 0) 6 (39, 40, 0, 0) 5 (40, 42, 2, 0) 2 (42, 44, 0, 0) 12 (53, 54, 0, 0) 9 (57, 58, 0, 0) 7 (60, 65, 0, 0) 8 (65, 69, 0, 0) 14 (76, 81, 0, 0)
Machine 2: 10 (38, 43, 0, 0) 11 (50, 55, 0, 0) 3 (65, 69, 0, 0) 1 (73, 76, 4, 0) 13 (76, 81, 0, 0)

Total penalty for crossdock: 8

Instance 3:

RG algorithm

Inbound area

Machine 1: 11 (7, 11, 2, 0) 12 (11, 14, 1, 0) 1 (14, 19, 0, 0) 20 (19, 23, 0, 2) 3 (24, 26, 0, 0) 13 (31, 35, 0, 0) 7 (38, 41, 0, 0)
Machine 2: 9 (10, 12, 0, 0) 6 (12, 16, 1, 0) 2 (16, 18, 3, 0) 16 (18, 22, 2, 0) 14 (30, 36, 0, 0) 10 (37, 40, 3, 0)
Machine 3: 17 (9, 11, 1, 0) 4 (11, 15, 0, 0) 15 (15, 17, 2, 0) 19 (17, 23, 0, 3) 5 (27, 29, 0, 0) 8 (32, 34, 3, 0) 18 (34, 40, 0, 0)

Outbound area

Machine 1: 6 (18, 21, 0, 0) 9 (22, 25, 0, 1) 5 (25, 29, 0, 1) 1 (41, 46, 4, 0) 10 (46, 50, 3, 0) 2 (50, 52, 1, 0) 16 (52, 56, 0, 0) 8 (60, 65, 0, 0) 19 (67, 71, 0, 0)
Machine 2: 20 (35, 36, 0, 0) 11 (40, 45, 0, 0) 17 (64, 68, 0, 0) 7 (73, 78, 0, 0)
Machine 3: 21 (15, 16, 0, 0) 3 (42, 44, 3, 0) 4 (44, 46, 4, 0) 15 (46, 49, 3, 0) 13 (49, 53, 5, 0) 12 (53, 55, 3, 0) 14 (55, 60, 0, 0) 18 (68, 72, 0, 0)

Total penalty for crossdock: 744

RGTS algorithm

Inbound area

Machine 1: 11 (7, 11, 2, 0) 12 (11, 14, 1, 0) 1 (14, 19, 0, 0) 20 (19, 23, 0, 2) 13 (31, 35, 0, 0) 8 (35, 37, 0, 0) 7 (38, 41, 0, 0)
Machine 2: 9 (10, 12, 0, 0) 6 (12, 16, 1, 0) 2 (16, 18, 3, 0) 16 (18, 22, 2, 0) 14 (30, 36, 0, 0) 10 (37, 40, 3, 0)
Machine 3: 17 (9, 11, 1, 0) 4 (11, 15, 0, 0) 15 (15, 17, 2, 0) 19 (17, 23, 0, 3) 3 (24, 26, 0, 0) 5 (27, 29, 0, 0) 18 (34, 40, 0, 0)

Outbound area

Machine 1: 6 (18, 21, 0, 0) 9 (22, 25, 0, 1) 20 (35, 36, 0, 0) 3 (45, 47, 0, 0) 10 (49, 53, 0, 0) 13 (54, 58, 0, 0) 8 (60, 65, 0, 0) 19 (67, 71, 0, 0)
Machine 2: 11 (40, 45, 0, 0) 1 (45, 50, 0, 0) 2 (51, 53, 0, 0) 14 (55, 60, 0, 0) 7 (73, 78, 0, 0)
Machine 3: 21 (15, 16, 0, 0) 5 (24, 28, 0, 0) 4 (47, 49, 1, 0) 15 (49, 52, 0, 0) 16 (52, 56, 0, 0) 12 (56, 58, 0, 0) 17 (64, 68, 0, 0) 18 (68, 72, 0, 0)

Total penalty for crossdock: 616

Instance 4:

RG algorithm

Inbound area

Machine 1: 15 (1, 5, 0, 0) 13 (5, 7, 0, 1) 24 (7, 12, 0, 0) 27 (12, 14, 1, 0) 22 (14, 19, 0, 0) 12 (19, 25, 0, 0) 20 (26, 32, 4, 0) 32 (32, 34, 4, 0) 30 (34, 37, 2, 0) 21 (37, 40, 0, 1) 4 (40, 45, 0, 2) 16 (46, 52, 0, 0)
Machine 2: 23 (1, 6, 3, 0) 1 (6, 11, 0, 0) 26 (11, 15, 0, 0) 3 (15, 17, 0, 0) 11 (20, 22, 0, 0) 9 (24, 29, 0, 0) 14 (32, 36, 0, 0) 10 (36, 42, 0, 0) 8 (45, 49, 0, 0) 31 (49, 53, 0, 1)
Machine 3: 25 (0, 3, 2, 0) 5 (3, 8, 0, 0) 17 (8, 14, 0, 1) 7 (14, 17, 0, 0) 19 (17, 20, 2, 0) 28 (20, 25, 0, 0) 6 (31, 36, 0, 0) 18 (36, 41, 0, 0) 2 (43, 49, 1, 0) 29 (49, 51, 0, 0)

Outbound area

Machine 1: 27 (37, 38, 0, 0) 24 (49, 52, 0, 0) 20 (53, 54, 0, 0) 13 (56, 61, 0, 0) 22 (61, 66, 0, 0) 30 (67, 72, 0, 0) 9 (74, 78, 0, 0)
Machine 2: 11 (22, 24, 0, 0) 32 (43, 47, 0, 0) 17 (51, 56, 3, 0) 34 (56, 58, 0, 0) 18 (58, 59, 1, 0) 29 (59, 61, 0, 0) 15 (64, 65, 0, 0) 3 (66, 68, 0, 0) 6 (68, 70, 0, 0) 28 (71, 73, 0, 0) 2 (78, 82, 0, 0)
Machine 3: 25 (26, 29, 0, 0) 4 (42, 43, 0, 0) 21 (50, 51, 0, 0) 12 (53, 57, 1, 0) 23 (57, 62, 0, 1) 10 (66, 71, 0, 0) 33 (72, 76, 0, 0) 5 (81, 84, 0, 0)
Machine 4: 16 (41, 43, 0, 0) 7 (51, 53, 0, 0) 19 (56, 58, 0, 0) 31 (60, 61, 0, 0) 8 (62, 64, 0, 0) 14 (66, 68, 0, 0) 26 (69, 72, 0, 0) 1 (76, 81, 0, 0)

Total penalty for crossdock: 724

RGTS algorithm

Inbound area

Machine 1: 15 (1, 5, 0, 0) 13 (5, 7, 0, 1) 24 (7, 12, 0, 0) 27 (12, 14, 1, 0) 22 (14, 19, 0, 0) 12 (19, 25, 0, 0) 20 (27, 33, 3, 0) 30 (33, 36, 3, 0) 21 (36, 39, 0, 0) 4 (39, 44, 0, 1) 16 (46, 52, 0, 0)
Machine 2: 23 (1, 6, 3, 0) 1 (6, 11, 0, 0) 26 (11, 15, 0, 0) 3 (15, 17, 0, 0) 11 (20, 22, 0, 0) 9 (24, 29, 0, 0) 14 (30, 34, 2, 0) 32 (34, 36, 2, 0) 10 (36, 42, 0, 0) 8 (45, 49, 0, 0) 31 (49, 53, 0, 1)
Machine 3: 25 (0, 3, 2, 0) 5 (3, 8, 0, 0) 17 (8, 14, 0, 1) 7 (14, 17, 0, 0) 19 (17, 20, 2, 0) 28 (20, 25, 0, 0) 6 (31, 36, 0, 0) 18 (36, 41, 0, 0) 2 (43, 49, 1, 0) 29 (49, 51, 0, 0)

Outbound area

Machine 1: 27 (37, 38, 0, 0) 20 (53, 54, 0, 0) 34 (54, 56, 2, 0) 13 (56, 61, 0, 0) 22 (61, 66, 0, 0) 30 (67, 72, 0, 0) 9 (74, 78, 0, 0)
Machine 2: 11 (22, 24, 0, 0) 32 (43, 47, 0, 0) 24 (49, 52, 0, 0) 17 (54, 59, 0, 0) 29 (59, 61, 0, 0) 15 (64, 65, 0, 0) 3 (66, 68, 0, 0) 6 (68, 70, 0, 0) 28 (71, 73, 0, 0) 2 (78, 82, 0, 0)
Machine 3: 25 (26, 29, 0, 0) 4 (42, 43, 0, 0) 21 (50, 51, 0, 0) 12 (54, 58, 0, 0) 18 (59, 60, 0, 0) 31 (60, 61, 0, 0) 10 (66, 71, 0, 0) 33 (72, 76, 0, 0) 5 (81, 84, 0, 0)
Machine 4: 16 (41, 43, 0, 0) 7 (51, 53, 0, 0) 19 (54, 56, 2, 0) 23 (56, 61, 0, 0) 8 (62, 64, 0, 0) 14 (66, 68, 0, 0) 26 (69, 72, 0, 0) 1 (76, 81, 0, 0)

Total penalty for crossdock: 423

Instance 5:

RG algorithm

Inbound area

Machine 1: 8 (10, 14, 0, 0) 29 (15, 17, 0, 0) 24 (19, 22, 0, 0) 28 (26, 30, 0, 0) 20 (30, 32, 0, 0) 16 (32, 35, 0, 2) 1 (36, 41, 0, 0) 14 (45, 51, 3, 0)
Machine 2: 9 (10, 12, 0, 0) 13 (12, 14, 0, 0) 12 (15, 20, 0, 0) 5 (20, 22, 0, 0) 19 (23, 27, 0, 0) 23 (29, 33, 0, 0) 30 (34, 39, 0, 0) 4 (39, 45, 2, 0) 3 (45, 48, 3, 0)
Machine 3: 18 (10, 16, 0, 0) 6 (18, 20, 0, 0) 22 (23, 25, 0, 0) 21 (26, 32, 1, 0) 15 (32, 37, 0, 0) 26 (40, 46, 0, 0)
Machine 4: 27 (4, 9, 0, 0) 10 (12, 18, 0, 0) 17 (19, 24, 0, 0) 7 (27, 33, 0, 0) 11 (33, 36, 0, 0) 2 (37, 41, 0, 0) 25 (42, 47, 0, 0)

Outbound area

Machine 1: 14 (25, 26, 0, 0) 8 (51, 54, 0, 0) 13 (55, 58, 0, 0) 4 (61, 62, 0, 0) 25 (63, 67, 0, 0) 5 (74, 79, 0, 0)
Machine 2: 1 (38, 40, 0, 0) 22 (51, 56, 0, 0) 11 (58, 61, 0, 0) 17 (62, 67, 0, 0) 12 (72, 75, 0, 0) 18 (89, 94, 0, 0)
Machine 3: 7 (32, 33, 0, 0) 29 (51, 55, 0, 0) 3 (56, 61, 0, 0) 27 (62, 65, 0, 0) 24 (66, 71, 0, 0) 6 (79, 82, 0, 0)
Machine 4: 28 (39, 43, 0, 0) 16 (52, 57, 0, 0) 20 (61, 66, 0, 0) 23 (69, 70, 0, 0) 2 (75, 80, 0, 0)
Machine 5: 21 (48, 52, 0, 0) 15 (54, 59, 0, 0) 19 (60, 63, 2, 0) 10 (63, 67, 0, 0) 26 (69, 72, 0, 0) 9 (80, 85, 0, 0)

Total penalty for crossdock: 211

RGTS algorithm

Inbound area

Machine 1: 8 (10, 14, 0, 0) 29 (15, 17, 0, 0) 24 (19, 22, 0, 0) 28 (26, 30, 0, 0) 20 (30, 32, 0, 0) 16 (32, 35, 0, 2) 1 (36, 41, 0, 0) 14 (45, 51, 3, 0)
Machine 2: 9 (10, 12, 0, 0) 13 (12, 14, 0, 0) 12 (15, 20, 0, 0) 5 (20, 22, 0, 0) 19 (23, 27, 0, 0) 23 (29, 33, 0, 0) 30 (34, 39, 0, 0) 4 (39, 45, 2, 0) 3 (45, 48, 3, 0)
Machine 3: 18 (10, 16, 0, 0) 6 (18, 20, 0, 0) 22 (23, 25, 0, 0) 21 (26, 32, 1, 0) 15 (32, 37, 0, 0) 26 (40, 46, 0, 0)
Machine 4: 27 (4, 9, 0, 0) 10 (12, 18, 0, 0) 17 (19, 24, 0, 0) 7 (27, 33, 0, 0) 11 (33, 36, 0, 0) 2 (37, 41, 0, 0) 25 (42, 47, 0, 0)

Outbound area

Machine 1: 14 (25, 26, 0, 0) 8 (51, 54, 0, 0) 13 (55, 58, 0, 0) 4 (61, 62, 0, 0) 25 (63, 67, 0, 0) 5 (74, 79, 0, 0)
Machine 2: 1 (38, 40, 0, 0) 22 (51, 56, 0, 0) 11 (58, 61, 0, 0) 17 (62, 67, 0, 0) 12 (72, 75, 0, 0) 18 (89, 94, 0, 0)
Machine 3: 7 (32, 33, 0, 0) 29 (51, 55, 0, 0) 3 (56, 61, 0, 0) 27 (62, 65, 0, 0) 24 (66, 71, 0, 0) 6 (79, 82, 0, 0)
Machine 4: 28 (39, 43, 0, 0) 16 (52, 57, 0, 0) 20 (61, 66, 0, 0) 23 (69, 70, 0, 0) 2 (75, 80, 0, 0)
Machine 5: 21 (48, 52, 0, 0) 15 (54, 59, 0, 0) 19 (60, 63, 2, 0) 10 (63, 67, 0, 0) 26 (69, 72, 0, 0) 9 (80, 85, 0, 0)

Total penalty for crossdock: 211

Instance 6:

RG algorithm

Inbound area

Machine 1: 14 (3, 7, 0, 0) 5 (11, 13, 0, 0) 13 (16, 20, 0, 0) 26 (22, 24, 0, 0) 16 (27, 32, 0, 0) 15 (39, 43, 0, 0) 18 (47, 50, 1, 0) 7 (50, 54, 0, 0)
Machine 2: 24 (7, 11, 0, 0) 10 (15, 18, 0, 0) 22 (18, 23, 0, 0) 20 (26, 29, 0, 0) 2 (35, 39, 0, 0) 25 (40, 42, 0, 0) 31 (43, 49, 2, 0) 27 (49, 54, 0, 0)
Machine 3: 28 (4, 7, 0, 0) 8 (12, 18, 0, 0) 32 (18, 22, 0, 0) 30 (23, 28, 0, 0) 3 (33, 39, 0, 0) 11 (42, 48, 0, 0) 19 (49, 55, 0, 0)
Machine 4: 12 (7, 9, 0, 0) 21 (12, 15, 0, 0) 9 (18, 22, 0, 0) 4 (23, 26, 0, 0) 23 (29, 31, 0, 0) 29 (36, 38, 0, 0) 17 (40, 46, 0, 0) 6 (47, 51, 1, 0) 1 (52, 56, 0, 0)

Outbound area

Machine 1: 27 (25, 26, 0, 0) 23 (39, 42, 0, 0) 22 (47, 50, 0, 0) 30 (63, 68, 0, 0) 13 (69, 73, 0, 0) 25 (73, 75, 0, 0) 3 (81, 85, 0, 0)
Machine 2: 18 (22, 23, 0, 0) 9 (36, 38, 0, 0) 29 (46, 48, 0, 0) 11 (54, 55, 0, 0) 15 (63, 65, 0, 0) 20 (66, 71, 1, 0) 24 (71, 74, 0, 0) 16 (75, 79, 0, 0)
Machine 3: 17 (25, 27, 0, 0) 21 (44, 47, 0, 0) 33 (51, 56, 0, 0) 1 (65, 70, 0, 0) 26 (71, 74, 0, 0) 2 (75, 80, 0, 0)
Machine 4: 28 (27, 28, 0, 0) 12 (45, 48, 0, 0) 6 (54, 58, 0, 0) 14 (66, 70, 0, 0) 8 (71, 75, 0, 0) 19 (81, 86, 0, 0)
Machine 5: 10 (27, 29, 0, 0) 31 (46, 49, 0, 0) 32 (58, 60, 0, 0) 4 (66, 71, 0, 0) 5 (71, 76, 0, 0) 7 (88, 91, 0, 0)

Total penalty for crossdock: 5

RGTS algorithm

Inbound area

Machine 1: 14 (3, 7, 0, 0) 5 (11, 13, 0, 0) 13 (16, 20, 0, 0) 26 (22, 24, 0, 0) 16 (27, 32, 0, 0) 15 (39, 43, 0, 0) 18 (47, 50, 1, 0) 7 (50, 54, 0, 0)
Machine 2: 24 (7, 11, 0, 0) 10 (15, 18, 0, 0) 22 (18, 23, 0, 0) 20 (26, 29, 0, 0) 2 (35, 39, 0, 0) 25 (40, 42, 0, 0) 31 (43, 49, 2, 0) 27 (49, 54, 0, 0)
Machine 3: 28 (4, 7, 0, 0) 8 (12, 18, 0, 0) 32 (18, 22, 0, 0) 30 (23, 28, 0, 0) 3 (33, 39, 0, 0) 11 (42, 48, 0, 0) 19 (49, 55, 0, 0)
Machine 4: 12 (7, 9, 0, 0) 21 (12, 15, 0, 0) 9 (18, 22, 0, 0) 4 (23, 26, 0, 0) 23 (29, 31, 0, 0) 29 (36, 38, 0, 0) 17 (40, 46, 0, 0) 6 (47, 51, 1, 0) 1 (52, 56, 0, 0)

Outbound area

Machine 1: 27 (25, 26, 0, 0) 23 (39, 42, 0, 0) 22 (47, 50, 0, 0) 30 (63, 68, 0, 0) 13 (69, 73, 0, 0) 25 (73, 75, 0, 0) 3 (81, 85, 0, 0)
Machine 2: 18 (22, 23, 0, 0) 9 (36, 38, 0, 0) 29 (46, 48, 0, 0) 11 (54, 55, 0, 0) 15 (63, 65, 0, 0) 20 (66, 71, 1, 0) 24 (71, 74, 0, 0) 16 (75, 79, 0, 0)
Machine 3: 17 (25, 27, 0, 0) 21 (44, 47, 0, 0) 33 (51, 56, 0, 0) 1 (65, 70, 0, 0) 26 (71, 74, 0, 0) 2 (75, 80, 0, 0)
Machine 4: 28 (27, 28, 0, 0) 12 (45, 48, 0, 0) 6 (54, 58, 0, 0) 14 (66, 70, 0, 0) 8 (71, 75, 0, 0) 19 (81, 86, 0, 0)
Machine 5: 10 (27, 29, 0, 0) 31 (46, 49, 0, 0) 32 (58, 60, 0, 0) 4 (66, 71, 0, 0) 5 (71, 76, 0, 0) 7 (88, 91, 0, 0)

Total penalty for crossdock: 5

Instance 7:

RG algorithm

Inbound area

Machine 1: 25 (9, 12, 0, 0) 17 (38, 41, 0, 0) 15 (49, 54, 0, 0) 28 (64, 66, 0, 0) 7 (76, 80, 0, 0) 26 (87, 92, 0, 0)
Machine 2: 6 (13, 19, 0, 0) 18 (49, 51, 0, 0) 1 (59, 62, 0, 0) 30 (71, 73, 0, 0) 5 (81, 86, 0, 0) 24 (88, 92, 0, 0)
Machine 3: 21 (31, 36, 0, 0) 11 (49, 52, 0, 0) 9 (62, 64, 0, 0) 23 (69, 74, 0, 0) 12 (84, 88, 0, 0) 3 (91, 95, 0, 0)
Machine 4: 29 (8, 10, 0, 0) 13 (32, 35, 1, 0) 27 (50, 52, 0, 0) 19 (63, 65, 0, 0) 16 (75, 78, 0, 0) 2 (87, 90, 0, 0)
Machine 5: 20 (17, 19, 0, 0) 14 (41, 43, 0, 0) 8 (57, 61, 0, 0) 4 (65, 71, 0, 0) 10 (81, 83, 0, 0) 22 (90, 92, 0, 0)

Outbound area

Machine 1: 6 (46, 47, 0, 0) 28 (73, 76, 0, 0) 19 (93, 98, 0, 0) 25 (106, 111, 0, 0) 20 (116, 119, 0, 0) 24 (124, 126, 0, 0)
Machine 2: 27 (58, 59, 0, 0) 1 (85, 86, 0, 0) 8 (97, 99, 0, 0) 17 (111, 112, 0, 0) 3 (117, 120, 0, 0) 7 (125, 130, 0, 0)
Machine 3: 4 (66, 69, 0, 0) 21 (89, 94, 0, 0) 16 (102, 107, 0, 0) 5 (114, 118, 0, 0) 30 (120, 123, 0, 0)
Machine 4: 12 (35, 37, 0, 0) 18 (71, 73, 0, 0) 13 (90, 94, 0, 0) 2 (102, 105, 0, 0) 26 (112, 117, 0, 0) 9 (118, 122, 0, 0) 10 (128, 131, 0, 0)
Machine 5: 14 (59, 62, 0, 0) 22 (92, 93, 0, 0) 29 (97, 101, 0, 0) 15 (113, 116, 0, 0) 23 (117, 120, 0, 0) 11 (124, 127, 0, 0)

Total penalty for crossdock: 1

RGTS algorithm

Inbound area

Machine 1: 25 (9, 12, 0, 0) 17 (38, 41, 0, 0) 15 (49, 54, 0, 0) 28 (64, 66, 0, 0) 7 (76, 80, 0, 0) 26 (87, 92, 0, 0)
Machine 2: 6 (13, 19, 0, 0) 18 (49, 51, 0, 0) 1 (59, 62, 0, 0) 30 (71, 73, 0, 0) 5 (81, 86, 0, 0) 24 (88, 92, 0, 0)
Machine 3: 21 (31, 36, 0, 0) 11 (49, 52, 0, 0) 9 (62, 64, 0, 0) 23 (69, 74, 0, 0) 12 (84, 88, 0, 0) 3 (91, 95, 0, 0)
Machine 4: 29 (8, 10, 0, 0) 13 (32, 35, 1, 0) 27 (50, 52, 0, 0) 19 (63, 65, 0, 0) 16 (75, 78, 0, 0) 2 (87, 90, 0, 0)
Machine 5: 20 (17, 19, 0, 0) 14 (41, 43, 0, 0) 8 (57, 61, 0, 0) 4 (65, 71, 0, 0) 10 (81, 83, 0, 0) 22 (90, 92, 0, 0)

Outbound area

Machine 1: 6 (46, 47, 0, 0) 28 (73, 76, 0, 0) 19 (93, 98, 0, 0) 25 (106, 111, 0, 0) 20 (116, 119, 0, 0) 24 (124, 126, 0, 0)
Machine 2: 27 (58, 59, 0, 0) 1 (85, 86, 0, 0) 8 (97, 99, 0, 0) 17 (111, 112, 0, 0) 3 (117, 120, 0, 0) 7 (125, 130, 0, 0)
Machine 3: 4 (66, 69, 0, 0) 21 (89, 94, 0, 0) 16 (102, 107, 0, 0) 5 (114, 118, 0, 0) 30 (120, 123, 0, 0)
Machine 4: 12 (35, 37, 0, 0) 18 (71, 73, 0, 0) 13 (90, 94, 0, 0) 2 (102, 105, 0, 0) 26 (112, 117, 0, 0) 9 (118, 122, 0, 0) 10 (128, 131, 0, 0)
Machine 5: 14 (59, 62, 0, 0) 22 (92, 93, 0, 0) 29 (97, 101, 0, 0) 15 (113, 116, 0, 0) 23 (117, 120, 0, 0) 11 (124, 127, 0, 0)

Total penalty for crossdock: 1

Instance 8:

RG algorithm

Inbound area

Machine 1: 6 (10, 16, 0, 0) 26 (24, 28, 0, 0) 17 (33, 38, 0, 0) 22 (38, 43, 0, 0) 38 (44, 46, 3, 0) 34 (56, 62, 0, 0)
Machine 2: 1 (6, 12, 0, 0) 15 (18, 24, 0, 0) 37 (26, 28, 0, 0) 31 (31, 37, 0, 0) 5 (37, 39, 0, 0) 10 (41, 47, 0, 0) 21 (49, 53, 0, 0) 19 (57, 63, 0, 0)
Machine 3: 35 (5, 7, 0, 0) 28 (13, 15, 0, 0) 7 (21, 26, 0, 0) 9 (28, 33, 1, 0) 30 (33, 36, 0, 0) 16 (37, 39, 0, 0) 39 (39, 41, 0, 0) 8 (42, 46, 0, 0) 14 (48, 51, 0, 0) 23 (57, 62, 0, 0)
Machine 4: 2 (7, 9, 0, 0) 40 (13, 15, 0, 0) 32 (21, 25, 0, 0) 36 (25, 30, 0, 0) 20 (33, 37, 0, 0) 13 (37, 39, 0, 0) 18 (41, 45, 0, 0) 11 (47, 52, 0, 0)
Machine 5: 24 (10, 14, 0, 0) 29 (24, 30, 0, 0) 12 (32, 35, 0, 0) 4 (36, 38, 0, 0) 3 (38, 42, 1, 0) 25 (42, 46, 2, 0) 27 (52, 55, 0, 0) 33 (59, 65, 0, 0)

Outbound area

Machine 1: 15 (42, 43, 0, 0) 21 (53, 56, 0, 0) 19 (63, 67, 1, 0) 28 (67, 72, 0, 0) 7 (75, 77, 0, 0) 25 (79, 84, 0, 0) 20 (100, 105, 0, 0)
Machine 2: 24 (44, 45, 0, 0) 18 (55, 58, 0, 0) 14 (64, 65, 0, 0) 34 (66, 70, 0, 0) 26 (72, 75, 0, 0) 36 (78, 80, 0, 0) 30 (80, 85, 0, 0) 1 (90, 95, 0, 0)
Machine 3: 27 (36, 37, 0, 0) 12 (46, 49, 0, 0) 4 (61, 65, 0, 0) 17 (67, 72, 0, 0) 8 (75, 79, 0, 0) 11 (79, 81, 0, 0) 37 (84, 89, 0, 0)
Machine 4: 10 (33, 34, 0, 0) 6 (45, 49, 0, 0) 2 (56, 61, 0, 0) 16 (64, 69, 0, 0) 33 (71, 76, 0, 0) 5 (77, 81, 0, 0) 29 (81, 82, 0, 0) 9 (83, 85, 0, 0) 32 (86, 91, 0, 0)
Machine 5: 3 (43, 44, 0, 0) 31 (53, 56, 0, 0) 35 (62, 64, 1, 0) 13 (64, 69, 0, 0) 22 (71, 76, 0, 0) 38 (79, 84, 0, 0) 23 (86, 89, 0, 0)

Total penalty for crossdock: 9

RGTS algorithm

Inbound area

Machine 1: 6 (10, 16, 0, 0) 26 (24, 28, 0, 0) 17 (33, 38, 0, 0) 22 (38, 43, 0, 0) 38 (44, 46, 3, 0) 34 (56, 62, 0, 0)
Machine 2: 1 (6, 12, 0, 0) 15 (18, 24, 0, 0) 37 (26, 28, 0, 0) 31 (31, 37, 0, 0) 5 (37, 39, 0, 0) 10 (41, 47, 0, 0) 21 (49, 53, 0, 0) 19 (57, 63, 0, 0)
Machine 3: 35 (5, 7, 0, 0) 28 (13, 15, 0, 0) 7 (21, 26, 0, 0) 9 (28, 33, 1, 0) 30 (33, 36, 0, 0) 16 (37, 39, 0, 0) 39 (39, 41, 0, 0) 8 (42, 46, 0, 0) 14 (48, 51, 0, 0) 23 (57, 62, 0, 0)
Machine 4: 2 (7, 9, 0, 0) 40 (13, 15, 0, 0) 32 (21, 25, 0, 0) 36 (25, 30, 0, 0) 20 (33, 37, 0, 0) 13 (37, 39, 0, 0) 18 (41, 45, 0, 0) 11 (47, 52, 0, 0)
Machine 5: 24 (10, 14, 0, 0) 29 (24, 30, 0, 0) 12 (32, 35, 0, 0) 4 (36, 38, 0, 0) 3 (38, 42, 1, 0) 25 (42, 46, 2, 0) 27 (52, 55, 0, 0) 33 (59, 65, 0, 0)

Outbound area

Machine 1: 15 (42, 43, 0, 0) 21 (53, 56, 0, 0) 19 (63, 67, 1, 0) 28 (67, 72, 0, 0) 7 (75, 77, 0, 0) 25 (79, 84, 0, 0) 20 (100, 105, 0, 0)
Machine 2: 24 (44, 45, 0, 0) 18 (55, 58, 0, 0) 14 (64, 65, 0, 0) 34 (66, 70, 0, 0) 26 (72, 75, 0, 0) 36 (78, 80, 0, 0) 30 (80, 85, 0, 0) 1 (90, 95, 0, 0)
Machine 3: 27 (36, 37, 0, 0) 12 (46, 49, 0, 0) 4 (61, 65, 0, 0) 17 (67, 72, 0, 0) 8 (75, 79, 0, 0) 11 (79, 81, 0, 0) 37 (84, 89, 0, 0)
Machine 4: 10 (33, 34, 0, 0) 6 (45, 49, 0, 0) 2 (56, 61, 0, 0) 16 (64, 69, 0, 0) 33 (71, 76, 0, 0) 5 (77, 81, 0, 0) 29 (81, 82, 0, 0) 9 (83, 85, 0, 0) 32 (86, 91, 0, 0)
Machine 5: 3 (43, 44, 0, 0) 31 (53, 56, 0, 0) 35 (62, 64, 1, 0) 13 (64, 69, 0, 0) 22 (71, 76, 0, 0) 38 (79, 84, 0, 0) 23 (86, 89, 0, 0)

Total penalty for crossdock: 9

Instance 9:

RG algorithm

Inbound area

Machine 1: 21 (4, 9, 1, 0) 15 (9, 15, 0, 0) 28 (18, 24, 0, 0) 9 (27, 30, 0, 0) 7 (31, 36, 1, 0) 2 (36, 42, 0, 0) 14 (44, 47, 0, 0) 41 (52, 56, 0, 0)
Machine 2: 20 (6, 12, 0, 0) 42 (15, 18, 0, 0) 19 (18, 22, 0, 0) 39 (22, 25, 0, 0) 10 (28, 33, 0, 0) 23 (33, 38, 3, 0) 24 (38, 41, 0, 0) 11 (45, 51, 0, 0)
Machine 3: 30 (6, 9, 2, 0) 1 (9, 12, 0, 0) 13 (12, 17, 0, 0) 4 (18, 24, 0, 0) 8 (26, 31, 0, 0) 16 (35, 40, 0, 0) 6 (43, 45, 0, 0) 5 (45, 51, 0, 0) 26 (55, 57, 0, 0)
Machine 4: 31 (5, 10, 1, 0) 18 (10, 13, 0, 0) 27 (14, 16, 0, 0) 33 (19, 22, 0, 0) 36 (25, 29, 0, 0) 35 (30, 34, 0, 0) 25 (35, 39, 1, 0) 40 (39, 44, 0, 1) 12 (45, 51, 0, 0) 29 (56, 59, 0, 0)
Machine 5: 34 (7, 10, 0, 0) 38 (10, 13, 0, 0) 37 (17, 23, 0, 0) 32 (25, 28, 0, 0) 17 (32, 36, 0, 0) 22 (37, 43, 0, 0) 3 (50, 55, 0, 0)

Outbound area

Machine 1: 20 (30, 31, 0, 0) 18 (39, 40, 0, 0) 43 (48, 49, 0, 0) 2 (55, 59, 0, 0) 24 (59, 61, 1, 0) 41 (61, 66, 0, 0) 4 (70, 72, 0, 0) 36 (74, 78, 0, 0) 15 (84, 86, 0, 0)
Machine 2: 9 (36, 38, 0, 0) 31 (44, 46, 0, 0) 7 (53, 55, 0, 0) 32 (55, 59, 0, 0) 34 (59, 60, 2, 0) 21 (60, 63, 0, 0) 19 (66, 67, 0, 0) 12 (70, 73, 0, 0) 33 (83, 86, 0, 0)
Machine 3: 26 (31, 32, 0, 0) 28 (36, 37, 0, 0) 37 (46, 49, 0, 0) 10 (53, 55, 0, 0) 22 (55, 58, 0, 0) 23 (60, 64, 0, 0) 6 (64, 69, 0, 0) 3 (72, 74, 0, 0) 13 (81, 83, 0, 0)
Machine 4: 39 (35, 36, 0, 0) 14 (42, 45, 0, 0) 29 (48, 49, 0, 0) 17 (53, 58, 1, 0) 30 (58, 62, 1, 0) 11 (62, 63, 0, 0) 16 (64, 69, 0, 0) 5 (73, 75, 0, 0) 8 (87, 92, 0, 0)
Machine 5: 35 (31, 32, 0, 0) 25 (39, 44, 0, 0) 1 (52, 55, 0, 0) 40 (58, 63, 0, 0) 27 (63, 66, 0, 0) 38 (72, 75, 0, 0) 42 (89, 93, 0, 0)

Total penalty for crossdock: 114

RGTS algorithm

Inbound area

Machine 1: 21 (4, 9, 1, 0) 15 (9, 15, 0, 0) 28 (18, 24, 0, 0) 9 (27, 30, 0, 0) 17 (32, 36, 0, 0) 2 (36, 42, 0, 0) 14 (44, 47, 0, 0) 41 (52, 56, 0, 0)
Machine 2: 20 (6, 12, 0, 0) 42 (15, 18, 0, 0) 19 (18, 22, 0, 0) 39 (22, 25, 0, 0) 10 (28, 33, 0, 0) 23 (33, 38, 3, 0) 24 (38, 41, 0, 0) 11 (45, 51, 0, 0)
Machine 3: 30 (6, 9, 2, 0) 1 (9, 12, 0, 0) 13 (12, 17, 0, 0) 4 (18, 24, 0, 0) 8 (26, 31, 0, 0) 16 (35, 40, 0, 0) 6 (43, 45, 0, 0) 5 (45, 51, 0, 0) 26 (55, 57, 0, 0)
Machine 4: 31 (5, 10, 1, 0) 18 (10, 13, 0, 0) 27 (14, 16, 0, 0) 33 (19, 22, 0, 0) 36 (25, 29, 0, 0) 35 (30, 34, 0, 0) 25 (35, 39, 1, 0) 40 (39, 44, 0, 1) 12 (45, 51, 0, 0) 29 (56, 59, 0, 0)
Machine 5: 34 (7, 10, 0, 0) 38 (10, 13, 0, 0) 37 (17, 23, 0, 0) 32 (25, 28, 0, 0) 7 (32, 37, 0, 0) 22 (37, 43, 0, 0) 3 (50, 55, 0, 0)

Outbound area

Machine 1: 20 (30, 31, 0, 0) 18 (39, 40, 0, 0) 43 (48, 49, 0, 0) 2 (55, 59, 0, 0) 24 (59, 61, 1, 0) 41 (61, 66, 0, 0) 4 (70, 72, 0, 0) 36 (74, 78, 0, 0) 15 (84, 86, 0, 0)
Machine 2: 9 (36, 38, 0, 0) 31 (44, 46, 0, 0) 7 (53, 55, 0, 0) 32 (55, 59, 0, 0) 21 (60, 63, 0, 0) 19 (66, 67, 0, 0) 12 (70, 73, 0, 0) 33 (83, 86, 0, 0)
Machine 3: 26 (31, 32, 0, 0) 28 (36, 37, 0, 0) 37 (46, 49, 0, 0) 10 (53, 55, 0, 0) 22 (55, 58, 0, 0) 23 (60, 64, 0, 0) 6 (64, 69, 0, 0) 3 (72, 74, 0, 0) 13 (81, 83, 0, 0)
Machine 4: 39 (35, 36, 0, 0) 14 (42, 45, 0, 0) 29 (48, 49, 0, 0) 17 (54, 59, 0, 0) 30 (59, 63, 0, 0) 16 (64, 69, 0, 0) 5 (73, 75, 0, 0) 8 (87, 92, 0, 0)
Machine 5: 35 (31, 32, 0, 0) 25 (39, 44, 0, 0) 1 (52, 55, 0, 0) 40 (56, 61, 2, 0) 34 (61, 62, 0, 0) 11 (62, 63, 0, 0) 27 (63, 66, 0, 0) 38 (72, 75, 0, 0) 42 (89, 93, 0, 0)

Total penalty for crossdock: 111

Instance 10:

RG algorithm

Inbound area

Machine 1: 27 (1, 3, 0, 0) 29 (10, 13, 0, 0) 1 (23, 27, 0, 0) 23 (28, 33, 0, 0) 12 (44, 49, 0, 0) 18 (52, 55, 0, 0)
Machine 2: 6 (3, 8, 0, 0) 14 (8, 13, 0, 0) 3 (20, 25, 0, 0) 10 (25, 31, 0, 0) 22 (32, 37, 0, 0) 16 (46, 50, 0, 0)
Machine 3: 5 (0, 3, 0, 0) 9 (9, 15, 0, 0) 32 (21, 27, 0, 0) 28 (27, 33, 0, 0) 24 (41, 47, 1, 0) 13 (51, 57, 0, 0)
Machine 4: 17 (8, 13, 0, 0) 8 (19, 25, 0, 0) 20 (27, 33, 0, 0) 25 (41, 47, 0, 0) 26 (49, 51, 0, 0)
Machine 5: 15 (6, 10, 0, 0) 30 (13, 18, 2, 0) 7 (23, 26, 0, 0) 31 (27, 29, 0, 0) 19 (29, 31, 0, 0) 4 (31, 36, 0, 0) 2 (44, 46, 0, 0) 21 (48, 51, 0, 0) 11 (52, 57, 0, 0)

Outbound area

Machine 1: 24 (25, 26, 0, 0) 17 (46, 48, 0, 0) 27 (55, 56, 0, 0) 33 (57, 59, 0, 0) 12 (59, 61, 0, 0) 23 (63, 66, 0, 0) 19 (67, 70, 0, 0) 32 (96, 100, 0, 0)
Machine 2: 1 (47, 52, 0, 0) 9 (57, 62, 0, 0) 35 (65, 69, 0, 0) 8 (82, 84, 0, 0)
Machine 3: 28 (38, 41, 0, 0) 31 (51, 55, 0, 0) 7 (59, 61, 0, 0) 16 (63, 67, 0, 0) 3 (70, 75, 0, 0) 29 (94, 98, 0, 0)
Machine 4: 21 (39, 40, 0, 0) 22 (53, 57, 0, 0) 20 (59, 64, 0, 0) 4 (65, 70, 0, 0) 30 (87, 91, 0, 0)
Machine 5: 34 (18, 19, 0, 0) 5 (45, 47, 0, 0) 26 (54, 56, 0, 0) 13 (60, 61, 0, 0) 2 (66, 69, 0, 0) 10 (92, 97, 0, 0)
Machine 6: 14 (35, 37, 0, 0) 11 (53, 54, 0, 0) 25 (58, 60, 0, 0) 6 (61, 66, 0, 0) 15 (68, 73, 0, 0) 18 (89, 90, 0, 0)

Total penalty for crossdock: 3

RGTS algorithm

Inbound area

Machine 1: 27 (1, 3, 0, 0) 29 (10, 13, 0, 0) 1 (23, 27, 0, 0) 23 (28, 33, 0, 0) 12 (44, 49, 0, 0) 18 (52, 55, 0, 0)
Machine 2: 6 (3, 8, 0, 0) 14 (8, 13, 0, 0) 3 (20, 25, 0, 0) 10 (25, 31, 0, 0) 22 (32, 37, 0, 0) 16 (46, 50, 0, 0)
Machine 3: 5 (0, 3, 0, 0) 9 (9, 15, 0, 0) 32 (21, 27, 0, 0) 28 (27, 33, 0, 0) 24 (41, 47, 1, 0) 13 (51, 57, 0, 0)
Machine 4: 17 (8, 13, 0, 0) 8 (19, 25, 0, 0) 20 (27, 33, 0, 0) 25 (41, 47, 0, 0) 26 (49, 51, 0, 0)
Machine 5: 15 (6, 10, 0, 0) 30 (13, 18, 2, 0) 7 (23, 26, 0, 0) 31 (27, 29, 0, 0) 19 (29, 31, 0, 0) 4 (31, 36, 0, 0) 2 (44, 46, 0, 0) 21 (48, 51, 0, 0) 11 (52, 57, 0, 0)

Outbound area

Machine 1: 24 (25, 26, 0, 0) 17 (46, 48, 0, 0) 27 (55, 56, 0, 0) 33 (57, 59, 0, 0) 12 (59, 61, 0, 0) 23 (63, 66, 0, 0) 19 (67, 70, 0, 0) 32 (96, 100, 0, 0)
Machine 2: 1 (47, 52, 0, 0) 9 (57, 62, 0, 0) 35 (65, 69, 0, 0) 8 (82, 84, 0, 0)
Machine 3: 28 (38, 41, 0, 0) 31 (51, 55, 0, 0) 7 (59, 61, 0, 0) 16 (63, 67, 0, 0) 3 (70, 75, 0, 0) 29 (94, 98, 0, 0)
Machine 4: 21 (39, 40, 0, 0) 22 (53, 57, 0, 0) 20 (59, 64, 0, 0) 4 (65, 70, 0, 0) 30 (87, 91, 0, 0)
Machine 5: 34 (18, 19, 0, 0) 5 (45, 47, 0, 0) 26 (54, 56, 0, 0) 13 (60, 61, 0, 0) 2 (66, 69, 0, 0) 10 (92, 97, 0, 0)
Machine 6: 14 (35, 37, 0, 0) 11 (53, 54, 0, 0) 25 (58, 60, 0, 0) 6 (61, 66, 0, 0) 15 (68, 73, 0, 0) 18 (89, 90, 0, 0)

Total penalty for crossdock: 3

Instance 11:

RG algorithm

Inbound area

Machine 1: 8 (6, 10, 1, 0) 37 (10, 13, 0, 0) 40 (20, 23, 0, 0) 13 (26, 31, 0, 0) 12 (38, 41, 0, 0) 9 (44, 49, 0, 0) 10 (54, 59, 0, 0)
Machine 2: 26 (6, 8, 0, 0) 18 (9, 14, 0, 0) 35 (14, 20, 0, 0) 4 (23, 29, 0, 0) 36 (31, 36, 0, 0) 33 (39, 45, 0, 0) 29 (45, 48, 0, 0) 21 (49, 52, 0, 0) 31 (58, 61, 0, 0)
Machine 3: 25 (6, 8, 1, 0) 5 (10, 12, 2, 0) 28 (14, 20, 0, 0) 39 (23, 28, 0, 0) 15 (28, 31, 0, 0) 14 (36, 42, 0, 0) 1 (44, 48, 0, 0) 6 (52, 54, 0, 0)
Machine 4: 16 (5, 11, 4, 0) 24 (11, 14, 0, 0) 7 (20, 22, 0, 0) 20 (27, 31, 0, 0) 38 (38, 42, 0, 0) 11 (43, 46, 0, 0) 22 (48, 52, 0, 0) 17 (55, 58, 0, 0)
Machine 5: 23 (5, 7, 0, 0) 3 (8, 12, 0, 0) 32 (14, 18, 0, 0) 2 (23, 29, 0, 0) 27 (33, 38, 0, 0) 19 (43, 49, 0, 0) 30 (52, 55, 0, 0) 34 (58, 61, 0, 0)

Outbound area

Machine 1: 33 (12, 13, 0, 0) 14 (41, 42, 0, 0) 1 (49, 50, 0, 0) 7 (50, 54, 0, 0) 15 (61, 66, 0, 0) 35 (68, 73, 0, 0) 10 (79, 82, 0, 0)
Machine 2: 41 (34, 35, 0, 0) 2 (46, 49, 0, 0) 28 (52, 55, 0, 0) 40 (62, 66, 0, 0) 31 (69, 71, 0, 0) 22 (77, 81, 0, 0) 37 (95, 100, 0, 0)
Machine 3: 8 (38, 40, 0, 0) 18 (47, 50, 2, 0) 4 (50, 51, 0, 0) 13 (53, 56, 0, 0) 34 (63, 66, 0, 0) 5 (67, 69, 0, 0) 12 (70, 71, 0, 0) 30 (72, 74, 0, 0) 43 (82, 86, 0, 0)
Machine 4: 42 (27, 29, 0, 0) 32 (45, 46, 0, 0) 19 (51, 56, 0, 0) 36 (65, 68, 0, 0) 21 (69, 71, 0, 0) 27 (73, 75, 0, 0) 38 (92, 97, 0, 0)
Machine 5: 3 (28, 30, 0, 0) 25 (43, 46, 0, 0) 11 (47, 52, 0, 0) 6 (56, 59, 0, 0) 26 (66, 71, 0, 0) 17 (77, 78, 0, 0)
Machine 6: 23 (35, 36, 0, 0) 20 (48, 50, 0, 0) 29 (52, 54, 0, 0) 16 (60, 62, 0, 0) 24 (66, 68, 0, 0) 39 (71, 72, 0, 0) 9 (87, 90, 0, 0)

Total penalty for crossdock: 10

RGTS algorithm

Inbound area

Machine 1: 8 (6, 10, 1, 0) 37 (10, 13, 0, 0) 40 (20, 23, 0, 0) 13 (26, 31, 0, 0) 12 (38, 41, 0, 0) 9 (44, 49, 0, 0) 10 (54, 59, 0, 0)
Machine 2: 26 (6, 8, 0, 0) 18 (9, 14, 0, 0) 35 (14, 20, 0, 0) 4 (23, 29, 0, 0) 36 (31, 36, 0, 0) 33 (39, 45, 0, 0) 29 (45, 48, 0, 0) 21 (49, 52, 0, 0) 31 (58, 61, 0, 0)
Machine 3: 25 (7, 9, 0, 0) 5 (9, 11, 3, 0) 24 (11, 14, 0, 0) 28 (14, 20, 0, 0) 39 (23, 28, 0, 0) 15 (28, 31, 0, 0) 14 (36, 42, 0, 0) 1 (44, 48, 0, 0) 6 (52, 54, 0, 0)
Machine 4: 16 (9, 15, 0, 0) 7 (20, 22, 0, 0) 20 (27, 31, 0, 0) 38 (38, 42, 0, 0) 11 (43, 46, 0, 0) 22 (48, 52, 0, 0) 17 (55, 58, 0, 0)
Machine 5: 23 (5, 7, 0, 0) 3 (8, 12, 0, 0) 32 (14, 18, 0, 0) 2 (23, 29, 0, 0) 27 (33, 38, 0, 0) 19 (43, 49, 0, 0) 30 (52, 55, 0, 0) 34 (58, 61, 0, 0)

Outbound area

Machine 1: 33 (12, 13, 0, 0) 14 (41, 42, 0, 0) 1 (49, 50, 0, 0) 7 (50, 54, 0, 0) 15 (61, 66, 0, 0) 35 (68, 73, 0, 0) 10 (79, 82, 0, 0)
Machine 2: 41 (34, 35, 0, 0) 2 (46, 49, 0, 0) 18 (49, 52, 0, 0) 28 (52, 55, 0, 0) 40 (62, 66, 0, 0) 31 (69, 71, 0, 0) 22 (77, 81, 0, 0) 37 (95, 100, 0, 0)
Machine 3: 8 (38, 40, 0, 0) 4 (50, 51, 0, 0) 13 (53, 56, 0, 0) 34 (63, 66, 0, 0) 5 (67, 69, 0, 0) 12 (70, 71, 0, 0) 30 (72, 74, 0, 0) 43 (82, 86, 0, 0)
Machine 4: 42 (27, 29, 0, 0) 32 (45, 46, 0, 0) 19 (51, 56, 0, 0) 36 (65, 68, 0, 0) 21 (69, 71, 0, 0) 27 (73, 75, 0, 0) 38 (92, 97, 0, 0)
Machine 5: 3 (28, 30, 0, 0) 25 (43, 46, 0, 0) 11 (47, 52, 0, 0) 6 (56, 59, 0, 0) 26 (66, 71, 0, 0) 17 (77, 78, 0, 0)
Machine 6: 23 (35, 36, 0, 0) 20 (48, 50, 0, 0) 29 (52, 54, 0, 0) 16 (60, 62, 0, 0) 24 (66, 68, 0, 0) 39 (71, 72, 0, 0) 9 (87, 90, 0, 0)

Total penalty for crossdock: 4

Instance 12:

RG algorithm

Inbound area

Machine 1: 39 (6, 9, 0, 0) 33 (11, 16, 0, 0) 29 (19, 23, 0, 0) 45 (23, 29, 0, 0) 41 (32, 35, 0, 0) 32 (38, 44, 0, 0) 9 (48, 51, 0, 0) 7 (51, 54, 0, 0) 26 (54, 58, 1, 0) 25 (58, 63, 0, 0) 54 (64, 68, 0, 0)
Machine 2: 6 (6, 11, 0, 0) 44 (12, 17, 0, 0) 43 (18, 24, 0, 0) 49 (25, 27, 0, 0) 48 (31, 35, 0, 0) 8 (35, 40, 0, 0) 2 (46, 51, 2, 0) 3 (51, 57, 2, 0) 35 (57, 62, 1, 0) 52 (62, 67, 0, 0)
Machine 3: 11 (2, 8, 0, 0) 42 (11, 15, 0, 0) 56 (15, 20, 1, 0) 22 (20, 23, 0, 1) 14 (23, 29, 0, 3) 20 (33, 37, 0, 0) 36 (46, 49, 0, 0) 37 (49, 55, 0, 0) 15 (56, 61, 1, 0) 19 (61, 64, 0, 2)
Machine 4: 17 (4, 6, 0, 0) 55 (8, 12, 0, 0) 10 (13, 15, 0, 0) 31 (15, 20, 2, 0) 13 (20, 24, 0, 0) 23 (25, 30, 0, 0) 30 (33, 38, 0, 0) 46 (42, 48, 0, 0) 4 (50, 55, 0, 0) 51 (55, 61, 1, 0) 1 (61, 63, 0, 2) 38 (63, 66, 0, 3)
Machine 5: 53 (3, 8, 0, 0) 21 (9, 13, 0, 0) 34 (13, 15, 1, 0) 16 (15, 21, 3, 0) 50 (21, 27, 0, 1) 28 (31, 35, 0, 0) 24 (38, 43, 0, 0) 47 (46, 50, 2, 0) 27 (50, 52, 2, 0) 12 (52, 54, 2, 0) 40 (54, 59, 3, 0) 18 (59, 61, 0, 0) 5 (61, 67, 0, 1)

Outbound area

Machine 1: 43 (30, 31, 0, 0) 54 (59, 62, 1, 0) 24 (62, 67, 3, 0) 23 (67, 69, 2, 0) 20 (69, 73, 1, 0) 46 (73, 77, 0, 0) 47 (78, 83, 0, 0) 1 (90, 92, 0, 0) 34 (96, 98, 0, 0)
Machine 2: 9 (37, 40, 0, 0) 12 (61, 62, 0, 0) 26 (65, 66, 0, 0) 5 (67, 72, 0, 1) 35 (73, 78, 0, 0) 16 (79, 84, 0, 0) 40 (86, 88, 0, 0) 44 (93, 96, 0, 0)
Machine 3: 14 (13, 14, 0, 0) 21 (54, 55, 0, 0) 27 (61, 65, 1, 0) 49 (65, 67, 4, 0) 25 (67, 70, 2, 0) 36 (70, 74, 2, 0) 22 (74, 78, 0, 0) 37 (78, 82, 0, 0) 53 (85, 90, 0, 0) 17 (95, 100, 0, 0)
Machine 4: 10 (16, 17, 0, 0) 56 (54, 58, 0, 0) 6 (62, 67, 9, 0) 30 (67, 72, 0, 2) 39 (72, 77, 0, 0) 52 (78, 80, 0, 0) 33 (80, 82, 0, 0) 29 (83, 84, 0, 0) 48 (91, 94, 0, 0)
Machine 5: 11 (32, 33, 0, 0) 45 (58, 59, 0, 0) 4 (63, 64, 0, 0) 51 (65, 66, 0, 0) 15 (66, 70, 2, 0) 55 (70, 71, 2, 0) 3 (71, 76, 1, 0) 19 (76, 78, 1, 0) 28 (78, 83, 0, 0) 42 (89, 94, 0, 0) 41 (102, 103, 0, 0)
Machine 6: 32 (47, 48, 0, 0) 57 (58, 62, 3, 0) 18 (62, 67, 5, 0) 38 (67, 70, 4, 0) 13 (70, 74, 3, 0) 2 (74, 76, 1, 0) 31 (76, 80, 0, 0) 8 (80, 84, 0, 0) 50 (85, 90, 0, 0) 7 (98, 103, 0, 0)

Total penalty for crossdock: 1671

RGTS algorithm

Inbound area

Machine 1: 11 (2, 8, 0, 0) 55 (8, 12, 0, 0) 44 (12, 17, 0, 0) 29 (19, 23, 0, 0) 45 (23, 29, 0, 0) 41 (32, 35, 0, 0) 24 (38, 43, 0, 0) 9 (47, 50, 1, 0) 7 (50, 53, 1, 0) 51 (53, 59, 3, 0) 25 (59, 64, 0, 1) 54 (64, 68, 0, 0)
Machine 2: 17 (4, 6, 0, 0) 6 (6, 11, 0, 0) 33 (11, 16, 0, 0) 43 (18, 24, 0, 0) 49 (25, 27, 0, 0) 48 (31, 35, 0, 0) 8 (35, 40, 0, 0) 47 (47, 51, 1, 0) 3 (51, 57, 2, 0) 15 (57, 62, 0, 0) 52 (62, 67, 0, 0)
Machine 3: 42 (10, 14, 1, 0) 31 (14, 19, 3, 0) 22 (19, 22, 0, 0) 50 (22, 28, 0, 2) 20 (33, 37, 0, 0) 46 (42, 48, 0, 0) 37 (49, 55, 0, 0) 35 (55, 60, 3, 0) 5 (60, 66, 0, 0)
Machine 4: 39 (6, 9, 0, 0) 10 (13, 15, 0, 0) 16 (15, 21, 3, 0) 13 (21, 25, 0, 1) 23 (25, 30, 0, 0) 30 (33, 38, 0, 0) 36 (46, 49, 0, 0) 4 (50, 55, 0, 0) 26 (55, 59, 0, 0) 1 (59, 61, 0, 0) 38 (61, 64, 0, 1)
Machine 5: 53 (3, 8, 0, 0) 21 (9, 13, 0, 0) 34 (13, 15, 1, 0) 56 (15, 20, 1, 0) 14 (20, 26, 0, 0) 28 (31, 35, 0, 0) 32 (38, 44, 0, 0) 2 (45, 50, 3, 0) 27 (50, 52, 2, 0) 12 (52, 54, 2, 0) 40 (54, 59, 3, 0) 18 (59, 61, 0, 0) 19 (61, 64, 0, 2)

Outbound area

Machine 1: 12 (61, 62, 0, 0) 24 (64, 69, 1, 0) 20 (69, 73, 1, 0) 46 (73, 77, 0, 0) 47 (78, 83, 0, 0) 40 (86, 88, 0, 0) 42 (89, 94, 0, 0) 7 (98, 103, 0, 0)
Machine 2: 21 (54, 55, 0, 0) 57 (61, 65, 0, 0) 26 (65, 66, 0, 0) 49 (66, 68, 3, 0) 15 (68, 72, 0, 0) 55 (72, 73, 0, 0) 35 (73, 78, 0, 0) 28 (78, 83, 0, 0) 53 (85, 90, 0, 0) 1 (90, 92, 0, 0)
Machine 3: 45 (58, 59, 0, 0) 27 (62, 66, 0, 0) 25 (66, 69, 3, 0) 23 (69, 71, 0, 0) 38 (71, 74, 0, 0) 22 (74, 78, 0, 0) 37 (78, 82, 0, 0) 50 (85, 90, 0, 0) 17 (95, 100, 0, 0)
Machine 4: 9 (37, 40, 0, 0) 6 (61, 66, 10, 0) 30 (66, 71, 0, 1) 39 (71, 76, 1, 0) 31 (76, 80, 0, 0) 33 (80, 82, 0, 0) 29 (83, 84, 0, 0) 48 (91, 94, 0, 0) 34 (96, 98, 0, 0)
Machine 5: 56 (54, 58, 0, 0) 54 (60, 63, 0, 0) 4 (63, 64, 0, 0) 51 (65, 66, 0, 0) 5 (66, 71, 0, 0) 36 (71, 75, 1, 0) 2 (75, 77, 0, 0) 19 (77, 79, 0, 0) 16 (79, 84, 0, 0)
Machine 6: 14 (13, 14, 0, 0) 10 (16, 17, 0, 0) 43 (30, 31, 0, 0) 11 (32, 33, 0, 0) 32 (47, 48, 0, 0) 18 (63, 68, 4, 0) 3 (68, 73, 4, 0) 13 (73, 77, 0, 0) 52 (78, 80, 0, 0) 8 (80, 84, 0, 0) 44 (93, 96, 0, 0) 41 (102, 103, 0, 0)

Total penalty for crossdock: 858

Instance 13:

RG algorithm

Inbound area

Machine 1: 22 (12, 15, 0, 0) 4 (19, 21, 0, 0) 28 (21, 23, 1, 0) 16 (23, 26, 0, 0) 32 (34, 39, 0, 0) 8 (53, 59, 0, 0)
Machine 2: 9 (2, 4, 0, 0) 21 (15, 18, 0, 0) 3 (18, 22, 4, 0) 24 (22, 24, 0, 0) 26 (24, 27, 0, 0) 12 (32, 38, 0, 0) 6 (47, 50, 0, 0)
Machine 3: 1 (13, 16, 0, 0) 29 (20, 26, 0, 0) 5 (31, 33, 0, 0) 7 (42, 47, 0, 0) 27 (57, 60, 0, 0)
Machine 4: 25 (10, 13, 0, 0) 11 (20, 26, 0, 0) 34 (27, 29, 0, 0) 23 (35, 39, 0, 0) 15 (47, 52, 0, 0)
Machine 5: 20 (6, 10, 0, 0) 2 (17, 22, 2, 0) 18 (22, 28, 0, 0) 10 (33, 37, 0, 0) 14 (47, 51, 0, 0) 19 (58, 60, 0, 0)
Machine 6: 33 (13, 19, 0, 0) 30 (20, 26, 0, 0) 17 (27, 29, 0, 0) 13 (36, 42, 0, 0) 31 (55, 57, 0, 0)

Outbound area

Machine 1: 9 (32, 33, 0, 0) 8 (61, 63, 0, 0) 25 (68, 70, 0, 0) 12 (71, 75, 0, 0) 10 (89, 93, 0, 0)
Machine 2: 28 (23, 24, 0, 0) 30 (46, 49, 0, 0) 31 (68, 70, 0, 0) 15 (70, 75, 0, 0) 2 (83, 88, 0, 0) 11 (98, 103, 0, 0)
Machine 3: 19 (39, 40, 0, 0) 23 (61, 66, 0, 0) 4 (70, 72, 0, 0) 14 (78, 83, 0, 0) 20 (90, 95, 0, 0)
Machine 4: 1 (38, 42, 0, 0) 13 (62, 67, 0, 0) 16 (70, 74, 0, 0) 24 (80, 85, 0, 0) 29 (92, 97, 0, 0)
Machine 5: 7 (41, 44, 0, 0) 22 (64, 65, 0, 0) 27 (69, 73, 0, 0) 32 (84, 86, 0, 0)
Machine 6: 17 (9, 10, 0, 0) 3 (42, 43, 0, 0) 26 (62, 67, 0, 0) 5 (70, 71, 0, 0) 18 (74, 75, 0, 0) 21 (79, 83, 0, 0) 6 (94, 95, 0, 0)

Total penalty for crossdock: 7

RGTS algorithm

Inbound area

Machine 1: 22 (12, 15, 0, 0) 4 (19, 21, 0, 0) 28 (21, 23, 1, 0) 16 (23, 26, 0, 0) 32 (34, 39, 0, 0) 8 (53, 59, 0, 0)
Machine 2: 9 (2, 4, 0, 0) 21 (15, 18, 0, 0) 3 (18, 22, 4, 0) 24 (22, 24, 0, 0) 26 (24, 27, 0, 0) 12 (32, 38, 0, 0) 6 (47, 50, 0, 0)
Machine 3: 1 (13, 16, 0, 0) 29 (20, 26, 0, 0) 5 (31, 33, 0, 0) 7 (42, 47, 0, 0) 27 (57, 60, 0, 0)
Machine 4: 25 (10, 13, 0, 0) 11 (20, 26, 0, 0) 34 (27, 29, 0, 0) 23 (35, 39, 0, 0) 15 (47, 52, 0, 0)
Machine 5: 20 (6, 10, 0, 0) 2 (17, 22, 2, 0) 18 (22, 28, 0, 0) 10 (33, 37, 0, 0) 14 (47, 51, 0, 0) 19 (58, 60, 0, 0)
Machine 6: 33 (13, 19, 0, 0) 30 (20, 26, 0, 0) 17 (27, 29, 0, 0) 13 (36, 42, 0, 0) 31 (55, 57, 0, 0)

Outbound area

Machine 1: 9 (32, 33, 0, 0) 8 (61, 63, 0, 0) 25 (68, 70, 0, 0) 12 (71, 75, 0, 0) 10 (89, 93, 0, 0)
Machine 2: 28 (23, 24, 0, 0) 30 (46, 49, 0, 0) 31 (68, 70, 0, 0) 15 (70, 75, 0, 0) 2 (83, 88, 0, 0) 11 (98, 103, 0, 0)
Machine 3: 19 (39, 40, 0, 0) 23 (61, 66, 0, 0) 4 (70, 72, 0, 0) 14 (78, 83, 0, 0) 20 (90, 95, 0, 0)
Machine 4: 1 (38, 42, 0, 0) 13 (62, 67, 0, 0) 16 (70, 74, 0, 0) 24 (80, 85, 0, 0) 29 (92, 97, 0, 0)
Machine 5: 7 (41, 44, 0, 0) 22 (64, 65, 0, 0) 27 (69, 73, 0, 0) 32 (84, 86, 0, 0)
Machine 6: 17 (9, 10, 0, 0) 3 (42, 43, 0, 0) 26 (62, 67, 0, 0) 5 (70, 71, 0, 0) 18 (74, 75, 0, 0) 21 (79, 83, 0, 0) 6 (94, 95, 0, 0)

Total penalty for crossdock: 7

Instance 14:

RG algorithm

Inbound area

Machine 1: 12 (14, 16, 0, 0) 23 (18, 21, 0, 0) 22 (21, 23, 3, 0) 30 (30, 36, 0, 0) 39 (41, 45, 0, 0) 44 (52, 56, 0, 0) 25 (62, 64, 0, 0)
Machine 2: 15 (16, 19, 0, 0) 1 (21, 27, 0, 0) 40 (30, 36, 0, 0) 18 (39, 45, 0, 0) 37 (52, 54, 0, 0) 19 (59, 64, 0, 0) 4 (67, 71, 0, 0)
Machine 3: 6 (9, 15, 0, 0) 14 (18, 23, 0, 0) 2 (25, 29, 0, 0) 26 (31, 33, 0, 0) 34 (35, 38, 0, 0) 41 (45, 49, 0, 0) 42 (52, 55, 3, 0) 45 (63, 69, 0, 0)
Machine 4: 49 (10, 14, 0, 0) 8 (18, 23, 0, 0) 29 (26, 32, 0, 0) 21 (34, 39, 0, 0) 24 (45, 49, 0, 0) 31 (58, 60, 0, 0) 7 (65, 70, 0, 0)
Machine 5: 28 (6, 8, 0, 0) 33 (16, 19, 0, 0) 5 (23, 29, 0, 0) 13 (31, 37, 0, 0) 47 (43, 47, 0, 0) 20 (54, 59, 0, 0) 9 (64, 69, 0, 0)
Machine 6: 11 (13, 17, 0, 0) 43 (17, 22, 2, 0) 16 (29, 34, 0, 0) 35 (37, 39, 0, 0) 46 (50, 53, 0, 0) 32 (58, 64, 0, 0) 17 (69, 75, 0, 0)
Machine 7: 50 (15, 21, 0, 0) 48 (23, 25, 0, 0) 3 (30, 32, 0, 0) 10 (32, 36, 0, 0) 36 (43, 47, 0, 0) 27 (55, 59, 0, 0) 38 (63, 68, 1, 0)

Outbound area

Machine 1: 35 (28, 29, 0, 0) 8 (47, 50, 0, 0) 6 (55, 60, 0, 0) 60 (69, 72, 0, 0) 38 (76, 81, 0, 0) 40 (84, 86, 0, 0) 39 (94, 99, 0, 0)
Machine 2: 55 (34, 37, 0, 0) 45 (51, 52, 0, 0) 18 (57, 59, 0, 0) 31 (68, 73, 0, 0) 52 (75, 78, 1, 0) 27 (78, 79, 0, 0) 15 (79, 84, 0, 0) 56 (88, 90, 0, 0) 14 (97, 101, 0, 0)
Machine 3: 28 (29, 30, 0, 0) 51 (49, 51, 0, 0) 1 (55, 56, 0, 0) 3 (65, 69, 0, 0) 9 (75, 80, 0, 0) 57 (81, 84, 0, 0) 24 (88, 93, 0, 0)
Machine 4: 46 (23, 24, 0, 0) 22 (46, 48, 0, 0) 19 (54, 56, 0, 0) 41 (61, 66, 0, 0) 29 (72, 77, 0, 0) 36 (77, 78, 0, 0) 59 (78, 82, 0, 0) 33 (88, 90, 0, 0) 49 (97, 101, 0, 0)
Machine 5: 48 (35, 37, 0, 0) 54 (50, 53, 0, 0) 30 (61, 66, 0, 0) 20 (75, 79, 0, 0) 26 (80, 82, 0, 0) 4 (85, 87, 0, 0) 2 (94, 98, 0, 0)
Machine 6: 34 (41, 42, 0, 0) 43 (53, 55, 0, 0) 11 (61, 63, 0, 0) 13 (69, 73, 0, 0) 17 (77, 80, 0, 0) 47 (80, 85, 0, 0) 16 (89, 94, 0, 0)
Machine 7: 5 (16, 17, 0, 0) 37 (45, 46, 0, 0) 53 (54, 58, 0, 0) 7 (65, 66, 0, 0) 42 (74, 78, 0, 0) 25 (78, 83, 0, 0) 21 (88, 91, 0, 0)
Machine 8: 50 (22, 23, 0, 0) 32 (47, 49, 0, 0) 44 (54, 58, 0, 0) 58 (68, 71, 0, 0) 10 (76, 80, 0, 0) 23 (81, 85, 0, 0) 12 (90, 92, 0, 0)

Total penalty for crossdock: 10

RGTS algorithm

Inbound area

Machine 1: 12 (14, 16, 0, 0) 23 (18, 21, 0, 0) 22 (21, 23, 3, 0) 30 (30, 36, 0, 0) 39 (41, 45, 0, 0) 44 (52, 56, 0, 0) 25 (62, 64, 0, 0)
Machine 2: 15 (16, 19, 0, 0) 1 (21, 27, 0, 0) 40 (30, 36, 0, 0) 18 (39, 45, 0, 0) 37 (52, 54, 0, 0) 19 (59, 64, 0, 0) 4 (67, 71, 0, 0)
Machine 3: 6 (9, 15, 0, 0) 14 (18, 23, 0, 0) 2 (25, 29, 0, 0) 26 (31, 33, 0, 0) 34 (35, 38, 0, 0) 41 (45, 49, 0, 0) 42 (52, 55, 3, 0) 45 (63, 69, 0, 0)
Machine 4: 49 (10, 14, 0, 0) 8 (18, 23, 0, 0) 29 (26, 32, 0, 0) 21 (34, 39, 0, 0) 24 (45, 49, 0, 0) 31 (58, 60, 0, 0) 7 (65, 70, 0, 0)
Machine 5: 28 (6, 8, 0, 0) 33 (16, 19, 0, 0) 5 (23, 29, 0, 0) 13 (31, 37, 0, 0) 47 (43, 47, 0, 0) 20 (54, 59, 0, 0) 9 (64, 69, 0, 0)
Machine 6: 11 (13, 17, 0, 0) 43 (17, 22, 2, 0) 16 (29, 34, 0, 0) 35 (37, 39, 0, 0) 46 (50, 53, 0, 0) 32 (58, 64, 0, 0) 17 (69, 75, 0, 0)
Machine 7: 50 (15, 21, 0, 0) 48 (23, 25, 0, 0) 3 (30, 32, 0, 0) 10 (32, 36, 0, 0) 36 (43, 47, 0, 0) 27 (55, 59, 0, 0) 38 (63, 68, 1, 0)

Outbound area

Machine 1: 35 (28, 29, 0, 0) 8 (47, 50, 0, 0) 6 (55, 60, 0, 0) 60 (69, 72, 0, 0) 38 (76, 81, 0, 0) 40 (84, 86, 0, 0) 39 (94, 99, 0, 0)
Machine 2: 55 (34, 37, 0, 0) 45 (51, 52, 0, 0) 18 (57, 59, 0, 0) 31 (68, 73, 0, 0) 52 (75, 78, 1, 0) 27 (78, 79, 0, 0) 15 (79, 84, 0, 0) 56 (88, 90, 0, 0) 14 (97, 101, 0, 0)
Machine 3: 28 (29, 30, 0, 0) 51 (49, 51, 0, 0) 1 (55, 56, 0, 0) 3 (65, 69, 0, 0) 9 (75, 80, 0, 0) 57 (81, 84, 0, 0) 24 (88, 93, 0, 0)
Machine 4: 46 (23, 24, 0, 0) 22 (46, 48, 0, 0) 19 (54, 56, 0, 0) 41 (61, 66, 0, 0) 29 (72, 77, 0, 0) 36 (77, 78, 0, 0) 59 (78, 82, 0, 0) 33 (88, 90, 0, 0) 49 (97, 101, 0, 0)
Machine 5: 48 (35, 37, 0, 0) 54 (50, 53, 0, 0) 30 (61, 66, 0, 0) 20 (75, 79, 0, 0) 26 (80, 82, 0, 0) 4 (85, 87, 0, 0) 2 (94, 98, 0, 0)

```
Machine 6: 34 (41, 42, 0, 0) 43 (53, 55, 0, 0) 11 (61, 63, 0, 0) 13 (69, 73, 0, 0) 17 (77, 80,
0, 0) 47 (80, 85, 0, 0) 16 (89, 94, 0, 0)
Machine 7: 5 (16, 17, 0, 0) 37 (45, 46, 0, 0) 53 (54, 58, 0, 0) 7 (65, 66, 0, 0) 42 (74, 78, 0,
0) 25 (78, 83, 0, 0) 21 (88, 91, 0, 0)
Machine 8: 50 (22, 23, 0, 0) 32 (47, 49, 0, 0) 44 (54, 58, 0, 0) 58 (68, 71, 0, 0) 10 (76, 80,
0, 0) 23 (81, 85, 0, 0) 12 (90, 92, 0, 0)

Total penalty for crossdock: 10
```

Instance 15:

RG algorithm

Inbound area

Machine 1: 37 (0, 6, 0, 0) 77 (9, 11, 0, 0) 90 (12, 15, 0, 0) 26 (16, 20, 2, 0) 16 (20, 26, 0, 0) 32 (30, 36, 0, 0) 19 (38, 40, 0, 0) 87 (42, 47, 1, 0) 74 (47, 49, 1, 0) 51 (49, 54, 0, 0) 82 (59, 62, 0, 0) 89 (65, 70, 1, 0)
Machine 2: 56 (5, 7, 0, 0) 69 (10, 13, 0, 0) 43 (16, 18, 0, 0) 6 (20, 26, 0, 0) 71 (30, 33, 1, 0) 45 (33, 38, 0, 0) 62 (40, 43, 0, 0) 78 (45, 48, 0, 0) 53 (48, 52, 0, 0) 83 (54, 59, 0, 0) 47 (64, 68, 0, 0)
Machine 3: 13 (8, 12, 0, 0) 4 (12, 14, 0, 0) 44 (17, 22, 0, 0) 18 (23, 27, 0, 0) 25 (31, 34, 1, 0) 81 (34, 40, 0, 0) 84 (43, 49, 0, 0) 22 (49, 53, 0, 0) 65 (58, 64, 0, 0) 63 (70, 73, 0, 0)
Machine 4: 34 (2, 4, 0, 0) 24 (8, 12, 0, 0) 14 (12, 17, 0, 0) 58 (20, 22, 0, 0) 11 (23, 25, 0, 0) 17 (27, 29, 0, 0) 60 (33, 35, 0, 0) 35 (35, 38, 0, 0) 9 (41, 47, 0, 0) 20 (47, 53, 0, 0) 46 (56, 59, 0, 0) 54 (63, 66, 0, 0)
Machine 5: 8 (5, 8, 0, 0) 29 (10, 13, 0, 0) 88 (16, 22, 0, 0) 5 (22, 24, 0, 0) 76 (26, 28, 0, 0) 31 (33, 36, 0, 0) 52 (37, 43, 0, 0) 72 (46, 48, 0, 0) 73 (48, 52, 0, 0) 68 (55, 60, 0, 0) 50 (64, 67, 0, 0)
Machine 6: 70 (4, 6, 0, 0) 28 (9, 13, 0, 0) 55 (14, 19, 0, 0) 57 (20, 22, 0, 0) 41 (23, 26, 0, 0) 33 (32, 35, 0, 0) 61 (35, 41, 0, 0) 15 (44, 46, 0, 0) 2 (47, 51, 0, 0) 12 (51, 57, 0, 0) 75 (61, 66, 0, 0)
Machine 7: 86 (5, 8, 0, 0) 67 (11, 15, 0, 0) 64 (18, 22, 0, 0) 49 (22, 24, 0, 0) 79 (27, 32, 0, 0) 7 (33, 35, 0, 0) 38 (36, 42, 0, 0) 85 (45, 50, 0, 0) 59 (51, 53, 0, 0) 66 (59, 61, 0, 0) 1 (65, 68, 0, 0)
Machine 8: 36 (6, 12, 0, 0) 30 (12, 15, 0, 0) 3 (17, 21, 1, 0) 23 (21, 23, 0, 0) 39 (24, 26, 0, 0) 27 (29, 35, 0, 0) 80 (36, 39, 0, 0) 48 (42, 45, 1, 0) 10 (45, 47, 2, 0) 42 (47, 51, 2, 0) 21 (56, 58, 0, 0) 40 (62, 66, 0, 0)

Outbound area

Machine 1: 31 (37, 38, 0, 0) 37 (50, 52, 0, 0) 53 (55, 57, 0, 0) 43 (59, 64, 0, 0) 14 (65, 69, 0, 0) 57 (70, 75, 0, 0) 81 (78, 80, 0, 0) 55 (81, 83, 0, 0) 56 (85, 89, 0, 0) 78 (90, 94, 0, 0) 35 (96, 100, 0, 0)
Machine 2: 33 (39, 42, 0, 0) 64 (51, 55, 0, 0) 32 (59, 62, 0, 0) 36 (64, 67, 0, 0) 13 (69, 74, 0, 0) 34 (78, 81, 0, 0) 17 (84, 87, 0, 0) 21 (89, 93, 0, 0) 11 (96, 100, 0, 0)
Machine 3: 71 (43, 47, 0, 0) 59 (51, 53, 0, 0) 4 (58, 61, 0, 0) 10 (64, 66, 0, 0) 79 (66, 70, 0, 0) 12 (71, 76, 0, 0) 69 (78, 80, 0, 0) 45 (80, 85, 0, 0) 48 (87, 92, 0, 0) 42 (93, 95, 0, 0) 9 (101, 104, 0, 0)
Machine 4: 76 (47, 51, 0, 0) 7 (54, 57, 0, 0) 28 (61, 66, 0, 0) 16 (68, 72, 0, 0) 61 (76, 80, 0, 0) 39 (80, 84, 0, 0) 24 (86, 90, 0, 0) 23 (91, 96, 0, 0) 75 (105, 110, 0, 0)
Machine 5: 88 (47, 49, 0, 0) 6 (54, 58, 0, 0) 1 (62, 63, 0, 0) 8 (65, 70, 0, 0) 41 (73, 77, 0, 0) 47 (78, 83, 0, 0) 65 (85, 89, 0, 0) 40 (91, 96, 0, 0) 70 (108, 110, 0, 0)
Machine 6: 26 (29, 32, 0, 0) 74 (48, 52, 0, 0) 29 (55, 57, 0, 0) 51 (61, 66, 0, 0) 54 (68, 70, 0, 0) 18 (73, 78, 1, 0) 62 (78, 80, 0, 0) 67 (81, 85, 0, 0) 25 (88, 92, 0, 0) 3 (92, 94, 0, 0) 84 (98, 102, 0, 0)
Machine 7: 22 (46, 48, 0, 0) 2 (54, 59, 0, 0) 73 (63, 66, 0, 0) 60 (67, 72, 0, 0) 72 (76, 80, 0, 0) 85 (82, 85, 0, 0) 80 (88, 93, 0, 0) 82 (96, 98, 0, 0)
Machine 8: 68 (43, 45, 0, 0) 83 (51, 54, 0, 0) 19 (58, 59, 0, 0) 77 (63, 67, 0, 0) 52 (69, 74, 0, 0) 66 (76, 80, 0, 0) 87 (80, 81, 0, 0) 46 (84, 88, 0, 0) 89 (89, 91, 0, 0) 63 (92, 96, 0, 0)
Machine 9: 27 (17, 18, 0, 0) 86 (47, 52, 0, 0) 30 (56, 59, 0, 0) 5 (63, 64, 0, 0) 50 (65, 68, 0, 0) 15 (69, 73, 0, 0) 58 (76, 80, 0, 0) 44 (80, 82, 0, 0) 38 (84, 88, 0, 0) 20 (89, 91, 0, 0) 49 (91, 96, 0, 0)

Total penalty for crossdock: 14

RGTS algorithm

Inbound area

Machine 1: 37 (0, 6, 0, 0) 77 (9, 11, 0, 0) 90 (12, 15, 0, 0) 26 (16, 20, 2, 0) 16 (20, 26, 0, 0) 32 (30, 36, 0, 0) 19 (38, 40, 0, 0) 87 (42, 47, 1, 0) 74 (47, 49, 1, 0) 51 (49, 54, 0, 0) 82 (59, 62, 0, 0) 89 (65, 70, 1, 0)
Machine 2: 56 (5, 7, 0, 0) 69 (10, 13, 0, 0) 43 (16, 18, 0, 0) 6 (20, 26, 0, 0) 71 (30, 33, 1, 0) 45 (33, 38, 0, 0) 62 (40, 43, 0, 0) 78 (45, 48, 0, 0) 53 (48, 52, 0, 0) 83 (54, 59, 0, 0) 47 (64, 68, 0, 0)
Machine 3: 13 (8, 12, 0, 0) 4 (12, 14, 0, 0) 44 (17, 22, 0, 0) 18 (23, 27, 0, 0) 25 (31, 34, 1, 0) 81 (34, 40, 0, 0) 84 (43, 49, 0, 0) 22 (49, 53, 0, 0) 65 (58, 64, 0, 0) 1 (65, 68, 0, 0)
Machine 4: 34 (2, 4, 0, 0) 24 (8, 12, 0, 0) 14 (12, 17, 0, 0) 58 (20, 22, 0, 0) 11 (23, 25, 0, 0) 17 (27, 29, 0, 0) 60 (33, 35, 0, 0) 35 (35, 38, 0, 0) 9 (41, 47, 0, 0) 20 (47, 53, 0, 0) 46 (56, 59, 0, 0) 54 (63, 66, 0, 0)
Machine 5: 70 (4, 6, 0, 0) 29 (10, 13, 0, 0) 88 (16, 22, 0, 0) 5 (22, 24, 0, 0) 76 (26, 28, 0,

0) 31 (33, 36, 0, 0) 52 (37, 43, 0, 0) 48 (43, 46, 0, 0) 72 (46, 48, 0, 0) 73 (48, 52, 0, 0) 50 (64, 67, 0, 0)
Machine 6: 8 (5, 8, 0, 0) 28 (9, 13, 0, 0) 55 (14, 19, 0, 0) 57 (20, 22, 0, 0) 41 (23, 26, 0, 0) 33 (32, 35, 0, 0) 38 (36, 42, 0, 0) 15 (44, 46, 0, 0) 2 (47, 51, 0, 0) 12 (51, 57, 0, 0) 75 (61, 66, 0, 0)
Machine 7: 86 (5, 8, 0, 0) 67 (11, 15, 0, 0) 64 (18, 22, 0, 0) 49 (22, 24, 0, 0) 79 (27, 32, 0, 0) 7 (33, 35, 0, 0) 61 (35, 41, 0, 0) 85 (45, 50, 0, 0) 21 (56, 58, 0, 0) 66 (59, 61, 0, 0) 63 (70, 73, 0, 0)
Machine 8: 36 (6, 12, 0, 0) 30 (12, 15, 0, 0) 3 (17, 21, 1, 0) 23 (21, 23, 0, 0) 39 (24, 26, 0, 0) 27 (29, 35, 0, 0) 80 (36, 39, 0, 0) 10 (45, 47, 2, 0) 42 (47, 51, 2, 0) 59 (51, 53, 0, 0) 68 (55, 60, 0, 0) 40 (62, 66, 0, 0)

Outbound area

Machine 1: 31 (37, 38, 0, 0) 37 (50, 52, 0, 0) 53 (55, 57, 0, 0) 43 (59, 64, 0, 0) 14 (65, 69, 0, 0) 57 (70, 75, 0, 0) 81 (78, 80, 0, 0) 55 (81, 83, 0, 0) 56 (85, 89, 0, 0) 78 (90, 94, 0, 0) 84 (98, 102, 0, 0)
Machine 2: 64 (51, 55, 0, 0) 32 (59, 62, 0, 0) 36 (64, 67, 0, 0) 60 (67, 72, 0, 0) 34 (78, 81, 0, 0) 17 (84, 87, 0, 0) 21 (89, 93, 0, 0) 11 (96, 100, 0, 0)
Machine 3: 71 (43, 47, 0, 0) 88 (47, 49, 0, 0) 4 (58, 61, 0, 0) 10 (64, 66, 0, 0) 79 (66, 70, 0, 0) 12 (71, 76, 0, 0) 69 (78, 80, 0, 0) 45 (80, 85, 0, 0) 48 (87, 92, 0, 0) 42 (93, 95, 0, 0) 9 (101, 104, 0, 0)
Machine 4: 27 (17, 18, 0, 0) 76 (47, 51, 0, 0) 7 (54, 57, 0, 0) 77 (63, 67, 0, 0) 16 (68, 72, 0, 0) 61 (76, 80, 0, 0) 39 (80, 84, 0, 0) 65 (85, 89, 0, 0) 23 (91, 96, 0, 0) 75 (105, 110, 0, 0)
Machine 5: 33 (39, 42, 0, 0) 74 (48, 52, 0, 0) 6 (54, 58, 0, 0) 1 (62, 63, 0, 0) 8 (65, 70, 0, 0) 41 (73, 77, 0, 0) 47 (78, 83, 0, 0) 24 (86, 90, 0, 0) 40 (91, 96, 0, 0) 82 (96, 98, 0, 0)
Machine 6: 26 (29, 32, 0, 0) 59 (51, 53, 0, 0) 29 (55, 57, 0, 0) 51 (61, 66, 0, 0) 54 (68, 70, 0, 0) 18 (73, 78, 1, 0) 62 (78, 80, 0, 0) 67 (81, 85, 0, 0) 25 (88, 92, 0, 0) 3 (92, 94, 0, 0) 35 (96, 100, 0, 0)
Machine 7: 22 (46, 48, 0, 0) 2 (54, 59, 0, 0) 73 (63, 66, 0, 0) 13 (69, 74, 0, 0) 72 (76, 80, 0, 0) 85 (82, 85, 0, 0) 80 (88, 93, 0, 0) 70 (108, 110, 0, 0)
Machine 8: 68 (43, 45, 0, 0) 83 (51, 54, 0, 0) 19 (58, 59, 0, 0) 28 (61, 66, 0, 0) 52 (69, 74, 0, 0) 66 (76, 80, 0, 0) 87 (80, 81, 0, 0) 46 (84, 88, 0, 0) 89 (89, 91, 0, 0) 63 (92, 96, 0, 0)
Machine 9: 86 (47, 52, 0, 0) 30 (56, 59, 0, 0) 5 (63, 64, 0, 0) 50 (65, 68, 0, 0) 15 (69, 73, 0, 0) 58 (76, 80, 0, 0) 44 (80, 82, 0, 0) 38 (84, 88, 0, 0) 20 (89, 91, 0, 0) 49 (91, 96, 0, 0)

Total penalty for crossdock: 13

Instance 16:

RG algorithm

Inbound area

Machine 1: 71 (2, 5, 0, 0) 56 (7, 10, 0, 0) 64 (13, 19, 0, 0) 86 (19, 24, 0, 0) 2 (36, 41, 0, 0) 35 (44, 50, 0, 0) 3 (51, 56, 0, 0) 48 (64, 70, 0, 0) 81 (73, 76, 0, 0)
Machine 2: 53 (3, 5, 0, 0) 77 (6, 8, 0, 0) 33 (13, 16, 0, 0) 10 (17, 20, 0, 0) 72 (24, 28, 0, 0) 32 (37, 42, 0, 0) 28 (45, 49, 1, 0) 16 (49, 55, 0, 0) 62 (57, 61, 0, 0) 73 (65, 67, 0, 0) 61 (67, 69, 3, 0)
Machine 3: 26 (5, 11, 0, 0) 47 (15, 20, 0, 0) 25 (21, 26, 0, 0) 6 (31, 34, 0, 0) 24 (39, 42, 0, 0) 39 (46, 52, 0, 0) 66 (52, 57, 0, 0) 60 (63, 67, 0, 0) 69 (69, 73, 0, 0)
Machine 4: 37 (5, 10, 0, 0) 54 (13, 19, 1, 0) 7 (19, 21, 0, 0) 89 (24, 29, 0, 0) 27 (38, 44, 0, 0) 18 (46, 52, 0, 0) 20 (53, 56, 0, 0) 70 (56, 62, 0, 0) 45 (67, 71, 0, 0) 9 (75, 80, 0, 0)
Machine 5: 83 (5, 8, 0, 0) 63 (9, 15, 0, 0) 38 (17, 23, 0, 0) 22 (34, 37, 0, 0) 42 (42, 44, 0, 0) 91 (46, 52, 0, 0) 51 (52, 58, 0, 0) 8 (64, 66, 0, 0) 57 (69, 71, 0, 0) 13 (75, 81, 0, 0)
Machine 6: 14 (3, 8, 0, 0) 23 (11, 13, 0, 0) 76 (15, 21, 0, 0) 41 (30, 36, 0, 0) 68 (40, 45, 3, 0) 36 (45, 51, 2, 0) 50 (51, 53, 0, 0) 84 (56, 62, 0, 0) 88 (65, 68, 0, 0) 30 (71, 73, 0, 0)
Machine 7: 52 (2, 7, 0, 0) 67 (12, 14, 0, 0) 31 (15, 18, 0, 0) 1 (18, 21, 0, 0) 15 (21, 25, 0, 0) 19 (31, 33, 0, 0) 85 (38, 44, 4, 0) 29 (44, 49, 3, 0) 17 (49, 53, 3, 0) 43 (56, 58, 0, 0) 82 (61, 63, 0, 0) 34 (67, 70, 0, 0) 44 (77, 81, 0, 0)
Machine 8: 79 (5, 7, 0, 0) 65 (8, 11, 0, 0) 75 (15, 17, 0, 0) 58 (17, 21, 2, 0) 87 (30, 33, 0, 0) 55 (42, 44, 0, 0) 74 (45, 47, 0, 0) 46 (48, 50, 0, 0) 11 (50, 54, 0, 0) 80 (56, 59, 1, 0) 5 (67, 70, 0, 0) 90 (72, 74, 0, 0)
Machine 9: 59 (5, 8, 0, 0) 49 (14, 20, 0, 0) 21 (30, 32, 0, 0) 40 (41, 43, 0, 0) 92 (46, 48, 0, 0) 78 (48, 53, 0, 0) 93 (54, 57, 0, 0) 12 (60, 66, 0, 0) 4 (69, 72, 0, 0)

Outbound area

Machine 1: 81 (27, 28, 0, 0) 27 (48, 49, 0, 0) 90 (58, 60, 0, 0) 21 (67, 69, 0, 0) 51 (71, 73, 0, 0) 71 (76, 78, 0, 0) 23 (83, 84, 0, 0) 12 (85, 89, 0, 0) 60 (89, 94, 2, 0) 78 (94, 96, 0, 0) 72 (96, 100, 0, 0) 73 (107, 109, 0, 0)
Machine 2: 25 (45, 46, 0, 0) 43 (58, 62, 0, 0) 28 (69, 74, 0, 0) 30 (77, 78, 0, 0) 44 (80, 85, 0, 0) 40 (88, 91, 0, 0) 85 (92, 97, 0, 0) 20 (108, 111, 0, 0)
Machine 3: 65 (41, 43, 0, 0) 39 (57, 61, 0, 0) 86 (66, 70, 0, 0) 1 (76, 79, 0, 0) 67 (83, 86, 0, 0) 70 (88, 90, 0, 0) 14 (91, 95, 1, 0) 35 (95, 98, 0, 0) 48 (101, 106, 0, 0)
Machine 4: 33 (36, 37, 0, 0) 52 (54, 58, 0, 0) 2 (67, 68, 0, 0) 94 (71, 72, 0, 0) 66 (75, 77, 0, 0) 57 (78, 81, 0, 0) 87 (84, 85, 0, 0) 59 (88, 90, 3, 0) 13 (90, 92, 2, 0) 22 (92, 97, 0, 0) 80 (98, 102, 0, 0) 29 (113, 117, 0, 0)
Machine 5: 6 (37, 39, 0, 0) 18 (53, 54, 0, 0) 76 (66, 70, 0, 0) 56 (72, 77, 0, 0) 91 (78, 82, 0, 0) 8 (85, 90, 0, 0) 3 (90, 95, 1, 0) 47 (95, 98, 0, 0) 55 (98, 99, 0, 0) 64 (103, 108, 0, 0) 19 (114, 117, 0, 0)
Machine 6: 89 (23, 24, 0, 0) 24 (50, 52, 0, 0) 7 (60, 62, 0, 0) 58 (71, 73, 0, 0) 11 (78, 80, 0, 0) 88 (83, 85, 0, 0) 15 (88, 90, 0, 0) 92 (92, 96, 0, 0) 34 (96, 99, 0, 0) 31 (104, 109, 0, 0)
Machine 7: 9 (28, 29, 0, 0) 32 (52, 54, 0, 0) 46 (60, 62, 0, 0) 37 (69, 71, 0, 0) 54 (74, 78, 0, 0) 75 (82, 87, 0, 0) 42 (88, 92, 3, 0) 16 (92, 96, 1, 0) 10 (96, 99, 0, 0) 62 (104, 107, 0, 0)
Machine 8: 77 (30, 31, 0, 0) 41 (50, 51, 0, 0) 50 (59, 62, 0, 0) 69 (69, 71, 0, 0) 68 (73, 76, 0, 0) 4 (78, 83, 0, 0) 84 (86, 90, 0, 0) 63 (90, 95, 1, 0) 79 (95, 98, 0, 0) 36 (100, 104, 0, 0) 82 (108, 111, 0, 0)
Machine 9: 74 (21, 23, 0, 0) 45 (45, 48, 0, 0) 61 (59, 64, 0, 0) 17 (71, 73, 0, 0) 49 (77, 79, 0, 0) 5 (82, 87, 1, 0) 26 (87, 89, 5, 0) 93 (89, 91, 3, 0) 83 (91, 93, 2, 0) 38 (93, 98, 0, 0) 53 (103, 106, 0, 0)

Total penalty for crossdock: 48

RGTS algorithm

Inbound area

Machine 1: 71 (2, 5, 0, 0) 56 (7, 10, 0, 0) 23 (11, 13, 0, 0) 64 (13, 19, 0, 0) 86 (19, 24, 0, 0) 2 (36, 41, 0, 0) 40 (41, 43, 0, 0) 92 (45, 47, 1, 0) 29 (47, 52, 0, 0) 51 (52, 58, 0, 0) 81 (73, 76, 0, 0)
Machine 2: 53 (3, 5, 0, 0) 77 (6, 8, 0, 0) 33 (13, 16, 0, 0) 10 (17, 20, 0, 0) 72 (24, 28, 0, 0) 32 (37, 42, 0, 0) 28 (45, 49, 1, 0) 16 (49, 55, 0, 0) 62 (57, 61, 0, 0) 60 (63, 67, 0, 0)
Machine 3: 26 (5, 11, 0, 0) 47 (15, 20, 0, 0) 89 (24, 29, 0, 0) 6 (31, 34, 0, 0) 22 (34, 37, 0, 0) 27 (38, 44, 0, 0) 39 (46, 52, 0, 0) 66 (52, 57, 0, 0) 73 (65, 67, 0, 0) 61 (67, 69, 3, 0) 69 (69, 73, 0, 0)
Machine 4: 37 (5, 10, 0, 0) 54 (13, 19, 1, 0) 7 (19, 21, 0, 0) 21 (30, 32, 0, 0) 24 (39, 42, 0, 0) 18 (45, 51, 1, 0) 3 (51, 56, 0, 0) 70 (56, 62, 0, 0) 88 (65, 68, 0, 0) 57 (69, 71, 0, 0) 13 (75, 81, 0, 0)

Machine 5: 83 (5, 8, 0, 0) 38 (17, 23, 0, 0) 42 (42, 44, 0, 0) 91 (45, 51, 1, 0) 50 (51, 53, 0, 0) 8 (64, 66, 0, 0) 45 (67, 71, 0, 0) 9 (75, 80, 0, 0)
Machine 6: 52 (2, 7, 0, 0) 63 (9, 15, 0, 0) 76 (15, 21, 0, 0) 41 (30, 36, 0, 0) 68 (42, 47, 1, 0) 36 (47, 53, 0, 0) 20 (53, 56, 0, 0) 84 (56, 62, 0, 0) 48 (64, 70, 0, 0) 30 (71, 73, 0, 0)
Machine 7: 14 (3, 8, 0, 0) 67 (12, 14, 0, 0) 31 (15, 18, 0, 0) 1 (18, 21, 0, 0) 15 (21, 25, 0, 0) 19 (31, 33, 0, 0) 35 (43, 49, 1, 0) 17 (49, 53, 3, 0) 93 (53, 56, 1, 0) 43 (56, 58, 0, 0) 82 (61, 63, 0, 0) 34 (67, 70, 0, 0) 44 (77, 81, 0, 0)
Machine 8: 79 (5, 7, 0, 0) 65 (8, 11, 0, 0) 75 (15, 17, 0, 0) 58 (17, 21, 2, 0) 87 (30, 33, 0, 0) 55 (42, 44, 0, 0) 74 (45, 47, 0, 0) 46 (48, 50, 0, 0) 11 (50, 54, 0, 0) 80 (56, 59, 1, 0) 5 (67, 70, 0, 0) 90 (72, 74, 0, 0)
Machine 9: 59 (5, 8, 0, 0) 49 (14, 20, 0, 0) 25 (21, 26, 0, 0) 85 (42, 48, 0, 0) 78 (48, 53, 0, 0) 12 (60, 66, 0, 0) 4 (69, 72, 0, 0)

Outbound area

Machine 1: 90 (58, 60, 0, 0) 21 (67, 69, 0, 0) 94 (71, 72, 0, 0) 71 (76, 78, 0, 0) 4 (78, 83, 0, 0) 23 (83, 84, 0, 0) 42 (88, 92, 3, 0) 13 (92, 94, 0, 0) 78 (94, 96, 0, 0) 72 (96, 100, 0, 0) 53 (103, 106, 0, 0)
Machine 2: 89 (23, 24, 0, 0) 25 (45, 46, 0, 0) 41 (50, 51, 0, 0) 28 (69, 74, 0, 0) 66 (75, 77, 0, 0) 30 (77, 78, 0, 0) 44 (80, 85, 0, 0) 40 (88, 91, 0, 0) 85 (91, 96, 1, 0) 10 (96, 99, 0, 0)
Machine 3: 39 (57, 61, 0, 0) 1 (76, 79, 0, 0) 67 (83, 86, 0, 0) 84 (86, 90, 0, 0) 14 (91, 95, 1, 0) 35 (95, 98, 0, 0) 48 (101, 106, 0, 0)
Machine 4: 52 (54, 58, 0, 0) 43 (58, 62, 0, 0) 2 (67, 68, 0, 0) 51 (71, 73, 0, 0) 57 (78, 81, 0, 0) 15 (88, 90, 0, 0) 59 (90, 92, 1, 0) 22 (92, 97, 0, 0) 80 (98, 102, 0, 0) 73 (107, 109, 0, 0) 29 (113, 117, 0, 0)
Machine 5: 74 (21, 23, 0, 0) 45 (45, 48, 0, 0) 24 (50, 52, 0, 0) 18 (53, 54, 0, 0) 86 (66, 70, 0, 0) 56 (72, 77, 0, 0) 91 (78, 82, 0, 0) 12 (85, 89, 0, 0) 3 (90, 95, 1, 0) 47 (95, 98, 0, 0) 55 (98, 99, 0, 0) 64 (103, 108, 0, 0) 82 (108, 111, 0, 0) 19 (114, 117, 0, 0)
Machine 6: 81 (27, 28, 0, 0) 9 (28, 29, 0, 0) 7 (60, 62, 0, 0) 68 (73, 76, 0, 0) 88 (83, 85, 0, 0) 8 (85, 90, 0, 0) 93 (90, 92, 2, 0) 92 (92, 96, 0, 0) 34 (96, 99, 0, 0) 62 (104, 107, 0, 0)
Machine 7: 33 (36, 37, 0, 0) 6 (37, 39, 0, 0) 32 (52, 54, 0, 0) 61 (59, 64, 0, 0) 37 (69, 71, 0, 0) 17 (71, 73, 0, 0) 54 (74, 78, 0, 0) 11 (78, 80, 0, 0) 87 (84, 85, 0, 0) 60 (88, 93, 3, 0) 16 (93, 97, 0, 0) 31 (104, 109, 0, 0)
Machine 8: 77 (30, 31, 0, 0) 50 (59, 62, 0, 0) 69 (69, 71, 0, 0) 75 (82, 87, 0, 0) 63 (88, 93, 3, 0) 83 (93, 95, 0, 0) 79 (95, 98, 0, 0)
Machine 9: 65 (41, 43, 0, 0) 27 (48, 49, 0, 0) 46 (60, 62, 0, 0) 76 (66, 70, 0, 0) 58 (71, 73, 0, 0) 49 (77, 79, 0, 0) 5 (83, 88, 0, 0) 70 (88, 90, 0, 0) 26 (91, 93, 1, 0) 38 (93, 98, 0, 0) 36 (100, 104, 0, 0) 20 (108, 111, 0, 0)

Total penalty for crossdock: 33

Appendix 11

Results of the RGLS and RGLSTS Algorithms for the first 14 Optimal Workers Allocation for the Crossdocking - JIT Scheduling Problem Instances shown in Appendix 8

The RGLS algorithm solution shown corresponds to its best RGLSTS algorithm solution for each problem instance.

Instance 1

RGLS algorithm
Current center point: (5, 6)
Current center point: (4, 5)
Current center point: (3, 4)
Current center point: (2, 3)
Current center point: (2, 2)
Current center point: (2, 1)

Inbound area

i **(s_i,c_i,e_i,t_i)**
Machine 1: 6 (1, 3, 0, 0) 4 (4, 6, 0, 0) 9 (6, 9, 1, 0) 8 (9, 15, 1, 0) 2 (15, 17, 0, 1) 7 (28, 34, 0, 0)
Machine 2: 10 (1, 4, 0, 0) 3 (6, 11, 2, 0) 5 (11, 16, 0, 0) 1 (22, 24, 0, 0)

Outbound area

j **(S_j, C_j, E_j, T_j)**
Machine 1: 5 (14, 15, 0, 0) 3 (21, 24, 2, 0) 11 (24, 26, 1, 0) 7 (26, 29, 0, 0) 9 (30, 32, 0, 0) 4 (44, 46, 0, 0) 2 (51, 54, 0, 0) 1 (56, 58, 0, 0) 10 (59, 64, 4, 0) 6 (64, 68, 0, 0) 8 (72, 77, 0, 0)

Total penalty for scheduling: 111

RGLSTS algorithm

Inbound area

Machine 1: 10 (1, 4, 0, 0) 9 (6, 9, 1, 0) 8 (9, 15, 1, 0) 2 (15, 17, 0, 1) 7 (28, 34, 0, 0)
Machine 2: 6 (1, 3, 0, 0) 4 (4, 6, 0, 0) 3 (6, 11, 2, 0) 5 (11, 16, 0, 0) 1 (22, 24, 0, 0)

Outbound area

Machine 1: 5 (14, 15, 0, 0) 3 (21, 24, 2, 0) 11 (24, 26, 1, 0) 7 (26, 29, 0, 0) 9 (30, 32, 0, 0) 4 (44, 46, 0, 0) 2 (51, 54, 0, 0) 1 (56, 58, 0, 0) 10 (59, 64, 4, 0) 6 (64, 68, 0, 0) 8 (72, 77, 0, 0)

Total penalty for scheduling: 111

```
Machines cost: $3000, Scheduling cost: $111, Total cost: $3111
```

Instance 2:

RGLS algorithm

```
Current center point: ( 8, 7 )
Current center point: ( 7, 6 )
Current center point: ( 6, 5 )
Current center point: ( 5, 4 )
Current center point: ( 4, 3 )
Current center point: ( 3, 2 )
Current center point: ( 2, 1 )
```

Inbound area

```
Machine 1: 2 (2, 6, 0, 0) 1 (14, 16, 3, 0) 3 (16, 19, 0, 0) 8 (19, 25, 0, 1) 15 (25, 27, 0, 1)
13 (27, 29, 0, 1) 9 (29, 34, 0, 1) 12 (34, 36, 2, 0) 11 (36, 42, 0, 3)
Machine 2: 7 (5, 11, 0, 0) 5 (14, 20, 2, 0) 10 (20, 26, 0, 1) 4 (26, 31, 0, 2) 6 (33, 35, 0, 0)
14 (35, 41, 0, 2)
```

Outbound area

```
Machine 1: 6 (36, 37, 3, 0) 4 (37, 40, 0, 0) 5 (40, 42, 2, 0) 2 (42, 44, 0, 0) 10 (44, 49, 0,
6) 12 (49, 50, 4, 0) 11 (50, 55, 0, 0) 9 (55, 56, 2, 0) 7 (56, 61, 4, 0) 3 (61, 65, 4, 0) 8
(65, 69, 0, 0) 14 (69, 74, 7, 0) 1 (74, 77, 3, 0) 13 (77, 82, 0, 1)

Total penalty for scheduling: 1936
```

RGLSTS algorithm

Inbound area

```
Machine 1: 2 (2, 6, 0, 0) 1 (14, 16, 3, 0) 3 (16, 19, 0, 0) 8 (19, 25, 0, 1) 15 (25, 27, 0, 1)
4 (27, 32, 0, 3) 6 (33, 35, 0, 0) 12 (35, 37, 1, 0) 11 (37, 43, 0, 4)
Machine 2: 7 (5, 11, 0, 0) 5 (14, 20, 2, 0) 10 (20, 26, 0, 1) 13 (26, 28, 0, 0) 9 (28, 33, 0,
0) 14 (33, 39, 0, 0)
```

Outbound area

```
Machine 1: 4 (32, 35, 5, 0) 5 (35, 37, 7, 0) 6 (37, 38, 2, 0) 10 (38, 43, 0, 0) 2 (43, 45, 0,
1) 11 (48, 53, 2, 0) 12 (53, 54, 0, 0) 9 (54, 55, 3, 0) 7 (55, 60, 5, 0) 3 (60, 64, 5, 0) 8
(64, 68, 1, 0) 14 (68, 73, 8, 0) 1 (73, 76, 4, 0) 13 (76, 81, 0, 0)

Total penalty for scheduling: 1148

Machines cost: $3000, Scheduling cost: $1148, Total cost: $4148
```

Instance 3:

RGLS algorithm

```
Current center point: ( 10, 11 )
Current center point: ( 9, 10 )
Current center point: ( 8, 9 )
Current center point: ( 7, 8 )
Current center point: ( 6, 7 )
Current center point: ( 5, 6 )
Current center point: ( 4, 5 )
Current center point: ( 3, 4 )
Current center point: ( 3, 3 )
Current center point: ( 3, 2 )
```

Inbound area

```
Machine 1: 17 (9, 11, 1, 0) 4 (11, 15, 0, 0) 19 (15, 21, 0, 1) 3 (24, 26, 0, 0) 14 (30, 36, 0,
0) 7 (38, 41, 0, 0)
Machine 2: 11 (7, 11, 2, 0) 12 (11, 14, 1, 0) 1 (14, 19, 0, 0) 16 (19, 23, 1, 0) 13 (31, 35, 0,
0) 8 (35, 37, 0, 0) 10 (37, 40, 3, 0)
Machine 3: 9 (10, 12, 0, 0) 6 (12, 16, 1, 0) 15 (16, 18, 1, 0) 2 (18, 20, 1, 0) 20 (20, 24, 0,
3) 5 (27, 29, 0, 0) 18 (34, 40, 0, 0)
```

Outbound area

```
Machine 1: 6 (20, 23, 0, 2) 5 (24, 28, 0, 0) 11 (40, 45, 0, 0) 3 (45, 47, 0, 0) 10 (47, 51, 2,
0) 2 (51, 53, 0, 0) 13 (53, 57, 1, 0) 12 (57, 59, 0, 1) 8 (60, 65, 0, 0) 19 (67, 71, 0, 0) 7
(73, 78, 0, 0)
Machine 2: 21 (15, 16, 0, 0) 9 (23, 26, 0, 2) 20 (35, 36, 0, 0) 1 (41, 46, 4, 0) 4 (46, 48, 2,
0) 15 (48, 51, 1, 0) 16 (51, 55, 1, 0) 14 (55, 60, 0, 0) 17 (64, 68, 0, 0) 18 (68, 72, 0, 0)

Total penalty for scheduling: 922
```

RGLSTS algorithm

Inbound area

```
Machine 1: 17 (9, 11, 1, 0) 4 (11, 15, 0, 0) 15 (15, 17, 2, 0) 19 (17, 23, 0, 3) 3 (24, 26, 0,
0) 14 (30, 36, 0, 0) 7 (38, 41, 0, 0)
Machine 2: 11 (7, 11, 2, 0) 12 (11, 14, 1, 0) 1 (14, 19, 0, 0) 16 (19, 23, 1, 0) 13 (31, 35, 0,
0) 8 (35, 37, 0, 0) 10 (37, 40, 3, 0)
Machine 3: 9 (10, 12, 0, 0) 6 (12, 16, 1, 0) 2 (16, 18, 3, 0) 20 (18, 22, 0, 1) 5 (27, 29, 0,
0) 18 (34, 40, 0, 0)
```

Outbound area

```
Machine 1: 6 (18, 21, 0, 0) 5 (24, 28, 0, 0) 11 (40, 45, 0, 0) 15 (45, 48, 4, 0) 4 (48, 50, 0,
0) 2 (50, 52, 1, 0) 13 (52, 56, 2, 0) 12 (56, 58, 0, 0) 8 (60, 65, 0, 0) 19 (67, 71, 0, 0) 7
(73, 78, 0, 0)
Machine 2: 21 (15, 16, 0, 0) 9 (23, 26, 0, 2) 20 (35, 36, 0, 0) 1 (40, 45, 5, 0) 3 (45, 47, 0,
0) 10 (47, 51, 2, 0) 16 (51, 55, 1, 0) 14 (55, 60, 0, 0) 17 (64, 68, 0, 0) 18 (68, 72, 0, 0)

Total penalty for scheduling: 629

Machines cost: $5000, Scheduling cost: $629, Total cost: $5629
```

Instance 4:

RGLS algorithm

```
Current center point: ( 16, 17 )
Current center point: ( 15, 16 )
Current center point: ( 14, 15 )
Current center point: ( 13, 14 )
Current center point: ( 12, 13 )
Current center point: ( 11, 12 )
Current center point: ( 10, 11 )
Current center point: ( 9, 10 )
Current center point: ( 8, 9 )
Current center point: ( 7, 8 )
Current center point: ( 6, 7 )
Current center point: ( 5, 6 )
Current center point: ( 4, 5 )
Current center point: ( 3, 4 )
Current center point: ( 3, 3 )
```

Inbound area

```
Machine 1: 25 (0, 3, 2, 0) 23 (3, 8, 1, 0) 1 (8, 13, 0, 2) 22 (13, 18, 1, 0) 19 (18, 21, 1, 0)
28 (21, 26, 0, 1) 20 (27, 33, 3, 0) 30 (33, 36, 3, 0) 18 (36, 41, 0, 0) 2 (43, 49, 1, 0) 29
(49, 51, 0, 0)
Machine 2: 5 (2, 7, 1, 0) 24 (7, 12, 0, 0) 26 (12, 16, 0, 1) 3 (16, 18, 0, 1) 11 (20, 22, 0, 0)
9 (24, 29, 0, 0) 14 (30, 34, 2, 0) 32 (34, 36, 2, 0) 21 (36, 39, 0, 0) 4 (39, 44, 0, 1) 16 (46,
52, 0, 0)
Machine 3: 15 (1, 5, 0, 0) 13 (5, 7, 0, 1) 17 (7, 13, 0, 0) 27 (13, 15, 0, 0) 7 (15, 18, 0, 1)
12 (19, 25, 0, 0) 6 (31, 36, 0, 0) 10 (36, 42, 0, 0) 8 (45, 49, 0, 0) 31 (49, 53, 0, 1)
```

Outbound area

```
Machine 1: 27 (37, 38, 0, 0) 4 (42, 43, 0, 0) 17 (49, 54, 5, 0) 13 (54, 59, 2, 0) 29 (59, 61,
0, 0) 30 (65, 70, 2, 0) 33 (70, 74, 2, 0) 9 (74, 78, 0, 0) 5 (81, 84, 0, 0)
Machine 2: 11 (22, 24, 0, 0) 16 (40, 42, 1, 0) 32 (42, 46, 1, 0) 24 (46, 49, 3, 0) 7 (49, 51,
2, 0) 20 (51, 52, 2, 0) 19 (52, 54, 4, 0) 23 (54, 59, 2, 0) 22 (59, 64, 2, 0) 14 (64, 66, 2, 0)
10 (66, 71, 0, 0) 1 (76, 81, 0, 0)
Machine 3: 25 (26, 29, 0, 0) 21 (50, 51, 0, 0) 34 (52, 54, 4, 0) 12 (54, 58, 0, 0) 18 (59, 60,
0, 0) 31 (60, 61, 0, 0) 8 (61, 63, 1, 0) 15 (63, 64, 1, 0) 3 (64, 66, 2, 0) 6 (66, 68, 2, 0) 26
(68, 71, 1, 0) 28 (71, 73, 0, 0) 2 (78, 82, 0, 0)

Total penalty for scheduling: 958
```

RGLSTS algorithm

Inbound area

```
Machine 1: 25 (0, 3, 2, 0) 5 (3, 8, 0, 0) 17 (8, 14, 0, 1) 22 (14, 19, 0, 0) 28 (20, 25, 0, 0)
20 (27, 33, 3, 0) 30 (33, 36, 3, 0) 18 (36, 41, 0, 0) 2 (43, 49, 1, 0) 29 (49, 51, 0, 0)
Machine 2: 23 (1, 6, 3, 0) 1 (6, 11, 0, 0) 26 (11, 15, 0, 0) 3 (15, 17, 0, 0) 19 (17, 20, 2, 0)
11 (20, 22, 0, 0) 9 (24, 29, 0, 0) 14 (30, 34, 2, 0) 32 (34, 36, 2, 0) 21 (36, 39, 0, 0) 4 (39,
44, 0, 1) 16 (46, 52, 0, 0)
Machine 3: 15 (1, 5, 0, 0) 13 (5, 7, 0, 1) 24 (7, 12, 0, 0) 27 (12, 14, 1, 0) 7 (14, 17, 0, 0)
12 (19, 25, 0, 0) 6 (31, 36, 0, 0) 10 (36, 42, 0, 0) 8 (45, 49, 0, 0) 31 (49, 53, 0, 1)
```

Outbound area

```
Machine 1: 27 (37, 38, 0, 0) 4 (42, 43, 0, 0) 17 (50, 55, 4, 0) 13 (55, 60, 1, 0) 31 (60, 61,
0, 0) 14 (65, 67, 1, 0) 30 (67, 72, 0, 0) 9 (74, 78, 0, 0) 5 (81, 84, 0, 0)
Machine 2: 11 (22, 24, 0, 0) 16 (41, 43, 0, 0) 32 (43, 47, 0, 0) 21 (50, 51, 0, 0) 7 (51, 53,
0, 0) 20 (53, 54, 0, 0) 19 (54, 56, 2, 0) 23 (56, 61, 0, 0) 22 (61, 66, 0, 0) 10 (66, 71, 0, 0)
28 (71, 73, 0, 0) 1 (76, 81, 0, 0)
Machine 3: 25 (26, 29, 0, 0) 24 (49, 52, 0, 0) 34 (52, 54, 4, 0) 12 (54, 58, 0, 0) 18 (58, 59,
1, 0) 29 (59, 61, 0, 0) 8 (62, 64, 0, 0) 15 (64, 65, 0, 0) 3 (65, 67, 1, 0) 6 (67, 69, 1, 0) 26
(69, 72, 0, 0) 33 (72, 76, 0, 0) 2 (78, 82, 0, 0)

Total penalty for scheduling: 434

Machines cost: $6000, Scheduling cost: $434, Total cost: $6434
```

Instance 5:

RGLS algorithm

```
Current center point: ( 15, 15 )
Current center point: ( 14, 14 )
Current center point: ( 13, 13 )
Current center point: ( 12, 12 )
Current center point: ( 11, 11 )
Current center point: ( 10, 10 )
Current center point: ( 9, 9 )
Current center point: ( 8, 8 )
Current center point: ( 7, 7 )
Current center point: ( 6, 6 )
Current center point: ( 5, 5 )
Current center point: ( 4, 4 )
Current center point: ( 4, 3 )
```

Inbound area

```
Machine 1: 18 (10, 16, 0, 0) 17 (19, 24, 0, 0) 21 (26, 32, 1, 0) 15 (32, 37, 0, 0) 26 (40, 46,
0, 0) 14 (46, 52, 2, 0)
Machine 2: 9 (10, 12, 0, 0) 13 (12, 14, 0, 0) 12 (15, 20, 0, 0) 19 (23, 27, 0, 0) 23 (29, 33,
0, 0) 11 (33, 36, 0, 0) 2 (37, 41, 0, 0) 25 (42, 47, 0, 0)
Machine 3: 27 (4, 9, 0, 0) 10 (12, 18, 0, 0) 24 (19, 22, 0, 0) 28 (26, 30, 0, 0) 20 (30, 32, 0,
0) 16 (32, 35, 0, 2) 1 (36, 41, 0, 0) 3 (45, 48, 3, 0)
Machine 4: 8 (10, 14, 0, 0) 29 (15, 17, 0, 0) 6 (18, 20, 0, 0) 5 (20, 22, 0, 0) 22 (23, 25, 0,
0) 7 (27, 33, 0, 0) 30 (34, 39, 0, 0) 4 (41, 47, 0, 0)
```

Outbound area

```
Machine 1: 14 (25, 26, 0, 0) 1 (38, 40, 0, 0) 16 (49, 54, 3, 0) 3 (54, 59, 2, 0) 27 (59, 62, 3,
0) 25 (62, 66, 1, 0) 24 (66, 71, 0, 0) 5 (74, 79, 0, 0) 9 (80, 85, 0, 0)
Machine 2: 7 (32, 33, 0, 0) 29 (44, 48, 7, 0) 21 (48, 52, 0, 0) 8 (52, 55, 0, 1) 13 (55, 58, 0,
0) 11 (58, 61, 0, 0) 4 (61, 62, 0, 0) 17 (62, 67, 0, 0) 26 (69, 72, 0, 0) 2 (75, 80, 0, 0) 18
(89, 94, 0, 0)
Machine 3: 28 (39, 43, 0, 0) 22 (46, 51, 5, 0) 15 (51, 56, 3, 0) 19 (56, 59, 6, 0) 20 (59, 64,
2, 0) 10 (64, 68, 0, 1) 23 (69, 70, 0, 0) 12 (72, 75, 0, 0) 6 (79, 82, 0, 0)

Total penalty for scheduling: 438
```

RGLSTS algorithm

Inbound area

```
Machine 1: 18 (10, 16, 0, 0) 17 (19, 24, 0, 0) 21 (26, 32, 1, 0) 15 (32, 37, 0, 0) 26 (40, 46,
0, 0) 14 (46, 52, 2, 0)
Machine 2: 9 (10, 12, 0, 0) 13 (12, 14, 0, 0) 12 (15, 20, 0, 0) 19 (23, 27, 0, 0) 23 (29, 33,
0, 0) 11 (33, 36, 0, 0) 2 (37, 41, 0, 0) 4 (41, 47, 0, 0)
Machine 3: 27 (4, 9, 0, 0) 10 (12, 18, 0, 0) 24 (19, 22, 0, 0) 28 (26, 30, 0, 0) 20 (30, 32, 0,
0) 16 (32, 35, 0, 2) 1 (36, 41, 0, 0) 3 (45, 48, 3, 0)
Machine 4: 8 (10, 14, 0, 0) 29 (15, 17, 0, 0) 6 (18, 20, 0, 0) 5 (20, 22, 0, 0) 22 (23, 25, 0,
0) 7 (27, 33, 0, 0) 30 (34, 39, 0, 0) 25 (42, 47, 0, 0)
```

Outbound area

```
Machine 1: 16 (44, 49, 8, 0) 15 (49, 54, 5, 0) 20 (54, 59, 7, 0) 27 (59, 62, 3, 0) 25 (62, 66,
1, 0) 24 (66, 71, 0, 0) 6 (79, 82, 0, 0)
Machine 2: 14 (25, 26, 0, 0) 1 (38, 40, 0, 0) 21 (48, 52, 0, 0) 8 (52, 55, 0, 1) 13 (55, 58, 0,
0) 11 (58, 61, 0, 0) 4 (61, 62, 0, 0) 17 (62, 67, 0, 0) 23 (69, 70, 0, 0) 12 (72, 75, 0, 0) 2
(75, 80, 0, 0) 9 (80, 85, 0, 0) 18 (89, 94, 0, 0)
Machine 3: 7 (32, 33, 0, 0) 28 (39, 43, 0, 0) 22 (46, 51, 5, 0) 29 (51, 55, 0, 0) 3 (55, 60, 1,
0) 19 (60, 63, 2, 0) 10 (63, 67, 0, 0) 26 (69, 72, 0, 0) 5 (74, 79, 0, 0)

Total penalty for scheduling: 338

Machines cost: $7000, Scheduling cost: $338, Total cost: $7338
```

Instance 6:

RGLS algorithm

```
Current center point: ( 16, 17 )
Current center point: ( 15, 16 )
Current center point: ( 14, 15 )
Current center point: ( 13, 14 )
Current center point: ( 12, 13 )
Current center point: ( 11, 12 )
Current center point: ( 10, 11 )
Current center point: ( 9, 10 )
Current center point: ( 8, 9 )
Current center point: ( 7, 8 )
Current center point: ( 6, 7 )
Current center point: ( 5, 6 )
Current center point: ( 4, 5 )
Current center point: ( 3, 4 )
Current center point: ( 3, 3 )
```

Inbound area

```
Machine 1: 12 (7, 9, 0, 0) 8 (11, 17, 1, 0) 32 (17, 21, 1, 0) 26 (21, 23, 1, 0) 4 (23, 26, 0,
0) 20 (26, 29, 0, 0) 3 (32, 38, 1, 0) 17 (38, 44, 2, 0) 6 (44, 48, 4, 0) 27 (48, 53, 1, 0) 1
(53, 57, 0, 1)
Machine 2: 24 (7, 11, 0, 0) 5 (11, 13, 0, 0) 13 (14, 18, 2, 0) 9 (18, 22, 0, 0) 30 (23, 28, 0,
0) 23 (29, 31, 0, 0) 29 (36, 38, 0, 0) 25 (40, 42, 0, 0) 11 (42, 48, 0, 0) 18 (48, 51, 0, 0) 7
(51, 55, 0, 1)
Machine 3: 14 (0, 4, 3, 0) 28 (4, 7, 0, 0) 21 (12, 15, 0, 0) 10 (15, 18, 0, 0) 22 (18, 23, 0,
0) 16 (27, 32, 0, 0) 2 (35, 39, 0, 0) 15 (39, 43, 0, 0) 31 (43, 49, 2, 0) 19 (49, 55, 0, 0)
```

Outbound area

```
Machine 1: 27 (25, 26, 0, 0) 10 (27, 29, 0, 0) 29 (46, 48, 0, 0) 33 (49, 54, 2, 0) 6 (54, 58,
0, 0) 4 (60, 65, 6, 0) 13 (65, 69, 4, 0) 25 (69, 71, 4, 0) 5 (71, 76, 0, 0) 19 (81, 86, 0, 0)
Machine 2: 28 (27, 28, 0, 0) 9 (36, 38, 0, 0) 12 (44, 47, 1, 0) 22 (47, 50, 0, 0) 11 (54, 55,
0, 0) 15 (60, 62, 3, 0) 14 (62, 66, 4, 0) 20 (66, 71, 1, 0) 8 (71, 75, 0, 0) 16 (75, 79, 0, 0)
3 (81, 85, 0, 0)
Machine 3: 18 (22, 23, 0, 0) 17 (25, 27, 0, 0) 23 (39, 42, 0, 0) 21 (43, 46, 1, 0) 31 (46, 49,
0, 0) 32 (56, 58, 2, 0) 30 (58, 63, 5, 0) 1 (63, 68, 2, 0) 26 (68, 71, 3, 0) 24 (71, 74, 0, 0)
2 (75, 80, 0, 0) 7 (88, 91, 0, 0)

Total penalty for scheduling: 256
```

RGLSTS algorithm

Inbound area

```
Machine 1: 28 (4, 7, 0, 0) 24 (7, 11, 0, 0) 8 (12, 18, 0, 0) 32 (18, 22, 0, 0) 26 (22, 24, 0,
0) 20 (26, 29, 0, 0) 29 (36, 38, 0, 0) 17 (38, 44, 2, 0) 6 (44, 48, 4, 0) 27 (48, 53, 1, 0) 1
(53, 57, 0, 1)
Machine 2: 5 (11, 13, 0, 0) 13 (14, 18, 2, 0) 22 (18, 23, 0, 0) 30 (23, 28, 0, 0) 23 (29, 31,
0, 0) 2 (35, 39, 0, 0) 25 (40, 42, 0, 0) 11 (42, 48, 0, 0) 18 (48, 51, 0, 0) 7 (51, 55, 0, 1)
Machine 3: 14 (3, 7, 0, 0) 12 (7, 9, 0, 0) 21 (12, 15, 0, 0) 10 (15, 18, 0, 0) 9 (18, 22, 0, 0)
4 (23, 26, 0, 0) 16 (27, 32, 0, 0) 3 (33, 39, 0, 0) 15 (39, 43, 0, 0) 31 (43, 49, 2, 0) 19 (49,
55, 0, 0)
```

Outbound area

```
Machine 1: 17 (25, 27, 0, 0) 23 (39, 42, 0, 0) 21 (43, 46, 1, 0) 29 (46, 48, 0, 0) 33 (51, 56,
0, 0) 1 (59, 64, 6, 0) 14 (64, 68, 2, 0) 26 (68, 71, 3, 0) 5 (71, 76, 0, 0) 7 (88, 91, 0, 0)
Machine 2: 18 (22, 23, 0, 0) 10 (27, 29, 0, 0) 9 (36, 38, 0, 0) 12 (44, 47, 1, 0) 22 (47, 50,
0, 0) 11 (54, 55, 0, 0) 32 (58, 60, 0, 0) 20 (62, 67, 5, 0) 13 (67, 71, 2, 0) 8 (71, 75, 0, 0)
16 (75, 79, 0, 0) 3 (81, 85, 0, 0)
Machine 3: 27 (25, 26, 0, 0) 28 (27, 28, 0, 0) 31 (46, 49, 0, 0) 6 (54, 58, 0, 0) 30 (58, 63,
5, 0) 15 (63, 65, 0, 0) 4 (65, 70, 1, 0) 24 (70, 73, 1, 0) 25 (73, 75, 0, 0) 2 (75, 80, 0, 0)
19 (81, 86, 0, 0)

Total penalty for scheduling: 238

Machines cost: $6000, Scheduling cost: $238, Total cost: $6238
```

Instance 7:

RGLS algorithm

```
Current center point: ( 15, 15 )
Current center point: ( 14, 14 )
Current center point: ( 13, 13 )
Current center point: ( 12, 12 )
Current center point: ( 11, 11 )
Current center point: ( 10, 10 )
Current center point: ( 9, 9 )
Current center point: ( 8, 8 )
Current center point: ( 7, 7 )
Current center point: ( 6, 6 )
Current center point: ( 5, 5 )
Current center point: ( 4, 4 )
Current center point: ( 3, 3 )
Current center point: ( 2, 2 )
```

Inbound area

```
Machine 1: 25 (9, 12, 0, 0) 6 (13, 19, 0, 0) 21 (31, 36, 0, 0) 14 (41, 43, 0, 0) 18 (47, 49, 2,
0) 15 (49, 54, 0, 0) 8 (57, 61, 0, 0) 9 (62, 64, 0, 0) 28 (64, 66, 0, 0) 23 (69, 74, 0, 0) 7
(76, 80, 0, 0) 5 (81, 86, 0, 0) 2 (86, 89, 1, 0) 22 (89, 91, 1, 0) 26 (91, 96, 0, 4)
Machine 2: 29 (8, 10, 0, 0) 20 (17, 19, 0, 0) 13 (32, 35, 1, 0) 17 (38, 41, 0, 0) 11 (47, 50,
2, 0) 27 (50, 52, 0, 0) 1 (59, 62, 0, 0) 19 (63, 65, 0, 0) 4 (65, 71, 0, 0) 30 (71, 73, 0, 0)
16 (75, 78, 0, 0) 10 (81, 83, 0, 0) 12 (84, 88, 0, 0) 24 (88, 92, 0, 0) 3 (92, 96, 0, 1)
```

Outbound area

```
Machine 1: 12 (35, 37, 0, 0) 27 (58, 59, 0, 0) 4 (66, 69, 0, 0) 28 (73, 76, 0, 0) 21 (89, 94,
0, 0) 8 (97, 99, 0, 0) 2 (102, 105, 0, 0) 25 (105, 110, 1, 0) 15 (110, 113, 3, 0) 5 (113, 117,
1, 0) 3 (117, 120, 0, 0) 30 (120, 123, 0, 0) 11 (124, 127, 0, 0) 10 (128, 131, 0, 0)
Machine 2: 6 (46, 47, 0, 0) 14 (59, 62, 0, 0) 18 (71, 73, 0, 0) 1 (85, 86, 0, 0) 13 (87, 91, 3,
0) 22 (91, 92, 1, 0) 19 (92, 97, 1, 0) 29 (97, 101, 0, 0) 16 (101, 106, 1, 0) 17 (106, 107, 5,
0) 26 (107, 112, 5, 0) 20 (112, 115, 4, 0) 23 (115, 118, 2, 0) 9 (118, 122, 0, 0) 24 (123, 125,
1, 0) 7 (125, 130, 0, 0)

Total penalty for scheduling: 535
```

RGLSTS algorithm

Inbound area

```
Machine 1: 25 (9, 12, 0, 0) 6 (13, 19, 0, 0) 21 (31, 36, 0, 0) 14 (41, 43, 0, 0) 18 (47, 49, 2,
0) 15 (49, 54, 0, 0) 8 (57, 61, 0, 0) 9 (62, 64, 0, 0) 28 (64, 66, 0, 0) 23 (69, 74, 0, 0) 7
(76, 80, 0, 0) 5 (81, 86, 0, 0) 2 (86, 89, 1, 0) 22 (89, 91, 1, 0) 26 (91, 96, 0, 4)
Machine 2: 29 (8, 10, 0, 0) 20 (17, 19, 0, 0) 13 (32, 35, 1, 0) 17 (38, 41, 0, 0) 11 (47, 50,
2, 0) 27 (50, 52, 0, 0) 1 (59, 62, 0, 0) 19 (63, 65, 0, 0) 4 (65, 71, 0, 0) 30 (71, 73, 0, 0)
16 (75, 78, 0, 0) 10 (81, 83, 0, 0) 12 (84, 88, 0, 0) 24 (88, 92, 0, 0) 3 (92, 96, 0, 1)
```

Outbound area

```
Machine 1: 12 (35, 37, 0, 0) 27 (58, 59, 0, 0) 4 (66, 69, 0, 0) 28 (73, 76, 0, 0) 21 (89, 94,
0, 0) 8 (95, 97, 2, 0) 29 (97, 101, 0, 0) 2 (102, 105, 0, 0) 25 (105, 110, 1, 0) 5 (110, 114,
4, 0) 20 (114, 117, 2, 0) 3 (117, 120, 0, 0) 30 (120, 123, 0, 0) 11 (124, 127, 0, 0) 10 (128,
131, 0, 0)
Machine 2: 6 (46, 47, 0, 0) 14 (59, 62, 0, 0) 18 (71, 73, 0, 0) 1 (85, 86, 0, 0) 13 (88, 92, 2,
0) 22 (92, 93, 0, 0) 19 (93, 98, 0, 0) 16 (101, 106, 1, 0) 26 (106, 111, 6, 0) 17 (111, 112, 0,
0) 15 (112, 115, 1, 0) 23 (115, 118, 2, 0) 9 (118, 122, 0, 0) 24 (123, 125, 1, 0) 7 (125, 130,
0, 0)

Total penalty for scheduling: 529

Machines cost: $4000, Scheduling cost: $529, Total cost: $4529
```

Instance 8:

RGLS algorithm

```
Current center point: ( 20, 19 )
Current center point: ( 19, 18 )
Current center point: ( 18, 17 )
Current center point: ( 17, 16 )
Current center point: ( 16, 15 )
Current center point: ( 15, 14 )
Current center point: ( 14, 13 )
Current center point: ( 13, 12 )
Current center point: ( 12, 11 )
Current center point: ( 11, 10 )
Current center point: ( 10, 9 )
Current center point: ( 9, 8 )
Current center point: ( 8, 7 )
Current center point: ( 7, 6 )
Current center point: ( 6, 5 )
Current center point: ( 5, 4 )
Current center point: ( 4, 3 )
```

Inbound area

```
Machine 1: 2 (7, 9, 0, 0) 40 (13, 15, 0, 0) 7 (21, 26, 0, 0) 37 (26, 28, 0, 0) 9 (28, 33, 1, 0)
20 (33, 37, 0, 0) 16 (37, 39, 0, 0) 18 (40, 44, 1, 0) 38 (44, 46, 3, 0) 27 (52, 55, 0, 0) 33
(59, 65, 0, 0)
Machine 2: 1 (6, 12, 0, 0) 28 (13, 15, 0, 0) 29 (24, 30, 0, 0) 12 (31, 34, 1, 0) 30 (34, 37, 0,
1) 13 (37, 39, 0, 0) 39 (39, 41, 0, 0) 10 (41, 47, 0, 0) 14 (48, 51, 0, 0) 34 (56, 62, 0, 0)
Machine 3: 35 (5, 7, 0, 0) 6 (10, 16, 0, 0) 32 (21, 25, 0, 0) 36 (25, 30, 0, 0) 17 (31, 36, 2,
0) 5 (36, 38, 1, 0) 3 (38, 42, 1, 0) 25 (42, 46, 2, 0) 21 (49, 53, 0, 0) 19 (57, 63, 0, 0)
Machine 4: 24 (10, 14, 0, 0) 15 (18, 24, 0, 0) 26 (24, 28, 0, 0) 31 (29, 35, 2, 0) 4 (35, 37,
1, 0) 22 (37, 42, 1, 0) 8 (42, 46, 0, 0) 11 (47, 52, 0, 0) 23 (57, 62, 0, 0)
```

Outbound area

```
Machine 1: 27 (36, 37, 0, 0) 24 (44, 45, 0, 0) 21 (52, 55, 1, 0) 4 (55, 59, 6, 0) 14 (59, 60,
5, 0) 19 (60, 64, 4, 0) 34 (64, 68, 2, 0) 33 (68, 73, 3, 0) 8 (73, 77, 2, 0) 29 (77, 78, 4, 0)
25 (78, 83, 1, 0) 9 (83, 85, 0, 0) 23 (86, 89, 0, 0) 1 (90, 95, 0, 0)
Machine 2: 10 (33, 34, 0, 0) 3 (43, 44, 0, 0) 12 (46, 49, 0, 0) 18 (55, 58, 0, 0) 35 (60, 62,
3, 0) 13 (62, 67, 2, 0) 28 (67, 72, 0, 0) 22 (72, 77, 0, 1) 36 (77, 79, 1, 0) 38 (79, 84, 0, 0)
37 (84, 89, 0, 0) 20 (100, 105, 0, 0)
Machine 3: 15 (42, 43, 0, 0) 6 (45, 49, 0, 0) 31 (51, 54, 2, 0) 2 (54, 59, 2, 0) 16 (59, 64, 5,
0) 17 (64, 69, 3, 0) 26 (69, 72, 3, 0) 7 (72, 74, 3, 0) 5 (74, 78, 3, 0) 11 (78, 80, 1, 0) 30
(80, 85, 0, 0) 32 (86, 91, 0, 0)

Total penalty for scheduling: 272
```

RGLSTS algorithm

Inbound area

```
Machine 1: 2 (7, 9, 0, 0) 40 (13, 15, 0, 0) 7 (21, 26, 0, 0) 37 (26, 28, 0, 0) 9 (28, 33, 1, 0)
20 (33, 37, 0, 0) 16 (37, 39, 0, 0) 18 (40, 44, 1, 0) 38 (44, 46, 3, 0) 27 (52, 55, 0, 0) 33
(59, 65, 0, 0)
Machine 2: 1 (6, 12, 0, 0) 28 (13, 15, 0, 0) 15 (18, 24, 0, 0) 29 (24, 30, 0, 0) 12 (32, 35, 0,
0) 4 (35, 37, 1, 0) 13 (37, 39, 0, 0) 39 (39, 41, 0, 0) 10 (41, 47, 0, 0) 14 (48, 51, 0, 0) 34
(56, 62, 0, 0)
Machine 3: 35 (5, 7, 0, 0) 6 (10, 16, 0, 0) 32 (21, 25, 0, 0) 36 (25, 30, 0, 0) 17 (31, 36, 2,
0) 5 (36, 38, 1, 0) 3 (38, 42, 1, 0) 25 (42, 46, 2, 0) 21 (49, 53, 0, 0) 19 (57, 63, 0, 0)
Machine 4: 24 (10, 14, 0, 0) 26 (23, 27, 1, 0) 31 (27, 33, 4, 0) 30 (33, 36, 0, 0) 22 (37, 42,
1, 0) 8 (42, 46, 0, 0) 11 (47, 52, 0, 0) 23 (57, 62, 0, 0)
```

Outbound area

```
Machine 1: 27 (36, 37, 0, 0) 24 (44, 45, 0, 0) 21 (52, 55, 1, 0) 2 (55, 60, 1, 0) 4 (60, 64, 1,
0) 14 (64, 65, 0, 0) 34 (65, 69, 1, 0) 33 (69, 74, 2, 0) 8 (74, 78, 1, 0) 25 (78, 83, 1, 0) 9
(83, 85, 0, 0) 23 (86, 89, 0, 0) 1 (90, 95, 0, 0)
Machine 2: 10 (33, 34, 0, 0) 3 (43, 44, 0, 0) 12 (46, 49, 0, 0) 18 (54, 57, 1, 0) 28 (57, 62,
10, 0) 19 (62, 66, 2, 0) 17 (66, 71, 1, 0) 22 (71, 76, 0, 0) 36 (76, 78, 2, 0) 29 (78, 79, 3,
0) 38 (79, 84, 0, 0) 37 (84, 89, 0, 0) 20 (100, 105, 0, 0)
Machine 3: 15 (42, 43, 0, 0) 6 (45, 49, 0, 0) 31 (53, 56, 0, 0) 16 (57, 62, 7, 0) 35 (62, 64,
1, 0) 13 (64, 69, 0, 0) 26 (69, 72, 3, 0) 7 (72, 74, 3, 0) 5 (74, 78, 3, 0) 11 (78, 80, 1, 0)
```

```
30 (80, 85, 0, 0) 32 (86, 91, 0, 0)
Total penalty for scheduling: 63
Machines cost: $7000, Scheduling cost: $63, Total cost: $7063
```

Instance 9:

RGLS algorithm

```
Current center point: ( 21, 22 )
Current center point: ( 20, 21 )
Current center point: ( 19, 20 )
Current center point: ( 18, 19 )
Current center point: ( 17, 18 )
Current center point: ( 16, 17 )
Current center point: ( 15, 16 )
Current center point: ( 14, 15 )
Current center point: ( 13, 14 )
Current center point: ( 12, 13 )
Current center point: ( 11, 12 )
Current center point: ( 10, 11 )
Current center point: ( 9, 10 )
Current center point: ( 8, 9 )
Current center point: ( 7, 8 )
Current center point: ( 6, 7 )
Current center point: ( 5, 6 )
Current center point: ( 5, 5 )
Current center point: ( 4, 4 )
Current center point: ( 4, 3 )
```

Inbound area

```
Machine 1: 20 (2, 8, 4, 0) 34 (8, 11, 0, 1) 38 (11, 14, 0, 1) 42 (15, 18, 0, 0) 4 (18, 24, 0,
0) 36 (25, 29, 0, 0) 35 (30, 34, 0, 0) 16 (35, 40, 0, 0) 40 (40, 45, 0, 2) 12 (45, 51, 0, 0) 26
(55, 57, 0, 0)
Machine 2: 21 (4, 9, 1, 0) 1 (9, 12, 0, 0) 27 (13, 15, 1, 0) 19 (15, 19, 3, 0) 28 (19, 25, 0,
1) 8 (26, 31, 0, 0) 7 (31, 36, 1, 0) 23 (36, 41, 0, 0) 6 (43, 45, 0, 0) 5 (45, 51, 0, 0) 29
(56, 59, 0, 0)
Machine 3: 31 (4, 9, 2, 0) 18 (9, 12, 1, 0) 13 (12, 17, 0, 0) 37 (17, 23, 0, 0) 32 (25, 28, 0,
0) 10 (28, 33, 0, 0) 25 (35, 39, 1, 0) 2 (39, 45, 0, 3) 14 (45, 48, 0, 1) 3 (50, 55, 0, 0)
Machine 4: 30 (6, 9, 2, 0) 15 (9, 15, 0, 0) 33 (19, 22, 0, 0) 39 (22, 25, 0, 0) 9 (27, 30, 0,
0) 17 (32, 36, 0, 0) 24 (36, 39, 2, 0) 22 (39, 45, 0, 2) 11 (45, 51, 0, 0) 41 (52, 56, 0, 0)
```

Outbound area

```
Machine 1: 26 (31, 32, 0, 0) 28 (36, 37, 0, 0) 25 (39, 44, 0, 0) 37 (45, 48, 1, 0) 10 (48, 50,
5, 0) 22 (50, 53, 5, 0) 32 (53, 57, 2, 0) 24 (57, 59, 3, 0) 11 (59, 60, 3, 0) 27 (60, 63, 3, 0)
19 (63, 64, 3, 0) 6 (64, 69, 0, 0) 3 (72, 74, 0, 0) 36 (74, 78, 0, 0) 15 (84, 86, 0, 0) 42 (89,
93, 0, 0)
Machine 2: 35 (31, 32, 0, 0) 9 (36, 38, 0, 0) 14 (42, 45, 0, 0) 43 (46, 47, 2, 0) 7 (47, 49, 6,
0) 17 (49, 54, 5, 0) 21 (54, 57, 6, 0) 23 (57, 61, 3, 0) 41 (61, 66, 0, 0) 4 (70, 72, 0, 0) 38
(72, 75, 0, 0) 13 (81, 83, 0, 0) 8 (87, 92, 0, 0)
Machine 3: 20 (30, 31, 0, 0) 39 (35, 36, 0, 0) 18 (39, 40, 0, 0) 31 (44, 46, 0, 0) 29 (47, 48,
1, 0) 1 (48, 51, 4, 0) 2 (51, 55, 4, 0) 34 (55, 56, 6, 0) 40 (56, 61, 2, 0) 30 (61, 65, 0, 2)
16 (65, 70, 0, 1) 12 (70, 73, 0, 0) 5 (73, 75, 0, 0) 33 (83, 86, 0, 0)

Total penalty for scheduling: 1482
```

RGLSTS algorithm

Inbound area

```
Machine 1: 31 (3, 8, 3, 0) 30 (8, 11, 0, 0) 38 (11, 14, 0, 1) 42 (15, 18, 0, 0) 28 (18, 24, 0,
0) 36 (25, 29, 0, 0) 35 (30, 34, 0, 0) 16 (35, 40, 0, 0) 40 (40, 45, 0, 2) 12 (45, 51, 0, 0) 26
(55, 57, 0, 0)
Machine 2: 21 (4, 9, 1, 0) 1 (9, 12, 0, 0) 13 (12, 17, 0, 0) 4 (18, 24, 0, 0) 9 (27, 30, 0, 0)
17 (32, 36, 0, 0) 2 (36, 42, 0, 0) 6 (43, 45, 0, 0) 11 (45, 51, 0, 0)
Machine 3: 34 (7, 10, 0, 0) 18 (10, 13, 0, 0) 27 (14, 16, 0, 0) 37 (17, 23, 0, 0) 32 (25, 28,
0, 0) 10 (28, 33, 0, 0) 23 (33, 38, 3, 0) 24 (38, 41, 0, 0) 14 (42, 45, 2, 0) 5 (45, 51, 0, 0)
41 (52, 56, 0, 0)
Machine 4: 20 (3, 9, 3, 0) 15 (9, 15, 0, 0) 19 (15, 19, 3, 0) 33 (19, 22, 0, 0) 39 (22, 25, 0,
0) 8 (26, 31, 0, 0) 7 (31, 36, 1, 0) 25 (36, 40, 0, 0) 22 (40, 46, 0, 3) 3 (50, 55, 0, 0) 29
(56, 59, 0, 0)
```

Outbound area

```
Machine 1: 37 (45, 48, 1, 0) 43 (48, 49, 0, 0) 22 (50, 53, 5, 0) 10 (53, 55, 0, 0) 32 (55, 59,
0, 0) 24 (59, 61, 1, 0) 34 (61, 62, 0, 0) 11 (62, 63, 0, 0) 27 (63, 66, 0, 0) 19 (66, 67, 0, 0)
```

```
4 (70, 72, 0, 0) 3 (72, 74, 0, 0) 36 (74, 78, 0, 0) 15 (84, 86, 0, 0) 8 (87, 92, 0, 0)
Machine 2: 35 (31, 32, 0, 0) 39 (35, 36, 0, 0) 9 (36, 38, 0, 0) 18 (39, 40, 0, 0) 14 (42, 45,
0, 0) 17 (45, 50, 9, 0) 1 (50, 53, 2, 0) 7 (53, 55, 0, 0) 2 (55, 59, 0, 0) 41 (59, 64, 2, 0) 16
(64, 69, 0, 0) 38 (72, 75, 0, 0) 13 (81, 83, 0, 0)
Machine 3: 20 (30, 31, 0, 0) 26 (31, 32, 0, 0) 28 (36, 37, 0, 0) 25 (39, 44, 0, 0) 31 (44, 46,
0, 0) 29 (47, 48, 1, 0) 40 (48, 53, 10, 0) 30 (53, 57, 6, 0) 21 (57, 60, 3, 0) 23 (60, 64, 0,
0) 6 (64, 69, 0, 0) 12 (70, 73, 0, 0) 5 (73, 75, 0, 0) 33 (83, 86, 0, 0) 42 (89, 93, 0, 0)

Total penalty for scheduling: 656

Machines cost: $7000, Scheduling cost: $656, Total cost: $7656
```

Instance 10:

RGLS algorithm

```
Current center point: ( 16, 18 )
Current center point: ( 15, 17 )
Current center point: ( 14, 16 )
Current center point: ( 13, 15 )
Current center point: ( 12, 14 )
Current center point: ( 11, 13 )
Current center point: ( 10, 12 )
Current center point: ( 9, 11 )
Current center point: ( 8, 10 )
Current center point: ( 7, 9 )
Current center point: ( 6, 8 )
Current center point: ( 5, 7 )
Current center point: ( 4, 6 )
Current center point: ( 4, 5 )
Current center point: ( 4, 4 )
Current center point: ( 4, 3 )
```

Inbound area

```
Machine 1: 15 (6, 10, 0, 0) 29 (10, 13, 0, 0) 8 (18, 24, 1, 0) 10 (24, 30, 1, 0) 19 (30, 32, 0,
1) 4 (32, 37, 0, 1) 2 (44, 46, 0, 0) 16 (46, 50, 0, 0) 18 (52, 55, 0, 0)
Machine 2: 6 (3, 8, 0, 0) 14 (8, 13, 0, 0) 3 (17, 22, 3, 0) 7 (22, 25, 1, 0) 31 (25, 27, 2, 0)
20 (27, 33, 0, 0) 24 (41, 47, 1, 0) 26 (49, 51, 0, 0) 11 (52, 57, 0, 0)
Machine 3: 27 (1, 3, 0, 0) 17 (8, 13, 0, 0) 30 (13, 18, 2, 0) 1 (23, 27, 0, 0) 28 (27, 33, 0,
0) 25 (41, 47, 0, 0) 21 (48, 51, 0, 0)
Machine 4: 5 (0, 3, 0, 0) 9 (9, 15, 0, 0) 32 (21, 27, 0, 0) 23 (27, 32, 1, 0) 22 (32, 37, 0, 0)
12 (44, 49, 0, 0) 13 (51, 57, 0, 0)
```

Outbound area

```
Machine 1: 34 (18, 19, 0, 0) 14 (35, 37, 0, 0) 17 (46, 48, 0, 0) 31 (51, 55, 0, 0) 27 (55, 56,
0, 0) 33 (57, 59, 0, 0) 20 (59, 64, 0, 0) 23 (64, 67, 0, 1) 4 (67, 72, 0, 2) 8 (82, 84, 0, 0)
10 (92, 97, 0, 0)
Machine 2: 28 (38, 41, 0, 0) 5 (45, 47, 0, 0) 22 (50, 54, 3, 0) 9 (54, 59, 3, 0) 6 (59, 64, 2,
0) 2 (64, 67, 2, 0) 19 (67, 70, 0, 0) 3 (70, 75, 0, 0) 18 (89, 90, 0, 0) 29 (94, 98, 0, 0)
Machine 3: 24 (25, 26, 0, 0) 21 (39, 40, 0, 0) 1 (47, 52, 0, 0) 11 (52, 53, 1, 0) 26 (53, 55,
1, 0) 25 (55, 57, 3, 0) 7 (57, 59, 2, 0) 13 (59, 60, 1, 0) 12 (60, 62, 0, 1) 16 (62, 66, 1, 0)
35 (66, 70, 0, 1) 15 (70, 75, 0, 2) 30 (87, 91, 0, 0) 32 (96, 100, 0, 0)

Total penalty for scheduling: 931
```

RGLSTS algorithm

Inbound area

```
Machine 1: 15 (6, 10, 0, 0) 29 (10, 13, 0, 0) 30 (13, 18, 2, 0) 8 (18, 24, 1, 0) 23 (24, 29, 4,
0) 19 (29, 31, 0, 0) 22 (32, 37, 0, 0) 2 (44, 46, 0, 0) 16 (46, 50, 0, 0) 13 (51, 57, 0, 0)
Machine 2: 6 (3, 8, 0, 0) 14 (8, 13, 0, 0) 3 (17, 22, 3, 0) 7 (22, 25, 1, 0) 31 (25, 27, 2, 0)
20 (27, 33, 0, 0) 24 (41, 47, 1, 0) 26 (49, 51, 0, 0) 11 (52, 57, 0, 0)
Machine 3: 27 (1, 3, 0, 0) 17 (8, 13, 0, 0) 32 (21, 27, 0, 0) 28 (27, 33, 0, 0) 25 (41, 47, 0,
0) 21 (48, 51, 0, 0)
Machine 4: 5 (0, 3, 0, 0) 9 (9, 15, 0, 0) 1 (21, 25, 2, 0) 10 (25, 31, 0, 0) 4 (31, 36, 0, 0)
12 (44, 49, 0, 0) 18 (52, 55, 0, 0)
```

Outbound area

```
Machine 1: 34 (18, 19, 0, 0) 24 (25, 26, 0, 0) 28 (38, 41, 0, 0) 17 (46, 48, 0, 0) 31 (51, 55,
0, 0) 7 (55, 57, 4, 0) 20 (57, 62, 2, 0) 23 (62, 65, 1, 0) 4 (65, 70, 0, 0) 29 (94, 98, 0, 0)
Machine 2: 14 (35, 37, 0, 0) 21 (39, 40, 0, 0) 5 (45, 47, 0, 0) 9 (47, 52, 10, 0) 11 (52, 53,
1, 0) 22 (53, 57, 0, 0) 12 (57, 59, 2, 0) 6 (59, 64, 2, 0) 2 (64, 67, 2, 0) 19 (67, 70, 0, 0) 3
(70, 75, 0, 0) 18 (89, 90, 0, 0) 10 (92, 97, 0, 0)
Machine 3: 1 (47, 52, 0, 0) 26 (52, 54, 2, 0) 27 (54, 55, 1, 0) 25 (55, 57, 3, 0) 33 (57, 59,
0, 0) 13 (59, 60, 1, 0) 16 (60, 64, 3, 0) 35 (64, 68, 1, 0) 15 (68, 73, 0, 0) 8 (82, 84, 0, 0)
30 (87, 91, 0, 0) 32 (96, 100, 0, 0)

Total penalty for scheduling: 51

Machines cost: $7000, Scheduling cost: $51, Total cost: $7051
```

Instance 11:

RGLS algorithm

```
Current center point: ( 20, 22 )
Current center point: ( 19, 21 )
Current center point: ( 18, 20 )
Current center point: ( 17, 19 )
Current center point: ( 16, 18 )
Current center point: ( 15, 17 )
Current center point: ( 14, 16 )
Current center point: ( 13, 15 )
Current center point: ( 12, 14 )
Current center point: ( 11, 13 )
Current center point: ( 10, 12 )
Current center point: ( 9, 11 )
Current center point: ( 8, 10 )
Current center point: ( 7, 9 )
Current center point: ( 6, 8 )
Current center point: ( 5, 7 )
Current center point: ( 4, 6 )
Current center point: ( 4, 5 )
Current center point: ( 4, 4 )
Current center point: ( 4, 3 )
```

Inbound area

```
Machine 1: 26 (6, 8, 0, 0) 3 (8, 12, 0, 0) 5 (12, 14, 0, 0) 35 (14, 20, 0, 0) 2 (22, 28, 1, 0)
15 (28, 31, 0, 0) 27 (33, 38, 0, 0) 38 (38, 42, 0, 0) 19 (43, 49, 0, 0) 30 (52, 55, 0, 0) 31
(58, 61, 0, 0)
Machine 2: 23 (5, 7, 0, 0) 18 (9, 14, 0, 0) 28 (14, 20, 0, 0) 39 (22, 27, 1, 0) 20 (27, 31, 0,
0) 33 (38, 44, 1, 0) 9 (44, 49, 0, 0) 6 (52, 54, 0, 0) 34 (58, 61, 0, 0)
Machine 3: 8 (6, 10, 1, 0) 37 (10, 13, 0, 0) 24 (13, 16, 0, 2) 40 (20, 23, 0, 0) 13 (26, 31, 0,
0) 14 (36, 42, 0, 0) 1 (44, 48, 0, 0) 21 (49, 52, 0, 0) 17 (55, 58, 0, 0)
Machine 4: 25 (6, 8, 1, 0) 16 (8, 14, 1, 0) 32 (14, 18, 0, 0) 7 (20, 22, 0, 0) 4 (23, 29, 0, 0)
36 (31, 36, 0, 0) 12 (38, 41, 0, 0) 11 (42, 45, 1, 0) 29 (45, 48, 0, 0) 22 (48, 52, 0, 0) 10
(54, 59, 0, 0)
```

Outbound area

```
Machine 1: 42 (27, 29, 0, 0) 8 (38, 40, 0, 0) 25 (43, 46, 0, 0) 4 (46, 47, 4, 0) 11 (47, 52, 0,
0) 29 (52, 54, 0, 0) 13 (54, 57, 0, 1) 36 (58, 61, 7, 0) 15 (61, 66, 0, 0) 35 (68, 73, 0, 0) 17
(77, 78, 0, 0) 43 (82, 86, 0, 0) 37 (95, 100, 0, 0)
Machine 2: 33 (14, 15, 0, 2) 41 (34, 35, 0, 0) 14 (41, 42, 0, 0) 2 (42, 45, 4, 0) 18 (45, 48,
4, 0) 7 (48, 52, 2, 0) 28 (52, 55, 0, 0) 6 (56, 59, 0, 0) 40 (60, 64, 2, 0) 26 (64, 69, 2, 0)
31 (69, 71, 0, 0) 39 (71, 72, 0, 0) 27 (73, 75, 0, 0) 10 (79, 82, 0, 0) 38 (92, 97, 0, 0)
Machine 3: 3 (28, 30, 0, 0) 23 (35, 36, 0, 0) 32 (45, 46, 0, 0) 20 (47, 49, 1, 0) 1 (49, 50, 0,
0) 19 (51, 56, 0, 0) 16 (59, 61, 1, 0) 34 (61, 64, 2, 0) 24 (64, 66, 2, 0) 5 (66, 68, 1, 0) 21
(68, 70, 1, 0) 12 (70, 71, 0, 0) 30 (72, 74, 0, 0) 22 (77, 81, 0, 0) 9 (87, 90, 0, 0)

Total penalty for scheduling: 540
```

RGLSTS algorithm

Inbound area

```
Machine 1: 26 (6, 8, 0, 0) 3 (8, 12, 0, 0) 5 (12, 14, 0, 0) 32 (14, 18, 0, 0) 7 (20, 22, 0, 0)
2 (22, 28, 1, 0) 15 (28, 31, 0, 0) 14 (36, 42, 0, 0) 19 (43, 49, 0, 0) 6 (52, 54, 0, 0) 17 (55,
58, 0, 0) 31 (58, 61, 0, 0)
Machine 2: 23 (5, 7, 0, 0) 25 (7, 9, 0, 0) 18 (9, 14, 0, 0) 39 (22, 27, 1, 0) 20 (27, 31, 0, 0)
27 (33, 38, 0, 0) 33 (38, 44, 1, 0) 9 (44, 49, 0, 0) 21 (49, 52, 0, 0) 34 (58, 61, 0, 0)
Machine 3: 8 (6, 10, 1, 0) 37 (10, 13, 0, 0) 28 (14, 20, 0, 0) 40 (20, 23, 0, 0) 13 (26, 31, 0,
0) 12 (38, 41, 0, 0) 1 (44, 48, 0, 0) 30 (52, 55, 0, 0)
Machine 4: 16 (5, 11, 4, 0) 24 (11, 14, 0, 0) 35 (14, 20, 0, 0) 4 (23, 29, 0, 0) 36 (31, 36, 0,
0) 38 (38, 42, 0, 0) 11 (42, 45, 1, 0) 29 (45, 48, 0, 0) 22 (48, 52, 0, 0) 10 (54, 59, 0, 0)
```

Outbound area

```
Machine 1: 3 (28, 30, 0, 0) 23 (35, 36, 0, 0) 11 (44, 49, 3, 0) 18 (49, 52, 0, 0) 29 (52, 54,
0, 0) 40 (58, 62, 4, 0) 34 (62, 65, 1, 0) 36 (65, 68, 0, 0) 35 (68, 73, 0, 0) 27 (73, 75, 0, 0)
22 (77, 81, 0, 0) 43 (82, 86, 0, 0)
Machine 2: 41 (34, 35, 0, 0) 14 (41, 42, 0, 0) 7 (44, 48, 6, 0) 20 (48, 50, 0, 0) 28 (50, 53,
2, 0) 13 (53, 56, 0, 0) 16 (60, 62, 0, 0) 26 (62, 67, 4, 0) 5 (67, 69, 0, 0) 31 (69, 71, 0, 0)
```

```
39 (71, 72, 0, 0) 9 (87, 90, 0, 0) 37 (95, 100, 0, 0)
Machine 3: 33 (14, 15, 0, 2) 42 (27, 29, 0, 0) 8 (38, 40, 0, 0) 25 (42, 45, 1, 0) 32 (45, 46,
0, 0) 2 (46, 49, 0, 0) 1 (49, 50, 0, 0) 4 (50, 51, 0, 0) 19 (51, 56, 0, 0) 6 (56, 59, 0, 0) 15
(61, 66, 0, 0) 24 (66, 68, 0, 0) 21 (68, 70, 1, 0) 12 (70, 71, 0, 0) 30 (72, 74, 0, 0) 17 (77,
78, 0, 0) 10 (79, 82, 0, 0) 38 (92, 97, 0, 0)

Total penalty for scheduling: 231

Machines cost: $7000, Scheduling cost: $231, Total cost: $7231
```

Instance 12:

RGLS algorithm

```
Current center point: ( 28, 29 )
Current center point: ( 27, 28 )
Current center point: ( 26, 27 )
Current center point: ( 25, 26 )
Current center point: ( 24, 25 )
Current center point: ( 23, 24 )
Current center point: ( 22, 23 )
Current center point: ( 21, 22 )
Current center point: ( 20, 21 )
Current center point: ( 19, 20 )
Current center point: ( 18, 19 )
Current center point: ( 17, 18 )
Current center point: ( 16, 17 )
Current center point: ( 15, 16 )
Current center point: ( 14, 15 )
Current center point: ( 13, 14 )
Current center point: ( 12, 13 )
Current center point: ( 11, 12 )
Current center point: ( 10, 11 )
Current center point: ( 9, 10 )
Current center point: ( 8, 9 )
Current center point: ( 7, 8 )
Current center point: ( 6, 7 )
Current center point: ( 6, 6 )
Current center point: ( 6, 5 )
```

Inbound area

```
Machine 1: 11 (2, 8, 0, 0) 10 (13, 15, 0, 0) 31 (15, 20, 2, 0) 14 (20, 26, 0, 0) 28 (31, 35, 0,
0) 46 (42, 48, 0, 0) 27 (52, 54, 0, 0) 1 (57, 59, 2, 0) 25 (59, 64, 0, 1)
Machine 2: 53 (3, 8, 0, 0) 42 (11, 15, 0, 0) 22 (19, 22, 0, 0) 50 (22, 28, 0, 2) 20 (33, 37, 0,
0) 9 (46, 49, 2, 0) 37 (49, 55, 0, 0) 35 (55, 60, 3, 0) 5 (60, 66, 0, 0)
Machine 3: 17 (4, 6, 0, 0) 21 (9, 13, 0, 0) 56 (15, 20, 1, 0) 13 (20, 24, 0, 0) 23 (25, 30, 0,
0) 8 (35, 40, 0, 0) 47 (48, 52, 0, 0) 12 (54, 56, 0, 0) 51 (56, 62, 0, 0) 52 (62, 67, 0, 0)
Machine 4: 6 (6, 11, 0, 0) 33 (11, 16, 0, 0) 29 (19, 23, 0, 0) 49 (25, 27, 0, 0) 48 (31, 35, 0,
0) 32 (38, 44, 0, 0) 7 (51, 54, 0, 0) 26 (54, 58, 1, 0) 19 (58, 61, 1, 0) 38 (61, 64, 0, 1)
Machine 5: 39 (6, 9, 0, 0) 34 (14, 16, 0, 0) 43 (18, 24, 0, 0) 41 (32, 35, 0, 0) 24 (38, 43, 0,
0) 2 (46, 51, 2, 0) 3 (51, 57, 2, 0) 15 (57, 62, 0, 0) 18 (62, 64, 0, 3)
Machine 6: 55 (8, 12, 0, 0) 44 (12, 17, 0, 0) 16 (17, 23, 1, 0) 45 (23, 29, 0, 0) 30 (33, 38,
0, 0) 36 (46, 49, 0, 0) 4 (50, 55, 0, 0) 40 (57, 62, 0, 0) 54 (64, 68, 0, 0)
```

Outbound area

```
Machine 1: 9 (37, 40, 0, 0) 12 (61, 62, 0, 0) 27 (62, 66, 0, 0) 5 (66, 71, 0, 0) 55 (71, 72, 1,
0) 13 (72, 76, 1, 0) 22 (76, 80, 0, 2) 33 (80, 82, 0, 0) 29 (83, 84, 0, 0) 50 (85, 90, 0, 0) 44
(93, 96, 0, 0)
Machine 2: 43 (30, 31, 0, 0) 56 (52, 56, 2, 0) 57 (56, 60, 5, 0) 24 (60, 65, 5, 0) 18 (65, 70,
2, 0) 6 (70, 75, 1, 0) 2 (75, 77, 0, 0) 19 (77, 79, 0, 0) 31 (79, 83, 0, 3) 40 (86, 88, 0, 0)
42 (89, 94, 0, 0) 41 (102, 103, 0, 0)
Machine 3: 14 (13, 14, 0, 0) 32 (47, 48, 0, 0) 54 (58, 61, 2, 0) 26 (61, 62, 4, 0) 49 (62, 64,
7, 0) 15 (64, 68, 4, 0) 38 (68, 71, 3, 0) 39 (71, 76, 1, 0) 52 (76, 78, 2, 0) 28 (78, 83, 0, 0)
8 (83, 87, 0, 3) 48 (91, 94, 0, 0) 7 (98, 103, 0, 0)
Machine 4: 10 (16, 17, 0, 0) 21 (54, 55, 0, 0) 4 (63, 64, 0, 0) 30 (67, 72, 0, 2) 36 (72, 76,
0, 0) 46 (76, 80, 0, 3) 47 (80, 85, 0, 2) 1 (90, 92, 0, 0) 17 (95, 100, 0, 0)
Machine 5: 11 (32, 33, 0, 0) 45 (55, 56, 3, 0) 51 (56, 57, 9, 0) 23 (57, 59, 12, 0) 25 (59, 62,
10, 0) 20 (62, 66, 8, 0) 3 (66, 71, 6, 0) 35 (71, 76, 2, 0) 37 (76, 80, 2, 0) 16 (80, 85, 0, 1)
53 (85, 90, 0, 0) 34 (96, 98, 0, 0)

Total penalty for scheduling: 2409
```

RGLSTS algorithm

Inbound area

```
Machine 1: 11 (2, 8, 0, 0) 10 (13, 15, 0, 0) 31 (15, 20, 2, 0) 14 (20, 26, 0, 0) 28 (31, 35, 0,
0) 46 (42, 48, 0, 0) 47 (48, 52, 0, 0) 27 (52, 54, 0, 0) 26 (54, 58, 1, 0) 25 (58, 63, 0, 0)
Machine 2: 53 (3, 8, 0, 0) 42 (11, 15, 0, 0) 16 (15, 21, 3, 0) 50 (21, 27, 0, 1) 20 (33, 37, 0,
0) 32 (38, 44, 0, 0) 37 (49, 55, 0, 0) 35 (55, 60, 3, 0) 5 (60, 66, 0, 0)
Machine 3: 17 (4, 6, 0, 0) 21 (9, 13, 0, 0) 56 (15, 20, 1, 0) 13 (20, 24, 0, 0) 23 (25, 30, 0,
```

0) 8 (35, 40, 0, 0) 2 (48, 53, 0, 0) 51 (53, 59, 3, 0) 19 (59, 62, 0, 0) 52 (62, 67, 0, 0)
Machine 4: 6 (6, 11, 0, 0) 33 (11, 16, 0, 0) 29 (19, 23, 0, 0) 49 (25, 27, 0, 0) 48 (31, 35, 0, 0) 9 (48, 51, 0, 0) 7 (51, 54, 0, 0) 12 (54, 56, 0, 0) 1 (57, 59, 2, 0) 18 (59, 61, 0, 0)
Machine 5: 39 (6, 9, 0, 0) 34 (14, 16, 0, 0) 43 (18, 24, 0, 0) 41 (32, 35, 0, 0) 24 (38, 43, 0, 0) 3 (51, 57, 2, 0) 15 (57, 62, 0, 0) 54 (64, 68, 0, 0)
Machine 6: 55 (8, 12, 0, 0) 44 (12, 17, 0, 0) 22 (19, 22, 0, 0) 45 (23, 29, 0, 0) 30 (33, 38, 0, 0) 36 (46, 49, 0, 0) 4 (50, 55, 0, 0) 40 (55, 60, 2, 0) 38 (60, 63, 0, 0)

Outbound area

Machine 1: 9 (37, 40, 0, 0) 27 (60, 64, 2, 0) 26 (64, 65, 1, 0) 51 (65, 66, 0, 0) 5 (66, 71, 0, 0) 55 (71, 72, 1, 0) 13 (72, 76, 1, 0) 31 (76, 80, 0, 0) 33 (80, 82, 0, 0) 29 (83, 84, 0, 0) 50 (85, 90, 0, 0) 44 (93, 96, 0, 0)
Machine 2: 43 (30, 31, 0, 0) 57 (58, 62, 3, 0) 4 (62, 63, 1, 0) 6 (63, 68, 8, 0) 23 (68, 70, 1, 0) 46 (70, 74, 3, 0) 22 (74, 78, 0, 0) 28 (78, 83, 0, 0) 40 (86, 88, 0, 0) 42 (89, 94, 0, 0) 41 (102, 103, 0, 0)
Machine 3: 14 (13, 14, 0, 0) 32 (47, 48, 0, 0) 54 (58, 61, 2, 0) 12 (61, 62, 0, 0) 15 (62, 66, 6, 0) 38 (66, 69, 5, 0) 49 (69, 71, 0, 0) 39 (71, 76, 1, 0) 19 (76, 78, 1, 0) 52 (78, 80, 0, 0) 8 (80, 84, 0, 0) 48 (91, 94, 0, 0) 7 (98, 103, 0, 0)
Machine 4: 10 (16, 17, 0, 0) 21 (54, 55, 0, 0) 18 (56, 61, 11, 0) 24 (61, 66, 4, 0) 30 (66, 71, 0, 1) 2 (71, 73, 4, 0) 35 (73, 78, 0, 0) 47 (78, 83, 0, 0) 1 (90, 92, 0, 0) 17 (95, 100, 0, 0)
Machine 5: 11 (32, 33, 0, 0) 56 (54, 58, 0, 0) 45 (58, 59, 0, 0) 3 (59, 64, 13, 0) 20 (64, 68, 6, 0) 25 (68, 71, 1, 0) 36 (71, 75, 1, 0) 37 (75, 79, 3, 0) 16 (79, 84, 0, 0) 53 (85, 90, 0, 0) 34 (96, 98, 0, 0)

Total penalty for scheduling: 298

Machines cost: $11000, Scheduling cost: $298, Total cost: $11298

Instance 13:

RGLS algorithm

```
Current center point: ( 17, 16 )
Current center point: ( 16, 15 )
Current center point: ( 15, 14 )
Current center point: ( 14, 13 )
Current center point: ( 13, 12 )
Current center point: ( 12, 11 )
Current center point: ( 11, 10 )
Current center point: ( 10, 9 )
Current center point: ( 9, 8 )
Current center point: ( 8, 7 )
Current center point: ( 7, 6 )
Current center point: ( 6, 5 )
Current center point: ( 5, 4 )
Current center point: ( 4, 3 )
Current center point: ( 4, 2 )
```

Inbound area

```
Machine 1: 9 (2, 4, 0, 0) 1 (12, 15, 1, 0) 4 (15, 17, 4, 0) 2 (17, 22, 2, 0) 28 (22, 24, 0, 0)
18 (24, 30, 0, 2) 10 (32, 36, 1, 0) 13 (36, 42, 0, 0) 14 (47, 51, 0, 0) 31 (55, 57, 0, 0)
Machine 2: 22 (12, 15, 0, 0) 3 (18, 22, 4, 0) 11 (22, 28, 0, 2) 5 (31, 33, 0, 0) 23 (35, 39, 0,
0) 6 (47, 50, 0, 0) 19 (58, 60, 0, 0)
Machine 3: 25 (10, 13, 0, 0) 21 (15, 18, 0, 0) 29 (18, 24, 2, 0) 26 (24, 27, 0, 0) 34 (27, 29,
0, 0) 32 (34, 39, 0, 0) 15 (47, 52, 0, 0) 27 (57, 60, 0, 0)
Machine 4: 20 (6, 10, 0, 0) 33 (11, 17, 2, 0) 30 (17, 23, 3, 0) 24 (23, 25, 0, 1) 16 (25, 28,
0, 2) 17 (28, 30, 0, 1) 12 (32, 38, 0, 0) 7 (42, 47, 0, 0) 8 (53, 59, 0, 0)
```

Outbound area

```
Machine 1: 28 (23, 24, 0, 0) 9 (32, 33, 0, 0) 19 (37, 38, 2, 0) 1 (38, 42, 0, 0) 3 (42, 43, 0,
0) 30 (46, 49, 0, 0) 22 (50, 51, 14, 0) 26 (51, 56, 11, 0) 25 (56, 58, 12, 0) 27 (58, 62, 11,
0) 13 (62, 67, 0, 0) 4 (67, 69, 3, 0) 5 (69, 70, 1, 0) 15 (70, 75, 0, 0) 14 (78, 83, 0, 0) 32
(84, 86, 0, 0) 10 (88, 92, 1, 0) 29 (92, 97, 0, 0)
Machine 2: 17 (9, 10, 0, 0) 7 (41, 44, 0, 0) 8 (58, 60, 3, 0) 23 (60, 65, 1, 0) 31 (65, 67, 3,
0) 16 (67, 71, 3, 0) 18 (71, 72, 3, 0) 12 (72, 76, 0, 1) 21 (76, 80, 3, 0) 24 (80, 85, 0, 0) 2
(85, 90, 0, 2) 20 (90, 95, 0, 0) 6 (95, 96, 0, 1) 11 (98, 103, 0, 0)

Total penalty for scheduling: 1290
```

RGLSTS algorithm

Inbound area

```
Machine 1: 9 (2, 4, 0, 0) 1 (13, 16, 0, 0) 2 (16, 21, 3, 0) 28 (21, 23, 1, 0) 18 (23, 29, 0, 1)
10 (32, 36, 1, 0) 13 (36, 42, 0, 0) 14 (47, 51, 0, 0) 31 (55, 57, 0, 0)
Machine 2: 22 (12, 15, 0, 0) 4 (15, 17, 4, 0) 30 (17, 23, 3, 0) 11 (23, 29, 0, 3) 5 (31, 33, 0,
0) 23 (35, 39, 0, 0) 6 (47, 50, 0, 0)
Machine 3: 25 (10, 13, 0, 0) 21 (15, 18, 0, 0) 29 (18, 24, 2, 0) 26 (24, 27, 0, 0) 34 (27, 29,
0, 0) 32 (34, 39, 0, 0) 15 (47, 52, 0, 0) 27 (55, 58, 2, 0) 19 (58, 60, 0, 0)
Machine 4: 20 (6, 10, 0, 0) 33 (12, 18, 1, 0) 3 (18, 22, 4, 0) 24 (22, 24, 0, 0) 16 (24, 27, 0,
1) 17 (27, 29, 0, 0) 12 (32, 38, 0, 0) 7 (42, 47, 0, 0) 8 (53, 59, 0, 0)
```

Outbound area

```
Machine 1: 28 (23, 24, 0, 0) 9 (32, 33, 0, 0) 1 (38, 42, 0, 0) 3 (42, 43, 0, 0) 30 (46, 49, 0,
0) 26 (51, 56, 11, 0) 25 (56, 58, 12, 0) 27 (58, 62, 11, 0) 13 (62, 67, 0, 0) 31 (67, 69, 1, 0)
5 (69, 70, 1, 0) 16 (70, 74, 0, 0) 18 (74, 75, 0, 0) 14 (78, 83, 0, 0) 2 (83, 88, 0, 0) 10 (88,
92, 1, 0) 29 (92, 97, 0, 0)
Machine 2: 17 (9, 10, 0, 0) 19 (39, 40, 0, 0) 7 (41, 44, 0, 0) 8 (56, 58, 5, 0) 23 (58, 63, 3,
0) 22 (63, 64, 1, 0) 15 (64, 69, 6, 0) 4 (69, 71, 1, 0) 12 (71, 75, 0, 0) 21 (75, 79, 4, 0) 24
(79, 84, 1, 0) 32 (84, 86, 0, 0) 20 (89, 94, 1, 0) 6 (94, 95, 0, 0) 11 (98, 103, 0, 0)

Total penalty for scheduling: 580

Machines cost: $6000, Scheduling cost: $580, Total cost: $6580
```

Instance 14:

RGLS algorithm

```
Current center point: ( 25, 30 )
Current center point: ( 24, 29 )
Current center point: ( 23, 28 )
Current center point: ( 22, 27 )
Current center point: ( 21, 26 )
Current center point: ( 20, 25 )
Current center point: ( 19, 24 )
Current center point: ( 18, 23 )
Current center point: ( 17, 22 )
Current center point: ( 16, 21 )
Current center point: ( 15, 20 )
Current center point: ( 14, 19 )
Current center point: ( 13, 18 )
Current center point: ( 12, 17 )
Current center point: ( 11, 16 )
Current center point: ( 10, 15 )
Current center point: ( 9, 14 )
Current center point: ( 8, 13 )
Current center point: ( 7, 12 )
Current center point: ( 6, 11 )
Current center point: ( 5, 10 )
Current center point: ( 5, 9 )
Current center point: ( 5, 8 )
Current center point: ( 5, 7 )
Current center point: ( 5, 6 )
Current center point: ( 5, 5 )
Current center point: ( 5, 4 )
```

Inbound area

```
Machine 1: 28 (6, 8, 0, 0) 50 (15, 21, 0, 0) 1 (21, 27, 0, 0) 40 (29, 35, 1, 0) 34 (35, 38, 0,
0) 39 (41, 45, 0, 0) 46 (50, 53, 0, 0) 42 (53, 56, 2, 0) 19 (59, 64, 0, 0) 7 (65, 70, 0, 0)
Machine 2: 11 (12, 16, 1, 0) 8 (16, 21, 2, 0) 22 (21, 23, 3, 0) 16 (29, 34, 0, 0) 21 (34, 39,
0, 0) 24 (45, 49, 0, 0) 20 (54, 59, 0, 0) 25 (62, 64, 0, 0) 9 (64, 69, 0, 0) 17 (69, 75, 0, 0)
Machine 3: 49 (10, 14, 0, 0) 33 (14, 17, 2, 0) 43 (17, 22, 2, 0) 2 (25, 29, 0, 0) 3 (30, 32, 0,
0) 13 (32, 38, 0, 1) 47 (43, 47, 0, 0) 37 (52, 54, 0, 0) 31 (58, 60, 0, 0) 38 (63, 68, 1, 0)
Machine 4: 12 (14, 16, 0, 0) 14 (18, 23, 0, 0) 5 (23, 29, 0, 0) 30 (30, 36, 0, 0) 35 (37, 39,
0, 0) 36 (43, 47, 0, 0) 44 (52, 56, 0, 0) 32 (58, 64, 0, 0) 4 (67, 71, 0, 0)
Machine 5: 6 (9, 15, 0, 0) 15 (15, 18, 1, 0) 23 (18, 21, 0, 0) 48 (23, 25, 0, 0) 29 (25, 31, 1,
0) 26 (31, 33, 0, 0) 10 (33, 37, 0, 1) 18 (39, 45, 0, 0) 41 (45, 49, 0, 0) 27 (55, 59, 0, 0) 45
(63, 69, 0, 0)
```

Outbound area

```
Machine 1: 35 (28, 29, 0, 0) 34 (41, 42, 0, 0) 8 (47, 50, 0, 0) 43 (53, 55, 0, 0) 6 (55, 60, 0,
0) 7 (63, 64, 2, 0) 58 (64, 67, 4, 0) 29 (67, 72, 5, 0) 27 (72, 73, 6, 0) 38 (73, 78, 3, 0) 15
(78, 83, 1, 0) 40 (83, 85, 1, 0) 4 (85, 87, 0, 0) 24 (88, 93, 0, 0) 49 (97, 101, 0, 0)
Machine 2: 46 (23, 24, 0, 0) 48 (35, 37, 0, 0) 32 (47, 49, 0, 0) 45 (51, 52, 0, 0) 19 (54, 56,
0, 0) 1 (56, 57, 0, 1) 18 (57, 59, 0, 0) 30 (61, 66, 0, 0) 31 (68, 73, 0, 0) 9 (73, 78, 2, 0)
59 (78, 82, 0, 0) 47 (82, 87, 0, 2) 56 (88, 90, 0, 0) 12 (90, 92, 0, 0) 39 (94, 99, 0, 0)
Machine 3: 5 (16, 17, 0, 0) 28 (29, 30, 0, 0) 37 (45, 46, 0, 0) 51 (49, 51, 0, 0) 53 (54, 58,
0, 0) 11 (61, 63, 0, 0) 3 (64, 68, 1, 0) 60 (68, 71, 1, 0) 42 (71, 75, 3, 0) 10 (75, 79, 1, 0)
25 (79, 84, 0, 1) 57 (84, 87, 0, 3) 33 (87, 89, 1, 0) 16 (89, 94, 0, 0) 14 (97, 101, 0, 0)
Machine 4: 50 (22, 23, 0, 0) 55 (34, 37, 0, 0) 22 (46, 48, 0, 0) 54 (50, 53, 0, 0) 44 (54, 58,
0, 0) 41 (61, 66, 0, 0) 13 (66, 70, 3, 0) 20 (70, 74, 5, 0) 36 (74, 75, 3, 0) 52 (75, 78, 1, 0)
17 (78, 81, 0, 1) 26 (81, 83, 0, 1) 23 (83, 87, 0, 2) 21 (88, 91, 0, 0) 2 (94, 98, 0, 0)

Total penalty for scheduling: 1359
```

RGLSTS algorithm

Inbound area

```
Machine 1: 28 (6, 8, 0, 0) 50 (15, 21, 0, 0) 1 (21, 27, 0, 0) 30 (29, 35, 1, 0) 34 (35, 38, 0,
0) 36 (43, 47, 0, 0) 42 (52, 55, 3, 0) 27 (55, 59, 0, 0) 19 (59, 64, 0, 0) 7 (65, 70, 0, 0)
Machine 2: 11 (13, 17, 0, 0) 8 (18, 23, 0, 0) 16 (29, 34, 0, 0) 21 (34, 39, 0, 0) 24 (45, 49,
0, 0) 20 (54, 59, 0, 0) 25 (62, 64, 0, 0) 9 (64, 69, 0, 0) 17 (69, 75, 0, 0)
Machine 3: 49 (10, 14, 0, 0) 33 (14, 17, 2, 0) 43 (17, 22, 2, 0) 2 (25, 29, 0, 0) 3 (30, 32, 0,
0) 13 (32, 38, 0, 1) 47 (43, 47, 0, 0) 37 (52, 54, 0, 0) 31 (58, 60, 0, 0) 38 (63, 68, 1, 0)
```

Machine 4: 12 (14, 16, 0, 0) 23 (18, 21, 0, 0) 22 (21, 23, 3, 0) 5 (23, 29, 0, 0) 40 (30, 36, 0, 0) 35 (37, 39, 0, 0) 39 (41, 45, 0, 0) 41 (45, 49, 0, 0) 46 (50, 53, 0, 0) 32 (58, 64, 0, 0) 4 (67, 71, 0, 0)
Machine 5: 6 (9, 15, 0, 0) 15 (15, 18, 1, 0) 14 (18, 23, 0, 0) 48 (23, 25, 0, 0) 29 (25, 31, 1, 0) 26 (31, 33, 0, 0) 10 (33, 37, 0, 1) 18 (39, 45, 0, 0) 44 (52, 56, 0, 0) 45 (63, 69, 0, 0)

Outbound area

Machine 1: 5 (16, 17, 0, 0) 35 (28, 29, 0, 0) 34 (41, 42, 0, 0) 32 (47, 49, 0, 0) 54 (50, 53, 0, 0) 19 (53, 55, 1, 0) 6 (55, 60, 0, 0) 29 (61, 66, 11, 0) 58 (66, 69, 2, 0) 60 (69, 72, 0, 0) 42 (72, 76, 2, 0) 52 (76, 79, 0, 0) 15 (79, 84, 0, 0) 40 (84, 86, 0, 0) 21 (88, 91, 0, 0) 39 (94, 99, 0, 0)
Machine 2: 46 (23, 24, 0, 0) 28 (29, 30, 0, 0) 48 (35, 37, 0, 0) 8 (47, 50, 0, 0) 45 (51, 52, 0, 0) 43 (53, 55, 0, 0) 1 (55, 56, 0, 0) 18 (57, 59, 0, 0) 41 (60, 65, 1, 0) 7 (65, 66, 0, 0) 9 (66, 71, 9, 0) 38 (71, 76, 5, 0) 10 (76, 80, 0, 0) 47 (80, 85, 0, 0) 4 (85, 87, 0, 0) 33 (87, 89, 1, 0) 16 (89, 94, 0, 0) 14 (97, 101, 0, 0)
Machine 3: 55 (34, 37, 0, 0) 53 (54, 58, 0, 0) 30 (60, 65, 1, 0) 3 (65, 69, 0, 0) 25 (69, 74, 9, 0) 17 (74, 77, 3, 0) 59 (77, 81, 1, 0) 57 (81, 84, 0, 0) 56 (88, 90, 0, 0) 12 (90, 92, 0, 0) 49 (97, 101, 0, 0)
Machine 4: 50 (22, 23, 0, 0) 37 (45, 46, 0, 0) 22 (46, 48, 0, 0) 51 (49, 51, 0, 0) 44 (54, 58, 0, 0) 11 (61, 63, 0, 0) 13 (64, 68, 5, 0) 31 (68, 73, 0, 0) 20 (73, 77, 2, 0) 36 (77, 78, 0, 0) 27 (78, 79, 0, 0) 26 (79, 81, 1, 0) 23 (81, 85, 0, 0) 24 (88, 93, 0, 0) 2 (94, 98, 0, 0)

Total penalty for scheduling: 268

Machines cost: $9000, Scheduling cost: $268, Total cost: $9268

References

ABERNATHY, William J.; BALOFF, Nicholas; HERSHEY, John C. and WANDEL, Sten (1973). A Three-Stage Manpower Planning and Scheduling Model: A Service-Sector Example. *Operations Research*, Vol. 21, No. 3, pp. 693-711.

ADAM, Everett E. and EBERT, Ronald, J. (1991). Production and Operations Management. 4th ed., Prentice-Hall, USA.

BAASE, Sara (1991). Computer Algorithms: Introduction to Design and Analysis. 2nd ed., Addison-Wesley Publishing Company, USA.

BAGCHI, Uttarayan; SULLIVAN, Robert S. and CHANG, Yih-Long (1986). Minimizing Mean Absolute Deviations of Completion Times about a Common Due Date. *Naval Research Logistics Quarterly*, Vol. 33, pp. 227-240.

BAKER, Kenneth R. (1976). Workforce Allocation in Cyclical Scheduling Problems: A Survey. *Operational Research Quarterly*, Vol. 27, No. 1, pp 155-167.

BAKER, Kenneth R. and SCUDDER, Gary D. (1990). Sequencing with Earliness and Tardiness Penalties: a Review. *Operations Research*, Vol. 38, No. 1, pp 22-36.

BAZARAA, Mokhtar S.; JARVIS, John J. and SHERALI, Hanif D. (1990). Linear Programming and Network Flows. 2nd ed., Wiley, Singapore.

BLAZEWICZ, Jacek; ECKER, Klaus H.; PESCH, Erwin; SCHMIDT, Günter and WEGLARZ, Jan (2001). Scheduling Computer and Manufacturing Processes. 2nd ed., Springer, Germany.

BRENNAN, Linda L. and ORWIG, Robert A. (2000). A Tale of Two Heuristics: Conflicting Work Allocation Approaches in Engineering Consulting. *Engineering Management Journal*, Vol. 12, No. 3, pp 18-25.

BURKE, Edmund K.; KENDALL, Graham and SOUBEIGA, Eric (2003). A Tabu Search Hyper-heuristic for Timetabling and Rostering. *Journal of Heuristics*, Vol 9, No. 6, pp 451-470.

CAMPBELL, Gerard M. and DIABY, Moustapha (2002). Development and Evaluation of an Assignment Heuristic for Allocating Cross-trained Workers. *European Journal of Operational Research*, Vol. 138, pp 9-20.

COFFMAN, Edward G. (1976). Computer and Job-Shop Scheduling Theory. Wiley-Interscience, USA.

CORMEN, Thomas H.; LEISERSON, Charles E. and RIVEST, Ronald L. (2001). Introduction to Algorithms. 2nd ed., MIT Press, USA.

COROMINAS, Albert; PASTOR, Rafael and RODRÍGUEZ, Ericka (2004). Rotational Allocation of Tasks to Multi-functional Workers in a Service Industry. *Technical Report IOC-DT-P-2004-09*. Universitat Polytècnica de Catalunya, Barcelona, Spain.

CRAINIC, Teodor G. and LAPORTE, Gilbert (1998). Fleet Management and Logistics. Kluwer Academic Publishers, USA.

DANTZIG, George B. and THAPA, Mukund N. (2003). Linear Programming, Vol 2: Theory and Extensions. Springer, USA.

DE ALBA, Karim (2004). Un Procedimiento Heurístico para un Problema de Diseño de Redes Multiproducto con Capacidad Finita y Cargos Fijos. Thesis. UANL, Mexico. (in Spanish)

DÍAZ, Adenso; GLOVER, Fred; GHAZIRI, Hassan M.; GONZÁLEZ-VELARDE, José L.; LAGUNA, Manuel; MOSCATO, Pablo and TSENG, Fan T. (1996). Optimización Heurística y Redes Neuronales. Ed. Paraninfo, Spain. (in Spanish)

FEO, Thomas A. and RESENDE, Mauricio G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, Vol. 8, pp 67-71.

FLEURENT, Charles and GLOVER, Fred (1999). Improved Constructive Multistart Strategies for the Quadratic Assignment Problem Using Adaptive Memory. *INFORMS Journal on Computing*, Vol. 11, No. 2, pp 198-204.

GAREY, Michael R.; TARJAN, Robert E. and WILFONG, Gordon T. (1988). One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties. *Mathematics of Operations Research*, Vol. 13, No. 2, pp 330-348.

GENDREAU, Michel (2002). An Introduction to Tabu Search, in Handbook of Metaheuristics. GLOVER, Fred and KOCHENBERGER, Gary A. (Eds.). Kluwer Academic Publishers, USA.

GLOVER, Fred (1977). Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, Vol. 8, pp 156-166.

GLOVER, Fred (1986). Future Paths to Integer Programming and Links to Artificial Intelligence. *Computers & Operations Research*, Vol. 5, No. 13, pp 533-549.

GLOVER, Fred and KOCHENBERGER, Gary A. (2003). Handbook of Metaheuristics. Kluwer Academic Publishers, USA.

GLOVER, Fred and LAGUNA, Manuel (1997). Tabu Search. Kluwer Academic Publishers, USA.

GOMAR, Jorge E.; HAAS, Carl T. and MORTON, David P. (2002). Assignment and Allocation Optimization of a Partially Multiskilled Workforce. *ASCE Journal of Construction Engineering and Management*, Vol. 128, No. 2, pp. 103-109.

GOTTFRIED, Byron S. (1997). Programación en C. 2nd ed., McGraw-Hill/Interamericana de España, Spain. (in Spanish)

GUE, Kevin R. (2001). Crossdocking: Just-In-Time for Distribution. *Technical Report*. Graduate School of Business and Public Policy Naval Postgraduate School, Monterey, CA, USA.

HALL, Nicholas G.; KUBIAK, Wieslaw and SETHI, Suresh P. (1991). Earliness-Tardiness Scheduling Problems, Vol II: Deviation of Completion Times about a Restrictive Common Due Date. *Operations Research*, Vol. 39, No. 5, pp 847-856.

HANSEN, Pierre (1986). The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. *Congress on Numerical Methods in Combinatorial Optimization*, Italy.

HART, Pirie and SHOGAN, Andrew W. (1987). Semi-greedy Heuristics: An Empirical Study. *Operations Research Letters*, Vol. 6, pp 107-114.

HEADY, Ronald B. and ZHU, Zhiwei (1998). Minimizing the Sum of Job Earliness and Tardiness in a Multi-machine System. *International Journal of Production Research*, Vol. 36, No. 6, pp 1619-1632.

HEYMANN, Elisa; SENAR, Miguel A.; LUQUE, Emilio and LIVNY, Miron (2000). Adaptive Scheduling for Master-Worker Applications on the Computational Grid. *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pp 214-227.

HOCHBAUM, Dorit S. (1997). Approximation Algorithms for NP-Hard Problems. PWS Publishing Company, USA.

HOROWITZ, Ellis; SAHNI, Sartaj and RAJASEKARAN, Sanguthevar (1998). Computer Algorithms. Computer Science Press, USA.

IIMA, Hitoshi and SANNOMIYA, Nobuo (2001). Module Type Genetic Algorithm for Modified Scheduling Problems with Worker Allocation. *Proceedings of the American Control Conference*, pp 856-861.

LAGUNA, Manuel and GONZÁLEZ-VELARDE, José L. (1991). A Search Heuristic for Just-in-Time Scheduling in Parallel Machines. *Journal of Intelligent Manufacturing*, Vol. 2, pp 253-260.
LAGUNA, Manuel and MARTÍ, Rafael (2003). Scatter Search: Methodology and Implementations in C. Kluwer Academic Publishers, USA.

LAKSHMINARAYAN, Sankaran; LAKSHMANAN, Ram; PAPINEAU, Robert L. and ROCHETTE, Rene (1978). Optimal Single-Machine Scheduling with Earliness and Tardiness Penalties. *Operations Research*, Vol. 26, No. 6, pp 1079-1082.

LAWLER, Eugene L.; LENSTRA, Jan K.; RINNOOY-KAN, Alexander H. and SHMOYS, David B. (1985). The Traveling Salesman Problem. Wiley-Interscience, Great Britain.

LEUNG, Joseph Y-T. (2004). Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapman and Hall/CRC, USA.

LEWIS, Gordon H.; SRINIVASAN, Ashok and SUBRAHMANIAN, Eswaran (1998). Staffing and Allocation of Workers in an Administrative Office. *Management Science*, Vol. 44, No. 4, pp 548-570.

LI, Yanzhi; LIM, Andrew and RODRIGUES, Brian (2004). Crossdocking - JIT Scheduling with Time Windows. *Journal of the Operational Research Society*, Vol. 55, No. 12, pp 1342-1351.

LIAW, Ching-Fang (1999). A Branch-and-Bound Algorithm for the Single Machine Earliness and Tardiness Scheduling Problem. *Computers & Operations Research*, Vol. 26, pp 679-693.

LIN, S. and KERNIGHAN, Brian W. (1973). An effective heuristic algorithm for the Traveling Salesman problem. *Operations Research*, Vol. 21, No. 2, pp 498-516.

MAZZINI, Renata and ARMENTANO, Vinícius A. (2001). A Heuristic for Single Machine Scheduling with Early and Tardy Costs. *European Journal of Operational Research*, Vol. 128, pp 129-146.

MURTY, Katta G. (1983). Linear Programming. John Wiley & Sons, USA.

NEAPOLITAN, Richard E. and NAIMIPOUR, Kumarss (1998). Foundations of Algorithms using C++ Pseudocode. 2nd ed., Jones and Bartlett Publishers, USA.

NEMHAUSER, George L. and WOLSEY, Laurence A. (1999). Integer and Combinatorial Optimization. Wiley-Interscience, USA.

PINEDO, Michael (2002). Scheduling: Theory, Algorithms, and Systems. 2nd ed., Prentice Hall, USA.

POLYA, George (1957). How to Solve it. 2nd ed., Princeton University Press, USA.

PRAIS, Marcelo and RIBEIRO, Celso C. (2000). Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, Vol. 12, pp 164-176.

RADHAKRISHNAN, Sanjay and VENTURA, José A. (2000). Simulated Annealing for Parallel Machine Scheduling with Earliness-Tardiness Penalties and Sequence-Dependent Set-up Times. *International Journal of Production Research*, Vol. 38, No. 10, pp 2233-2252.

RESENDE, Mauricio G. and GONZÁLEZ-VELARDE, José L. (2003). GRASP: Procedimientos de Búsqueda Miopes, Aleatorizados y Adaptativos. *Inteligencia Artificial*, No. 19, pp 61-76. (in Spanish)

RESENDE, Mauricio G. and RIBEIRO, Celso C. (2001). Greedy Randomized Adaptive Search Procedures, in Handbook of Metaheuristics. GLOVER, Fred and KOCHENBERGER, Gary A. (Eds.). Kluwer Academic Publishers, USA.

RIVERA, José G. (1996). Desarrollo de Métodos Heurísticos para el Problema de Secuenciación Justo a Tiempo considerando Tiempos de Preparación. Thesis. ITESM, Campus Monterrey, Mexico. (in Spanish)

ROSAS, Rosario (1991). Diseño de un Modelo de Solución por medio del Metaheurístico Tabu Search para el Problema de Programación de Tareas en Dos Máquinas. Thesis. ITESM, Campus Monterrey, Mexico. (in Spanish)

SIDNEY, Jeffrey B. (1977). Optimal Single-Machine Scheduling with Earliness and Tardiness Penalties. *Operations Research*, Vol. 25, No. 1, pp 62-69.

SIERKSMA, Gerard (2001). Linear and Integer Programming: Theory and Practice. 2nd ed., Marcel Dekker Inc., USA.

SIFERD, Sue P. and BENTON, W. C. (1992). Workforce Staffing and Scheduling: Hospital Nursing Specific Models. *European Journal of Operational Research*, Vol. 60, pp 233-246.

SIVRIKAYA-SERIFOGLU, Funda and ULUSOY, Gündüz (1999). Parallel Machine Scheduling with Earliness and Tardiness Penalties. *Computers & Operations Research*, Vol. 26, pp 773-787.

THARMMAPHORNPHILAS, Wipawee and NORMAN, Bryan A. (2004). A Quantitative Method for Determining Proper Job Rotation Intervals. *Annals of Operations Research*, Vol. 128, pp 251-256.

ZANAKIS, Stelios H. and EVANS, James R. (1981). Heuristic “Optimization”: Why, When, and How to Use It. *Interfaces*, Vol. 11, No. 5, pp 84-91.