UNIVERSITÀ DEGLI STUDI DELL'AQUILA
DIPARTIMENTO DI INGEGNERIA E SCIENZE DELL'INFORMAZIONE E MATEMATICA (DISM)

Dottorato di Ricerca in: Information and Communication Technology (ICT)

Curriculum: Emerging computing models: algorithms architectures and intelligent systems

Ciclo: XXXIII

Titolo della tesi

Optimization models and algorithms for part routing and scheduling in a wafer fab

SSD: MAT/09

Dottoranda

Fatemeh Kafash Ranjbar

Coordinatore del corso

Prof. Vittorio Cortellessa

Tutor

Prof. Claudio Arbib

A.A. 2020/2021

"The important thing is not to stop questioning. Curiosity has its own reason for existing. One cannot help but be in awe when he contemplates the mysteries of eternity, of life, of the marvelous structure of reality. It is enough if one tries merely to comprehend a little of this mystery every day."

Albert Einstein

Declaration of Authorship

I, **Fatemeh Kafash Ranjbar**, declare that this thesis titled, “Optimization models and algorithms for part routing and scheduling in a wafer fab” and the work presented in it are my own. I confirm that:

- ✓ Where I have consulted the published work of others, this is always clearly attributed.
- ✓ Where I have quoted from the work of others, the source is always given.
- ✓ I have acknowledged all main sources of help.

Signed:

Date:

University of L'Aquila

Dept. of Information Engineering, Computer Science and Mathematics

Doctoral program in Information engineering and science, cycle XXXIII

(Emerging computing models: algorithms architectures and intelligent systems)

Optimization models and algorithms for part routing and scheduling in a wafer fab

By Fatemeh Kafash Ranjbar

Abstract

In the first part of our investigation, we consider the problem of moving production lots within the clean room of *LFoundry*, an important Italian manufacturer of micro-electronic devices. The model adopted is a DIAL-A-RIDE PROBLEM (DARP). We propose a math-heuristic, specifically designed for use in a dynamic environment, with the objective of balancing vehicle workloads. This objective is achieved by minimizing, at each optimization cycle, the makespan, i.e., the largest completion time among the vehicles. A *cluster-first route-second* heuristic is devised for on-line use and compared to the actual practice through a computational experience based on real plant data. In the perspective of a comparison of the solutions obtained to the optimum or to lower bounds, and also to address such details as in-process inventory, we developed an original INVENTORY DARP and formulated it in terms of Mixed Integer Programming (MIP). This problem has not been considered yet in the vehicle routing literature. Due to its inherent complexity, our MIP formulation was just useful to find optima in small problem instances, being inapplicable to those derived from real size. Anyway, this piece of research gave us interesting indications for future studies.

In the second part of the thesis, we present a mathematical model to deal with a problem that often arises in those manufacturing contexts where industrial parts are obtained by cutting sheets or bars of raw material. In such contexts, the production quality might be harmed by the occurrence of faults in the sheets. In our study, we focus on a ONE-DIMENSIONAL BIN PACKING PROBLEM (1-BPP) and consider the problem of finding limited trim-loss solutions which also minimize the expected loss of parts derived from small faulty areas. We first address the problem of computing the expected loss, then the more complex one of finding a BIN PACKING that is most robust against random faults. Two heuristics based on the solution of a suitable MULTIPROCESSOR SCHEDULING PROBLEM are designed and tested.

Keywords: Vehicle Routing, Dial-A-Ride, Bin Packing, Multiprocessor Scheduling, Robust Optimization, Math-heuristics, Mixed Integer Linear Programming.

Acknowledgements

As an industrial PHD student, I had this opportunity to have collaboration with both academic and industrial people. First, I would like to thank my supervisor **Prof. Claudio Arbib** for all his assistance and guidance during my doctoral program, for giving me the possibility to work on many different research projects, and for teaching me a lot about Operations Research and Combinatorial Optimization. He was always helpful and motivated me to cultivate different approaches and new ideas. He was there for any question, doubt or concern that would come up to me. He really is a remarkable supervisor and professor.

I also want to thank our Doctoral Program Coordinator, **Prof. Vittorio Cortellessa,** for all his sincere support during the whole program.

I would also like to thank all the other components of the group of Operations Research of our department, **Prof. Stefano Smriglio** and **Prof. Fabrizio Rossi**, for assisting me to learn more in Operations Research field.

I really appreciated the head of our department **Prof. Guido Proietti**, and all the colleagues in the Department of Information Engineering, Computer Science and Mathematics of the University of L'Aquila for all their advice and support.

I also want to thank the Ministero dell'Istruzione, dell'Università e della Ricerca (**MIUR**) for the financial support provided with a **PON** scholarship (PON Ricerca e innovazione) to my Ph.D. program.

Moreover, I'd really like to thank all of people in **IT, ICT, and Manufacturing Execution System (MES) group of LFoundry**, for giving me an opportunity to work in a professional environment during my doctoral program.

Thanks also to the members of Department of Operations and Information Systems of the University of Graz, specially to **Prof. Ulrich Pferschy**, for all the help given during my stay there, in the difficult period of Covid19 pandemics.

Finally, I would like to thank **my family** for supporting my decision of starting a new stage of my life, studying, and working abroad. They always encouraged me in my career endeavors and empowered me in complicated periods through every obstacle that emerged during my time abroad.

Fatemeh (Fatima) Kafash Ranjbar

L'Aquila, Italy, August 2021

Contents

List of Figures

List of Tables

1 Introduction

1.1 Quick overview

This work originates from an industrial PON grant spent at *LFoundry S.r.l.* (www.lfoundry.com), a major semiconductor manufacturer located in Avezzano (L'Aquila, Italy). The main problem modeled and analyzed is the routing and scheduling of a fleet of manually operated vehicles (the *carts*) in the heart of the plant: the *clean room* where the core production process takes place. The fleet offers transportation services based on requests that originate from hi-tech machines which, organized by group-technology into fourteen separate work areas, perform the necessary production operations. Guaranteeing an efficient transportation service is crucial for a plant that employs several hundreds of such machines, and where each individual product lot must undergo some 700 distinct production steps. This issue has also increased its importance since the evolution of the process from mass manufacturing to on-demand production.

Optimal transportation services are the subject of a huge body of research. In this field, the model that appeared to fit our problem best is known as the DIAL-A-RIDE PROBLEM (DARP): this is a particular VEHICLE ROUTING PROBLEM (VRP) – more properly, a PICKUP AND DELIVERY VRP (PDVRP). In the PDVRP, one or more commodities must be moved from an origin to a destination, and in the one-to-one case of the problem, commodities are replaced by individual resources with a single origin and a single destination; pickup/deliveries can be done once for all or periodically repeated, depending on application. The DARP entails servicing a series of customer transport requests at the lowest possible cost, each specified by a specific origin-destination pair, while adhering to operational and quality-of-service restrictions. Load and capacity, total length covered, and transit time constraints, as well as time windows, are common examples of such restrictions.

In our work, we frame the practical problem at *LFoundry* in the VRP literature, fit it into a formal model that includes all necessary details and takes advantage of special layout conditions in the plant, try and solve it by exact and heuristic algorithm designed on purpose, and assess the quality of the solutions found via an extensive computational experience built on data obtained by the plant operation.

A main step in this research was the construction of mixed integer programming models for the problem or for subproblems originated by its decomposition. These models include, among the constraint, sort-of bin packing inequalities that ensure that each vehicle load does not exceed the relevant vehicle capacity. In the ONE-DIMENSIONAL BIN PACKING PROBLEM (1-BPP) items of different lengths must be assigned to a minimum number of bins of unit length (the problem is referred

to as ONE-DIMENSIONAL CUTTING STOCK PROBLEM (1-CSP) when multiple copies of the same item are considered). Reciprocally, in the MULTIPROCESSOR SCHEDULING PROBLEM (MSP), a set of bins of unbounded length is given and one must fill them with all the items in such a way that the largest bin load (also called the makespan) is reduced to a minimum.

In a branch of our research, we then considered a situation in which bins represent standard fixed length bars of some material, from which one wants to cut items of various lengths. A defect can however occur in a bar, and items cut from a faulty zone should be discarded. Depending on defect position, cuts can or cannot be reconfigured to save all items, and we consider a ROBUST BIN PACKING PROBLEM (RBPP), that is the problem of finding a bin packing that minimizes the expected item loss due to defects.

The main problems considered in our work are all NP-hard, which implies that no polynomial-time algorithm is presently known for their exact solution. Considering the industrial interest of our research, we basically concerned on heuristics, that is, algorithms that find possibly good primal solutions without being able to ensure their optimality. In fact, the tackled real-world case entails an amount of transportation requests so large that any known exact solution procedure is unpractical because not compliant with on-line operation. The results achieved, evaluated through computational experience carried out with real-world data or instances taken from literature, are encouraging and corroborate the done effort. Further research needs include: for the DARP, on one hand studying alternative formulation for the inventory-constrained version, and on the other hand implementing the heuristic methodology developed at the clean room operation level; for the RBPP, studying new MIP model or finding good bounds to improve/assess the quality of the primal solutions found.

1.2 Thesis outline

The thesis focuses on the following general issues:

- Analysis and model of the part of the internal logistics of *LFoundry*, a relevant microchip manufacturer, which refers to transportation services of product components processed by the plant.
- Mathematical formulation of an original optimization problem (INVENTORY DARP) with a focus on rack capacity, and placement of this problem in the VRP literature.
- Development of an original math-heuristic algorithm to solve the DARP modeled, and prototypal implementation in Python making use of Mixed Integer Programming (MIP) solvers (GuRoBi®).

- Computational experience with real data from the plant and quality assessment of the methodologies developed.
- Definition of an original model for evaluating the robustness of bin packing solutions against random point-shaped defects in the bins.
- Development of original math-heuristic algorithms to find robust packing with a given number of bins, and prototypal implementation in Python making use of MIP solvers (GuRoBi®).
- Computational experience with data from the bin packing literature and quality assessment of the methodologies developed.

The work is organized as follows:

In Chapter **2** we describe the application context. This description includes the Company's involvement in the program, their goals and needs, the application details, the routing and scheduling system, the data structures used, the Manufacturing Execution System, along with general information about the problem and the Company's approach to it.

Chapter **3** is divided in two parts. First, we will survey some fundamental concepts of Mathematical Programming, Linear Programming, Mixed Integer Programming, Branch-and-Bound, and the Cutting Planes Approach. Then, we discuss the main models and algorithmic techniques used in our research: these include such models as the VRP, the DARP, the MATCHING (ASSIGNMENT) PROBLEM, the BOTTLENECK MATCHING PROBLEM, and such methods as the Hungarian and the Bottleneck Matching algorithm.

In chapter **4**, we define the INVENTORY DIAL-A-RIDE PROBLEM, a general MIP model that incorporates all the main feature of the *LFoundry* application. In particular, the model addresses the capacity limitation at the racks which serve the fourteen work areas in which the clean room is divided. We stress the model to evaluate the size at which exact solutions can no longer be provided by the state-of-the-art MIP solver used, GuRoBi®. Even if one removes rack capacities, this size is quite smaller than the one derived from real data and justifies the recourse to heuristics, such as the one illustrated in the subsequent chapter.

The math-heuristic described in chapter **5**. decomposes the general DARP in three subproblems or phases, following a *first-cluster then-route* approach. The first phase consists in solving a MIP model to partition requests into as many clusters as available carts in the fleet. This phase exploits in a crucial way the rectilinear layout of the plant. A second phase consists then in computing the time required of each cart to schedule the requests of each cluster (we limited our attention to simple double-sweep

schedules). Finally, in phase three, each cart is assigned to one request cluster by solving a BOTTLENECK MATCHING PROBLEM. The chapter is concluded by exhibiting our computational findings: these are organized in two parts, corresponding to a static and a dynamic scenario, with real data from the clean room.

Chapter **6** regards the latter problem as a model for deciding raw material cutting operations that obtain a given set of items from rectilinear bars in such a way that trim loss is minimized. In this scenario, we consider the effect on production quality of defects that randomly arise in the bars. We propose a mathematical model to deal with this situation, first by evaluating the expected loss due to defective areas, then trying to construct solutions that minimize such a loss using a given number of bars. The latter problem, of considerable complexity, is approached by two math-heuristics based on the solution of particular MPS problems. The efficiency and effectiveness of the solution methods is evaluated on problem instances taken from the bin packing literature.

2 Application context

In this chapter, we describe the application context. The description includes the Company's involvement in the program, their goals and needs, the application details, the routing and scheduling system, the data structures used, the Manufacturing Execution System, along with general information about the problem and the way the Company is presently approaching it.

2.1 Generalities on *LFoundry*

LFoundry S.r.l. (www.lfoundry.com) produces high-tech micro-electronic components with a very high integration scale in an advanced production plant located in Avezzano (L'Aquila, Italy). The firm has an advanced 200mm manufacturing fab and proprietary technologies at 150 and 110 nm nodes, with MPW[1] and MLM[2] services available. *LFoundry* provides special capabilities and know-how for CMOS Image Sensors through CIS optimized processes down to 90nm, as well as Back Side Illumination technology. They also provide excellent technology support for Optoelectronics such as SiPM, SPAD, X-Ray, as well as for DBI Bonding (3D-Stacking) and Smart Power and a vast range of applications for the automotive, industrial, medical, security, science, and space imaging industries. As a SMIC Company, *LFoundry* can leverage skills and capabilities of one of the leading semiconductor foundries in the world and the largest and most advanced foundry in mainland China. Fig. 2-1 gives an overview of SMIC global presence.

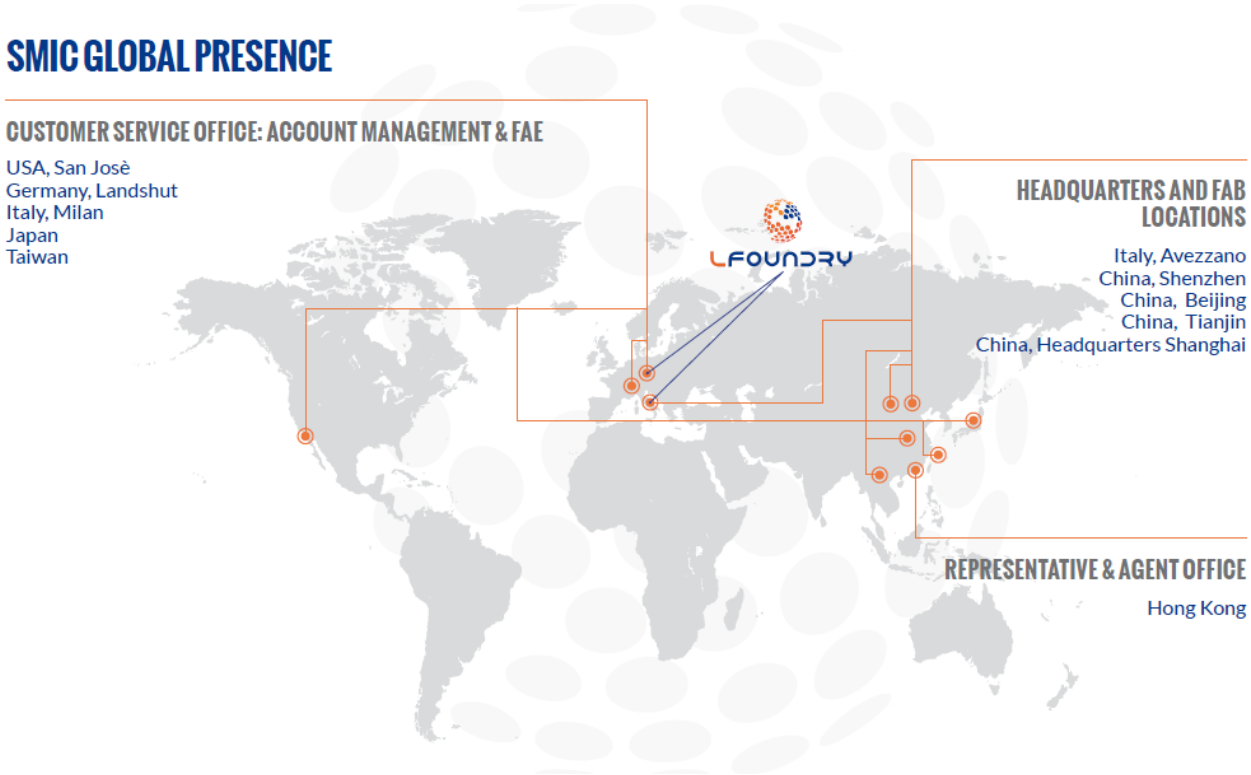


Fig. 2-1) SMIC global presence.

[1] *Multi Project Wafer*
[2] *Multi Layer Mask*

Since 2006, the Avezzano site (former *Micron Technology*) has been manufacturing customer imaging process technologies and products using 180nm to 90nm technologies at the 200mm, including a volume copper Back End of Line (BEOL), Back Side Illumination processes (BSI) and extensive testing capabilities.

The Company provides automotive ISO-TS16949, telecom TL9000 and ISO9001 quality management system certifications as well as OHSAS 18001 certification for health and safety management system and ISO 14001 certification for environmental management system.

Fig. 2-2 shows the company's most relevant numbers and data.

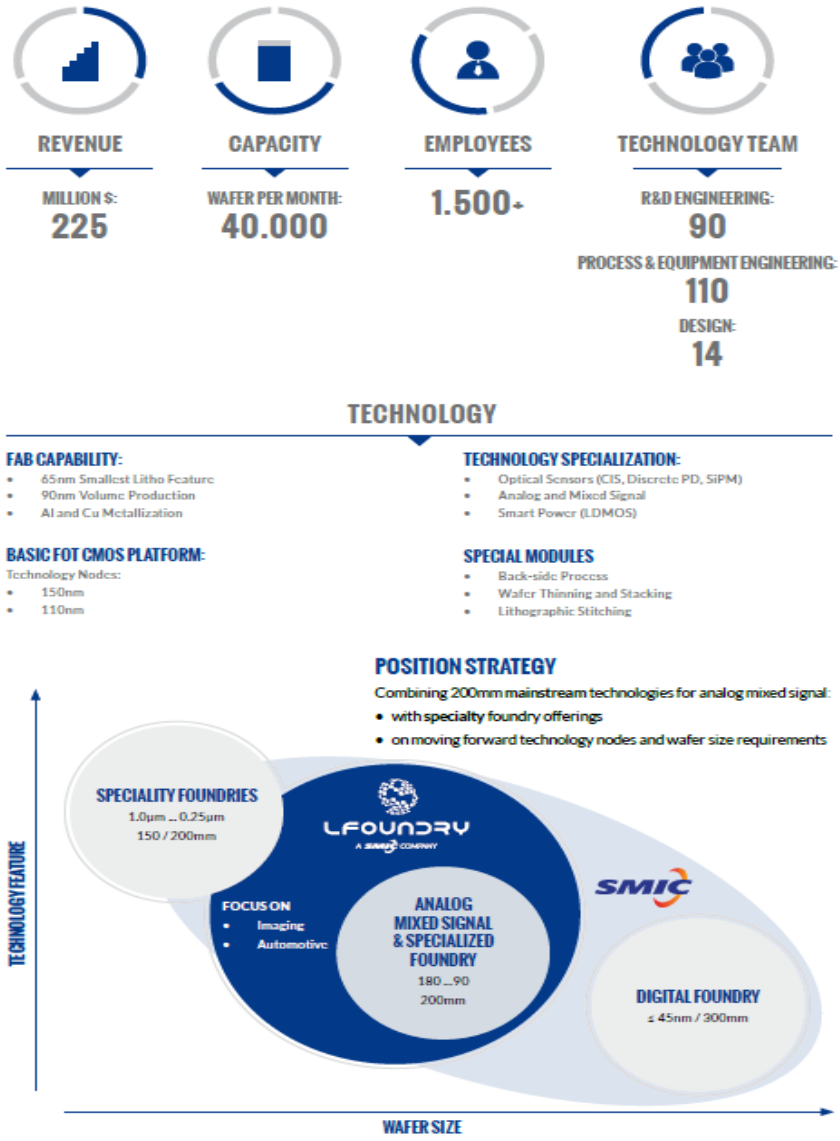


Fig. 2-2) LFoundry's base data.

2.2 Production line and Clean Room

The main production activities of the Avezzano plant take place in a special area called *clean room,* i.e., a space that has HEPA filtration to remove particles from the air and is therefore characterized by controlled and extremely low levels of air pollution. Clean rooms are used for manufacturing where high level of cleanliness and sterility are required. Common application are medical devices, pharmaceutical and semiconductor manufacturing.

Clean rooms are classified according to the number of particles permitted per volume of air, distinguished by size. *LFoundry*'s clean room in Avezzano is ISO3 class. People who work in the clean room must obey severe norms and restrictions and must use special suits to prevent bringing contamination inside it (Fig. 2-3). In fact, semiconductors manufacturers produce devices with ultra-small super dense features (examples are computer chips for our cell phone or PC): if contamination were to get on the chip during manufacturing, they would not work.

Fig. 2-3) Clean room facilities.

The *LFoundry*'s clean room is 162 m long and 53 m large, for a covered area of 8586 m^2, and is equipped with around 700 machines clustered according to group technology (i.e., by functionality,

Fig. 2-4) to achieve economies of scale in the necessary technological support systems. There are 7 areas in the clean room identified according to technology: *Photo*, *Dry Etch*, *CVD/PVD*, *Diffusion*, *Wet*, *Implant*, *CMP*.

In addition, there is one more area called *Probe and Analytics* which is the last destination of the production lots. In this area, lots pass a testing process after which – if succeeded – they are ready for shipment. Lots not passing the test are sent to the *Failure Analysis Lab*.

Our study is concerned on the lot transfer operations inside the clean room that involve the 7 areas above (i.e., *Probe and Analytics* excluded).

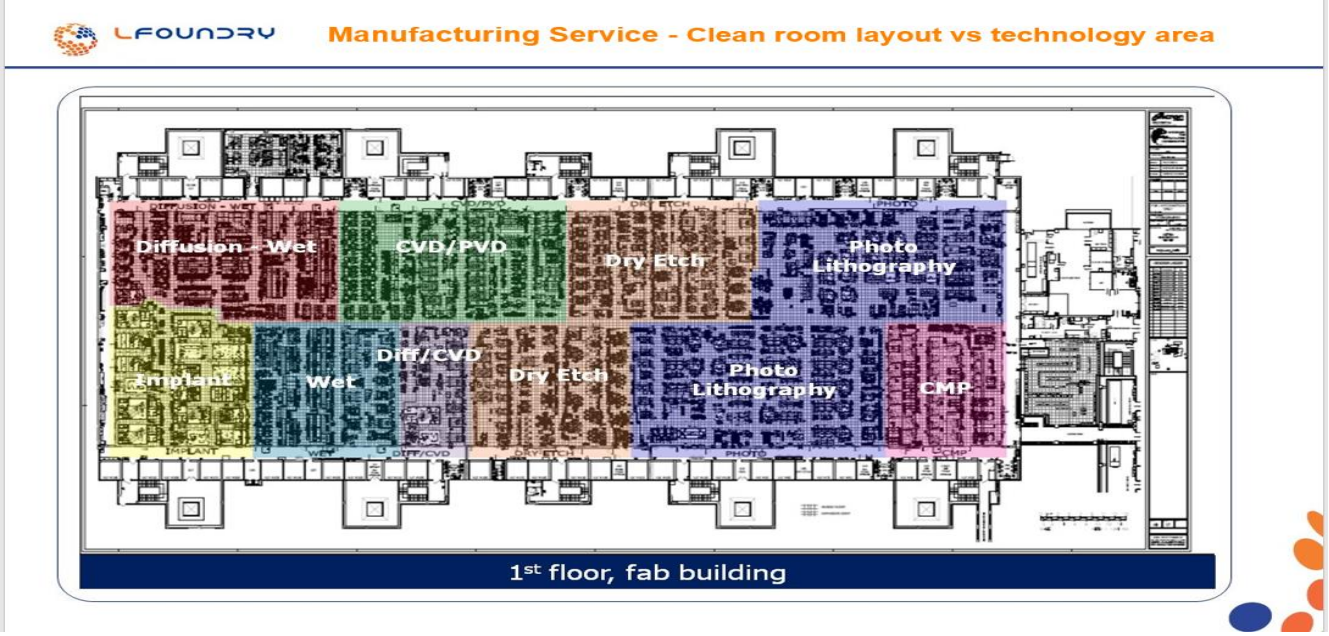


Fig. 2-4) Clean room layout vs technology area.

Machines can be tooled to perform specific operations. Each lot requires a large number of operations, typically from 250 to 750. A lot is moved 200 times at least before process end, and some move about 2000 times. This, broadly speaking, gives the manufacturing system the aspect of a (very big and complex) flexible job-shop.

A schematic view of the basic elements in the clean room is given by Fig. 2-5. We see:

1) A machine area divided into fourteen sectors.

2) An area dedicated to warehouses.

3) A corridor (or aisle) interposed between the two areas above.

Basically, the central aisle distributes production flows between warehouse and machine area, and within machine sub-areas. Machines are organized in fourteen dedicated sub-aisles, that lay orthogonally to the main one and are provided at their heads with a rack for in-process inventory. Lots are moved from rack to rack by a fleet of manually operated carts that commute back and forth along the main aisle, and by individual operators between the machines and the relevant head racks.

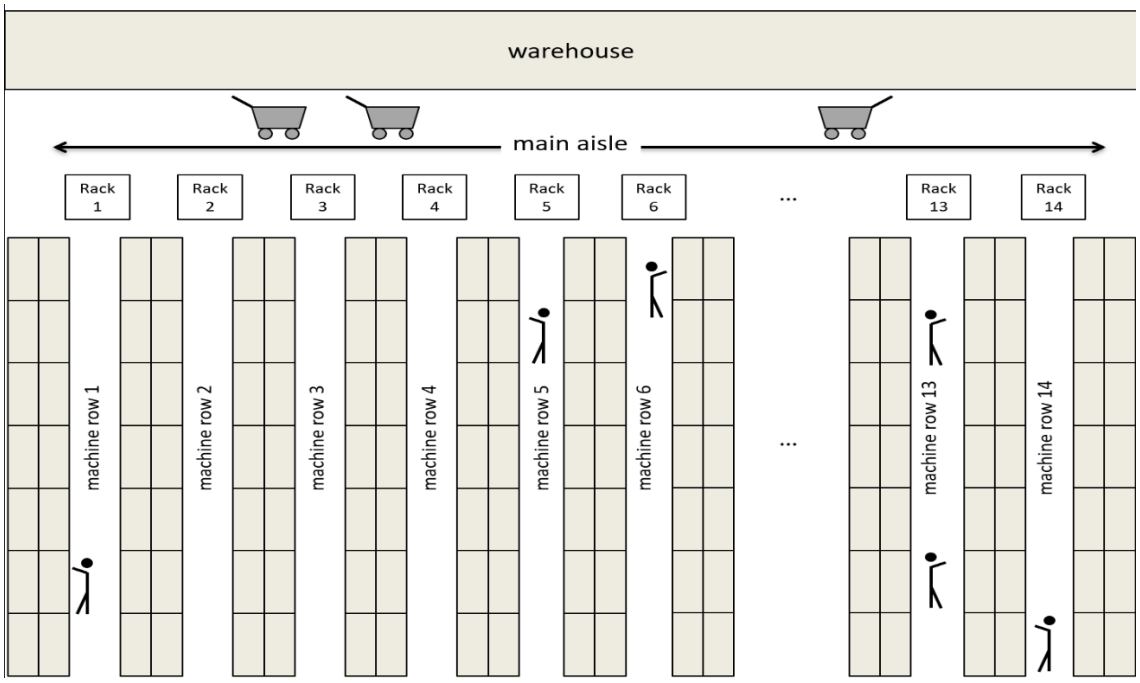
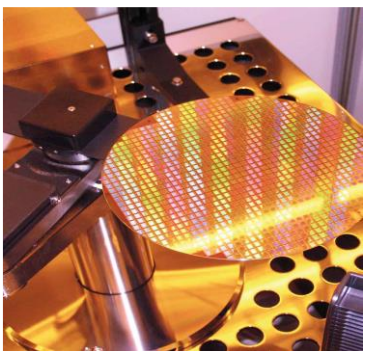


Fig. 2-5) Elementary representation of clean room organization and layout.

A production lot consist of a set of wafers closed inside a box. Boxes have different colors, and each color indicates a particular material group. For example, green indicates the group of metal, white no-metal and red indicates copper. Each lot has a Radio Frequency Identification Device (RFID) tag, which operators take advantage of in order to track the lot position.



(a)

(b)

(c)

(d)

(e)

(f)

Fig. 2-6) (a): a single wafer, (b): a lot (closed box), (c): a trolley for carrying lots, (d): a wafer carrier while carrying lots, (e): a wafer carrier when putting lots in the machine, (f): racks in the clean room.

Two distinct staff groups are responsible of lot transfers in the clean room: *wafer carriers* and *machine operators*. Wafer carriers are responsible for moving lots from rack to rack in the main aisle, machine operators move lots into or out from machines. In our study, we focus on wafer carriers and therefore on lot movements in the main aisle (rack 1 to 14 of the clean room). Fig. 2-7 shows the details of lot movement in the clean room.

At the beginning of every work shift, cart positions are determined and known: a parking space for carts is located between rack 3 and 4. The cart fleet normally consists of 8 vehicles, each one featuring an initial capacity of 12 lots. Rack groups are available for lot pick-up or drop, their positions being identified by 14 columns. A rack group is of 292 lots on average (minimum 70, maximum 550 lots): Fig. 2-8 shows rack state as it is stored in the factory information system. Racks are equipped with RFID ports, so any lot is RFID-trackable while in the racks. We collected the following measures that are relevant to lot transfer operation:

- distance between consecutive columns (that is, between rack groups) = 10 m
- average speed at which wafer carrier move =1.2 m/s (according to Company's reports)
- lot pickup time = about 10 s
- lot delivery time = 5 s

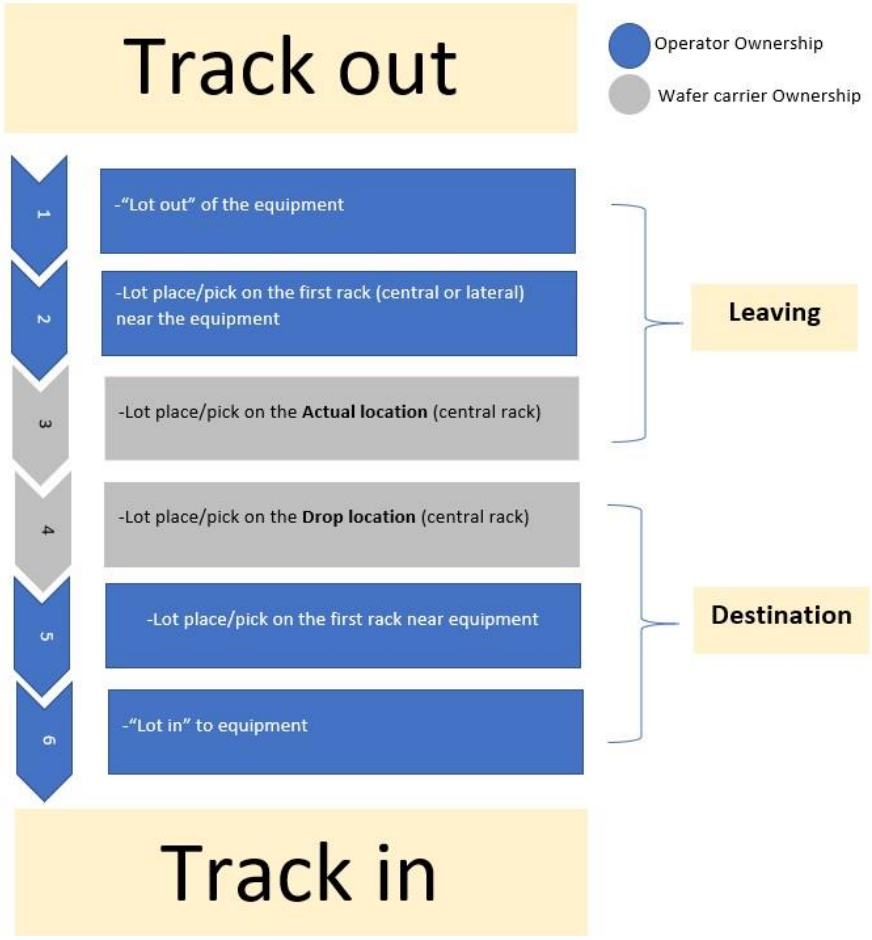


Fig. 2-7) Detail of lot transfer in the clean room.

H1067 =TRIM(MID(A1067,7,4))

	A	B	C	D	E	F	G	H	I	J	K
1	Interrogator	Rack Area desc	Rack Group	Rack Name	Shelf	capacity	TAG SCAN	CANALINA	RACK ID	SHELVES ID	MONTA TAG SCAN (S/N)
1054	s112dr0420p	WEB13	WETBOX10	D10	11..15	5	112d	0420	WEB13-D10	WEB13-D10-11..15	N
1055	s112dr0423p	WEB13	WETBOX10	D10	16..20	5	112d	0423	WEB13-D10	WEB13-D10-16..20	N
1056	s112dr0421p	WEB13	WETBOX10	D10	21..25	5	112d	0421	WEB13-D10	WEB13-D10-21..25	N
1057	s112dr0231p	WEB13	WETBOX10	D10	26..30	5	112d	0231	WEB13-D10	WEB13-D10-26..30	S
1058	s1127r0447p	WEB13	WETBOX23	S01	01..05	5	1127	0447	WEB13-S01	WEB13-S01-01..05	N
1059	s1127r0197p	WEB13	WETBOX23	S01	06..10	5	1127	0197	WEB13-S01	WEB13-S01-06..10	N
1060	s1127r0442p	WEB13	WETBOX23	S01	11..15	5	1127	0442	WEB13-S01	WEB13-S01-11..15	N
1061	s1127r0198p	WEB13	WETBOX23	S01	16..20	5	1127	0198	WEB13-S01	WEB13-S01-16..20	S
1062	s1127r0441p	WEB13	WETBOX23	S01	21..25	5	1127	0441	WEB13-S01	WEB13-S01-21..25	N
1063	s1127r0229p	WEB13	WETBOX23	S01	26..30	5	1127	0229	WEB13-S01	WEB13-S01-26..30	N
1064	s1112r0561p	WEB13	WETBOX23	S02	01..05	5	1112	0561	WEB13-S02	WEB13-S02-01..05	N
1065	s1112r0562p	WEB13	WETBOX23	S02	06..10	5	1112	0562	WEB13-S02	WEB13-S02-06..10	N
1066	s1112r0563p	WEB13	WETBOX23	S02	11..15	5	1112	0563	WEB13-S02	WEB13-S02-11..15	N
1067	s1112r0537p	WEB13	WETBOX23	S03	06..10	5	1112	0537	WEB13-S03	WEB13-S03-06..10	N

Fig. 2-8) Rack status as reported in the fab information system.

2.3 *LFoundry* scheduling system

To manage transportation requests as well as operations on lots and any information relevant to production, the plant is equipped with a complex information system. This is the *Manufacturing Execution System* (MES) and is responsible, among other tasks, of assigning each operation that any lot must undergo to one among the various machines capable of performing it. The assignment considers the occupancy status of the machines available for the operation, the expected duration of the operation, the possible need of preliminary machine set-ups and other essential constraints or alerts for the correct performance of production, such as precedence, time windows (ready times and due dates), urgency, etc.

Since each lot must undergo a prescribed sequence of operations on different machines, the MES must continuously manage a set of transfer requests from one rack in the corridor to another. To manage such material flows basically means for the MES to assign the received requests to the carts in a certain time frame, and to ensure that the relevant transfer operations are carried out. The multiple ways to achieve this task, although equivalent from a functional point of view, are not at all equivalent in terms of efficiency.

Controlling WORK-IN-PROCESS (WIP) is one of the crucial issues to be considered in order to optimize flow management. Lots not currently worked by some machine, wait for operation in the racks positioned in the main aisle. The RFID system continuously feed the information system with the actual position of every lot waiting for or worked by a machine. These conditions suggest a proper vehicle routing/scheduling system performance and reduce the WIP.

The former *LFoundry* scheduling system was called LOT DISPATCHING INTERFACE (LDI). Dispatching here indicates a procedure used to assign clients to tools when the client calls for it (e.g., Calling for Taxicab). LDI is a dispatch list that all lots at current step are check validated, its rules have built in real-time. It produces a prioritized list of lots for each area and workstation; objects are lots that are assigned due dates, but with fast workstation LDI cannot keep the tools fully loaded.

Recently, *LFoundry* has updated the scheduler by introducing a new system called MICRON ACCELERATED CYCLE-TIME HETERODOXY (MACH), which improves planning tools and scheduling applications, implements changes in policies and procedures, and enhancements to databases and data collection used to reduce CYCLE TIME (CT). The MACH Scheduler (Fig. 2-9) manages the WIP, not the lots, determines how many lots of each part type-step should be completed in the current shift, and assign lots to individual tools.

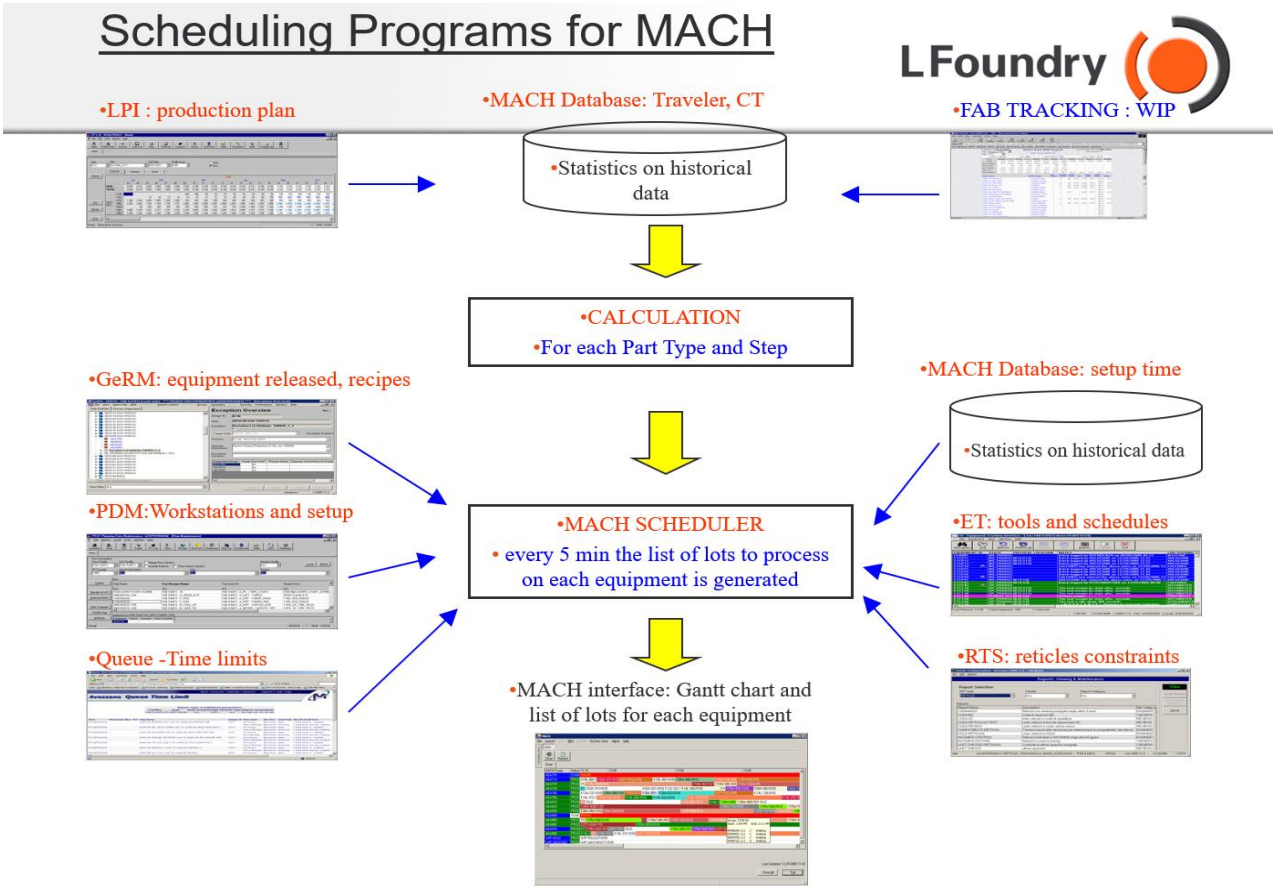


Fig. 2-9) The MACH Scheduler in use at LFoundry.

However, lot transfer procedures have not been updated. In the past, a production with slow changes recommended to decide for a fixed cart routing, obtained via statistics identifying the most frequent transportation requests. On this basis, three static routes were defined: the longest, covering all the fourteen stations in the main aisle; the shortest, covering its central part; and an intermediate one, strictly contained in the former and strictly containing the latter. The cart fleet was then three-partitioned and statically assigned to those routes. But, due to changes in market positioning and end user of products, *LFoundry* has recently moved from mass production with large inventory levels to one oriented to individual customer needs, where smaller lots of different types are due by specified time ends. The objectives of production planning were consequently reviewed, and, in this perspective, the production scheduling system needs to be completely redesigned.

The problem posed by the Company therefore consists in optimizing appropriate quality indicators associated with the movement and loading and unloading of lots in production, indicators that

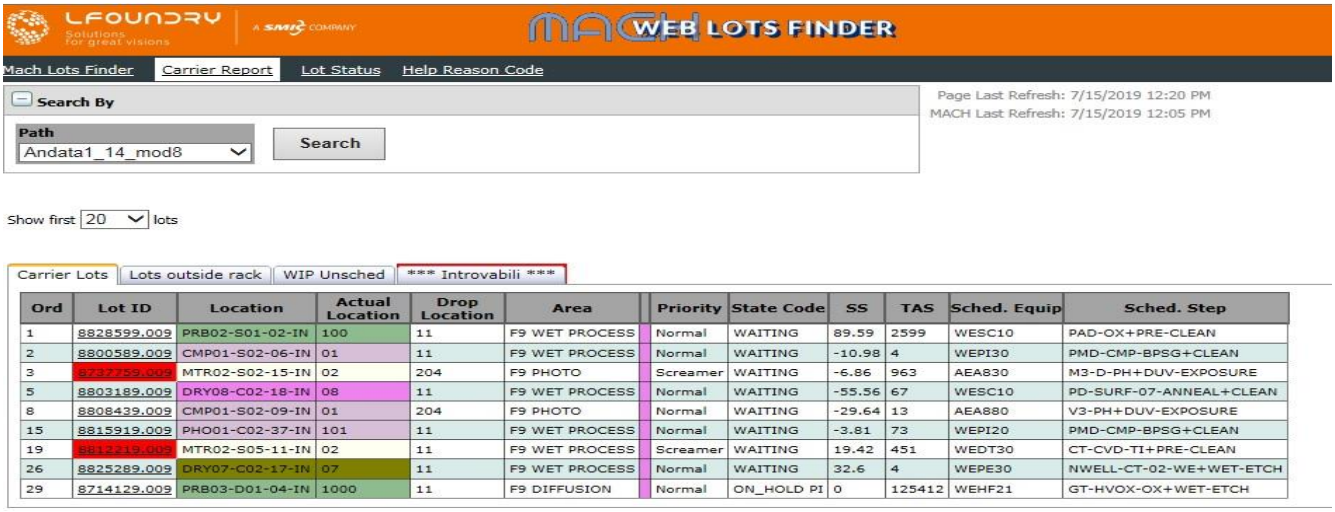
essentially consider the time required to complete the required handling operations. With the new situation, the *LFoundry* technical staff agreed to redesign the material handling system to dynamically adapt cart routes and schedules to requests changes over time.

Summarizing, fleet management calls for finding optimal cart routes and schedules in accordance to newly designed objectives. This is the goal posed to the present research.

2.4 Data from the plant

According to the Company's view, we focused on wafer carriers' movements in the clean room. Our goal consists in suggesting a better scheduling for carts movement in the clean room. In order to reach this goal, we started by investigating wafer carriers' reports accessed from a server that the Company made available for our use. The resident database contains a list of snapshots that describe, minute by minute, the complete state of the lots. From this information it is also possible to deduce the wafer carriers' movement in the clean room. The necessary information we extracted from *LFoundry* data base was relevant to the actual and drop locations of the lots involved in each snapshot. This information basically forms the list of requests that must be fulfilled with the available carts (see Fig. 2-10). Those request lists will form the main input to our model.

Our observations summarize to a mean of 82 requests per snapshot. On average, 55.7% of the requests must travel short distance (meaning less than 6 columns) and 1.8% must travel long distance (including between columns 1 and 14).



Ord	Lot ID	Location	Actual Location	Drop Location	Area	Priority	State Code	SS	TAS	Sched. Equip	Sched. Step
1	8828599.009	PRB02-S01-02-IN	100	11	F9 WET PROCESS	Normal	WAITING	89.59	2599	WESC10	PAD-OX+PRE-CLEAN
2	8800589.009	CMP01-S02-06-IN	01	11	F9 WET PROCESS	Normal	WAITING	-10.98	4	WEPI30	PMD-CMP-BPSG+CLEAN
3	8737759.009	MTR02-S02-15-IN	02	204	F9 PHOTO	Screamer	WAITING	-6.86	963	AEA830	M3-D-PH+DUV-EXPOSURE
5	8803189.009	DRY08-C02-18-IN	08	11	F9 WET PROCESS	Normal	WAITING	-55.56	67	WESC10	PD-SURF-07-ANNEAL+CLEAN
8	8808439.009	CMP01-S02-09-IN	01	204	F9 PHOTO	Normal	WAITING	-29.64	13	AEA880	V3-PH+DUV-EXPOSURE
15	8815919.009	PHO01-C02-37-IN	101	11	F9 WET PROCESS	Normal	WAITING	-3.81	73	WEPI20	PMD-CMP-BPSG+CLEAN
19	8812219.009	MTR02-S05-11-IN	02	11	F9 WET PROCESS	Screamer	WAITING	19.42	451	WEDT30	CT-CVD-TI+PRE-CLEAN
26	8825289.009	DRY07-C02-17-IN	07	11	F9 WET PROCESS	Normal	WAITING	32.6	4	WEPE30	NWELL-CT-02-WE+WET-ETCH
29	8714129.009	PRB03-D01-04-IN	1000	11	F9 DIFFUSION	Normal	ON_HOLD PI	0	125412	WEHF21	GT-HVOX-OX+WET-ETCH

Fig. 2-10) Wafer carrier reports in the clean room.

2.5 Vehicle routing models (generalities)

On-demand transport – DEMAND RESPONSIVE TRANSPORT (DRT), DIAL-A-RIDE, or even PARATRANSIT – is a sustainable mobility tool implemented in various realities to support local public transport systems. It exploits a fleet of small vehicles that allow customized travel based on user requests, with origin and destination chosen each time, transporting a certain number of users at a time, and managing the concatenation of routes with a certain level of flexibility to be able to cover all requests.

These on-call services are very effective in satisfying the mobility needs of homogeneous groups of users (disabled, elderly, students, tourists, etc.), especially in areas that are not served by other systems or in times of low demand. In fact, the intensification of scheduled services is not always a viable solution, because it tends to produce huge costs for the transport companies and the municipalities involved and to generate externalities, difficult to quantify but not negligible, in terms of pollution, road wear, safety and increased congestion. On-call systems, on the other hand, make it possible to create a wide range of intermediate services between the taxi and the bus. In basic cases, users can only book a trip between predefined stops. In the most advanced cases, the service is carried out door to door and can also be requested in real time.

The on-call model can also be applied to freight transport: in this scenario the requests to the system represent the movement of one or more units of goods from one location to another, and the main difference is played by the quality-of-service requirements (which obviously consider different indicators for goods and people). The theme addressed in this work starts from a problem of industrial logistics, which falls within the DIAL-A-RIDE type for load transport.

Transport services are typically associated with combinatorial optimization problems classified under the general term of VEHICLE ROUTING PROBLEM (VRP). The situation described in the previous sections naturally leads to a particular VRP called DIAL-A-RIDE PROBLEM (DARP). In general terms, a DARP solution is required to match, in an optimal way with respect to appropriate cost functions, n individual travel requests using a fleet of m vehicles. In its practical applications, the DARP is configured as a multicriteria optimization problem with conflicting objectives, such as:

- offer a quality service to users;
- serve as many requests as possible;
- reduce operating cost to a minimum.

Operating costs are mostly related to fleet size and distance traveled, while user discomfort is often measured in terms of time: time to reach each user destination and, for the driver, excessive driving time. A common technique for determining the non-dominated (Pareto-optimal[3]) solutions of a multi-objective problem consists in tracing it back to the repeated solution of single-objective constrained problems: one chooses one among the objectives and the others are treated through appropriate inequalities that decision variables are called to satisfy. For example, a minimum cost solution can be sought among those that serve at least a certain number of requests and guarantee a level of quality not lower than a prescribed threshold.

The problem can be static or dynamic depending on the time horizon in which travel requests are collected and the related information on the instants required for picking up and reaching the destination. In general terms, the problem is considered static when the solution satisfies requests collected in a relatively long-time interval and dynamic when the solution is recalculated and updated every time a new request (or a small set of requests) arises with very close timelines.

The DARP generalizes a complex VEHICLE ROUTING PROBLEM WITH PICKUP AND DELIVERY (VRP-PD), and as such it is in turn computationally difficult. On the other hand, it is a generalization of the TRAVELING SALESMAN PROBLEM WITH TIME WINDOWS (TSP-TW). So, calculating an optimal solution of a DARP is NP-hard. Consequently, many methods developed in the literature are of heuristic type. However, if the problem is limited in size, it is possible to devise exact solution methods based on mathematical programming models.

The particular DARP examined in this work belongs to the class of PICKUP AND DELIVERY PROBLEMS WITH TIME WINDOWS (PDP-TW), i.e., routing and scheduling problems where each transportation request is represented by a pair of nodes of a graph (pick-up and delivery) and is associated with a time window whose extremes specify the minimum pick-up time and the maximum desired (or required) time for delivery. The problem consists in choosing, for each vehicle, a route that serves the requests assigned to it in compliance with time constraints (and, where present, load capacity), minimizing an appropriate global cost function.

The cost function to be minimized can be linked to delivery dates (due-date related), and in this case it can, for example, measure the maximum delay of a delivery (maximum lateness, L_{max}); or the – possibly weighted – sum of the delays incurred; or the – again, possibly weighted – number of late

[3] ***Pareto optimality*** *is a situation where no individual or preference criterion can be better off without making at least one individual or preference criterion worse off or without any loss thereof.*

deliveries. Where appropriate, it can instead be released from due dates, and take for instance the form of the maximum time (C_{max}) necessary for the fleet to carry out all the required deliveries. The size of the fleet needed to make deliveries can be another cost factor to be considered. In this case, the problem requires to establish the minimum number (and possibly type) of vehicles needed to fulfill transport requests with a given quality assurance.

The problem arising in *LFoundry* is a DARP with homogeneous fleet consisting of identical vehicles with finite capacity. Also, the capacity of racks at the possible origins and destinations can be considered – and this adds to the problem a novel feature not considered, to the best of our knowledge, in the literature.

For this problem we will consider both a static and a dynamic scenario. The time horizon is in fact long enough to prefer the second scenario, but a decomposition that locally solve and concatenate static instances that group a reasonably large number of requests can indeed be afforded with the computational resources available.

3 Background: optimization models and methods

This chapter contains basic concepts on solution methods for general integer linear optimization problems and for some combinatorial problems that revealed to be of interest for the case studied. The chapter can easily be skipped by readers with ground skills in combinatorial optimization. The theme is subdivided in two parts. First, we survey some fundamental concepts of Mathematical Programming, Linear Programming, Mixed Integer Programming, Branch-and-Bound, and the Cutting Planes Approach. Then, we discuss the main models and algorithmic techniques used in our research: these include such models as the VRP, the DARP, the MATCHING (ASSIGNMENT) PROBLEM, the BOTTLENECK MATCHING PROBLEM, and such methods as the Hungarian and the Bottleneck Matching algorithm.

3.1 Mathematical programming

MATHEMATICAL PROGRAMMING (MP) is the field of mathematics dealing with the theory and methods for determining the extreme values of functions on sets specified by linear and non-linear constraints (equalities and inequalities) in a finite-dimensional vector space. In turn, MP is a branch of Operations Research that encompasses a broad range of methods using finite-dimensional extremum problems as mathematical models.

MP issues have applications in a variety of fields of human activity where one of several options must be chosen, such as in handling multiple problems of control and planning of manufacturing processes, as well as in design and long-term planning. The word "programming" refers to the objective of addressing diverse issues by selecting action programs [1].

3.1.1 Linear programming

LINEAR PROGRAMMING (LP) is a subsection of MP. In a generic LP problem, a cost vector $c = (c_1, \ldots, c_n)$ is supplied with the goal of minimizing a linear cost function $c'x = \sum_{i=1}^{n} c_i x_i$ across all n-dimensional vectors $x = (x_1, \ldots, x_n)$, subject to a set of linear equality and inequality constraints. The following is an example of a linear programming problem: calculate the minimal value of a linear function, that is, find

$$min\,(\sum_{j=1}^{n} c_j x_j) \qquad \textit{(Eq. 3-1)}$$

under following conditions:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad i = 1,2,\ldots,m \qquad \textit{(Eq. 3-2)}$$

$$x_j \geq 0 \quad j = 1,2,\ldots,n \qquad \textit{(Eq. 3-3)}$$

where the coefficients c_j, a_{ij}, and b_i are known real numbers. Equations ((Eq. 3-2) and (Eq. 3-3)) are the algebraic description of an n-dimensional convex polyhedron.

The variables $x_1, \ldots, x_n$ are referred to as *decision variables*, and a vector x satisfying all constraints is referred to as a *feasible solution*. The *feasible set* or feasible area is the set of all possible feasible solutions and coincides with the said polyhedron. The *objective function* is defined as $c'x = \sum_{i=1}^{n} c_i x_i$.

An *optimal solution*, when one exists, is a feasible solution x^* that minimizes the objective function; that is, $c'x^* \leq c'x$ for every feasible x.

LP is used in a wide variety of situations [2]. For example, to find optimal production levels in manufacturing facilities where decisions must be made to optimize overall income while also fulfilling material availability and output needs. Further applications may be found in organizations' multiperiod planning processes: for example, an electric power plant must select whether to grow electric capacity based on coal or nuclear power plants, and the lowest cost capacity expansion plan is required. Linear programming is also used in transportation and communication networks, such as the multi commodity flow problem and pattern classification problem [3].

After the fundamental contribution of L. Khachiyan in 1979, LP problems have been recognized to be solvable in polynomial time if defined by rational coefficients. The algorithm, known as the *Ellipsoid Method*, is a general tool to solve convex optimization problems. Before that, LP was solved by the celebrated *Simplex Method* originally developed by G.B. Dantzig. Differing from the former, which successively improves interior points of the polyhedron, the Simplex Method proceeds by extreme points corresponding to the vertices of the polyhedron (algebraically expressed by *basic feasible solutions*) which are finitely many and can be associated with subsets of columns of the matrix describing the polyhedron when in standard form. Although not proved to converge in polynomial time in any case, the Simplex Method is very efficient in practice and its performance is often comparable to that of interior points methods. With respect to those algorithms, it also offers the capability of recovering the feasibility of a basis by simple dual pivots. In any case, efficient algorithms for LP play a crucial role in methods designed to solve more difficult problems, such as Mixed Integer Programming.

3.1.2 Mixed Integer Programming

The general framework for MIXED INTEGER PROBLEMS (MIPs) has the following aspect:

$$max\ \ (cx + hy) \qquad \textit{(Eq. 3-4)}$$

subject to,

$$Ax + Gy \leq b \qquad \textit{(Eq. 3-5)}$$

$$x \geq 0 \textit{ and integer} \qquad \textit{(Eq. 3-6)}$$

$$y \geq 0 \qquad \textit{(Eq. 3-7)}$$

Here $c = (c_1, \ldots, c_n)$, $h = (h_1, \ldots, h_p)$ are real row vectors, $A = (a_{ij})$ is an $m \times n$ real matrix, $G = (g_{ij})$ is a $m \times p$ real matrix, and $b = (b_1, \ldots, b_m)'$ is an m-dimensional column vector. The objective function and all restrictions are linear; the variables x_i must be nonnegative integers, while the variables y_i can take any nonnegative real value. Clearly, for $p = 0$ one obtains a pure LP, that may then be regarded as a particular instance of a MIP.

A feasible solution x is one that meets all restrictions. The collection of feasible solutions can be formally defined as follows:

$$S \coloneqq \{(x, y) \in Z_+^n \times R_+^p : Ax + Gy \leq b\} \quad \textit{(Eq. 3-8)}$$

A particular case is Mixed 0-1 LP, where integer variables can only get values 0 or 1:

$$S \coloneqq \{(x, y) \in \{0,1\}^n \times R_+^p : Ax + Gy \leq b\} \quad \textit{(Eq. 3-9)}$$

Pure 0-1 LP is in particular useful to model Combinatorial Optimization problems.

There is a clear distinction between the *optimization* and the *feasibility problem*. In the first we want to determine a solution in the feasible region, if one exists, that corresponds to the best possible value of the objective function (minimum or maximum). The second problem, instead, just calls for finding a point (if any) in the feasible set. The methods developed over time to tackle these problems employ two main techniques: *branch-and-bound* and *cutting plane*. Both techniques exploit the fact that LP solution is algorithmically practical. In fact, the best solution of a problem derived from a MIP by removing integrality clauses gives a (lower or upper) bound to the best value achievable in the original MIP. The LP so obtained is called the *linear relaxation* of the MIP, and the difference between its optimal solution and the best feasible mixed-integer solution of the MIP found at any time during computation, is called the *optimality gap*. This gap can be computed by the Simplex Method or any other LP interior point method and is generally used in branch-and-bound (see Section 3.1.3) to estimate the distance from the optimum of a primal solution and to possibly halt enumeration.

For instance, in the binary case of 0-1 LP, $x \in \{0,1\}$ is replaced by $0 \leq x \leq 1$. The original feasible set of the MIP is then contained in the convex polyhedron

$$P_0 \coloneqq \{(x, y) \in R_+^n \times R_+^p : Ax + Gy \leq b\}$$

and the linear program $max\{cx + hy : (x, y) \in P_0\}$ is obtained as the linear programming relaxation of.(Eq. 3-4) - (Eq. 3-7) [4].

Fundamental tools of decision making [5], MIP models can however be either too weak or too large to be solved efficiently and call for sophisticated MIP techniques [6]. In fact, the same problem can be

formulated by (potentially infinite) different MIP models, each one with different linear relaxation and gap. In particular, the best relaxation of a 0-1 LP corresponds to the polyhedron identified by the convex hull of its feasible (0-1) solutions and is characterized by a null optimality gap: that is, it directly provides an integer optimum. Unfortunately, finding this formulation is non-trivial in general, although nice theoretical results can help in some cases (as e.g., the MATCHING PROBLEM, see Section 3.4). An inclusion-wise sequence of linear relaxations can however be obtained by successive additions of valid linear inequalities whose origin (a hyperplane) cuts the polyhedron of the current relaxation: this procedure is at the basis of the Cutting Plane method, see Section 3.1.4.

3.1.3 Branch-and-Bound

Branch-and-bound is a well-studied approach to solve MIP problems. The algorithm takes the MIP formulation as its input and removes initially all integrality requirements. As said in Section 3.1.2, this operation produces an LP relaxation, the solution of which provides an optimal solution that may or may not be feasible for the MIP. In the former case, that is if the solution meets all the initial MIP criteria, it is returned as MIP-optimal, and the procedure terminates. Otherwise, a decision variable constrained to integrality is chosen and a dichotomy operated. Let us describe this operation by a simple example.

Suppose that in the optimum of the LP relaxation some (originally binary) variable x is set to 0.7. The feasible region of the original MIP, P_0, is then split into two sub-regions, P_1 and P_2 according to the value assumed by x, which is 0 in P_1 and 1 in P_2. Problems P_1 and P_2 are solved to their optimality. As a result, the original optimization problem, defined in P_0, is replaced by two MIP problems, P_1 and P_2, none of which include the LP optimum, which is MIP-infeasible and is therefore cut off.

As the described dichotomy procedure is associated to a fractional variable and produces two distinct MIP sub-problems branching, it is also called *variable branching* [4], and successive branching results in a search tree as shown in Fig. 3-1. The MIPs produced during the search are associated with tree nodes, in particular P_0 corresponds to the root node. At any time during computation, the tree leaves indicate MIPs that are not solved yet.

Searching the tree requires to define logic used to process individual nodes and attempt to solve the corresponding MIPs. Suppose we are solving a minimization problem and are currently exploring a node in the search tree. The solution of the relevant LP relaxation has four possible outcomes:

1. If all integrality constraints are satisfied, a feasible solution to the original MIP has been discovered. This node is then labeled as fathomed and does not need to be further branched.

The feasible solution acquired here is then compared to the best feasible solution found so far (so *called incumbent*): if it improves the incumbent, then replaces it as the new incumbent, and we go ahead with the search by selecting another node.

2. Another possibility is that the branch that led to the present node includes a constraint that causes the infeasibility of the LP relaxation, and consequently of the MIP associated with the node. Also in this case, the node is fathomed and the search proceeds with a different node.
3. In the third case, the optimum value of the LP relaxation exceeds that of the incumbent. The MIP associated to the current node cannot clearly produce a better integral solution, and therefore is fathomed also in this case.
4. Finally, it might be the case that the optimum value of the LP relaxation is lower than that of the incumbent. In this case the node cannot be fathomed and becomes the local root of a new branch.

Summarizing, a branch in a search tree can be cut, or stopped of being investigated, in three circumstances:

- An integer solution is locally found at the node.
- The node corresponds to an infeasible subproblem.
- The best potential objective value cannot improve the incumbent.
- To conclude our excursion, two more issues must be illustrated. First, an incumbent solution is found, its value gives an upper bound to the optimal solution of the MIP (supposing it a minimization problem). On the other hand, LP relaxations continuously provide and update a lower bound during the search, determined by taking the minimum among all the current LP relaxations values at the leaf nodes. As said, the optimality gap is the difference between the best upper and lower bounds found so far. When the gap equals zero, we have achieved at optimality.
- To summarize, **Algorithm 1** (Fig. 3-2) presents a pseudo-code of a branch-and-bound algorithm as applied to a minimization MIP. Let $\mathcal{L}$ be the number of nodes that still need to be solved and $\underline{z}$ a lower bound to the optimum z^*. The initial upper bound can be obtained via a heuristic solution or set to $-\infty$.

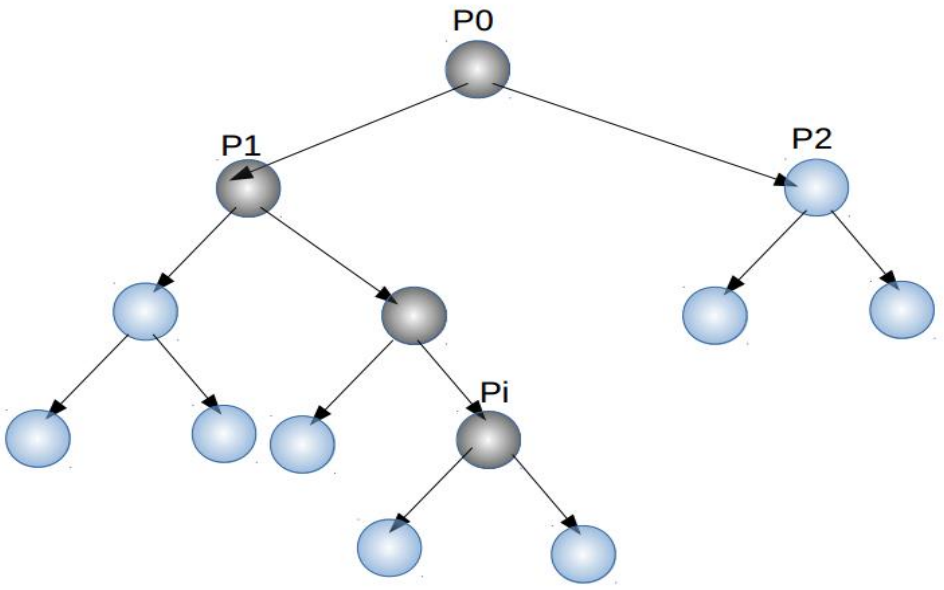


Fig. 3-1) Branch-and-Bound search tree.

1 **Initialize**: $\mathcal{L} := \{N_0\}$, $\underline{z} := -\infty$, $(x^*, y^*) := \emptyset$.
2 **Check**,
If $\mathcal{L} = \emptyset$, then the solution (x^*, y^*) is optimal.
3 **Select node**: Choose a node N_i in $\mathcal{L}$ and delete it from $\mathcal{L}$.
4 **Bound**: Solve LP_i. If it is infeasible, go to Step 2. Else, let (x^i, y^i) be an optimal solution of LP_i and z_i its objective value.
5 **Prune**:
If $z_i \leq \underline{z}$, go to Step 2.
If (x^i, y^i) is feasible to MIP, set $\underline{z} := z_i$, $(x^*, y^*) := (x^i, y^i)$ and go to Step 2.
Otherwise
6 **Branch**: From LP_i, construct $k \geq 2$ linear programs $LP_{i_1}, ..., LP_{i_k}$ with smaller feasible regions whose union does not contain (x^i, y^i), but contains all the solutions of LP_i with $x \in \mathbb{Z}^n$.
Add the corresponding new nodes $N_{i_1}, ..., N_{i_k}$ to $\mathcal{L}$ and got to Step 2.

Fig. 3-2) Branch-and-Bound Algorithm.

Let us conclude by observing that, in general, three important issues must be addressed to improve the performance of a branch-and-bound method:

- Formulation (to obtain a gap as small as possible)
- Heuristics (to determine an appropriate upper bound)
- Node selection for branching [7].

3.1.4 Cutting plane technique

Adding cutting planes to the polyhedron associated with a branch-and-bound node is a method for tightening the formulation and thus solving the MIP problem faster. Consider the MIP provided in (Eq. 3-4) - (Eq. 3-7), and recall the set $S \coloneqq \{(x, y) \in Z_+^n \times R_+^p\colon Ax + Gy \leq b\}$. Suppose that (x_0, y_0) is the best solution in the polyhedron P_0 of the LP relaxation of S, but is MIP-infeasible. The cutting plane technique is based on finding an inequality $\alpha x + \gamma y \leq \beta$ that is satisfied by every point in S and such that $\alpha x_0 + \gamma y_0 > \beta$. That is, a cutting plane removes (x_0, y_0) from S because it violates the associated inequality. Define

$$P_1 \coloneqq P_0 \cap \{(x, y)\colon \alpha x + \gamma y \leq \beta\}. \qquad \textit{(Eq. 3-10)}$$

Because $S \subseteq P_1 \subseteq P_0$, the LP relaxation of the MIP based on P_1 is better than the original LP relaxation, in the sense that the optimal value of P_1 is at least as good an upper-bound on the value z^* as the original LP relaxation. The cutting plane method results from the recursive application of this approach.

A cutting plane approach could fit within a branch-and-bound method and provide tighter upper bounds during the tree search. This gives rise to the *branch-and-cut method*, which is one of the foremost effective strategies for tackling mixed-integer programs. Its integration in **Algorithm 1** (Fig. 3-1) is obtained by inserting a cutting-plane step (**Algorithm 2** (Fig. 3-2)) before the branching step. In practice, many cuts are often added at the root node N_0, whereas fewer cuts are added at in-depth nodes of the search tree [4].

1 **Initialize**: $i = 0$
2 **Recursive Step.** Solve the linear program $\max\{cx + hy : (x, y) \in P_i\}$.
If the associated optimal solution (x^i, y^i) belongs to S, stop.
Otherwise, solve the *separation problem*:
Find a cutting plane $\alpha x + \gamma y \leq \beta$ separating (x^i, y^i) from S.
Set $P_{i+1} := P_i \cap \{(x, y) : \alpha x + \gamma y \leq \beta\}$ and repeat the recursive step.

Fig. 3-3) Cutting Planes Algorithm.

3.2 Vehicle Routing Problem

The VEHICLE ROUTING PROBLEM (VRP) is a combinatorial optimization (and hence 0-1 LP) problem that requires to find an *optimal set of routes* and assign them to a fleet of vehicles with the operational constraint of serving a given set of customers scattered in different locations of a geographic area. In

the VRP, the quality of a solution is measured by the total distance covered by the fleet, in turn computed after a given *distance matrix*. In this sense the problem generalizes the well-known TRAVELING SALESMAN PROBLEM (TSP), where the fleet consists of a single carrier.

Determining an optimal solution for VRP is an NP-hard problem, and this limits the size of problems that can be optimally solved. For this reason, due to the size and frequency of real world VRPs, dedicated solvers tend to use heuristics.

The routes forming a solution are classically assumed to start and end in the same location, called the vehicle *depot* (Fig. 3-4). In typical applications, the distance matrix is measured on road network described by a graph whose arcs represent the roads and whose vertices include customers' locations and the depot. In this model, each arc has an associated cost that normally corresponds to the length or travel time of the road segment (with the latter possibly depending on vehicle type).

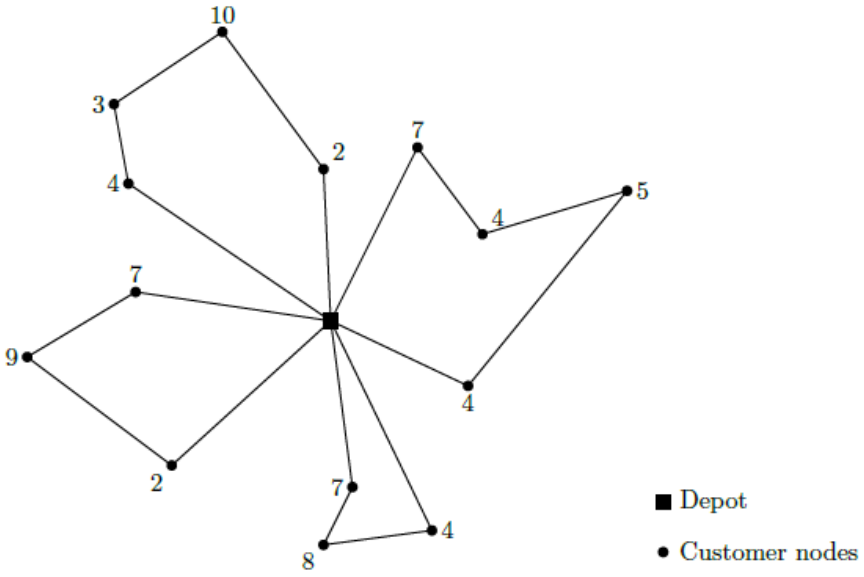


Fig. 3-4) Possible routes that form a solution of a Vehicle Routing Problem.

Sometimes it is impossible to fulfill all customer requests; in these cases, the solution algorithm can reduce the request set and leave some customers unserved. To address such situations, it is possible to introduce customer priorities and/or associate penalties for missed customer service.

The objective function of a VRP can vary a lot depending on application. Typical goals are:

- Minimize the overall transport costs based on the total traveled distance and possible fixed costs associated with vehicles and drivers.
- Minimize the number of vehicles needed to serve all customers.

- Minimize the largest variation of vehicles' travel time and load.

Fig. 3-5 reports the main VRP variants and specializations studied in the literature. Their classification begins with a base case called CAPACITATED VEHICLE ROUTING PROBLEM (CVRP): a set of customers must be served using a fleet of homogeneous vehicles with identical finite capacities. One wants to find a depot-centered route per vehicle minimizing the total traveled distance and respecting vehicle capacities. In the DISTANCE-CONSTRAINED CAPACITATED VEHICLE ROUTING PROBLEM (DCVRP), each vehicle is imposed an upper limit to the distance traveled. If time constraints for visiting nodes are added to the CVRP, then the VEHICLE ROUTING PROBLEM WITH TIME WINDOWS (VRPTW) is obtained. The VEHICLE ROUTING PROBLEM WITH BACKHAULS (VRPB) is an extension in which customer nodes are partitioned into two subsets: *linehaul* nodes, to which resources taken from the depot must be delivered; and *backhaul* nodes, from where resources must be withdrawn and brought back to the depot. Backhaul nodes can only be served after serving linehaul nodes. Finally, in the VEHICLE ROUTING PROBLEM WITH PICKUP AND DELIVERY (VRPPD), each customer is associated with a *pickup* node, where some quantity of resource is to be withdrawn, and a *delivery* node, where the same resource must eventually be taken after picking it up.

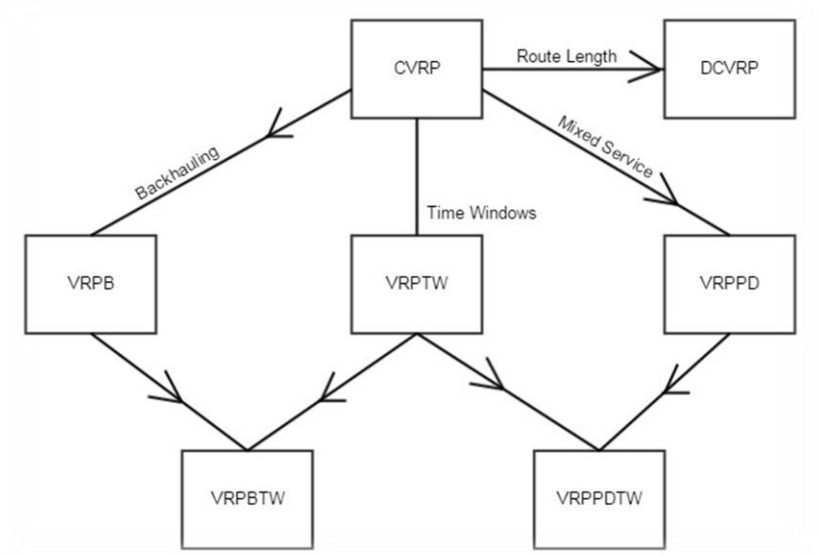


Fig. 3-5) The different types of VRP.

There are three main approaches to VRP modeling:

- Vehicle Flow Formulations. Models of this type of associate integer variables with arcs, each counting how many times the arc is crossed by a vehicle. This approach is generally

used for basic VRPs and is convenient when the solution cost is expressed by the sum of all arc costs but becomes unpractical for more complex applications.

- Commodity Flow Formulations. Further (integer) arc-variables are added to represent the flow of goods along the routes crossed by the vehicles.
- Set Partitioning Formulations. These models have 0-1 decision variables, each defining a feasible route, that is, the set of customer nodes touched by the route. The VRP is therefore formulated as the problem of partitioning the customer nodes into subsets, each to be served by a suitable route. As there are 2^n distinct subsets of a given subset of n elements, the model can entail exponentially many decision variables.

Due to the difficulty of finding optimal solutions to large-scale VRP instances, an important research effort has been devoted to (meta)heuristics. Some of the newest and most efficient metaheuristics for vehicle routing problems achieve solutions within 0.5% or 1% gap for problem instances with hundreds or even thousands delivery points. Also, metaheuristics are often privileged for large-scale applications with complicated constraints and series of decisions.

3.3 The Dial-a-ride problem

PICKUP AND DELIVERY PROBLEMS (PDP) form an important class of VRPs in which freight or people are to be transported from given origins to given destinations. Most of the existing PDPs are encompassed by the general picture of Fig. 3-6 and share the following notation.

Let $G = (V, A)$ be a complete directed graph with node set $V = \{0, \dots, n\}$ and arc set $A = \{(i,j): i,j \in V, i \neq j\}$. Node 0 represents the *warehouse* (or *depot*) and the other represent the customers. Each arc $(i,j) \in A$ is associated with a non-negative cost c_{ij}, typically measuring travel time and fulfilling *triangle inequality*. Let then $H = \{1, \dots, p\}$ be a set of *resources* modelling the freight or people to be transported. Each node, including the depot, can request or provide a non-negative amount of resource. Let $\mathbf{D} = (d_{ih})$ be a *demand matrix* where d_{ih} is the amount of resource h provided by node i and $-d_{ih}$ is the amount of resource h requested from node i. It is assumed $\sum_{i \in V} d_{ih} = 0$ for any resource $h \in H$: that is, for any resource, supply and demand are in equilibrium. Let also $K = \{1, \dots, m\}$ be the fleet of available vehicles: vehicle *k* deploys a capacity Q_{kh} devoted to resource h. A subset $T \in V$ models possible intermodal exchange nodes, where a vehicle can unload a resource to be collected later, perhaps by another vehicle. A vehicle can both unload and load the entire

amount of resource requested from or available at a node. As usual, a route is a circuit starting from and ending at the depot and touching some set of intermediate nodes.

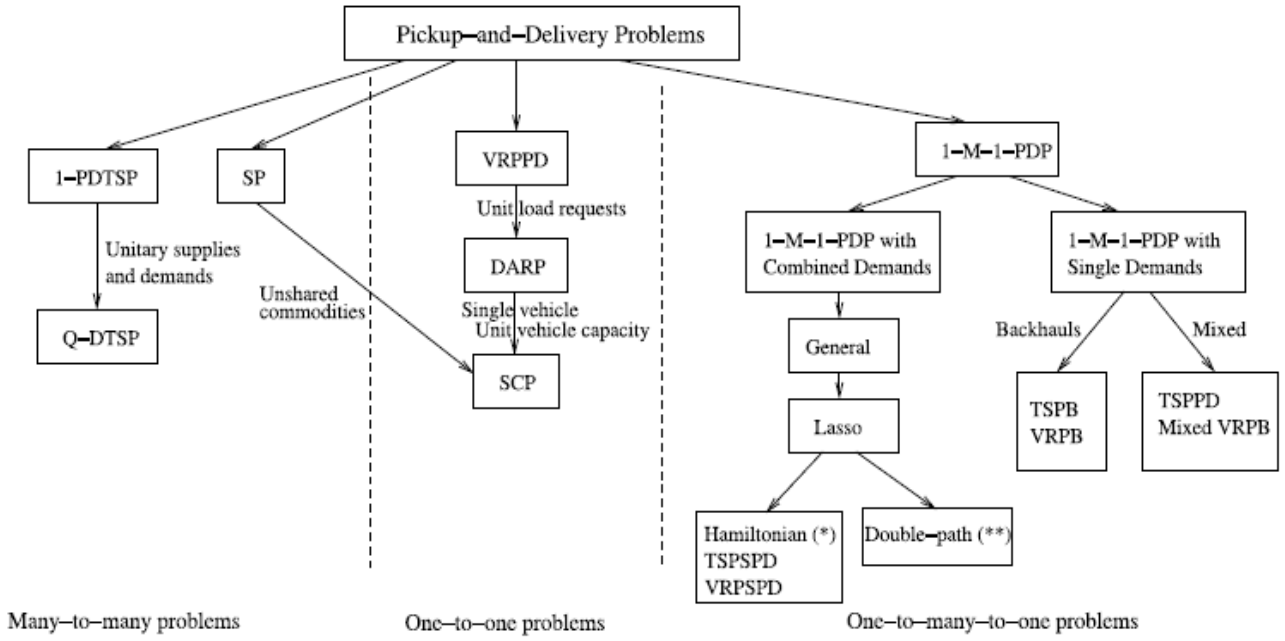


Fig. 3-6) Classification of Pickup and Delivery Problems (PDPs).

The PDP, in its basic form, has the following features:

- All pickup and delivery requests are satisfied.
- No intermodal transfer is specified for nodes in $V \setminus T$.
- The load of each resource on any vehicle never exceeds that vehicle capacity for that resource.
- The sum of the costs of the routes is minimized.

The DIAL-A-RIDE PROBLEM (DARP) is a one-to-one PDP, that is, each resource has a unique origin and destination. Pickup and deliveries can be performed one or more times, and the number of vehicles required for the solution could be bounded or unbounded. The DARP consists in covering at a minimum cost all customer transport requests, each defined by a specific destination-origin pair, respecting constraints related to operational considerations and service quality. These typically include load and capacity constraints, total duration, and travel time constraints, as well as time windows. Customer requests can be *outbound*, specifying a desired arrival time at destination, or *inbound*, giving instead a desired departure time from the origin.

Exact solution methods for the DARP include dynamic programming (for the single vehicle case) and mathematical programming (for multi-vehicle fleets). The introduction of primal heuristic methods, able to find (hopefully good) solutions but generally unable to certify optimality or approximation ranges, is justified by the fact that real cases present so many requests to make the convergence of exact methods impossible, at least within the time required to service provision. Heuristics can basically be classified into:

- Incomplete optimization algorithms, which heuristically limit the search in the solution space.
- Constructive heuristics, which try to build a feasible solution starting from the initial data.
- Improvement heuristics, which attempt to improve a primal solution by subsequent local changes.
- Metaheuristics, which combine local search algorithms with artificial intelligence techniques.

Constructive heuristics work by inserting one or a few users at a time into a solution being formed. The best placement is chosen based on figures of merit[4]. These heuristics are mainly close to Greedy Methods: very fast but unable to consider the effect of subsequent insertions. They generally lead to solutions of poor quality, in unfortunate cases very bad.

The poor quality of a solution generated by constructive heuristics can be improved by exploring its surroundings: to do this, a move is applied to the solution that leads to another point in the solution space. A wide range of moves is generally available: typical ones just change the visiting order of two or more nodes in the same route, or remove a request served and then reinsert it in a new position, serving it with the same or another vehicle. The new solution is accepted only if it improves the objective function. By its very nature, an improvement heuristic ends in a local optimum, that is, in the best solution achievable with the definition of neighborhood adopted. Further improvements are then no longer possible, unless move type is changed, that is, without changing the definition of the neighborhood of a solution.

The algorithms based on search metaheuristics do not stop the search in the first local minimum found but continue the exploration of the space of solutions by temporarily accepting, according to appropriate criteria, even pejorative or inadmissible solutions; some of these techniques are general and

[4] *A **figure of merit** is a quantity used to characterize the performance of a device, system, or method, relative to its alternatives.*

very widespread, also in other areas, such as SIMULATED ANNEALING, TABU SEARCH and GENETIC ALGORITHMS.

There is a rich literature on the DARP. One of the simplest cases of the DARP is where all users are served by a single vehicle. This case was introduced by Psaraftis [8]- [9], who formulated the problem considering the minimization of a combination of travel completion time and customer dissatisfaction and solved it by dynamic programming. Customer dissatisfaction is expressed as a weighted combination of waiting time before pickup and driving time. Customers do not specify time windows; rather, the transporter imposes "maximum position shift" constraints that limit the difference between a customer position in the calling list and his/her position in the route. This algorithm was subsequently updated by the same author to manage customer-given time windows. As it is often in the case of dynamic programming, the algorithm can only find optimal solutions in small instances: in fact, the procedure proposed has a complexity $O(n^2 3^n)$, and the largest instance resolved by this approach contains nine users.

Desrosiers et al. [10] reformulated the problem and the formulation includes time windows, vehicle capacity and precedence constraints, and is solved exactly by dynamic programming. Using a dual labeling scheme, the authors were able to identify and subsequently eliminate dominated states and state transitions. Optimal solutions were obtained for $n = 40$.

Much more research has been conducted on the multi-vehicle case.

One of the first heuristics for static DARP with many vehicles was proposed by Jaw et al. [11]. The model considered imposes a time window on the arrival time for outbound journeys and one on the departure time for return journeys. A maximum travel time, calculated as a linear function of the direct travel time, is imposed on each request. Furthermore, vehicles are not allowed to park if they are carrying passengers. To evaluate solution quality, heuristics resort to a non-linear objective function that measures and combines various types of user inconvenience. The heuristics selects users in order of admissible departure times and gradually inserts them into vehicle routes to obtain the smallest possible increase in the objective function each time. The algorithm was tested on an artificial instance with 250 requests and on a real dataset with 2617 requests and 28 vehicles.

A commonly used technique consists in defining clusters of users to be served by the same vehicle, and then route them separately (cluster-first route-second). This idea was exploited by Bodin and Sexton [12]- [13] (and subsequently taken up by others) to build clusters first, then apply to each cluster Bodin-Sexton's single vehicle routing algorithm, and finally try to obtain improvements by exchanging

requests between clusters. The results are presented on two problems extracted from a Baltimore database with approximately 85 users per instance.

A real problem, linked to disabled transport in the city of Bologna, was addressed by Toth and Vigo [14]. Users impose a time window on the departure and arrival times; moreover, stay on board is subject to an upper limit proportional to direct travel time. Transportation is provided by a fleet of minibuses and special vehicles. As an exception, taxis can also be used but, as they are not the best vehicles for transporting the disabled, solutions that use them are penalized. The goal is to minimize the total service cost. Toth and Vigo developed a heuristic that first assigns requests to routes via a parallel insertion procedure, then applies exchanges within the route and between different transportations. Tests with requests ranging between 276 and 312 show significant improvements over previous hand-built solutions.

Another study, conducted by Borndörfer et al. [15], uses a two-step approach that first builds user clusters, then group them together to form eligible routes for vehicles. A cluster is defined as "the minimum subtour for which a vehicle is not empty". Its starting and ending points correspond to the first user pickup and the last user drop-off, respectively. In the first phase, a large set of good clusters is built, and a set partitioning problem is solved to choose a subset of clusters that serve each user exactly once. In the second step, the eligible routes are enumerated by combining the clusters and a second set partitioning problem is solved to choose the best set of routes that cover each cluster exactly once. Both set partitioning problems are solved by branch-and-cut. On real instances, the algorithm does not always converge to an optimum, and therefore must be prematurely interrupted at the best-known solution. The algorithm was applied to real Berlin instances, which include 859 to 1771 transport requests per day.

Cordeau and Laporte [16] proposed a Tabu Search algorithm for the DARP in which users can specify a time window on the arrival time for outbound journeys, and one on the departure time for return journeys. A maximum limit is also associated with the time on board of each request served. This limit can be the same for each request or, as in [14], can be calculated using a maximum deviation factor with respect to the direct travel time of each individual request. Capacity and maximum duration constraints are imposed on vehicles. The search algorithm iteratively removes a request from one route to re-enter it in another. Ineligible solutions are temporarily accepted but penalized by an appropriate cost function, as typical of these contexts. The algorithm was tested on randomly generated instances with requests ranging between 24 and 144, and on six datasets, with 200 and 295 requests, provided by a Danish transporter. Compared to some alternative methods, such as column generation and branch-

and-cut, the taboo search can be easily adapted to a wide range of constraints and objectives, even non-linear.

Bent and Van Hentenryck [17] developed a two-phase heuristic for PDPTW: the first phase uses Simulated Annealing[5] to minimize the number of paths needed, the second phase minimizes the duration of paths using a LARGE NEIGHBORHOOD SEARCH (LNS) and a truncated branch-and-bound. Simulated Annealing uses a hierarchical objective function which first minimizes the number of paths, then maximizes the number of requests served by a single path, and finally minimizes the routing cost of the solution. The algorithm was tested on numerous literature instances with 100, 200 and 600 requests, allowing the identification of many new improved solutions.

Ropke and Pisinger [18] developed an ADAPTIVE LARGE NEIGHBORHOOD SEARCH (ALNS) metaheuristics to minimize the weight of routing cost, vehicle travel time and number of unserved requests. The algorithm is organized as a framework in which many simple explorations moves around a solution are used, and alternates a request removal phase to an insertion phase. The new solution so obtained is evaluated through SIMULATED ANNEALING. The heuristics used by the removal and insertion procedures are selected on the basis of the success they had in the past. The method can solve many different types of VEHICLE ROUTING PROBLEMS in an excellent way and was tested extensively on a set of 594 instances and on different datasets. The algorithm is competitive both in terms of execution time and in the quality of the solutions.

More recently, Cordeau and Laporte [19] presented some valid inequalities and a branch-and-cut algorithm for the multi-vehicle DARP. In this way the authors found exact solutions to instances involving up to 32 requests.

Ropke et al. [20] later presented stronger formulations and new valid inequalities for the DARP and the Pickup and Delivery Problem with Time Windows (PDPTW). Initially, the models are solved by relaxing some of the constraints. During the branching process, separation algorithms are applied to identify the violated constraints among those initially relaxed or the valid inequalities. These constraints are then introduced, and the process ends when the search tree has been explored according to the usual branch-and-bound rules. The largest instance resolved with the first algorithm contains 36 requests, while the largest resolved by the second algorithm contains 96 requests.

---

[5] Simulated annealing is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space for an optimization problem.

Both Cordeau and Laporte, and Ropke et al. formulate the DARP as an integer linear programming problem (with some continuous variables, in the first case).

Ropke and Cordeau [21] therefore proposed a branch-and-cut-and-price method for PDPTW. At the moment, this algorithm definitely represents state of the art among exact methods. The authors tested the algorithm to compare it with previous works by Ropke et al., Cordeau to determine the maximum limitation on the number of requests that allow optimal resolution. The algorithm was then tested on instances, known in literature, having from 30 to 200 requests, with a maximum computation time of two hours. The results demonstrate an improvement in the performance of the previous algorithm but highlight problems in dealing with larger instances: already some cases with 100 requests are not solved in the best possible time in the available time. However, some preliminary experiments were conducted on instances with 500 requests.

Very recently, Bartolini [22] formulated the PDPTW problem as a partitioning problem with additional cuts and solved it by a branch-and-cut-and-price algorithm. His method outperformed Ropke and Cordeau's in terms of both bound quality and computation time.

To conclude this section, let us observe that dial-a-ride transport systems can be managed in either static or dynamic mode.

In the static mode, transport requests are all known beforehand, therefore also vehicle routes can be planned in advance. The static problem can also be used for long-term strategical decision: in this case, the information about requests is constructed using historical data or predictions. Several scenarios are thus created and solved so that, in some sense, solutions are generated as in a simulation process. The results can also be used as a reference for better understanding the effect of possible events or trends.

In dynamic mode, transport requests are not known or only partially known in advance. Requests are in fact collected during the planning horizon as customer calls. Therefore, vehicles routes are constructed in real time. If system dynamics is not too fast, the problem can be tackled by decomposing it into a sequence of static problems of small-medium size, the first one defined by an initial request set and the following by updating the requests not yet fulfilled with incoming ones. To be viable, the approach requires to find a solution of each static problem in a time compliant with vehicle operation, which may or may not be possible depending on static problem sizes. In fact, in a dynamic scenario, solutions must be provided very quickly as the customers must be informed on-time (often meaning, while at the phone) whether their request can be satisfied or not and, if the case, at what time. To fasten computation, one can think to reoptimize and update the current solution whenever a new single request pops up. This on-line approach may however converge in the long run to very bad solutions.

3.4 Matching and assignment problems

The MATCHING PROBLEM (sometimes called coupling problems) is among the most important and studied graph optimization problems, and has multiple applications, both as a model of real problems and as a tool to solve more complex optimization problems.

Consider an undirected graph $G = (N, A)$ with n nodes and m arcs. A matching M is a subset of arcs such that every node of G is extreme of at most one arc of M. If $(i, j) \in M$, then node i (node j) is said to be *coupled*, and node j (node i) is called its mate; on the other hand, a node not belonging to any arc of M is said to be *exposed*. A matching M is maximal if adding to it an edge not in M returns a set which is not a matching. In other words, M maximal in G means that M is not a proper subset of any matching of G, or that any edge of G intersects at least one edge of M.

A *maximum matching* is one containing the largest possible number of edges and is not necessarily unique in a graph. Note that every maximum matching is maximal, but the reverse is not true in general. The MAXIMUM MATCHING PROBLEM calls for finding the largest matching of G. If edges are given numerical weights, one can consider the MAXIMUM WEIGHTED MATCHING PROBLEM, where one seeks a matching whose edge weights sum up to a maximum.

A *perfect matching* (Fig. 3-7) touches all vertices of the graph, namely, is at the same time a matching and an *edge-cover*. Every perfect matching is maximum and therefore maximal among the matchings of G, and minimum among its edge-covers. The MAX (MIN) WEIGHTED PERFECT MATCHING PROBLEM calls for finding a perfect matching of maximum (minimum) weight. If the edge set of G can be partitioned into perfect matchings, then we say that G is *factorizable*. In that case, G turns out *regular*, that is, all its nodes are incident on the same number of edges (= the node degree) and the chromatic index of G coincides with its degree.

If G is bipartite, that is, if its node set can be partitioned into two stable sets, we speak of *bipartite matching*. A graph can contain in general a very large number of matchings: for example, a complete bipartite graph with two stable sets of n nodes each (usually denoted by $K_{n,n}$) contains exactly $n!$ perfect matchings. But although the possible solutions of a matching problem can be so many, all the matching problems described above can be solved by polynomial-time algorithms.

Studies identified more elaborated forms of matching problems, such as STABLE MARRIAGE and ENVY-FREE MATCHING, that have interesting applications. The former can be solved in $O(n^2)$ time, the latter in $O(n^3)$ in the perfect matching case, see [23] and [24]. Adding such type of constraints as forbidden

edge pairs has useful applications as well (e.g., in flight slot allocation, see [25]) but turns out to be NP-hard.

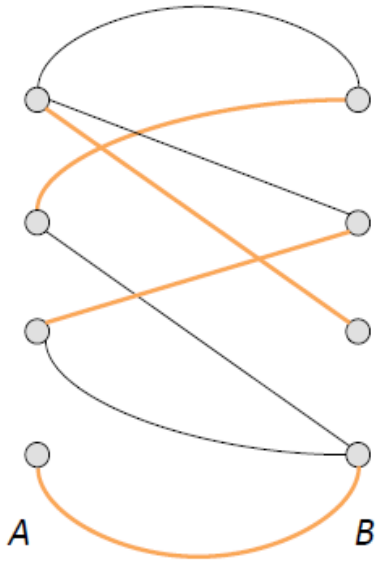


Fig. 3-7) Example of perfect matching in a bipartite graph.

Most algorithms for matching problems improve the current matching M by exploiting paths that are *augmenting* with respect to M. A matching is in fact maximum if and only if it has no augmenting path. The special case in which G is complete bipartite and edges are weighted with real numbers is also known as the (LINEAR SUM) ASSIGNMENT PROBLEM. To exemplify the problem, consider a situation in which n agents are required to perform n tasks. Any agent can be assigned any task, incurring some cost depending on which task is assigned which agent. All tasks must be assigned exactly one agent and vice-versa, and the total cost of this assignment is to be minimized.

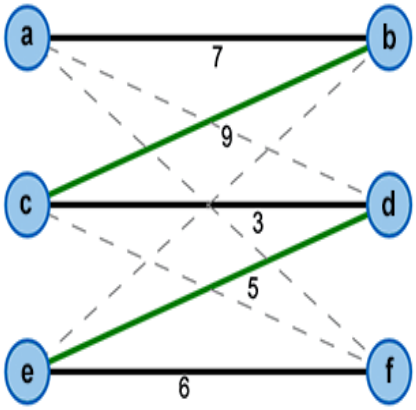


Fig. 3-8) Example of an assignment problem.

Problem data are collected in a non-negative cost matrix $\mathbf{C} = (c_{ij})$, and a solution correspond to matching each row to a different column so that the sum of the corresponding items is minimized. This corresponds to a perfect matching in a complete bipartite graph G where the two stable sets identify on one hand the agents and on the other hand the tasks. The case in which some assignments are prohibited or infeasible can be dealt with by assuming the corresponding cost so large that it will not be considered by any solution. This combinatorial optimization problem admits the following simple 0-1 LP formulation:

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \qquad \textit{(Eq. 3-11)}$$

subject to:

$$\sum_{i=1}^{n} x_{ij} = 1 \quad for \;\; j = 1, \dots, n \qquad \textit{(Eq. 3-12)}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad for \;\; i = 1, \dots, n \qquad \textit{(Eq. 3-13)}$$

$$x_{ij} \in \{0,1\} \quad for \;\; i,j = 1, \dots, n \qquad \textit{(Eq. 3-14)}$$

that boils down to an LP since the matrix associated with the equations written is totally unimodular, and the right-hand side is integral.

Rather than the Simplex Method, one can take advantage of very efficient combinatorial algorithms. The worst-case complexity of the best sequential algorithms for LSAP is $O\ (n^{2.5})$, where n is the size of the problem. Most sequential algorithms for LSAP can be classified into primal-dual algorithms and simplex-based algorithms. The primal-dual algorithms work with a pair of an infeasible primal and a feasible dual solution that satisfies the relaxed complementarity conditions. Such algorithms iteratively update the solutions until the primal solution becomes feasible, while keeping the complementarity conditions satisfied. At this point, according to LP duality, the primal solution is also optimal.

Simplex-based algorithms are special implementations of the primal-dual simplex algorithm for linear programming. Although simplex algorithms are not generally polynomial, there are special polynomial implementations in the case of LSAP.

3.4.1 The Hungarian method

The Hungarian method is a combinatorial optimization algorithm which solves the assignment problem in polynomial time, and which anticipates subsequent primal-dual methods. It was developed and published in 1955 by Harold Kuhn [26]. James Munkres [27] revised the algorithm in 1957 and noted that it is (strongly) polynomial. Since the algorithm has also been known as the Kuhn-Munkres or Munkres algorithm. The time complexity of the original algorithm was $O\ (n^4)$, however Edmonds and Karp [28], and later Tomizawa [29], noted that it could be modified to obtain an $O\ (n^3)$ running time. Ford and Fulkerson [30] extended the method to general transportation problems.

Let C be a non-negative $N \times N$ matrix, in which the element in the $i-th$ row and in the $j-th$ column represents the cost of assigning the $j-th$ work to the $i-th$ operator. We need to find an assignment for workers that has a minimal cost. If the goal is to find the assignment that produces the maximum cost, the problem can be modified to fit the setup by replacing each cost with the maximum cost subtracted from the cost.

The algorithm is easier to describe if we formulate the problem using a bipartite graph. We have a complete bipartite graph $G = (U,V;\ E)$ with n work vertices (U) and n work vertices (T) and each edge has a non-negative cost $C\ (i,j)$. We want to find a perfect match of minimum cost.

The basic idea of the Hungarian algorithm is to manipulate the cost coefficients, always through appropriate operations of adding or subtracting constants to rows and / or columns of C, in order to make more and more zeros appear, until it is possible to find an assignment that makes use of only those costs. At the end, the value of the optimal solution will be given by the sum of the original cost coefficients of the arcs used in the perfect matching.

The algorithm starts from any possible solution, even partial of the problem and tries to improve it by checking for the presence of unassigned elements.

At each iteration, the algorithm tries to find a path that increases the number of elements in the solution. The first iteration starts from an unassigned element of $u \in U$. There are two possibilities:

- There is an arc that connects the element u to an element $v \in V$ not part of the solution.
- There are only arcs that connect the element u to elements of V already assigned.

In the first case, the assignment between u and v is added to the solution.

Using a matrix interpretation, we will apply the following steps to find an optimal assignment in matrix C, which is a square matrix of order N:

1. We subtract the entry with the smallest value in each row from all the boxes in its row. We will now have a matrix with at least one zero for each row. Given the cost matrix C, for each row i we calculate the minimum on that row: $c_i^0 = min_j\, c_{ij}$. This value will be subtracted from each element of row i and this is done for all rows. A matrix C_1 is thus obtained which in position (i,j) will have the value $c_{ij} - c_i^0$.
2. We subtract the box with the smallest value in each column from all entries in its column. The matrix C_1 is further transformed by calculating the minimum on each of its column's j: $c_j^1 = min_i\, (c_{ij} - c_i^0)$. We subtract this from each element of column j. The result is a matrix C_2 which in position (i,j) will have the value $c_{ij}^2 = c_{ij} - c_i^0 - c_j^1 \geq 0$. Note that all the new elements of the matrix are non-negative.
3. We draw lines on rows and columns so that all entries in the cost matrix containing a zero are covered and so that the minimum number of lines is used.
4. Optimality test: (i) if the minimum number of coverage lines is n, an optimal assignment of zeros is possible and the algorithm ends, (ii) if the minimum number of coverage lines is less than N, an optimal assignment of zeros is not yet possible. In this case, go to step 5.
5. We determine the entry with the smallest value not covered by any line. We subtract this value from elements not covered by lines and instead add it to each element where two lines cross. Finally, return to step 3.

To determine the fewest number of lines needed to cover all formed zeros (step 3) the following algorithm can be used:

a. Determine an assignment (even incomplete, i.e., an assignment in which there may be some row not assigned to any column).
b. Mark the unassigned lines.
c. For each marked row not yet examined, mark the columns with a zero on that row.
d. For each unmarked column, mark the rows that have an assignment to those columns.
e. Repeat from step 3 until all the marked lines not yet visited are exhausted.
f. The minimum number of lines is obtained by selecting the marked columns and the unmarked lines.

3.4.2 The Bottleneck Matching Problem

The BOTTLENECK MATCHING PROBLEM (BMP, also known as LINEAR BOTTLENECK ASSIGNMENT) has the same solution set as bipartite matching, but with a different objective. Consider again n agents

and n tasks: any agent can be assigned any task, incurring a particular cost, all tasks must be done, and each task must be assigned exactly one agent. Instead, however, to minimize the sum of assignment costs, the goal is now to minimize the largest cost of all the assignments. The term "bottleneck" refers to this min-max objective and is clearly explained by a common application of this combinatorial problem, where costs are task durations that vary from agent to agent, and – assuming n parallel task executions – one wants to minimize the time at which all tasks are eventually completed. In other words, the "maximum cost" is the "longest duration", i.e., the time bottleneck of the overall workplan.
So, in a BMP formulation the objective reads:

$$min \max_{i,j=1,\dots,N} c_{ij}x_{ij} \qquad \textit{(Eq. 3-15)}$$

A classical algorithm to solve the problem was devised by Gross [31], and consists of the following steps:

1. Start by selecting one among the possible $n!$ feasible solutions. We can imagine this solution as defining *n* entries of matrix **C** that do not share a row or a column.
2. Compute $V = max\{c_{ij} \mid x_{ij} = 1\}$: that is, take the largest cost among the edges in the feasible solution selected in step 1.
3. Choose each element, denoted by (i^*, j^*), taken from the set $B = \{(i,j) | c_{ij} = V, x_{ij} = 1\}$ and create a loop on **C** starting and ending in (i^*, j^*), as explained below:
    a. From each entry of the solution, pass to an unassigned entry in on the same column with cost less than V.
    b. From each unassigned entry, pass to an assigned entry in the same row.
4. If no such loop exists, then the current assignment is optimal. Otherwise, the current assignment is updated by setting $x'_{ij} = 1 - x_{ij}$ for all entries in the cycle, and one returns to step 2.

For a non-optimal solution, it is easy to see that such an "improvement cycle" always exists.
Algorithm convergence to an optimum is ensured in no more than n^2 iterations since no cell can be repeated as origin of an improvement cycle.
More algorithms were proposed to reduce problem complexity, some of which belonging to the family of “threshold algorithms”.
A slight generalization of BMP is presented by minimum at maximum cost of MATCHING PROBLEM with bottleneck objective function: let $G = (U, V; E)$ be a bipartite graph with edge set E. Edge (i,j) has a length c_{ij}. The minimum at maximum cost of MATCHING PROBLEM can be formulated as

follows: find a maximum matching in G such that the longest edge in the matching is the shortest possible:

$$min\{max_{(i,j)\in M}\, c_{ij} : M\ is\ a\ maximum\ matching\} \qquad \textit{(Eq. 3-16)}$$

The idea of threshold methods for the BMP dates back to Garfinkel [31]. A threshold algorithm generally alternates two phases. In the first, a cost element c^*, called threshold value, is chosen and a threshold matrix $\bar{C}$ is defined by:

$$\bar{c}_{ij} = \begin{cases} 1 & if\ c_{ij} > c^* \\ 0 & otherwise \end{cases} \qquad \textit{(Eq. 3-17)}$$

In the second phase, one check whether the cost matrix $\bar{C}$ admits an assignment of cost 0. To this aim, one constructs a bipartite graph $G = (U, V; E)$ with $|U| = |V| = n$ and with $(i,j) \in E$ if and only if $c_{ij} = 0$. In this way, the problem is reduced to check whether a bipartite graph with threshold matrix $\bar{C}$ contains or not a perfect matching. The smallest value of c^* for which the graph has a perfect matching gives the optimum of the BMP. In the next chapter we will present a threshold algorithm for the BOTTLENECK MATCHING PROBLEM.

4 THE INVENTORY Dial-A-Ride Problem (IDARP)

In this chapter, we present the INVENTORY DIAL-A-RIDE PROBLEM, that is, a DARP with capacity constraints at the nodes (corresponding, in our case study, to inventory limits at the racks). We developed a complete Integer Linear Programming formulation for this new problem and tested it numerically on some small examples. An optimal solution could only be found up to eleven requests, and after long CPU time. This preliminary test shows therefore that the formulation does not fit the needs of *LFoundry*, which presents over sixty requests minimum. The exercise would however serve as a basis for further research on this challenging problem.

4.1 Pick-up and delivery with finite location capacities

We here address a special PICKUP AND DELIVERY PROBLEM whose novel feature is that each location has a finite inventory capacity. Therefore, not only vehicle capacities but also local buffer at the nodes must be considered when building a feasible route that fulfills transportation requests. This consideration introduces a difficulty in problem decoupling by a cluster-first route-second approach because, by increasing or reducing the capacity available at the nodes, pick-ups or deliveries operated by one vehicle affect the routes of other vehicles. A similar notation is used in [20].

The other problem characteristics are the same as in an ordinary DIAL-A-RIDE PROBLEM, so to highlight the above distinguishing feature we call this problem INVENTORY DARP (IDARP). The problem elements are described by three sets (Fig. 4-1):

- K set of m carts that form the fleet used to move parts in the clean room: cart k can carry at most c^k lots.
- I set of n positions, corresponding to the racks headed at each machine aisle and used to temporarily host production lots: the rack in position i can host at most b_i lots.
- $R \subseteq I^2$, set of q service requests: request r consists of picking up l_r lots in position p_r and delivering them to position $d_r \in I$.

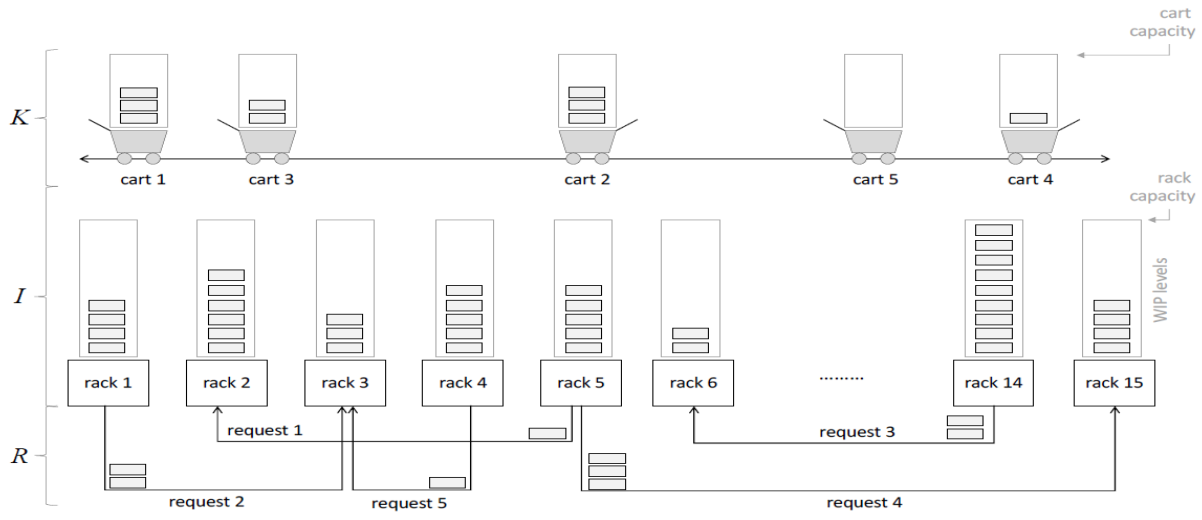


Fig. 4-1) Problem elements.

In the figure, cart and rack capacities are the same for all carts and racks, but this assumption can be relaxed according to needs. Here is a general description of the problem:

Problem 1. Find cart routes to deliver all the lots requested, minimizing the largest completion time of any request, and respecting possible restrictions on cart and rack capacity.

For simplicity of presentation, we formulate Problem 1 in two steps: we begin in §4.2 with a mixed integer linear programming core formulation that defines cart routes and schedules neglecting capacity restrictions; then we introduce capacity constraints, with further decision variables, in §4.3 and so complete our model.

4.2 Core formulation

Cart routes and schedules for Problem 1 are supported by a directed graph $G = (N, A)$ with $N = \{s^1, \cdots, s^m, s\} \cup U$. U is the set of all pick-up and delivery nodes. A node in N is called a service: those in U are proper cart services, the others are just dummy services.

There are two nodes of U per request $r \in R$: one associated with its pick-up position p_r, the other with its delivery position d_r. The arc set A contains:

- All the arcs of the form $(s^k, u), k \in K \; and \; u \in U$.
- All the arcs of the form $(u, s), u \in U$.
- All the possible arcs in U^2, except those of the form $(d_r, p_r), r \in R$.

For any $u \in N$, we let $\delta^-(u)$ (let $(\delta^+(u))$) denote the set of arcs with first (respectively, second) extreme in u. Node $s^k, k \in K$ is associated with the time instant t^k from which cart k is available for operation. For $u, v \in U$, arc $a = (u, v)$ has a weigh τ_a equal to the time necessary to operate service u and to travel from the relevant position to the one where service v must be done. A null time is required to reach the final dummy service s, i.e., τ_{us} coincides with the service time of u, for all $(u, s) \in A$. Arcs $a = (s^k, u)$ are instead weighted by t^k plus the time cart k needs to travel from its initial position to that of service u. In general, we suppose that the triangle inequality holds: $\tau_{uv} \leq \tau_{uw} + \tau_{wv}$ for any $u, v, w \in N$. The core formulation has two main types of decision variables:

$x_a^k \in \{0,1\}$ defined for $a \in A, k \in K$ and used to determine cart routes.

$y_{uv} \in \{0,1\}$ defined for any pair $u, v \in U$, and used to express service precedence relations.

To these variables we add a further real variable $C_{max} = C_s$, designed to give the largest completion time of the required services.

The introduced variables obey the conditions below:

$$C_s - \sum_{a \in A} \tau_a \, x_a^k \geq t^k \quad k \in K \qquad \textit{(Eq. 4-1)}$$

$$\sum_{a \in \delta^-(s_k)} x_a^k \leq 1 \quad k \in K \qquad \textit{(Eq. 4-2)}$$

$$\sum_{a \in \delta^+(u)} x_a^k - \sum_{a \in \delta^-(u)} x_a^k = 0 \quad u \in U, k \in K \qquad \textit{(Eq. 4-3)}$$

$$\sum_{k \in K} \sum_{a \in \delta^+(u)} x_a^k = 1 \quad u \in U \qquad \textit{(Eq. 4-4)}$$

$$\sum_{a \in \delta^+(p_r)} x_a^k - \sum_{a \in \delta^+(d_r)} x_a^k = 0 \quad k \in K, r \in R \qquad \textit{(Eq. 4-5)}$$

$$y_{uv} + y_{vu} = 1 \quad u, v \in U \qquad \textit{(Eq. 4-6)}$$

$$y_{uv} + y_{vw} + y_{wu} \leq 2 \quad u, v, w \in U \qquad \textit{(Eq. 4-7)}$$

$$x_a^k - y_{uv} \leq 0 \quad a = (u, v) \in A, k \in K \qquad \textit{(Eq. 4-8)}$$

$$y_{p_r d_r} = 1 \quad r \in R \qquad \textit{(Eq. 4-9)}$$

and the objective is:

$$min \;\; C_s \qquad \textit{(Eq. 4-10)}$$

The inequalities written are fulfilled by cart schedules

(Eq. 4-1) completed when the last cart has terminated its duty, in a time minimized by (Eq. 4-10),

(Eq. 4-2) with each active cart leaving its initial location,

(Eq. 4-3) with exactly one route per active cart,

(Eq. 4-4) touching every service of U exactly once,

(Eq. 4-5) such that, if the $k-th$ route touches the pick-up service of r, then it also touches the delivery service of r,

(Eq. 4-6) that define service precedence according to anti-symmetry

(Eq. 4-7) and transitive property,

(Eq. 4-8) that implement precedence as specified by the route arcs,

(Eq. 4-9) that pick up any request before the corresponding delivery.

4.3 Capacity and time constraints

Let us now extend the core formulation of (4.2) to consider work-in-process at the various positions, cart loads, and issues related to the completion of individual deliveries. In this way, we can deal with inventory constraints and limited cart capacity, as well as impose deadlines on specific deliveries. In the following, M denotes a sufficiently large number that changes from case to case according to need.

Time constraints and delivery deadlines

For these restrictions, introduce variables $C_u \in R$ defining the completion time of service $u \in U$. This is expressed as the least C_v fulfilling

$$C_v \geq C_u + \tau_{uv} x_{uv}^k - M(1 - y_{uv}) \quad k \in K, (u,v) \in A \qquad \textit{(Eq. 4-11)}$$

In fact, if u does not precede v, then the inequality does not bind completion times to each other. Otherwise, $C_v \geq C_u$ and if u is moreover an immediate predecessor of v in some route k, then C_v is at least C_u augmented of the time τ_{uv} required to complete service u and then reach v. (Eq. 4-1) is implied by (Eq. 4-11) and can be removed.

Imposing a deadline t_r on the completion of request r means enforcing:

$$C_u + \tau_{uv} x_{uv}^k - M(1 - y_{uv}) \leq t_r \quad k \in K, (u,v) \in A \qquad \textit{(Eq. 4-12)}$$

for the delivery service u corresponding to request r. Due-date related constraints and objectives can be treated in a similar fashion.

Inventory constraints:

As per the notation introduced, let l_r be the amount of lots involved (for either pick-up or delivery) in request r. For the corresponding services, let us set $\ell_u = l_r$ if $u = d_r$ and $\ell_u = -l_r$ if $u = p_r$. Let also U_i denote the set of all the services insisting on position $i \in I$. Then, for any $v \in U_i$, $\sum_{u \in U_i} \ell_u y_{uv}$ represents the WIP observed in position i immediately before service v is operated by some cart. If position i can only host a limited volume b_i of lots, we enforce

$$\ell_v + \sum_{u \in U_i} \ell_u y_{uv} \leq b_i \quad i \in I, v \in U_i \qquad \textit{(Eq. 4-13)}$$

Conditions (Eq. 4-13) are coupled with constraints (Eq. 4-11) that relate service completion to service precedence, and of course with (Eq. 4-6), (Eq. 4-7) of the core formulation.

Notice that (Eq. 4-13) may not be fulfilled by any cart schedule. A necessary and sufficient condition for fulfillment is that, in any position, the volume of inbound material minus that of outbound material (including the material transferred between racks and machine areas) does not exceed the residual capacity at that position. This condition is immediately checked by inspection of problem data.

Limited cart capacity:

The above defined ℓ_u can also weigh the arcs visited by a specific route k, and so keep track of the total cart load after a given service v. Let $L_u^k \geq 0$ be cart k load immediately after service u. With $L_{s_k}^k$ equal to the known initial cart loads, we require

$$L_v^k \geq L_u^k + \ell_v x_{uv}^k - M(1 - x_{uv}^k) \quad k \in K, (u,v) \in A \qquad \textit{(Eq. 4-14)}$$

In fact, either u does not immediately precede v on route k and hence the cart load at u does not directly affect L_v^k, or it comes immediately before v and therefore cart k load is modified by the operation of v. If c^k is the capacity of cart k, we enforce

$$L_u^k + \ell_v x_{uv}^k - M(1 - x_{uv}^k) \leq c^k \quad k \in K, (u,v) \in A \qquad \textit{(Eq. 4-15)}$$

4.4 A small example of formulation

Fig. 4-2, illustrates some of the constraints introduced: below, it shows an example with eight racks and nine requests: above, the nodes of graph G with the arcs of two routes constructed according to (Eq. 4-3) and (Eq. 4-5). The input for this example is summarized in Tab. 4-1. Arcs are weighted by the τ_a (in black). Just to illustrate the method, we assume one time unit to move between adjacent racks, and one time unit to load/unload a single lot into/from the rack. We also suppose cart 1 initially located in position 6 and cart 2 in position 1.

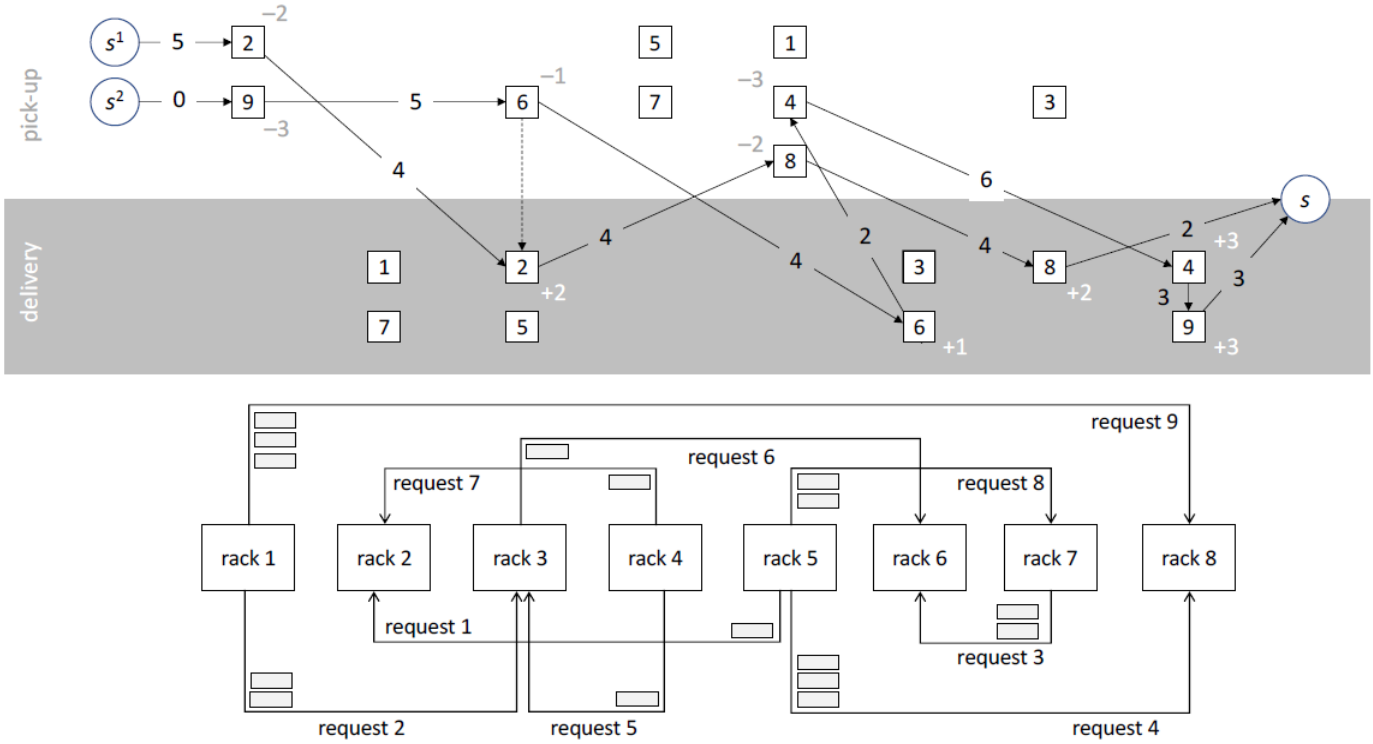


Fig. 4-2) Two cart routes: below, problem elements; above, graph G partially represented.

Requests	Positions		Volume
r	Pickup (p_r)	Delivery (d_r)	l_r
1	5	2	1
2	1	3	2
3	7	6	2
4	5	8	3
5	4	3	1
6	3	6	1
7	4	2	1
8	5	7	2
9	1	8	3

Tab. 4-1) Input data for the example of Fig. 4-2.

According to inequality (Eq. 4-1), the route of cart 1 is completed 19 time-units after cart release, while that of cart 2 ends 24 time-units from release.

Nodes, visually aligned to rack positions and separated according to pickup and delivery, are labeled by requests, and weighted by the ℓ_u (in grey and white). A single precedence relation is shown, indicated with a dashed arrow. It corresponds to $y_{uv} = 1$, a value compliant with inequality (Eq. 4-11) where u is the pickup service of request 6 and v the delivery service of request 2. Note that the companion inequality $C_u \geq C_v + \tau_{vu} x_{vu}^k - M(1 - y_{vu})$ sets $y_{vu} = 0$ since $C_u < C_v$. According to the left-hand side of inequality (Eq. 4-13), the initial work-in-progress in position 3 is reduced by 1 lot after pickup service 6, and then augmented of 2 lots after delivery service 2.

4.5 Numerical tests

We ran a numerical experience to understand the range of application of the formulation proposed for the IDARP. We assumed 2 trolleys available for 8 positions and considered finite capacities for both trolleys and racks. Requests were generated by choosing random pairs of positions. We implemented our model in Python3 (§5.4.2) and solved the relevant problems by GuRoBi® (§5.4.1).

Number of requests	C_{max}	**Completion time (sec.)**	**Gap (%)**
8	23	25.93	0%
9	25	51.54	0%
10	27	1274.95	0%
11	29	3448.28	0%
12	30	34132	6.45%

Tab. 4-2) Numerical results for Inventory DARP instances.

Column 2 reports the C_{max} value of the best solution found, column 4 the final gap. As one can see, already with 12 requests over 9 hours CPU time was not sufficient for the solver to find an optimal solution.

4.6 Conclusions

Unfortunately, according to the experiments carried out, the formulation cannot be used in the industrial context of our work. This indicates a possible direction of future research, not only to develop new formulations that are possibly manageable in real cases, but also to design heuristics able to cope with the problem peculiarity, which is the presence of finite capacities at pick-up and drop locations, that as said makes it difficult to directly adapt classical *cluster-first route-second* methods.

Moreover, the formulation shows its limit in numerical application even if the local capacity constraints are relaxed. This motivated us to develop a *cluster-first route-second* method to solve the relaxed problem, that is the subject of the next section.

5 Modelling and solving the *LFoundry* DARP

In this chapter we devise a DARP model for routing vehicles in the *LFoundry* clean room without capacity constraint at drop locations. We describe a math-heuristic that decomposes the problem in three subproblems or phases, following a *first-cluster then-route* approach. The first phase consists in solving a MIP model to partition requests into as many clusters as available carts in the fleet. This phase exploits in a crucial way the rectilinear layout of the plant. A second phase consists then in computing the time required of each cart to schedule the requests of each cluster (we limited our attention to simple double-sweep schedules). Finally, in phase three, each cart is assigned to one request cluster by solving a BOTTLENECK MATCHING PROBLEM. The chapter is concluded by exhibiting our computational findings: these are organized in two parts, corresponding to a static and a dynamic scenario, with real data from the clean room.

5.1 Introduction

The model described in Section 4 is here relaxed by removing rack capacity constraints at locations. In this way, we can adopt a *cluster-first route-second* heuristic algorithm that overtakes the convergence problem of the MIP formulation and can tackle real data. The aforementioned relaxation can possibly return infeasible solutions: we deal with this possibility preventively, by adopting simple individual cart routes that try to reduce the risk of infeasibility as far as possible by prioritizing pick-up to delivery operations, thus moving as far as possible the feasibility issue from the rack capacities to the local capacity of each cart. In this way, external interactions of carts at the locations are minimized, and each cart can be individually routed in the cluster it is assigned to. If the chances of an infeasibility are reduced, there is no guarantee that an infeasibility never occurs: however, one can put a remedy to it by simply slowing down some individual cart, thus worsening performance but not having to completely revise clusters and cart routes.

The algorithm developed is a math-heuristic, as it makes use of a mathematical optimization model to find intermediate solutions. The model is formulated as a MIP and its solution found by a commercial solver. To this aim, software licenses should be purchased by *LFoundry*; if not, however, a tailored heuristic method can be designed to find a primal solution to the MIP: this task is however beyond the scope of the work here presented.

The overall method is divided into three phases. The first phase consists in solving the said mathematical problem, which consists of partitioning the given set of requests into linear intervals called *spans*. This formulation exploits a noticeable geometrical feature of the application, that is: carts always move along a rectilinear layout. A span is then defined by a generic pair of stations on the main aisle. Also, requests are defined by station pairs, but in this case, we distinguish the origin from the destination, and therefore we represent a generic request by an oriented interval. In conclusion, the MIP formulation just models the assignment of oriented intervals (requests) to non-oriented ones (spans), forming in this way the clusters to be successively assigned to carts for routing. The goal is to minimize the largest completion time of a route, but since routes are still unknown at this point of the computation, the formulation uses an estimate of this time.

To obtain a precise measure of routing time, the second phase schedules all the requests assigned to each span using all possible carts and considering their individual starting points. Those schedules are then determined for each pair (cart, span), and their determination includes the direction to be undertaken and the costs related to different distances between cart and span. Driven by the need of

both simplifying the scheduling problem (NP-hard for general cart capacity) and, as said, reducing the risk of violating local rack capacities, we resorted for this phase to elementary sweep schedules. This phase, in conclusion, returns a reliable evaluation of route duration for any cart-span pair.

The third phase has finally the goal of assigning the spans selected in the first phase to the available carts that will physically carry out operation. This problem is solved by considering the route durations determined in the second phase and, because of the objective of balancing cart workloads, has the shape of a BOTTLENECK MATCHING PROBLEM.

The general assumptions on the model, also made in the numerical experiments done, are:

i. There are 14 rack positions equally spaced in the aisle at a distance of 10 m. Carts are identical and are available in the number of p = 8. Carts move at constant nominal speed of 1.2 $^{m}/_{s}$ so that, for any cart, passing from position i to position $i + 1$ or $i - 1$ requires 12 second.

ii. At the beginning of a work shift carts are parked between rack 3 and 4; we however designed the algorithm for use in a dynamic environment, thus considering any possible initial position of any cart in the aisle.

5.2 The span model

As with generic DARPs, our problem can initially be described as in Fig. 5-1 by a direct graph $G = (N, A)$ in which: The nodes $n \in N$ represent the rack positions in the aisle (black squares in the figure).

- The arcs $(i, j) \in A$ represent transport requests of lots from rack i to rack j (orange and blue arrows in the figure: each arrow starts from the pick-up position (where the lots are waiting for a cart) and ends in a delivery position (where a cart is expected to drop the lots involved in the request).
- Arc (i, j) has a weight that represents the number of lots to be moved from node i to node j.

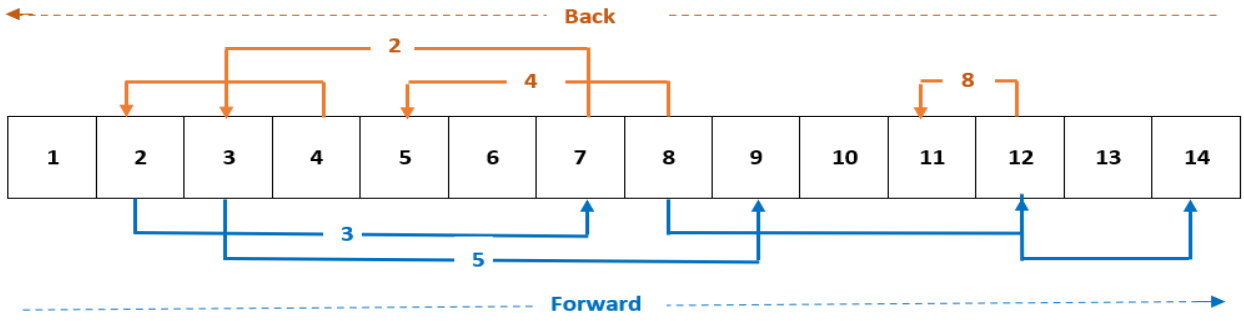


Fig. 5-1) Initial graph of requests and positions.

Given the available cart fleet and knowing the travel time between any position, we want to cover all requests minimizing the time required by the longest among the cart missions. To this aim we move from the graph described and model another auxiliary graph, in which the idea of span is introduced. The goal is to assign each cart a certain span in which it will have to move. As said in the introduction, a span is an unordered pair $\{u, v\}$ of positions: since order is not significant, we will always represent a span by the notation $[u, v]$ with $u < v$; conversely, a request is an ordered pair (u, v) and since here the order distinguish the request origin (u) from the destination (v), we may have $u < v$ as well as the opposite.

The new graph derived from $G = (N, A)$ is bipartite and undirected, and is denoted by $B = (S \cup R, A)$:

- The two node sets respectively represent spans (S, only if deemed significant) and requests (R).
- An arc (r, s) represents the coverage of request r by span s.

A span can only cover requests that falls within its range: as we see in Fig. 5-2, for example, span [2,7] can cover the requests (2,7), (7,3) but not (8, 5); span [3,10] can cover both (3,9) and (8,5), but not (4,2), etc. So, for any $s \in S$ we denote by R_s the set of requests potentially covered by s and, conversely, for any $r \in R$ we let S_r denote the set spans that can cover request r.

Note that graph B has a special structure because both requests and spans are sub-intervals of a given interval, and therefore can be ordered according to the positions of their extremes.

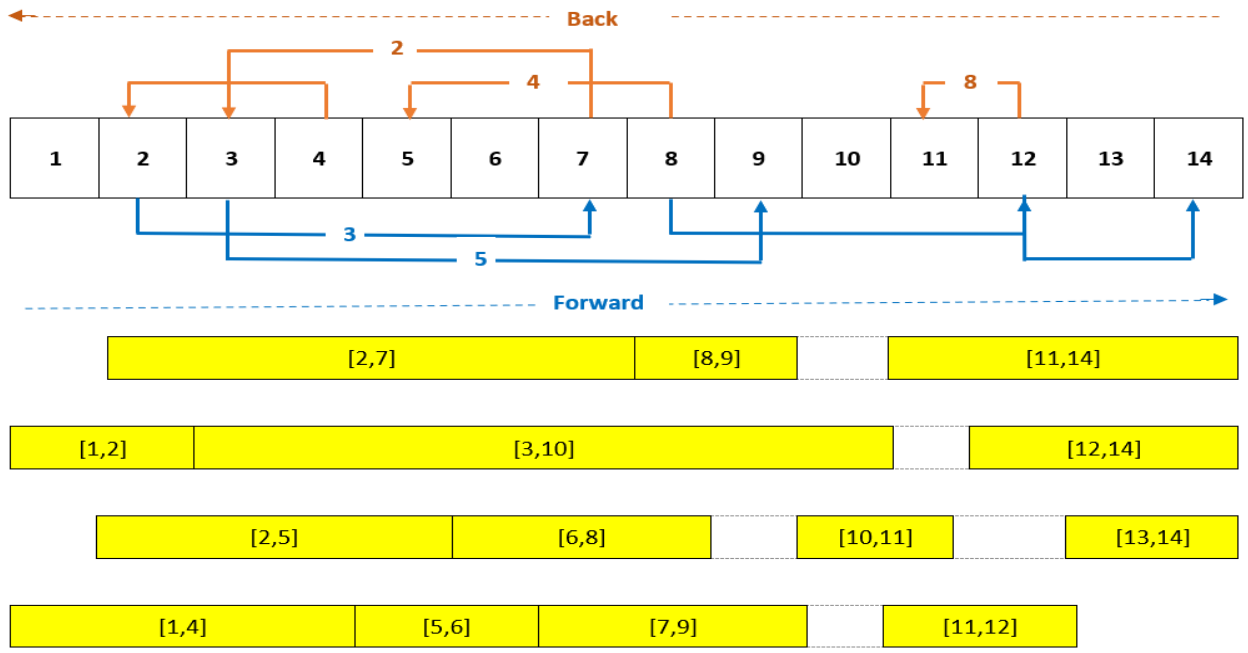


Fig. 5-2) Spans associated with a set of requests.

Only significant spans are considered, i.e., those that cover at least one request and are of *minimal length*, which means that further reducing the length will miss to cover at least one request. 0-length interval such as a [1,1], [2,2] etc. will have no meaning and are hence discarded. Fig. 5-3 indicates in green the significant spans found in Fig. 5-2 (those that do not cover any request are colored white).

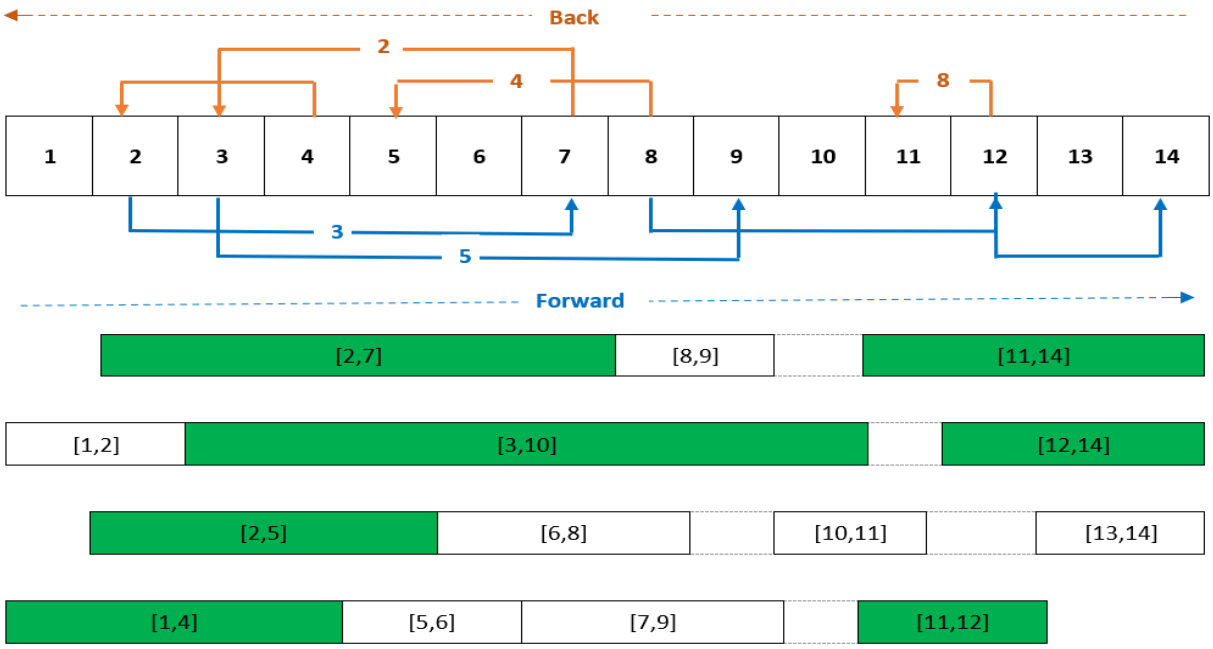


Fig. 5-3) Significant spans for a set of requests.

The request nodes r of graph B are labelled by the number of lots q_r that must be transported for that request. Also span nodes s have a label that gives an estimation d_s of the time required to sweep the span from one extreme to the other. Finally, we let p be the number of carts forming the available fleet and assume that each cart k has capacity c_k.

The problem solved in the first phase consists in assigning each request to a span so that:

1. The spans selected in the assignment do not exceed the number of available carts.
2. All requests are assigned.
3. The maximum time estimated to cover the requests assigned to a span is minimized.

An example of feasible solution is given in Fig. 5-4. At this level of computation, the time it takes for a cart to move within its span is to be estimated from the length of the span d_s and the total number of requests assigned to the span. The problem so formulated is approached as the MIP model described below.

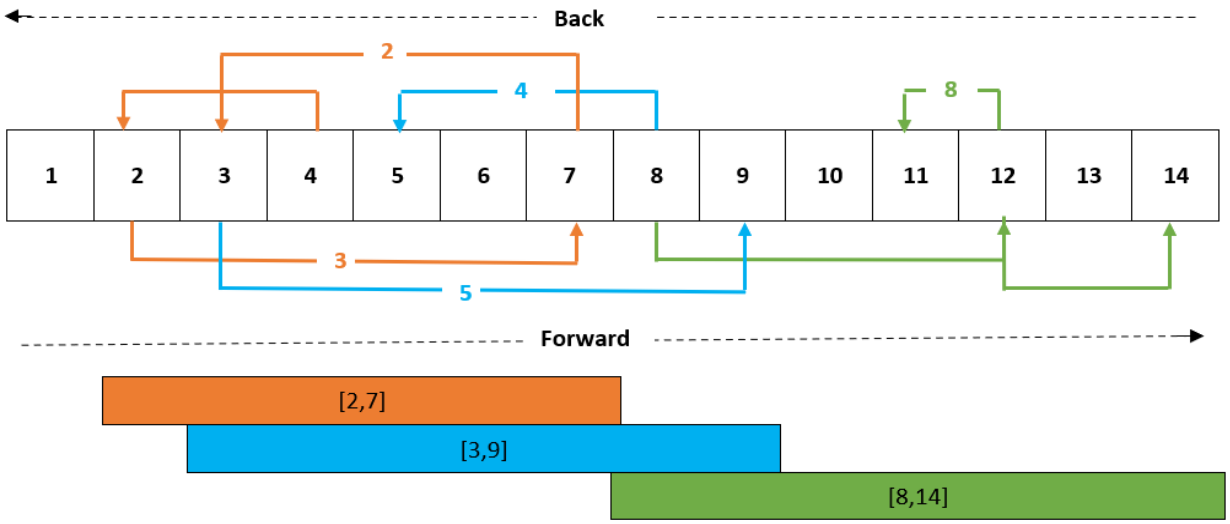


Fig. 5-4) Example of a solution for the problem.

5.2.1 The span model as a Mixed Integer Linear Program

The Span Model described in the previous section can be expressed as a mixed integer linear program. The program has the following features:

5.2.1.1 Model parameters

Sets:

- Set of spans *S*.
- Set of requests *R*.
- Set of requests covered by a span s: R_s.
- Set of requests with forward direction: R^F.
- Set of requests with backward direction: R^B.

Resources and technical coefficients:

- Number of carts available for request fulfillment: p
- Number of lots per request: q_r
- Cart *k* capacity: c_k
- Distance traveled to cover span s from end to end: d_s
- Normalization coefficient: α

5.2.1.2 Decision variables

The formulation uses the following decision variables:

$$x_{sr} = \begin{cases} 1 & if\ request\ r\ is\ assigned\ to\ span\ s \\ 0 & otherwise \end{cases} \quad \textit{(Eq. 5-1)}$$

$$y_s = \begin{cases} 1 & if\ span\ s\ was\ chosen \\ 0 & otherwise \end{cases} \quad \textit{(Eq. 5-2)}$$

Variable x_{sr} indicates whether request r is or not attributed to span s. Variable y_s indicates whether span s is or not assigned a request and is used to ensure that the spans selected are as many as the carts available.

A problem arises in estimating the time spent to cover a span and the associated requests: the span length d_s is a fine estimate provided that all the requests assigned to the span go in the same direction (forward or backward). If this is not the case and the requests go both ways, the above evaluation becomes too rough: a better approximation of travel time is in this case $2d_s$, and to introduce it in the model we need two more 0-1 variables per span s, y_s^B and y_s^F. When set to 1, these variables indicate that the span is used respectively by backward or forward requests only.

$$y_s^F = \begin{cases} 1 & if\ span\ s\ is\ covered\ in\ the\ forward\ direction \\ 0 & otherwise \end{cases} \quad \textit{(Eq. 5-3)}$$

$$y_s^B = \begin{cases} 1 & if\ span\ s\ is\ covered\ in\ the\ backward\ direction \\ 0 & otherwise \end{cases} \quad \textit{(Eq. 5-4)}$$

Finally, a real variable c_{max} is introduced to represent the largest estimated duration required to cover a generic span.

5.2.1.3 The objective

The goal of our problem is to minimize c_{max} , that measures an estimate of the largest time a cart employs to travel in the span and cover the requests assigned to it:

$$min\ (c_{max}) \quad \textit{(Eq. 5-5)}$$

This variable, as we will see in the next section, is constrained to the time cost associated with each span selected in the assignment.

5.2.1.4 Constraints

We here list the constraints required to obtain a feasible assignment and to measure its cost. First, we must ensure that each request r is assigned exactly one span:

$$\sum_{s \in S_r} x_{sr} = 1 \qquad \forall\, r \in R \tag{Eq. 5-6}$$

Each span can serve requests that fall within its $R_s$range. The following inequalities (Eq. 5-7) ensure that, if a span is not active, no request can be assigned to it, while only in the opposite case it can cover some requests:

$$x_{sr} \leq y_s \qquad \forall\, r \in R_s, \forall\, s \in S \tag{Eq. 5-7}$$

Let p be the number of available trolleys, then with the following constraint (Eq. 5-8) we enforce that the number of spans that will be assigned requests will comply with the fleet available.

$$\sum_{s \in S} y_s = p \tag{Eq. 5-8}$$

The next inequalities (Eq. 5-9) ensure the respect of cart capacities:

$$\sum_{r \in R_s} q_r\, x_{sr} \leq c_k \qquad \forall\, s \in S \tag{Eq. 5-9}$$

The following constraints clarify the role of variables y_s^B and y_s^F.

$$x_{sr} \leq y_s^F \qquad \forall\, r \in R^F, \forall\, s \in S \tag{Eq. 5-10}$$

$$x_{sr} \leq y_s^B \qquad \forall\, r \in R^B, \forall\, s \in S \tag{Eq. 5-11}$$

These variables define the directions in which the span must be traveled. We distinguish between forward and backward requests (u, v): in the former, $u < v$, in the latter $u > v$. If a span is assigned a forward request, then y_s^F must be set to 1, that is, the span is to be traveled in the forward direction, and conversely for backward requests.

Now, a span traveled in some direction (either forward or backward) means a span assigned at least one request. Therefore

$$y_s^F \leq y_s \qquad \forall\, s \in S \tag{Eq. 5-12}$$

$$y_s^B \leq y_s \qquad \forall\, s \in S \tag{Eq. 5-13}$$

In other words, by (Eq. 5-12) and (Eq. 5-13) , a span which is not activated will not be traveled in any direction.

The introduction of the above variables and constraints allows to improve the estimation of the time to cover a span, which in general will include two terms: a *pure travel time*, and the time required to fulfill the requests that have been assigned to it by *loading* and *unloading* the relevant lots.

Since spans are not currently assigned to carts, in first attempt we approximated travel time to twice the distance d_s to cover the associated interval (as there is the possibility that the span is assigned requests in both forward and backward direction). With this approach, we constraint the real variable c_{max} representing the largest time required to cover a span in the following way:

$$c_{max} \geq c_s = (2d_s\, y_s + \alpha \sum_{r \in R_s} x_{sr}) \qquad \forall\, s \in S \qquad \textit{(Eq. 5-14)}$$

where the summation counts the number of requests assigned to the span, and α is a positive real number that normalizes load/unload time to travel time. However, as noticed, when the requests assigned to a span are one way, a better approximation of the travel time is just d_s. So, we introduced variables y_s^F, y_s^B and rewrite constraint (Eq. 5-14) in order to get a better time estimate:

$$c_{max} \geq d_s(y_s^F + y_s^B) + \alpha \sum_{r \in R_s} x_{sr} \qquad \forall\, s \in S \qquad \textit{(Eq. 5-15)}$$

If the request management time is considered negligible (therefore $\alpha = 0$), then to avoid the spans activated in the solution with no assigned request (and therefore a cart remains unused) the following constraint is added:

$$\sum_{r \in R_s} x_{sr} \geq y_s \qquad \forall\, s \in S \qquad \textit{(Eq. 5-16)}$$

In this way, a span can be activated only if at least one request has been assigned to it. In cases where the management time is considered (i.e., $\alpha > 0$), this constraint is redundant to minimize the value of c_{max}, as all the trolleys will certainly be activated.

5.2.1.5 The MIP model, resumed

$$min\ (c_{max}) \qquad \textit{(Eq. 5-5)}$$

$$\sum_{s \in S_r} x_{sr} = 1 \qquad \forall\, r \in R \qquad \textit{(Eq. 5-6)}$$

$$x_{sr} \leq y_s \qquad \forall\, r \in R_s, \forall\, s \in S \qquad \textit{(Eq. 5-7)}$$

$$\sum_{s \in S} y_s = p \qquad \textit{(Eq. 5-8)}$$

$$\sum_{r \in R_s} q_r \, x_{sr} \leq c_k \qquad \forall \, s \in S \tag{Eq. 5-9}$$

$$x_{sr} \leq y_s^F \qquad \forall \, r \in R^F, \forall \, s \in S \tag{Eq. 5-10}$$

$$x_{sr} \leq y_s^B \qquad \forall \, r \in R^B, \forall \, s \in S \tag{Eq. 5-11}$$

$$y_s^F \leq y_s \qquad \forall \, s \in S \tag{Eq. 5-12}$$

$$y_s^B \leq y_s \qquad \forall \, s \in S \tag{Eq. 5-13}$$

$$c_{max} \geq d_s(y_s^F + y_s^B) + \alpha \sum_{r \in R_s} x_{sr} \qquad \forall \, s \in S \tag{Eq. 5-15}$$

$$x \in \{0,1\}^{|A|} \tag{Eq. 5-17}$$

$$y \in \{0,1\}^{|S|} \tag{Eq. 5-18}$$

$$y_s^F \in \{0,1\}^{|S|} \tag{Eq. 5-19}$$

$$y_s^B \in \{0,1\}^{|S|} \tag{Eq. 5-20}$$

5.3 Cart-to-span assignment costs

The cost of assigning a span to a specific cart is the time at which that cart will complete the deliveries of the resources assigned to the span, and depends by the cart release date, initial position, and route. The route, that is the sequence in which lots are picked up from origins and delivered to destination, must also fulfill cart (and possibly rack) capacity constraints.

In detail, a cart k has the following characteristics:

- A release date t_k from which the cart is available, which depends on the cart busy state. If $t_k = 0$, then the cart is immediately available.
- q_r is the quantity of lots and each cart k has capacity c_k.
- A starting position, which is assumed to be one of the rack locations, that correspond to the place in which the cart is found when available.

Clearly, the distance of any chosen span from each particular cart depends on the cart initial positions; the completion time of the requested assigned to the span depends then on this distance, and also on the cart release date and on the order in which the span requests are scheduled. Regarding the scheduling, to reduce complexity we consider two very simple sweep policies:

- **Policy 1**: The cart picks up parts in the same order as it finds them in the travel and delivers them in the same order as destinations are encountered.
- **Policy 2:** The cart first operates all the pick-ups in the order found in the travel; once all parts are collected, the cart delivers them in the order in which destinations are encountered.

Policy 1 also aims at minimizing the part stack in the cart, while the goal of Policy 2 is to minimize the work-in-process at the racks and the time each cart waits for another: in fact, with Policy 2 the pick-up operations precede every delivery, so we use from the very beginning the whole cart capacity available to free racks as far as possible.

According to the above policies, three sweeping modes are only possible:

- Single: the cart moves along a single direction, either forward (F) or backward (B).
- Double: the cart moves in two opposite directions, either forward first, then backward (FB) or backward first, then forward (BF).
- Triple: this case applies only when the cart initial position lies within the span and for pick-ups and deliveries in a particular order that forces this mode: either the cart first moves forward, then backward, then again forward (FBF) or first backward, then forward, then again backward (BFB).

With this choice, the scheduling time for a specific cart is easily computed and depends only on the initial direction in which the span is swept in particular, when the span is swept in both directions, one must only check from which span end the cart will start to manage the requests. Of the listed possibilities, we will then accept the shortest one. Summarizing, to compute each span-to-cart assignment cost we proceed through the following elementary steps:

1. Detect the cart position with respect to the span, in particular, check whether the cart lies within the span range or outside (either left or right to it).
2. Take note of the direction the cart takes (either to the right or to the left) to reach the pick-up position of the first request to be covered.
3. Take note of the node that the cart will reach first (which can represent either a span end or the position of the first request if this does not coincide with a span end).
4. Choose the sweeping mode: B, F, BF, FB, BFB, FBF.

The computation will produce a $p \times p$ matrix in which each entry represents the shortest time d_{sk} that cart k would employ to complete the requests assigned to span s.

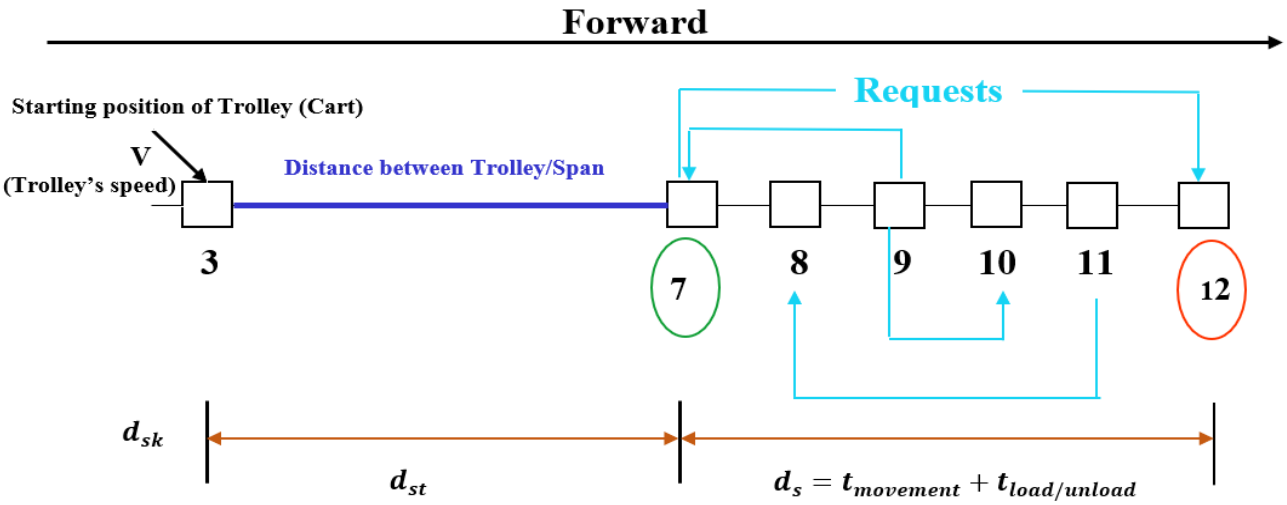


Fig. 5-5) Example of calculating the distance between a cart and span [7, 12].

5.3.1 Cart-to-span matching

Assigning the p carts available to the p spans selected means finding a perfect matching (Fig. 5-6) in graph M. The weight of a solution, however, would not be the sum of the weights of the arcs in the matching – as in the classical MIN COST PERFECT MATCHING PROBLEM: carts in fact operate in parallel, and therefore their completion times will not sum up. Rather, because we wish to balance cart workloads as far as possible, we try and minimize the largest completion time T_{max} among all the carts, which is the maximum among the weights t_{sk} of the arcs (s, k) in the matching.

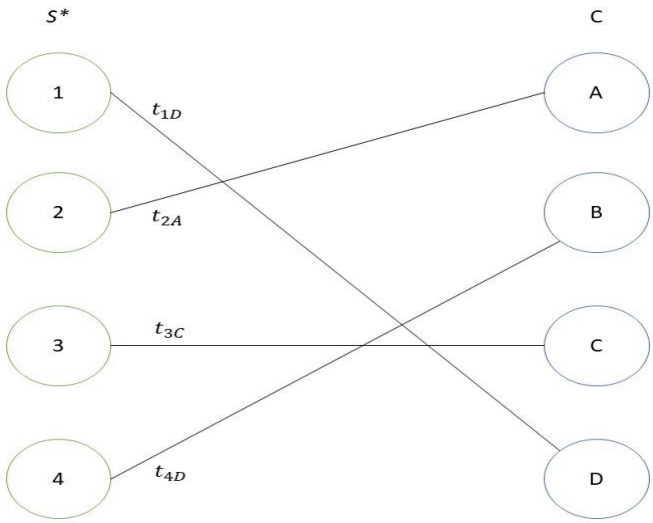


Fig. 5-6) Example of matching between spans and trolleys.

The formulation has the features illustrated below.

Sets:

- Set S^* of optimal spans, output of phase 1
- Set C of available carts

Resources and technical coefficients:

- Number p of carts available transport requests
- Total completion time t_{sk} if cart *k* fulfills the requests of span *s*

The model uses the following decision variables:

$$z_{sc} = \begin{cases} 1 & \text{if trolley c is assigned to span s} \\ 0 & otherwise \end{cases} \quad \textit{(Eq. 5-21)}$$

$$T_{max} = \text{ the largest completion time among the carts selected} \quad \textit{(Eq. 5-22)}$$

The objective then reads:

$$Min\, T_{max} \quad \textit{(Eq. 5-23)}$$

and is sought under the following constraints:

$$\sum_{s \in S^*} z_{sc} = 1 \qquad \forall\, c \in C \quad \textit{(Eq. 5-24)}$$

$$\sum_{c \in C} z_{sc} = 1 \qquad \forall\, s \in S^* \quad \textit{(Eq. 5-25)}$$

Equations (Eq. 5-24) and (Eq. 5-25) above are the classical constraints of a perfect matching problem. The following constraints (Eq. 5-26 assign to variable $T_{\max}$ the largest weight among those of the matching, since we are looking for a solution that minimizes $T_{\max}$.

$$T_{max} \geq t_{sc} z_{sc} \qquad \forall\, c \in C, \forall\, s \in S^* \quad \textit{(Eq. 5-26)}$$

All in all, the formulation then reads

$$Min\, T_{max} \quad \textit{(Eq. 5-23)}$$

subject to:

$$\sum_{s \in S^*} z_{sc} = 1 \qquad \forall\, c \in C \quad \textit{(Eq. 5-24)}$$

$$\sum_{c \in C} z_{sc} = 1 \qquad \forall\, s \in S^* \quad \textit{(Eq. 5-25)}$$

$$T_{max} \geq t_{sc} z_{sc} \qquad \forall c \in C, \forall s \in S^* \qquad \textit{(Eq. 5-26)}$$

$$z \in \{0,1\}^{|A|} \qquad \textit{(Eq. 5-27)}$$

5.3.2 A combinatorial algorithm for Bottleneck Matching

The problem defined in section 5.3.1, if formulated on a complete bipartite graph $G = (U, V, A)$ with $|U| = n$ and $|V| = n$, is generally known as the ASSIGNMENT PROBLEM. This problem has a min-sum objective and, as already seen in section 3.4.1, the most famous combinatorial algorithm for solving it is the Hungarian method. However, we here, deal with a min-max objective

$$min \max_{i,j=1,\ldots,n} c_{ij} z_{ij} \qquad \textit{(Eq. 5-28)}$$

where $\mathbf{C} = \{c_{ij}\}$ is an $n \times n$ cost matrix, and our problem has the form of a BOTTLENECK (PERFECT) MATCHING PROBLEM (BMP).

Several combinatorial algorithms for this problem are available in literature. Among them, we find the so-called "threshold methods". These form a class of bottleneck algorithms in which the value V of an initial solution is progressively increased, and for each value obtained one checks the existence of a feasible solution of cost V: the first solution that passes this test is the optimum. To solve the problem, we adopted the threshold method proposed by Pundir et al. [32], which uses the Hungarian Method as a subroutine. It consists of the following steps:

1. Set $c^* = \max\left(\min_i c_{ij}, \min_j c_{ij}\right)$
2. Construct an auxiliary cost matrix **C**' by replacing with +∞ all the entries of **C** that are greater than c^*
3. Apply the Hungarian Method to find an optimal solution to the problem defined by the **C** so updated. If a finite value solution exists, then c^* is optimal for the BMP, otherwise go to step 4.
4. All the entries of the original matrix $\mathbf{C} = \{c_{ij}\}$ are taken and sorted in ascending order, and the value following c^* in the matrix is considered as new c^*. Steps 2 and 3 are then repeated until a solution is obtained.

Note that the Hungarian Method is used to find a perfect matching of minimum cost and in general deals with matrices with finite entries. But in step 2 of the algorithm, entries with values +∞ are introduced: hence the Hungarian Method may not be able to return a finite value solution. In cases like this, the algorithm moves on to step 4, updates the threshold c^* to the value immediately larger than the previous (which cannot be accomplished), and tries again to find a perfect matching of minimum cost.

To explain the procedure described, consider the following example. Let the cost matrix be:

$$\mathbf{C} = \begin{pmatrix} 8 & 2 & 3 & 3 \\ 2 & 7 & 5 & 8 \\ 0 & 9 & 8 & 4 \\ 2 & 5 & 6 & 3 \end{pmatrix}$$

An optimal solution (which always exists) is obtained with the following steps:

I. We first set $c^* = \max\{2, 2, 0, 2, 0, 2, 3, 3\} = 3$.

II. Now we modify matrix $\mathbf{C}$ by inserting the entries with value +∞; we obtain

$$\mathbf{C}' = \begin{pmatrix} \infty & 2 & 3 & 3 \\ 2 & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ 2 & \infty & \infty & 3 \end{pmatrix}$$

III. At this point, however, the Hungarian Method cannot find a finite solution, so we move to step 4 of the algorithm.

IV. The ordered list of the entries of matrix $\mathbf{C}$ is $(0, 2, 2, 2, 3, 3, 3, \mathbf{4}, 5, 5, 6, 7, 8, 8, 8, 9)$. The threshold is then updated by selecting the value immediately after 3: so, we set $c^* = 4$. We modify again $\mathbf{C}$ by inserting the entries with value +∞, obtaining now

$$\mathbf{C}' = \begin{pmatrix} \infty & 2 & 3 & 3 \\ 2 & \infty & \infty & \infty \\ 0 & \infty & \infty & 4 \\ 2 & \infty & \infty & 3 \end{pmatrix}$$

V. The Hungarian Method again fails to find a finite value solution: in the ordered list we then take $c^* = 5$ and modify $\mathbf{C}$ by inserting the entries with value +∞. We now get

$$\mathbf{C}' = \begin{pmatrix} \infty & \mathbf{2} & 3 & 3 \\ 2 & \infty & \mathbf{5} & \infty \\ \mathbf{0} & \infty & \infty & 4 \\ 2 & 5 & \infty & \mathbf{3} \end{pmatrix}$$

VI. The Hungarian Method now finds the finite value solution $(x_{12},\ x_{23}, x_{31}, x_{44})$ indicated in bold, with arc weights $(2, 5, 0, 3)$. This solution identifies an optimal bottleneck matching of value $c^* = 5$.

5.4 Implementation and Results

5.4.1 Technological tools: GuRoBi® solver

A solver is a software, often implemented in the form of a standalone program or software library, that solves a mathematical problem. A solver takes problem data as input in a specified format and outputs

one or more possible solutions. For mathematical programming problems, and more particularly linear programming problems, a wide range of commercial or academic solvers are available. Generally, these are codes licensed for use, and the license can be free or paid.

Application problems normally have a consistent number of parameters, variables, and logic-mathematical relationships. For this reason, it is necessary to use automatic calculation systems that can be easily interfaced with databases or spreadsheets. There are many systems, differing in performance and data management capabilities, with these characteristics. Some products integrate a modeling language with the solver (s) into a single commercial package. These are optimization products including the computing environment, called "stand-alone" modeling systems, which can provide the complete interface between the different levels of formulation, solution, and analysis. Stand-alone systems tend to be the most advantageous at the prototype construction level, when the work is focused on building an acceptable model and demonstrating that the approach is promising enough to warrant higher investments.

GuRoBi® Optimizer is an engine used to transform data into smarter decisions. It allows users to declare their toughest business problems as mathematical models and then automatically consider billions or even trillions of possible solutions to find the best one. Zonghao Gu, Ed Rothberg and Bob Bixby, founded GuRoBi in 2008. In case you were wondering, the Company name is derived from the first two letters of the founders' last names (Gu-Ro-Bi).

GuRoBi® has been used to produce measurable improvements in a wide range of high-value business functions, including manufacturing, distribution, purchasing, finance, capital investment and human resources. GuRoBi® has proven to be both robust and scalable and can solve problems involving millions of decision variables.

GuRoBi® Optimizer is a cutting-edge solver for mathematical programming. GuRoBi® Optimizer solvers have been designed from the ground up to take advantage of modern architectures and multicore processors, using the most advanced implementations of the latest algorithms. It includes the following solvers: 1) linear programming solver (LP), 2) mixed-integer linear programming solver (MILP); 3) mixed-integer quadratic programming solver (MIQP); 4) quadratic programming solver (QP); 5) quadratically constrained programming solver (QCP); 6) mixed-integer quadratically constrained programming solver (MIQCP).

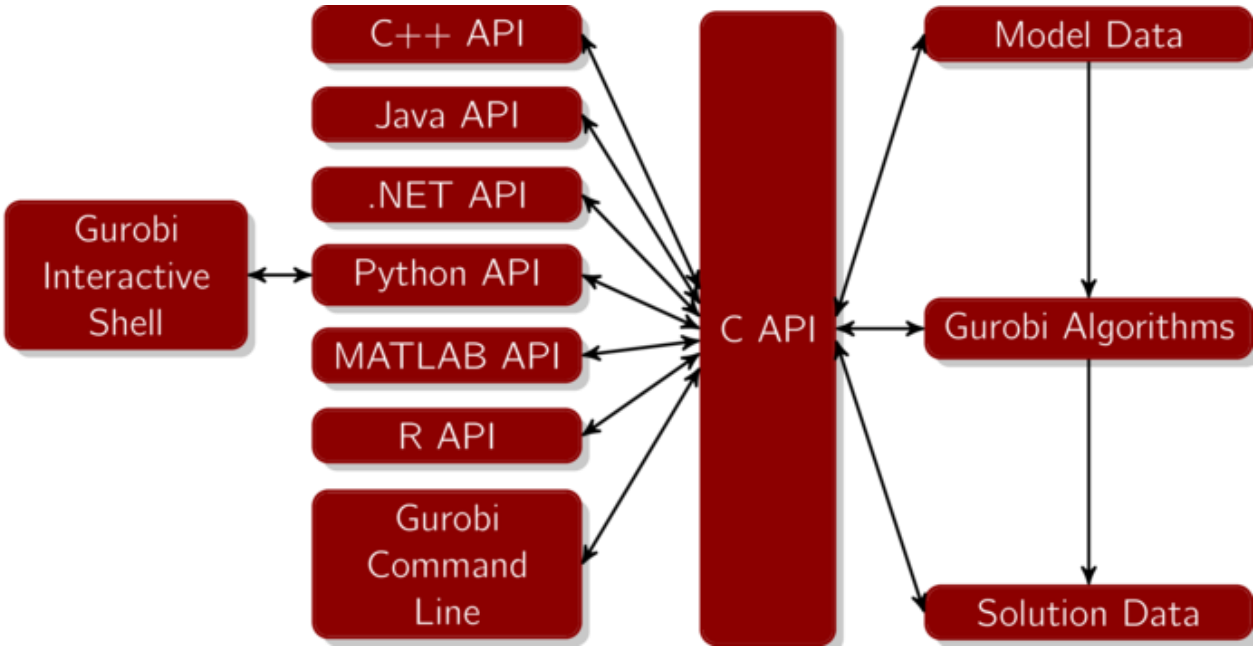


Fig. 5-7) The Gurobi Optimizer models.

To maximize productivity, it supports interfaces for a variety of programming and modeling languages: 1) object-oriented interfaces for C ++, Java, .NET, and Python; 2) matrix-oriented interfaces for C, MATLAB, and R; 3) link to standard modeling languages: AIMMS, AMPL, GAMS and MPL; 4) link to Excel through the Premium Solver Platform and the Risk Solver Platform.

GuRoBi® algorithms are all implemented in C (for performance and portability reasons). All the other *APPLICATION PROGRAMMING INTERFACES* (APIs) are just thin wrappers around the C API. Everything is handled transparently, so one does not see a C program in the background. Of course, translating from an API into C has some small overhead, but usually negligible, except for cases where one wants to solve hundreds or thousands of small models per second, so that most of the time is spent on building the model and passing around data.

GuRoBi® offers all object-oriented interfaces and matrices are implemented as modern and lightweight APIs. As a result, they are faster and use less memory than competing alternatives. Furthermore, the interfaces are designed to be consistent and intuitive.

The GuRoBi® interactive interface is based on the object-oriented Python API. A significant advantage is the ability to use this interface not only as an easily accessible environment for running and testing models, but also as a development environment that can be used to build complex models and then pass these models to full applications. Python offers a full range of preconfigured libraries to support full application development, including exceptional data access capabilities [33].

5.4.2 Technological tools: Anaconda and Jupyter Notebook

Anaconda is a free and open-source distribution of the *Python* and *R* programming languages for scientific computing (data science, machine learning applications, large scale data processing, predictive analytics, etc.), which aims to simplify the package management and implementation.
Package versions are managed by the package management system. The *Anaconda* distribution is used by over 6 million users and includes over 1400 popular data science packages suitable for Windows, Linux and MacOS.

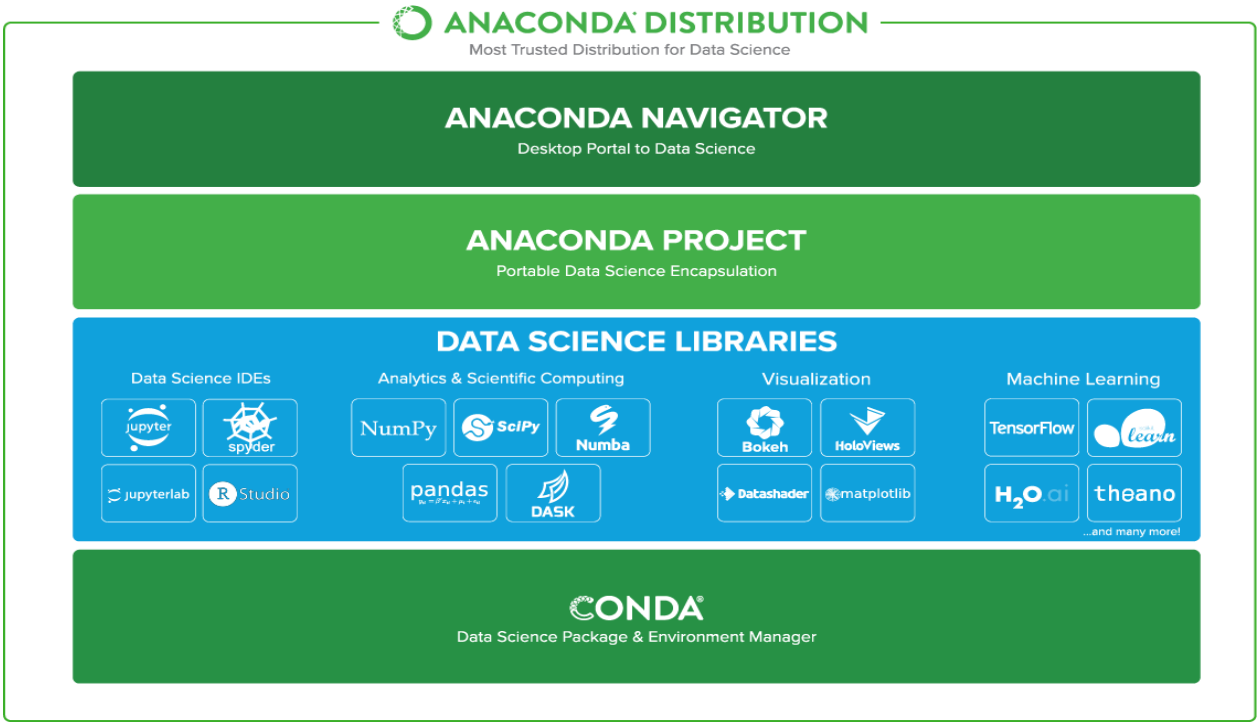


Fig. 5-8) Services provided by Anaconda Distribution.

Anaconda Distribution contains:

- The *Conda* package manager, used to install, run, and update *Anaconda* packages and their dependencies.
- *Anaconda Navigator*, a virtual environment manager.
- Hundreds of community-curated science packages for download and use.

This eliminates the need to learn how to install each library independently: it is in fact possible to download the packages together with their dependencies with a simple command “conda install {package name}”.

Anaconda is also useful for installing python on different platforms, separating different environments by switching between them locally, managing the lack of correct privileges, and preparing and running specific libraries and packages.

Anaconda Navigator is a desktop graphical user interface included in Anaconda Distribution that allows users to be able to launch various applications and manage conda packages, environments, and channels without having to use commands via the command line.

Navigator can search for packages on the *Anaconda Cloud* or in a local *Anaconda Repository*, install them in an environment, run the packages and update them. Ultimately it is widely used because:

- Many scientific packages depend on specific versions of other packages to run. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.
- The *conda* program is both a package manager and an environment manager, to help data scientists ensure that each version of each package has all the necessary dependencies and works correctly.
- Navigator is an easy way to work with packages and environments without the need to type *conda* commands in a terminal window. One can use it to find the packages one wants, install them in an environment, run the packages and update them, all within *Navigator* [34].

Jupyter Notebook (formerly *IPython Notebooks*) is a Web-based interactive computing environment for creating documents for Jupyter notebooks. The term "notebook" can colloquially refer to many different entities, mainly the *Jupyter web application*, the *Jupyter Python web server*, or the *Jupyter document format*, depending on the context. A *Jupyter Notebook* document is a JSON document, which follows a versioned pattern and contains an ordered list of input / output cells which can contain code, text (using Markdown), math, graphics, and more, usually ending with ". ipynb "extension.

A *Jupyter Notebook* can be converted to several open standard output formats (HTML, presentation slides, LaTeX, PDF, ReStructuredText, Markdown, Python).

Jupyter Notebook can connect to many kernels to allow programming in different languages. By default, *Jupyter Notebook* is directed towards the *IPython* kernel. A *Jupyter kernel* is a program responsible for handling various types of requests (code execution, code completion, inspection) and providing a response [35].

5.4.3 Computational resources and algorithm breakdown

The method developed employs the GuRoBi® to solve the MIP model described in section 5.2.1, while an ad-hoc *Python 3* implementation [36] was devised for the part concerning distance calculations, cart scheduling and cart-to-span assignment (section 5.3).

The algorithm was encoded in Python 3.7.6. and run on an Intel core 7,1.8-1.99 GHz processor, 16 GB RAM, operation system Windows 10 Pro 64-bit version 2004.

The algorithm developed is broken down into the following separate stages:

1. We start from the requests-positions graph representing the problem and the related requests-span graph is created with the necessary attributes associated with nodes and arcs.
2. Through the GuRoBi® package, a MIP model of the problem is created from data of the request-span graph: initially the model and the variables of the problem are created, then the constraints are inserted and the objective function, finally the solver is called to find an optimal solution.
3. The solution obtained by GuRoBi® contains a set of as many spans as carts available: a new bipartite graph is then created, with node partition representing the carts and the spans chosen.
4. Costs of all the possible individual span-to-cart assignments are used to label the arcs of the bipartite graph. These costs update the span labels by considering the (possibly non-null) time required of the cart to reach the span, the cart movement within the span and the time for request load/unload: this computation is carried out by a Python code that, according to the directions assigned to the span (forward, backward, and single, double or triple) and according to the initial position of the cart (within, to the left or to right of the span), determines the shortest possible schedule (therefore, cart scheduling is in this way simultaneously computed).
5. The coefficients obtained for each cart-span pair are used to construct an instance of BOTTLENECK MATCHING PROBLEM. Two subroutines are called: one performs the steps described in section 5.3.2, the other encodes the Hungarian Method.

The method so developed returns as output

a. the value of the largest (bottleneck) completion time of a cart schedule,
b. the request-to-span and span-to-cart assignments,

c. the shortest cart schedule per request cluster (i.e., per span selected), including such details as the direction that each cart has to take to reach the span assigned, and the order in which all the assigned requests are to be covered.

5.5 Experimental setting and numerical tests

5.5.1 The static scenario

In the static scenario, we considered single snapshots taken from real data obtained from *LFoundry* (see Section 2.4) as an input for our model. Tab. 5-1 below gives details on these static tests.

Snapshots (*LFoundry* Server)	**Number of requests**	c_{max} **first phase**	**CPU time (sec)**	c_{max} **second phase**
S1	65	350.8	3.5	470.8
S2	60	327.5	4.7	480.9
S3	60	337.5	4.1	500.8
S4	59	362.5	4.7	457.5
S5	55	314.2	3.6	470.3
S6	65	337.5	4.5	410.6
S7	65	327.6	4.3	413.8
S8	77	419.3	4.1	485.2
S9	72	404.2	3.9	590.8
S10	63	343.2	4.4	440.3
S11	63	344.2	10.9	464.2
S12	64	359.2	9.7	430.4
S13	63	322.5	12.8	495.9
S14	65	328.6	10.8	530.8
S15	68	322.5	10.4	515.2
S16	61	344.2	8.9	440.3
S17	61	343.8	11.6	440.7
S18	60	322.5	9.1	529.2
S19	62	329.2	10.1	485.1
S20	58	322.5	11.4	485.83
Average	63	343.2	7.4	476.9

Tab. 5-1) The results of static scenario from plant.

Models and algorithms were encoded in Python 3.7.6, and all integer programs were solved by GuRoBi® 9.0.2 with default settings using 8 threads. All algorithms ran on an Intel core 7,1.8-1.99 GHz processor, 16 GB RAM, operating system Windows 10 Pro 64-bit version 2004.

The first column in the table reports each single snapshot used as input for the model. The second column gives the number of requests in each snapshot. The third column displays the optimal c_{max} obtained after the first phase of our method, explained in section 5.2, where we employed GuRoBi® to solve the MIP model and find an optimal request-to-span assignment. Optimal (that is, 0% gap) solutions were found for each snapshot: CPU times can be read in column 4. Finally, column 5 shows the corrected optimal c_{max} found after the second phase of our method, explained in section 5.3, where we matched the carts available to the spans selected in the first phase.

5.5.2 The Dynamic scenario

The samples that we use for dynamic scenario are described below:

- Each sample describes a work shift of over 6 hours and is formed by a series of snapshots (one per minute) of request status. Besides other information, a snapshot gives the requests pending, those in process and newly arrived ones, all recorded according to the actual practice of cart routing and scheduling.
- On average, 55.7% of the requests are short distance, that is, cover < 6 stations in the aisle, and only 1.8% are long-distance, that is, range over all the 14 stations (as we said in section (2.4)). More features are given in Tab. 5-2, for the five samples used in the numerical experience in dynamic scenario.

Sample	**Actual Duration (min) #Snapshots**	**Requests per snapshot**				**Requests per sample**
		min	**max**	**average**	σ	
1	374	38	96	91	0.2	3,110
2	351	32	95	86	0.6	3,076
3	343	41	96	80	0.7	2,984
4	342	45	94	86	0.5	2,902
5	347	54	95	82	0.8	2,943
Total	**1,757**					**15,015**

Tab. 5-2) Sample work shift features.

Recall that by our assumption on scheduling (second phase of our model (5.3)), unless an exception occurs, every cart moves end-to-end in the assigned span minimizing direction changes. This simple idea can be implemented by different policies that we mentioned in section (5.3). We will see later that the best scheduling method adopted in our experiments follows Policy 1.

The computational experience in the dynamic scenario was carried out with the following procedure, see also the scheme in Fig. 5-9:

Step 1. **Initialize**. Construct the list L of all the requests in a sample, each with the time of its arrival. Let $t = 0$ and let R_0 contain the requests of the first snapshot.

Step 2. **Cycle**. Run the math heuristic model (5.2.1) on all requests R_t, let C_{max} and C_{min} be the time length of the longest and the shortest route of a cart. Take note of the status (minimum ready time t_k of cart and particular position of that on that time (P_k)) of all the carts at $t + C_{max}$.

Step 3. **Update and Repeat**. Set $t := t + C_{min}$, create R_t including in it all the requests of L that arrived within $[t - C_{min}, t]$. Considering cart status, go back to step 2 and repeat until all the requests in L have been fulfilled.

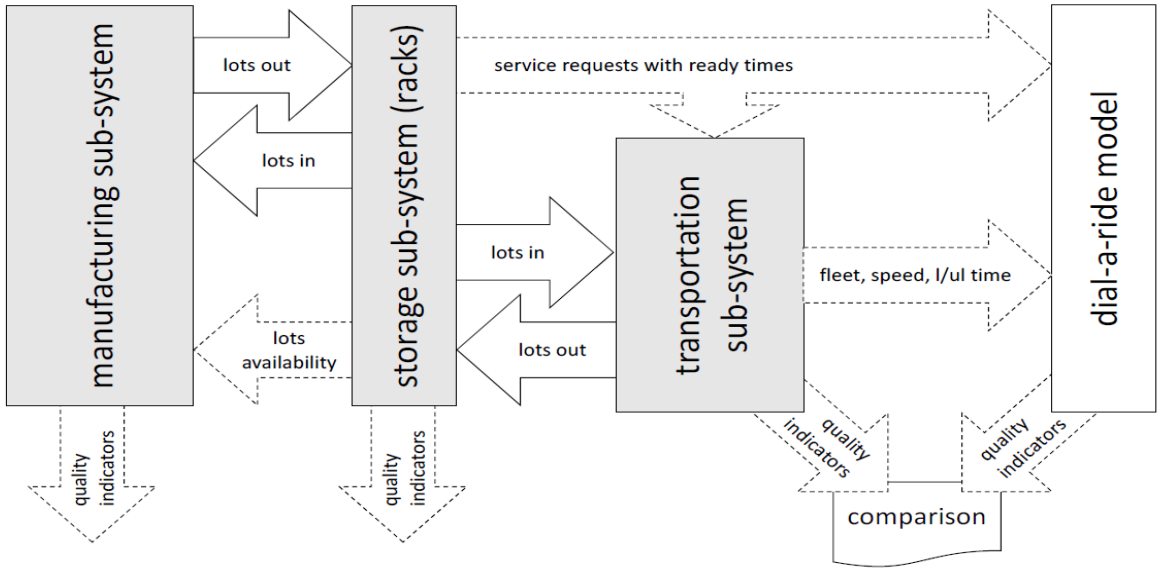


Fig. 5-9) Scheme of computational experience: plant sub-systems (grey blocks), software components (white blocks), physical flows (solid arrows), main information flows (dashed arrows).

We took note of the time t required to complete all the deliveries in L and compared it to the actual practice.

We used the same environment and computational resources as in the static scenario: in particular, the MIP solver adopted is GuRoBi® 9.0.2 with default settings using 8 threads. The number of variables ranges from a minimum of 2,667 to a maximum of 6,767, that of constraints from 2,669 to 6,333. All problems were solved to proven optimality.

The outcome is reported in Tab. 5-3. All in all, we processed 15,015 requests for a total work time of 1,757 minutes, see column 2. The result of a numerical test gives the values in columns 3-4. The test results in a rough 30% improvement over actual practice, amounting in absolute terms to over eight hours and $^{3}/_{4}$, potentially saved in five work shifts.

Sample id	Actual duration	Our result	%Improvement
1	374	252	32.7%
2	351	254	27.5%
3	343	248	27.7%
4	342	246	28.1%
5	347	231	33.5%
Total	**1,757**	**1,231**	**29.9%**

Tab. 5-3) Algorithm performance. Minutes to complete operation, actual duration vs. tests with our model and percentage reductions.

Every cycle (step 2 of the above procedure) took in this case about 46.4 seconds CPU time on average, ranging from a minimum of 4.9 to a maximum of 147.8 seconds, all values compliant to on-line request management.

We also tested our optimization method under Policy 2. Over all samples, this scheduling policy returned a completion time of 1,477 minutes, that is 246 minutes (about 20%) worse than Policy 1 but still 280 minutes (16%) better than actual operation.

Besides makespan, it is interesting to observe how cart workloads are distributed with the proposed approaches. Tab. 5-4, gives, per sample, the mileage covered by the carts. Also, under this respect Policy1 gives the best result, returning an average cart mileage 15.7% shorter than that attained by Policy 2 (last row of Tab. 5-4). As for route length distribution, the largest cart mileage with Policy 1 is from 11.6 to 12.9 km shorter than with Policy 2. In absolute terms, Policy 1 also improves the mileage unbalance of Policy 2 of amounts ranging between 15.4% and 16.8%.

However, the relative unbalance

$$U_R = \frac{longest\ route - shortest\ route}{longest\ route} \qquad \textit{(Eq. 5-29)}$$

is quite large with both policies, from 48.0% to 51.6% (51.2% with Policy 2). To correctly read this information one has however to consider that integer linear programming problems tends to assign less lots to longer routes, to compensate travel and load/unload time.

Sample id	Cart mileage (km)					
	Policy 1			Policy 2		
	average	max	diff.	average	max	diff.
1	41.1	63.9	31.5	49.5	76.7	37.8
2	42.6	64.2	30.8	50.6	77.1	37.0
3	39.3	62.8	31.7	46.2	75.4	38.0
4	39.6	62.1	31.2	46.5	74.5	37.4
5	38.5	63.8	31.5	48.9	76.6	37.5
Total average	**41.2**			**48.9**		

Tab. 5-4) Mileage balance. Kilometers per cart to complete operation: average, maximum and unbalance (diff.) per sample and scheduling policy.

Finally, a simple test of the robustness of the method against parameter variation was done with a new run of all samples, randomly choosing cart speed and load/unload time uniformly picked in $1.2 \pm 0.1\ m/s$ and $15 \pm 2\ s$. With both Policy 1 and 2, this experiment returned a total completion time substantially close to the values found with constant parameters.

5.6 Conclusions and future research

We tackled a challenging VEHICLE ROUTING PROBLEM arising in the wafer fab of *LFoundry*, Italy. The problem has the form of INVENTORY DIAL-A-RIDE on a rectilinear topology. Since an exact MIP model turned out not viable for both problem size and on-line requirements, we developed a math-heuristic that, after removing inventory constraints, decomposes the problem following a *cluster-first route-second* approach. Clusters are obtained by solving an integer linear program, then are assigned to vehicles by solving a bottleneck matching problem: carts schedules are found by simple sweep heuristic method, exploiting the rectilinear geometry of the plant. One of the sweep heuristics is conceived to maximize the chances that inventory levels at the pick-up and drop locations are maintained within capacity limits.

The output of the method developed was used in a large-scale numerical test, carried out in both a static and a dynamic scenario, from which we got clear indications of possible improvement of plant operation. The improvement becomes more sensible if a more accurate estimate of scheduling time is used in the routing: this entails a trade-off between CPU time and solution quality although, in general, the method is very efficient in computational terms and compliant with on-line operation. To address the rare cases of non-compliancy, one should accelerate the resolution of integer linear program, which largely dominates the total solution time.

Future research could then consider exploiting and strengthen, if possible, the highlighted p-centre structure of the model. Further improvements of solution quality could be obtained by refining individual cart scheduling, also considering different policies. Prior to algorithm implementation in the plant, additional effort should finally be devoted to construct a simulation model of the manufacturing sub-system to observe the effect of an optimized transportation sub-system on the process of request generation.

6 Robust BIN PACKING

In this chapter we regard the BIN PACKING problem as a model for deciding the cutting operations that are necessary to obtain a given set of items from rectilinear bars in such a way that trim loss is minimized. In this scenario, we consider the effect on production quality of defects that may randomly arise in the bars. We propose a mathematical model to deal with this situation, first by defining and measuring the robustness of a cutting pattern, then by evaluating the Expected Economic Loss (EEL) due to defects in the bar, and finally trying to construct solutions that minimize such a loss for a given number of bars. We observe that even determining the robustness of a given pattern is NP-hard. The problem of finding a robust bin packing, of greater complexity, is approached by two math-heuristics based on the solution of specific MULTI-PROCESSOR SCHEDULING (MPS) problems. The efficiency and effectiveness of the solution methods is evaluated on problem instances taken from the bin packing literature.
The efficiency and effectiveness of our method to reduce the EEL is evaluated on examples taken from literature [37]. The result of numerical tests is determined by the recourse option chosen. Consider the following example:

1. When a solution is computed, the bins that make up the solution are delivered in order, each with its own set of items. Although item assignment cannot be changed, in the situation of a defective bin, the items should be rearranged to minimize the actual economic loss.
2. The bins do not arrive in any order, even though the solution cannot be changed: As a result, one can discover defective bins in advance and choose the most advantageous item-to-bin assignments dictated by the solution, rearranging the things inside each faulty bin to minimize economic loss.

Option 1 is the one considered in this study, Option 2 and possible variants will be the subject of future research.

6.1 Basics of the BIN PACKING PROBLEM

The ONE-DIMENSIONAL BIN PACKING PROBLEM (1-BPP) requires to packing a given set I of items of distinct lengths $w_i \in \mathbb{Q}_+ \,,\ i \in I$, into a minimum number of bins of the same length $w > w_i$. The cost of a solution is traditionally defined by the number of bins used for the packing. In a slightly different version of the problem, we may consider this cost against the utility generated from accommodating items. If $e_i \in \mathbb{R}_+$ is the economic value of the generic item i (a value that we suppose normalized to the bin value), the problem of choosing the items to pack so that net utility, i.e., the total value of the items packed minus the number of bins used, is maximized. Under this assumption we consider a situation in which no more than d_i items of type i can be produced.

The 1-BPP is closely related to the ONE-DIMENSIONAL CUTTING STOCK PROBLEM (1-CSP): here, items must be cut from bars of length w (which is in fact equivalent to packing). The difference is that each item I is possibly demanded in multiple copies. The 1-CSP appears then as a generalization of the 1-BPP, but the basic impact of this difference is on the bit complexity of problem instances.

Both the 1-BPP and the 1-CSP admit a MIP formulation based on assignment variables x_{ij}, which give the number of items of type i that a solution assigns to the $j-th$ bin (for the 1-BPP, this value is either 0 or 1, while for the 1-CSP is a positive integer). This formulation, attributed to Kantorovich, presumes the knowledge of an upper bound m to the number of bins involved in a solution. Indicating by J this set of m bins, we can write:

$$max \sum_{i \in I} \sum_{j \in J} e_i x_{ij} - \sum_{j \in J} y_j \qquad \textit{(Eq. 6-1)}$$

subject to

$$\sum_{i \in I} w_i x_{ij} \leq\ w y_j \qquad \forall\, j \in J \qquad \textit{(Eq. 6-2)}$$

$$\sum_{j \in J} x_{ij} \leq d_i \qquad \forall\, i \in I \qquad \textit{(Eq. 6-3)}$$

$$x_{ij} \geq 0\ and\ integer\ \forall\, i \in I, j \in J \qquad \textit{(Eq. 6-4)}$$

$$y \in\ \{0,1\}^m \qquad \textit{(Eq. 6-5)}$$

Variables y_j count the number of bins used, that is, $y_j = 1$ if and only if the $j-th$ bin contains an item. The above formulation gives a model for utility maximization. If item requirements must be fulfilled, then (Eq. 6-3) must be written with the sign "=" and in this case the total economic value of production amounts to a constant value.

A different formulation, due to Gilmore and Gomory, uses the notion of *cutting pattern* (in short, *pattern*). A pattern describes how to fit a subset of *I* into a single bin. In the 1-dimensional case, it has only to fulfil

$$w_0 = w - \sum_{i \in I} w_i \, a_i \geq 0 \qquad \textit{(Eq. 6-6)}$$

If $a_i = 1$, item i is packed in the bin, if $a_i = 0$ it is not; w_0 gives the unused space in the bin and is called the pattern *leftover*. A pattern is called *maximal* if no additional item from *I* can be added to it while satisfying inequality (Eq. 6-6). A pattern can therefore be defined as any solution $a = (a_1, \ldots, a_{|I|})$ of a 0-1 (or, for the 1-CSP, INTEGER) KNAPSACK PROBLEM. In the following, we will also indicate a pattern by a pair (w_0, P), where *P* is the set denoted by the incidence vector a.

6.2 Pattern robustness

In the 1-BPP, the order in which a bin is filled with items from *P* is clearly irrelevant. In cutting stock applications, however, this sequence can have a significant impact. In fact, if bins represent physical bars that are possibly prone to defects, positioning the items in the bin may or may not cause an item to be hit by a defect. This consideration raises then an issue of solution robustness with respect to possible defects.

In detail, when the pieces of *P* can be packed so that the defect hits the leftover section, we say that *P* is *robust* for that defect position. The operation of rearranging item positions within the bin is called the *pattern recovery*. So, a problem first arises of understanding whether a pattern is or not recoverable.

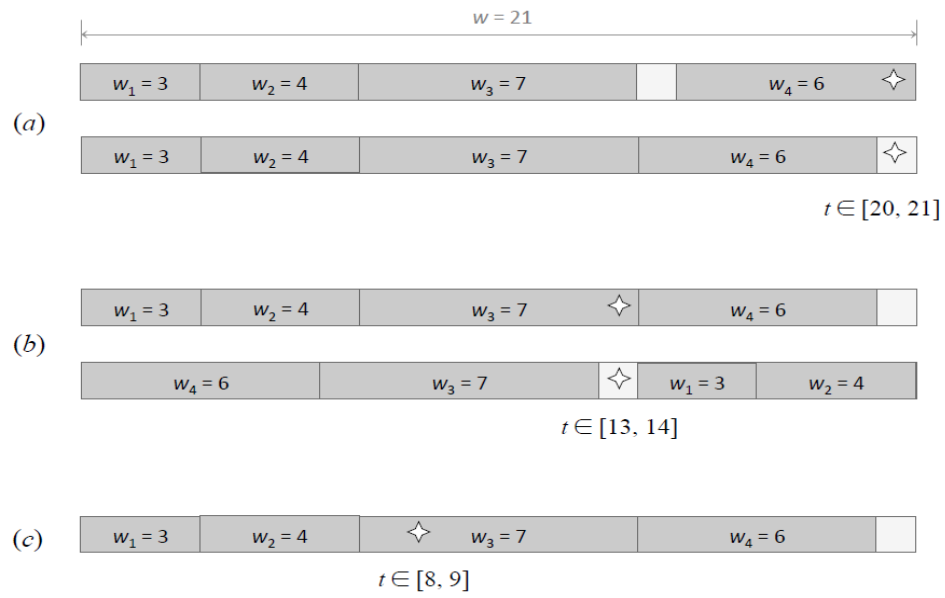


Fig. 6-1) (a)-(b) Reconfigurable and (c) non-reconfigurable pattern for fault in position t.

Fig. 6-1, shows a pattern and the possibility of recovering, depending on defect position. In practice, P can be recovered if its items can be partitioned in two sets L and $R = P - L$, so that those in L precede those in R and the former (the latter) lie all to the left (to the right) of the fault position t. Note that if P is t-reconfigurable for some t, it is also $(w - t)$-reconfigurable[6].

To the best of our knowledge, the earliest reference to a problem of this kind is [38], which considers a problem called MIN DEFECTIVE SUBSET SUM. A single defective interval is present at a specific position in the bin, and the goal is to permute items so that the number of lost products is reduced. This problem is shown to be NP-complete, is reformulated as MULTIPLE SUBSET SUM and is solved by branch-and-bound. Unlike [38], however, in our setting the defect position is not specified a-priori, and we define the *pattern robustness* as the chance of recovering all items in presence of a defect randomly occurred in the bin.

6 The notion can be generalized to r bin defects: for $1 = t_0 \le t_1 \le \cdots \le t_r \le t_{r+1} = w$. We say that P is $(t_1, \dots, t_r)$-reconfigurable if it has an $(r+1)$-partition in subsets S_j such that $\sum_{i \in S_j} w_i \le t_j - t_{j-1}$ for $j = 1, \dots, r+1$. We however suppose that the simultaneous occurrence of more defects in a bin is a rare event, and hence only focus on t-reconfigurability. Non-infinitesimal spots of given length ϵ are an easy generalization, as it suffices to change the right-hand sides of ((Eq. 6-8) and (Eq. 6-9)) into $t - 1 - \frac{\epsilon}{2}$ and $t - w + \frac{\epsilon}{2}$.

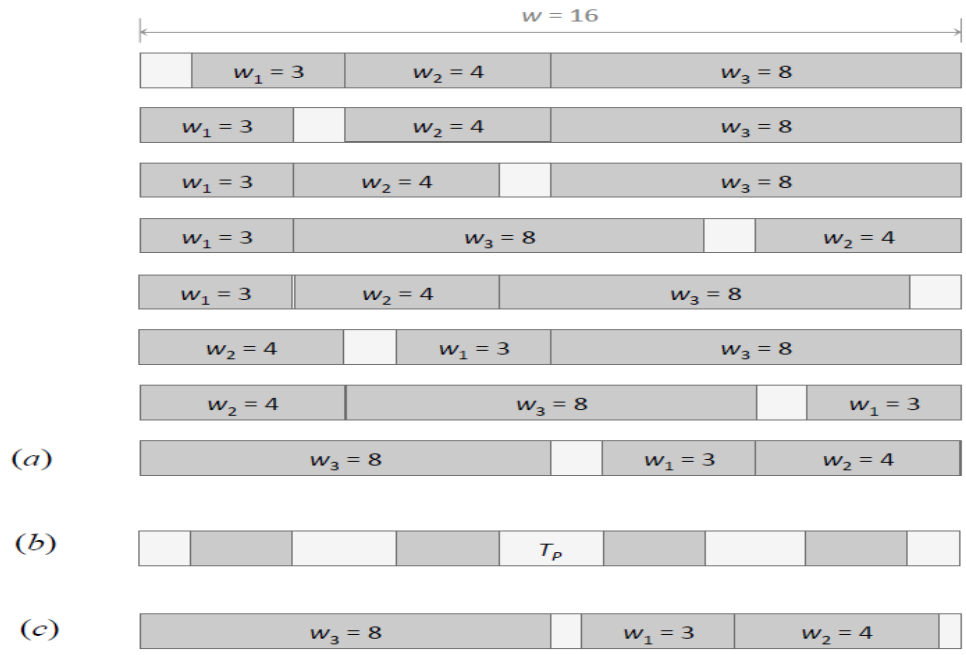


Fig. 6-2) (a) Reconfiguration intervals, (b) reconfiguration set and (c) a generic reconfiguration implied by T_P.

Let us describe the way we adopt to measure the robustness of a pattern P. All the $t \in [1, w]$ for which P is t-reconfigurable form a set $T_P \subseteq [0, w]$. Fig. 6-2 shows how T_P is built from a finite number of reconfiguration intervals. These are 2^n (Fig. 6-2-a), and are generated from subsets $L \subseteq P$ by moving all the items of L to the left side of the bin and the remaining ones to the right: Fig. 6-2-c shows that no more sets are necessary to build T_P, as any other way of t-reconfiguring a pattern is implied by a subset of reconfiguration intervals. Clearly, $T_Q \supseteq T_P$ for any subset Q of P.

Referring to a unit length bin ($w = 1$) and normalized item lengths $w_i \in [0,1]$, the measure of T_P gives the *probability of saving all the items of P* after a single fault uniformly distributed at random in the bin:

$$\pi_P = \mu(T_P) = \int_0^1 t\eta_P(t)dt = 2\int_0^{\frac{1}{2}} t\eta_P(t)dt \qquad \textit{(Eq. 6-7)}$$

where, for any $t \in [0,1]$, $\eta_P(t) = 0$, if P is t-reconfigurable and 1 otherwise. (i.e., $\eta_P(t)$ is the characteristic function of T_P). By (Eq. 6-7), the larger the π_P, the more P is robust against faults, thus the formula provides a measure of pattern robustness.

6.3 Computing pattern robustness

In this section we assimilate a defect to a point-shaped spot in the bar and assume integer item and bin lengths. In this case formula (Eq. 6-7) reduces to the solution of w 0-1 ILPs. Two cases in fact occur of a given pattern P: either the defect can be isolated within the pattern trim loss (Fig. 6-1, (a)-(b)), or this operation is impossible and therefore an item should be discarded (Fig. 6-1, (c)). Specifically, let P be a pattern assigning $n = \sum_{i \in I} a_i$ items to a bin, and let $t \in [1, w]$. Then P is t-reconfigurable if there exists $L \subseteq P$ such that

$$\sum_{i \in L} w_i \leq t - 1 \qquad \textit{(Eq. 6-8)}$$

$$\sum_{i \notin L} w_i \leq w - t \qquad \textit{(Eq. 6-9)}$$

We then easily derive a mathematical model for t-reconfigurability:

Model 6-1

Variables:

- $x_i \in \{0,1\}$ for $i \in I$ with $x_i = 1$, if and only if $i \in I$ is assigned to L
- z: continuous variable representing $w(L)$

Constraints:

$$\sum_{i \in P} w_i x_i \geq z \qquad \textit{(Eq. 6-10)}$$

$$\sum_{i \in P} w_i x_i \leq t - 1 \qquad \textit{(Eq. 6-11)}$$

Objective: Maximize z

Once given an optimal solution L of value z^* is found, P is $t-$reconfigurable if $z^* \geq (\sum_{i \in L} w_i) - w + t$. Hence, π_P is computed by repeatedly solving the problem for $t = 1, \ldots, w - 1$, counting how many times the above inequalities are fulfilled by z^*, and dividing this number by w.

6.4 Expected Economic Loss

If a pattern is found to be unrecoverable for a certain defect position, a loss is incurred because an item will be for certain affected by the fault. Let e_i denote the value of item i: we either assume this value independent on the item, or proportional to the item length w_i.

While meaningless for a scheduling problem, in cutting problems this value may derive from material recycle but not from reuse (because, say, reworking the leftover to obtain an item is not worth the effort).

If all the items have (the leftover has) a length-independent economic value e_1 (e_0), the economic loss $\ell(P,t)$ due to a defect in position t is e_0 if P is t- reconfigurable and e_1 otherwise. Hence, the Expected Economic Loss (EEL) for a uniformly random fault in the bin is

$$ee\ell(P) = e_0\pi_P + e_1(1-\pi_P) \qquad \textit{(Eq. 6-12)}$$

For different economic values of the items, the EEL for a single fault in a bin has a more complex expression (Eq. 6-12). In general, calling π_i the probability of losing item i, and writing $\pi_0 = \pi_P$, we have for the EEL

$$ee\ell(P) = e_0\pi_{P_0} + \sum_{i\in P} e_i\pi_{P_i} \qquad \textit{(Eq. 6-13)}$$

Note that the EEL computation is subject to a sort of "optimized" choice, as one will always try to discard the less valuable items in P (remark that computing the EEL is not the same as the deterministic problem of finding a minimum cost reconfiguration after a defect is found in t).

For integer item and bin lengths, the *Expected Economic Loss* (EEL) is easily obtained by averaging the values of the defective item that cannot be avoided, for t varying between 0 and w. One can easily see that, under the assumptions made, if a pattern is not recoverable for some t, the loss is limited to the item of smallest value, that is, e_1. Therefore, the EEL can be computed by summing the following values for $t = 1,\dots,w$: e_1 if P is not t −recoverable, 0 if P is t −recoverable

6.5 Best pattern reconfiguration

Let us now address the following problem:

> Given a pattern P with a defect in position t and the economic values $e_1,\dots,e_n$ of its items, reconfigure P to minimize the economic loss $\ell(P,t)$.

This problem is easily formulated as an integer linear program to be solved for integer $t \in [1, w]$. Based on these solutions, we can then maximize the expected gain (EG) and expected profit (EP) of a given packing into m bins.

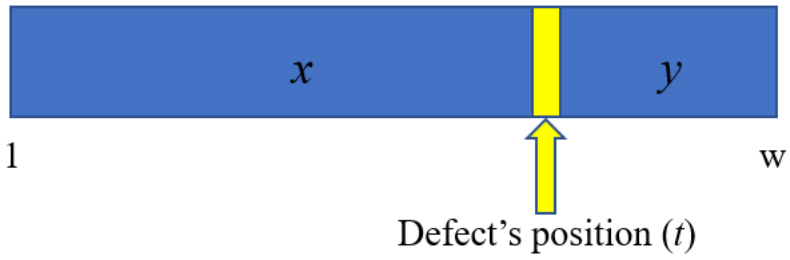


Fig. 6-3) Defect 's position in a bin and presenting new model with minimum loss.

Model 6-2

Parameters:

- P: set of items in the bin, $|P| = n$
- t: defect position, integer $\in [1, w]$

Variables:

- $x_i \in \{0,1\}$ for $i \in I$, with $x_i = 1$, if $i \in I$ lies to the left of t
- $y_i \in \{0,1\}$ for $i \in I$, with $y_i = 1$, if $i \in I$ lies to the right of t

Constraints:

$$x_i + y_i \leq 1 \quad i \in I \qquad \textit{(Eq. 6-14)}$$

$$\sum_{i=1}^{n} w_i x_i \leq t - 1 \qquad \textit{(Eq. 6-15)}$$

$$\sum_{i=1}^{n} w_i y_i \leq w - t \qquad \textit{(Eq. 6-16)}$$

Objective:

$$z^*(t) = max\left(\sum_{i=1}^{n} w_i y_i + \sum_{i=1}^{n} w_i x_i\right) \qquad \textit{(Eq. 6-17)}$$

Expected Gain and Profit are then computed by averaging on the $z^*(t)$ obtained for all possible values of t.

Expected Gain (EG):

$$EG = \frac{1}{W}\sum_{t=1}^{W}(\ z^*(t))\ \ (for\ any\ bin) \qquad \textit{(Eq. 6-18)}$$

Expected Profit (EP):

PF: profit factor, p: defect probability and bw: unit cost of raw material for each bin

$$EP = PF * \left(p * EG + \ (1-p)\sum_{i=1}^{I_b} w_i \right) - (bw)\ (for\ each\ bin\ or\ pattern) \qquad \textit{(Eq. 6-19)}$$

6.6 Finding a robust BIN PACKING

The problem of finding a robust BIN PACKING consists in minimizing either the EEL for a given number of bins, or the total expected production cost, which is the EEL plus the cost of the bins used to pack the item set. Specifically, the latter problem is defined as follows:

> Given a set I of items with lengths $w_i \in \mathbb{Q}_+$ and economic values $e_i \in \mathbb{R}_+,\ i \in I$, accommodate them into bins of length w with patterns $P_1, \cdots, P_m$minimizing the total EEL, that is,
>
> $$eel(P_1, \cdots, P_m) = \sum_{k=1}^{m} P_k + m \qquad \textit{(Eq. 6-20)}$$

Both problems are particularly complex, so we address them heuristically starting from the trivial observation that robustness normally increases with leftover. Indeed, as a general rule and in rough terms, the robustness of a pattern *P* against point-shaped faults increase with

- the pattern leftover w_0, and
- the pattern assortment, i.e., the number of different lengths of items in P.

Let in fact $W_0 = mw - \sum_{i \in I} w_i$ be the total leftover obtained for an m-packing of I. For the first of the issues listed above, it makes sense to distribute W_0 as much evenly as possible among the patterns that form the packing. Consider in fact the following example: $I\ = \{1,2,3,4\}, w = 1, w_1 =\ \ w_2 = \frac{1}{2}\ , w_3 =$

$w_4 = \frac{1}{4}$, There are two minimum bin sets that allow to pack all the items of I: $\mathcal{A}$, consisting of $P_1 = \{1,2\}$ and $P_2 = \{3,4\}$, and $\mathcal{B}$, consisting of $P_3 = \{1,3\}$ and $P_4 = \{2,4\}$ (or equivalent combinations like $\{1,4\}$ etc.). P_1 has 0 leftover but is not robust at all as $\pi_{P_1} = 0$. P_2, P_3, P_4 are instead totally robust, as $\pi_{P_2} = \pi_{P_3} = \pi_{P_4} = 1$. Therefore

$$eel(\mathcal{A}) > 0\,, eel(\mathcal{B}) = 0 \qquad \textit{(Eq. 6-21)}$$

So, we have two solutions that are equivalent from the viewpoint of total leftover but not from that of potential value loss caused by a single random fault: value loss is minimized by a solution that most fairly distributes leftover among the bins.

Suppose that I admits a minimum BIN PACKING (in the ordinary sense) with m^* bins and let ϱ be the probability of a fault in any of those bins. For any pattern P in the solution, known the economic values $e_1, \dots, e_n$ of its items, we compute $\pi_1, \dots, \pi_n$ and $eel(P)$. Then, summing up the optimal values found multiplied by ϱ, we obtain the EEL of the whole packing. That said, we can heuristically attempt to decrease the EEL by solving, for bin sets J of increasing cardinality $m \geq m^*$, the following MULTIPROCESSOR SCHEDULING PROBLEM:

Model 6-3 (Way I)

Variables:

- $x_{ij} \in \{0,1\}$ defined for $i \in I$ and $j \in J$, with $x_{ij} = 1$, if item $i \in I$ is assigned to bin $j \in J$
- C_{max}: continuous variable

Constraints:

$$\sum_{j \in J} x_{ij} = 1 \qquad i \in I \qquad (each\ item\ is\ assigned\ to\ one\ bin) \qquad \textit{(Eq. 6-22)}$$

$$\sum_{i \in I} w_i\, x_{ij} \leq w \qquad j = 1, 2, \dots, m \qquad (bin\ capacity\ is\ not\ violated) \qquad \textit{(Eq. 6-23)}$$

$$\sum_{i \in I} w_i\, x_{ij} \leq C_{max} \qquad j = 1, 2, \dots, m \qquad \textit{(Eq. 6-24)}$$

Objective: Minimize C_{max}, which corresponds to balance bin loads as far as possible.

An alternative way to balance bin loads is to minimize the difference between the largest (C_{max}) and the smallest (C_{min}) load assigned to a bin. The formulation has the same variables above, but different constraint set and objective.

Model 6-4 (Way II)

Constraints:

$$\sum_{j \in J} x_{ij} = 1 \qquad i \in I \qquad (each\ item\ is\ assigned\ to\ one\ bin) \tag{Eq. 6-22}$$

$$\sum_{i \in I} w_i\, x_{ij} \leq w \qquad j = 1, 2, \dots, m \qquad (bin\ capacity\ is\ not\ violated) \tag{Eq. 6-23}$$

$$\sum_{i \in I} w_i\, x_{ij} \leq C_{max} \qquad j = 1, 2, \dots, m \tag{Eq. 6-24}$$

$$\sum_{i \in I} w_i\, x_{ij} \geq C_{min} \qquad j = 1,2, \dots, m \tag{Eq. 6-25}$$

Objective: Minimize ($C_{max} - C_{min}$), that is the difference between the largest and the smallest load of a bin.

6.7 Computational experience

6.7.1 The test bed

Numerical tests were carried out to evaluate the performance of Model 6-3 (Way I) and Model 6-4 (Way II) in terms of reduction of total EEL with respect to a BPP solution that just minimizes the number of bins used.

We assumed w, w_i integer for all $i \in I$, bins prone to a defect with identical probabilities p, and defect position t uniformly distributed in the bin length $[1, w]$, then Model 6-2, solved for every pattern *P* in the solution and for integers $t \in [1, w]$.

The tests were done on BIN PACKING instances taken from [37] . Algorithms were coded in Python3 programming [36] with Anaconda3 [34], Jupyter notebook [35] using GuRoBi® 9.0.2 [33] as mathematical programming solver. For each instance we were provided with the optimal m^* bins solution given by [37]. We then computed:

- a robust solution via Model 6-3 (Way I) with $m = m^*, m^* + 1, m^* + 2$ bins
- a robust solution via Model 6-4 (Way II) with $m = m^*, m^* + 1, m^* + 2$ bins
- the EG via Model 6-2, for both the robust solutions above.

Instances id [37]	$\|I\|$	**w**	m^*	C_{max} **Model 6-3** **(Way I)**	**CPU time Model 6-3 (Sec.)**	**EG Model 6-2**	**CPU time Model 6-2 (Sec.)**
N1C1W1_A.BPP	50	100	25	95	0.35	1082.14	53.51
N1C1W1_B.BPP	50	100	31	83	0.71	1251.66	69.46
N1C1W1_C.BPP	50	100	20	98	0.84	1057.42	46.66
N1C1W1_D.BPP	50	100	28	86	0.56	1153.06	69.7
N1C1W1_E.BPP	50	100	26	88	0..36	1154.5	51.64
N1C1W1_F.BPP	50	100	27	90	0.64	1181.22	58.13
N1C1W1_G.BPP	50	100	25	95	0.47	1060.38	56.47
N1C1W1_H.BPP	50	100	31	83	0.65	1254.1	71.86
N1C1W1_I.BPP	50	100	25	89	0.41	1252.92	54.57
N1C1W1_J.BPP	50	100	26	92	4.53	1115.94	55.71
N1C1W1_K.BPP	50	100	26	89	0.46	1217.04	48.07
N1C1W1_L.BPP	50	100	33	71	0.57	1349.36	72.05
N1C1W1_M.BPP	50	100	30	86	0.69	1151.62	82.26
N1C1W1_N.BPP	50	100	25	91	0.4	1025.34	56.69
N1C1W1_O.BPP	50	100	32	81	0.52	1196.82	67.9
N1C1W1_P.BPP	50	100	26	89	0.54	1329.4	54.23
N1C1W1_Q.BPP	50	100	28	83	0.39	1361.2	52.72
N1C1W1_R.BPP	50	100	25	92	1.78	1222.62	52.03
N1C1W1_S.BPP	50	100	28	90	0.36	1085.34	53.27
N1C1W1_T.BPP	50	100	28	84	0.39	1248.5	52.07
N2C1W1_A.BPP	100	100	48	95	34.82	2326.04	104.72
N2C1W1_C.BPP	100	100	46	97	6.4	2288.41	96.41
N2C1W1_D.BPP	100	100	50	94	5.8	2271.12	99.31
N2C1W1_E.BPP	100	100	58	89	12.6	2372.74	203.68
N2C1W1_F.BPP	100	100	50	96	16.51	2165.66	110.86
N2C1W1_G.BPP	100	100	60	81	6.36	2596.48	129.93
N2C1W1_I.BPP	100	100	62	81	2.21	2466.66	118.16
N2C1W1_J.BPP	100	100	59	86	5.2	2411.42	110.86
N2C1W1_K.BPP	100	100	55	90	3.68	2287.78	107.55
N2C1W1_L.BPP	100	100	55	91	5.04	2229.16	119.35
N2C1W1_M.BPP	100	100	46	98	2.94	2205.5	101.02
N2C1W1_N.BPP	100	100	48	96	4.9	2269.12	99.52
N2C1W1_O.BPP	100	100	48	96	3.32	2025.06	103.44
N2C1W1_P.BPP	100	100	54	92	7.68	2262.16	119.01
N2C1W1_Q.BPP	100	100	46	98	4.33	2174.56	100.81
N2C1W1_R.BPP	100	100	56	92	3.4	2265.86	114.61
N2C1W1_S.BPP	100	100	45	97	9.23	2193.54	90.32

N2C1W1_T.BPP	100	100	52	94	2.81	2233.7	101.33
N1W4B1R0.BPP	50	1000	6	913	1.15	5474.032	180.34
N1W4B1R1.BPP	50	1000	6	932	1.19	5566.99	176.4
N1W4B1R2.BPP	50	1000	6	919	7.78	5499.83	178.39
N1W4B1R3.BPP	50	1000	6	919	0.23	5495.05	179.24
N1W4B1R4.BPP	50	1000	6	925	12.34	5526.07	182.75
N1W4B1R5.BPP	50	1000	6	908	0.844	5447.77	177.53
N1W4B1R6.BPP	50	1000	6	914	13.8	5475.73	176.16
N1W4B1R7.BPP	50	1000	6	926	2.61	5526.63	183.92
N1W4B1R8.BPP	50	1000	6	896	89.61	5376	191.96
N1W4B1R9.BPP	50	1000	6	902	258.53	5412.39	194.74
N1W4B2R0.BPP	50	1000	6	951	0.8	5691.8	196.05
N1W4B2R1.BPP	50	1000	6	842	1.35	5054	186.63
N1W4B2R2.BPP	50	1000	6	890	0.31	5341	182.6
N1W4B2R3.BPP	50	1000	6	951	0.61	5670.11	194.17
N1W4B2R4.BPP	50	1000	6	935	1.07	5605.14	191.68
N1W4B2R5.BPP	50	1000	6	984	0.45	5834.1	188.04
N1W4B2R6.BPP	50	1000	6	927	0.86	5561.43	183.96
N1W4B2R7.BPP	50	1000	6	864	0.87	5188	179.9
N1W4B2R8.BPP	50	1000	6	996	0.24	5781.04	202.07
N1W4B2R9.BPP	50	1000	6	880	1.16	5284	188.68
N1W4B3R0.BPP	50	1000	6	865	0.48	5191	204.94
N1W4B3R1.BPP	50	1000	6	922	0.61	5535.63	213.8
N1W4B3R2.BPP	50	1000	7	901	0.69	6283.48	218.76
N1W4B3R3.BPP	50	1000	6	899	0.54	5396	208.09
N1W4B3R4.BPP	50	1000	6	920	0.35	5460.99	208.69
N1W4B3R5.BPP	50	1000	7	865	0.69	6059	234.71
N1W4B3R6.BPP	50	1000	6	921	0.39	5530	210.67
N1W4B3R7.BPP	50	1000	6	881	0.34	5286	192.99
N1W4B3R8.BPP	50	1000	6	967	0.83	5766.14	196.08
N1W4B3R9.BPP	50	1000	6	969	1.63	5777.77	208.48
N2W4B1R0.BPP	100	1000	12	939	690.35	11172.85	391.23
N2W4B1R1.BPP	100	1000	12	924	3.5	11046.89	401.02
N2W4B1R2.BPP	100	1000	12	941	176.39	11094.71	373.99
N2W4B1R3.BPP	100	1000	12	935	1020.65	11122.08	443.21
N2W4B1R4.BPP	100	1000	12	931	40.86	11102.89	406.43
N2W4B1R5.BPP	100	1000	12	932	43	11110.6	418.31
N2W4B1R6.BPP	100	1000	11	998	3.33	10340.6	490.9
N2W4B1R7.BPP	100	1000	12	924	22.34	11044.85	426.08
N2W4B1R8.BPP	100	1000	11	994	63.16	10408.85	403.2
N2W4B1R9.BPP	100	1000	11	994	31.94	10380.96	399.99

N2W4B2R0.BPP	100	1000	11	968	10.74	10599.01	392.88
N2W4B2R1.BPP	100	1000	11	969	10.06	10541.42	407.97
N2W4B2R2.BPP	100	1000	11	994	7.77	10679.57	417.67
N2W4B2R3.BPP	100	1000	11	947	8.33	10385.41	391.89
N2W4B2R4.BPP	100	1000	12	941	16.46	11258.3	434.76
N2W4B2R5.BPP	100	1000	12	938	53.07	11244.32	425.52
N2W4B2R6.BPP	100	1000	12	929	23.26	11123.38	447.22
N2W4B2R7.BPP	100	1000	11	986	12.87	10691.92	429.57
N2W4B2R8.BPP	100	1000	12	978	103.89	11442.79	442.92
N2W4B2R9.BPP	100	1000	11	984	2.74	10728.69	385.63
N2W4B3R0.BPP	100	1000	12	919	13.42	11029	400.42
N2W4B3R1.BPP	100	1000	12	927	3.45	11118.78	417.97
N2W4B3R2.BPP	100	1000	12	986	11.27	11702.48	450.89
N2W4B3R3.BPP	100	1000	11	999	0.97	10682.32	500.99
N2W4B3R4.BPP	100	1000	12	929	12.2	11150.94	422.4
N2W4B3R5.BPP	100	1000	10	997	0.31	9846.5	443.71
N2W4B3R6.BPP	100	1000	11	965	13.16	10599.06	411.49
N2W4B3R7.BPP	100	1000	11	977	4.24	10719.34	417.13
N2W4B3R8.BPP	100	1000	11	989	4.82	10737.11	415.85
N2W4B3R9.BPP	100	1000	12	980	19	11603.61	457.21

Tab. 6-1) Result of Model 6-3 (Way I) with m^* bins.

Instances id [37]	C_{max} **Model 6-3 (Way I)**	**CPU time Model 6-3 (Sec.)**	**EG Model 6-2**	**CPU time Model 6-2 (Sec.)**
N1C1W1_A.BPP	91	1.54	1177.7	65.1
N1C1W1_B.BPP	76	1.6	1294.16	74.31
N1C1W1_C.BPP	93	6.57	1159.16	52.96
N1C1W1_D.BPP	80	1.83	1196.84	76.68
N1C1W1_E.BPP	86	3.18	1219.3	61.32
N1C1W1_F.BPP	86	3.08	1250.06	62.98
N1C1W1_G.BPP	90	1.63	1136.72	59.51
N1C1W1_H.BPP	81	1.75	1320.04	76.76
N1C1W1_I.BPP	86	1.33	1312.64	57.89
N1C1W1_J.BPP	89	1.52	1197.3	61.63
N1C1W1_K.BPP	85	1.16	1281.5	57.95
N1C1W1_L.BPP	68	1.52	1404.82	75.94
N1C1W1_M.BPP	80	2.12	1203.68	77.72
N1C1W1_N.BPP	91	1.64	1118.88	57.64
N1C1W1_O.BPP	79	1.54	1269.82	73.49

N1C1W1_P.BPP	82	1.29	1393.32	59.66
N1C1W1_Q.BPP	81	1.27	1422.84	64.57
N1C1W1_R.BPP	88	2.95	1294.98	60.19
N1C1W1_S.BPP	87	1.4	1158.58	65.79
N1C1W1_T.BPP	82	1.29	1310.94	62.17
N1W4B1R0.BPP	782	3	5479	199.67
N1W4B1R1.BPP	799	2.3	5596	201.99
N1W4B1R2.BPP	788	12.89	5516	195.13
N1W4B1R3.BPP	788	37.35	5517	199.49
N1W4B1R4.BPP	793	19.7	5551	207.63
N1W4B1R5.BPP	778	1.37	5452	198.22
N1W4B1R6.BPP	783	1.31	5484	198.67
N1W4B1R7.BPP	794	5.21	5559	210.41
N1W4B1R8.BPP	768	1.22	5376	208.82
N1W4B1R9.BPP	773	46.29	5413	228.58
N1W4B2R0.BPP	815	2.23	5711	216.59
N1W4B2R1.BPP	722	1.14	5054	210.87
N1W4B2R2.BPP	763	1.5	5341	212.39
N1W4B2R3.BPP	815	1.14	5709	220.16
N1W4B2R4.BPP	801	2.16	5612	219.41
N1W4B2R5.BPP	843	1.01	5906	205.29
N1W4B2R6.BPP	794	1.47	5564	201.75
N1W4B2R7.BPP	741	2.16	5188	196.25
N1W4B2R8.BPP	854	2.78	5980	215.94
N1W4B2R9.BPP	754	1.93	5284	211.17
N1W4B3R0.BPP	741	0.63	5191	217.8
N1W4B3R1.BPP	790	0.7	5536	229.8
N1W4B3R2.BPP	789	3.88	6313	249.51
N1W4B3R3.BPP	770	1.53	5396	227.34
N1W4B3R4.BPP	788	0.85	5520	228.56
N1W4B3R5.BPP	757	1.69	6059	265.04
N1W4B3R6.BPP	790	1.42	5530	239.09
N1W4B3R7.BPP	755	1.14	5286	210.94
N1W4B3R8.BPP	829	0.97	5805	238.54
N1W4B3R9.BPP	831	3.93	5818	238.76

Tab. 6-2) Result of Model 6-3 (Way I) with (m^*+1) bins.

Instances id [37]	C_{max} **Model 6-3 (Way I)**	**CPU time Model 6-3 (Sec.)**	**EG Model 6-2**	**CPU time Model 6-2 (Sec.)**
N1C1W1_A.BPP	87	1.88	1220.74	65.41
N1C1W1_B.BPP	74	1.63	1342.42	80.42
N1C1W1_C.BPP	89	40.59	1234.56	56.04
N1C1W1_D.BPP	76	2	1228.54	87.33
N1C1W1_E.BPP	85	1.75	1291	66.58
N1C1W1_F.BPP	82	1.66	1309.2	66.04
N1C1W1_G.BPP	86	6.62	1196.96	67.6
N1C1W1_H.BPP	76	1.76	1370.58	76.65
N1C1W1_I.BPP	82	1.34	1377.1	62.73
N1C1W1_J.BPP	82	1.92	1251.22	64.86
N1C1W1_K.BPP	82	2.26	1341.12	59.11
N1C1W1_L.BPP	64	1.6	1456.18	80.91
N1C1W1_M.BPP	76	1.81	1258.92	104.94
N1C1W1_N.BPP	86	2.07	1196.72	61.58
N1C1W1_O.BPP	72	1.72	1312.08	74.9
N1C1W1_P.BPP	80	1.3	1434.34	66.34
N1C1W1_Q.BPP	76	1.57	1467.22	69.4
N1C1W1_R.BPP	83	1.49	1372.96	66.41
N1C1W1_S.BPP	84	1.43	1207.28	64.57
N1C1W1_T.BPP	79	1.18	1388.48	68.31
N1W4B1R0.BPP	684	1.35	5479	217.42
N1W4B1R1.BPP	699	2.71	5596	211.2
N1W4B1R2.BPP	689	13.36	5516	207.7
N1W4B1R3.BPP	689	145.14	5517	212.93
N1W4B1R4.BPP	693	2.14	5551	218.81
N1W4B1R5.BPP	681	83.5	5452	207.23
N1W4B1R6.BPP	685	2.37	5484	211.52
N1W4B1R7.BPP	694	265.32	5559	233.27
N1W4B1R8.BPP	672	32.64	5376	223.57
N1W4B1R9.BPP	676	9.17	5413	233.74
N1W4B2R0.BPP	713	2.71	5711	232.71
N1W4B2R1.BPP	631	4.53	5054	232.08
N1W4B2R2.BPP	667	2.08	5341	231.18
N1W4B2R3.BPP	713	3.78	5709	237.16
N1W4B2R4.BPP	701	4.78	5612	235.34
N1W4B2R5.BPP	738	6.74	5906	226.44
N1W4B2R6.BPP	695	2.8	5564	215.98

N1W4B2R7.BPP	648	2.78	5188	215.36
N1W4B2R8.BPP	747	3.89	5980	231.12
N1W4B2R9.BPP	660	3.7	5284	235.45
N1W4B3R0.BPP	648	1.2	5191	236.05
N1W4B3R1.BPP	692	3.81	5536	254.27
N1W4B3R2.BPP	701	128.21	6313	272.25
N1W4B3R3.BPP	674	2.81	5396	244.87
N1W4B3R4.BPP	690	4.11	5520	220.58
N1W4B3R5.BPP	673	3.22	6059	293.68
N1W4B3R6.BPP	691	3.22	5530	255.91
N1W4B3R7.BPP	660	0.35	5286	228.03
N1W4B3R8.BPP	725	1.32	5805	266.74
N1W4B3R9.BPP	727	3.47	5818	261.23

Tab. 6-3) Result of Model 6-3 (Way I) with (m^*+2) bins.

Instances id [37]	$\|I\|$	**w**	m^*	$[C_{max}, C_{min}]$ **Model 6-4 (Way II)**	**CPU time Model 6-4 (Sec.)**	**EG Model 6-2**	**CPU time Model 6-2 (Sec.)**
N1C1W1_A.BPP	50	100	25	[99.0, 95.0]	0.37	1085.56	46.87
N1C1W1_B.BPP	50	100	31	[100.0, 83.0]	0.83	1249.66	68.62
N1C1W1_C.BPP	50	100	20	[100.0, 98.0]	0.79	1055.42	38.12
N1C1W1_D.BPP	50	100	28	[100.0, 86.0]	0.61	1152.86	49.4
N1C1W1_E.BPP	50	100	26	[100.0, 88.0]	0.52	1154.18	48.66
N1C1W1_F.BPP	50	100	27	[99.0, 90.0]	0.54	1182.56	46.43
N1C1W1_G.BPP	50	100	25	[100.0, 95.0]	0.84	1061.16	46.8
N1C1W1_H.BPP	50	100	31	[100.0, 83.0]	0.66	1255.18	56.17
N1C1W1_I.BPP	50	100	25	[96.0, 89.0]	0.86	1246.16	58.59
N1C1W1_J.BPP	50	100	26	[99.0, 92.0]	0.88	1119.72	58.23
N1C1W1_K.BPP	50	100	26	[96.0, 89.0]	0.55	1225.98	48.42
N1C1W1_L.BPP	50	100	33	[99.0, 71.0]	0.86	1334.7	60.42
N1C1W1_M.BPP	50	100	30	[100.0, 86.0]	0.47	1150.62	54.71
N1C1W1_N.BPP	50	100	25	[100.0, 91.0]	0.42	1002.58	44.01
N1C1W1_O.BPP	50	100	32	[100.0, 81.0]	0.77	1197.13	57.11
N1C1W1_P.BPP	50	100	26	[96.0, 89.0]	0.44	1329.72	46.25
N1C1W1_Q.BPP	50	100	28	[97.0, 83.0]	0.52	1362.86	51.21
N1C1W1_R.BPP	50	100	25	[98.0, 92.0]	2.43	1228.36	47.89
N1C1W1_S.BPP	50	100	28	[100.0, 90.0]	0.45	1085.66	51.76
N1C1W1_T.BPP	50	100	28	[100.0, 84.0]	0.43	1249.24	64.19
N2C1W1_C.BPP	100	100	46	[99.0, 97.0]	9.51	2278.58	88.98
N2C1W1_D.BPP	100	100	50	[99.0, 94.0]	2.28	2267.38	95.46

N2C1W1_E.BPP	100	100	58	[100.0, 89.0]	4.83	2374.62	110.7
N2C1W1_F.BPP	100	100	50	[100.0, 96.0]	12.67	2166.66	101.12
N2C1W1_G.BPP	100	100	60	[99.0, 81.0]	4.94	2597.72	112.53
N2C1W1_H.BPP	100	100	52	[99.0, 93.0]	6.21	2342.34	102.02
N2C1W1_I.BPP	100	100	62	[100.0, 81.0]	19.61	2467.94	120.99
N2C1W1_J.BPP	100	100	59	[100.0, 86.0]	5.88	2411.42	112.19
N2C1W1_K.BPP	100	100	55	[100.0, 90.0]	1.95	2288.18	109.14
N2C1W1_L.BPP	100	100	55	[100.0, 91.0]	5.23	2230.1	109.5
N2C1W1_M.BPP	100	100	46	[100.0, 98.0]	4.29	2215.38	90.61
N2C1W1_N.BPP	100	100	48	[100.0, 96.0]	5.48	2268.04	94.2
N2C1W1_O.BPP	100	100	48	[100.0, 96.0]	4.24	2050.76	97.53
N2C1W1_P.BPP	100	100	54	[100.0, 92.0]	1.75	2260.82	114.87
N2C1W1_Q.BPP	100	100	46	[100.0, 98.0]	4.3	2180.04	99.45
N2C1W1_R.BPP	100	100	56	[100.0, 92.0]	1.46	2266.84	113.82
N2C1W1_S.BPP	100	100	45	[100.0, 97.0]	4.83	2191.18	87.43
N2C1W1_T.BPP	100	100	52	[100.0, 94.0]	4.39	2244.32	103.25
N1W4B1R0.BPP	50	1000	6	[914.0, 913.0]	1.05	5474.4	173.22
N1W4B1R1.BPP	50	1000	6	[933.0, 932.0]	2.5	5566.21	207.11
N1W4B1R2.BPP	50	1000	6	[920.0, 919.0]	2.34	5500.32	214.18
N1W4B1R3.BPP	50	1000	6	[920.0, 919.0]	0.37	5500.82	205.8
N1W4B1R4.BPP	50	1000	6	[926.0, 925.0]	1.14	5523.81	198.39
N1W4B1R5.BPP	50	1000	6	[909.0, 908.0]	0.82	5449.09	213.59
N1W4B1R6.BPP	50	1000	6	[914.0, 914.0]	1.67	5475.63	207.87
N1W4B1R7.BPP	50	1000	6	[927.0, 926.0]	0.35	5537.65	206.27
N1W4B1R8.BPP	50	1000	6	[896.0, 896.0]	1.89	5376	211.29
N1W4B1R9.BPP	50	1000	6	[903.0, 902.0]	0.86	5412.8	204.62
N1W4B2R0.BPP	50	1000	6	[952.0, 951.0]	1.44	5619.11	198.18
N1W4B2R1.BPP	50	1000	6	[843.0, 842.0]	1.02	5054	200.49
N1W4B2R2.BPP	50	1000	6	[891.0, 890.0]	0.57	5341	193.4
N1W4B2R3.BPP	50	1000	6	[952.0, 951.0]	0.75	5690.19	223.06
N1W4B2R4.BPP	50	1000	6	[936.0, 935.0]	0.65	5610.85	200.26
N1W4B2R5.BPP	50	1000	6	[985.0, 984.0]	1.42	5811.24	222.14
N1W4B2R6.BPP	50	1000	6	[928.0, 927.0]	2.36	5552.33	218.91
N1W4B2R7.BPP	50	1000	6	[865.0, 864.0]	1.83	5188	211.35
N1W4B2R8.BPP	50	1000	6	[997.0, 996.0]	0.73	5735.42	226.43
N1W4B2R9.BPP	50	1000	6	[881.0, 880.0]	0.85	5284	210.47
N1W4B3R0.BPP	50	1000	6	[866.0, 865.0]	0.3	5191	171.93
N1W4B3R1.BPP	50	1000	6	[923.0, 922.0]	0.64	5535.17	202.8
N1W4B3R2.BPP	50	1000	7	[902.0, 901.0]	0.73	6298.69	207
N1W4B3R3.BPP	50	1000	6	[900.0, 899.0]	0.74	5396	210.49
N1W4B3R4.BPP	50	1000	6	[920.0, 920.0]	2.2	5520	206.11

N1W4B3R5.BPP	50	1000	7	[866.0, 865.0]	1.32	6059	214.38
N1W4B3R6.BPP	50	1000	6	[922.0, 921.0]	1.9	5530	200.89
N1W4B3R7.BPP	50	1000	6	[881.0, 881.0]	3.06	5286	209.38
N1W4B3R8.BPP	50	1000	6	[968.0, 967.0]	0.32	5773.55	183.37
N1W4B3R9.BPP	50	1000	6	[970.0, 969.0]	1.01	5766.31	181.26
N2W4B1R0.BPP	100	1000	12	[940.0, 939.0]	26.41	11163.55	387.28
N2W4B1R3.BPP	100	1000	12	[936.0, 935.0]	118.28	11131	400.68
N2W4B1R4.BPP	100	1000	12	[932.0, 931.0]	49.84	11116.28	403.13
N2W4B1R5.BPP	100	1000	12	[933.0, 932.0]	133.11	11100.87	404.07
N2W4B1R7.BPP	100	1000	12	[925.0, 924.0]	51.47	11047	379.14
N2W4B1R8.BPP	100	1000	11	[995.0, 994.0]	17.34	10354.87	367.05
N2W4B1R9.BPP	100	1000	11	[995.0, 994.0]	67.34	10447.26	373.36
N2W4B2R0.BPP	100	1000	11	[969.0, 968.0]	11.99	10604.39	397.09
N2W4B2R1.BPP	100	1000	11	[970.0, 969.0]	13.08	10552.53	379.91
N2W4B2R2.BPP	100	1000	11	[994.0, 994.0]	105.2	10719.61	408.33
N2W4B2R3.BPP	100	1000	11	[948.0, 947.0]	12.21	10398.87	364.83
N2W4B2R4.BPP	100	1000	12	[942.0, 941.0]	24.15	11277.49	409.4
N2W4B2R5.BPP	100	1000	12	[939.0, 938.0]	16.12	11249.52	403.24
N2W4B2R6.BPP	100	1000	12	[930.0, 929.0]	54.12	11138.04	412.15
N2W4B2R7.BPP	100	1000	11	[987.0, 986.0]	20.11	10700.34	417.68
N2W4B2R8.BPP	100	1000	12	[979.0, 978.0]	17.42	11589.82	433.82
N2W4B2R9.BPP	100	1000	11	[985.0, 984.0]	3.23	10675.1	409.24
N2W4B3R0.BPP	100	1000	12	[920.0, 919.0]	8.06	10991.82	432.53
N2W4B3R1.BPP	100	1000	12	[928.0, 927.0]	10.5	11128.96	434.63
N2W4B3R2.BPP	100	1000	12	[987.0, 986.0]	15.81	11673.39	452.05
N2W4B3R3.BPP	100	1000	11	[1000.0, 999.0]	3.54	10697.39	479.18
N2W4B3R4.BPP	100	1000	12	[930.0, 929.0]	7.54	11152.01	423.21
N2W4B3R5.BPP	100	1000	10	[998.0, 997.0]	1.16	9846.44	421.37
N2W4B3R6.BPP	100	1000	11	[966.0, 965.0]	10.38	10556.55	389.93
N2W4B3R7.BPP	100	1000	11	[978.0, 977.0]	10.05	10713.23	420.07

Tab. 6-4) Result of Model 6-4 (Way II) with m^* bins.

Instances id [37]	$[C_{max}, C_{min}]$ **Model 6-4 (Way II)**	**CPU time Model 6-4 (Sec.)**	**EG Model 6-2**	**CPU time Model 6-2 (Sec.)**
N1C1W1_A.BPP	[99.0, 91.0]	1.18	1172.4	54.08
N1C1W1_B.BPP	[100.0, 76.0]	1.58	1300.7	73.6
N1C1W1_C.BPP	[95.0, 93.0]	6.18	1084.12	45.43
N1C1W1_D.BPP	[100.0, 80.0]	1.56	1202.2	59.13
N1C1W1_E.BPP	[97.0, 86.0]	16.67	1229.36	52.57

N1C1W1_F.BPP	[99.0, 86.0]	1.06	1262.26	54.34
N1C1W1_G.BPP	[100.0, 90.0]	4.93	1127.3	58.75
N1C1W1_H.BPP	[97.0, 81.0]	1.33	1322.58	64.84
N1C1W1_I.BPP	[95.0, 86.0]	1.38	1320.88	67.79
N1C1W1_J.BPP	[99.0, 89.0]	1.5	1205.86	63.35
N1C1W1_K.BPP	[96.0, 85.0]	0.99	1287.74	53.81
N1C1W1_L.BPP	[99.0, 68.0]	1.3	1405.96	69.68
N1C1W1_M.BPP	[100.0, 80.0]	1.09	1206.56	60.96
N1C1W1_N.BPP	[100.0, 91.0]	0.91	1124.54	52.89
N1C1W1_O.BPP	[100.0, 79.0]	1.37	1267.46	62.76
N1C1W1_P.BPP	[96.0, 82.0]	1.13	1374.06	56.05
N1C1W1_Q.BPP	[97.0, 81.0]	1.16	1424.66	60.09
N1C1W1_R.BPP	[95.0, 88.0]	1.14	1303.74	55.09
N1C1W1_S.BPP	[100.0, 87.0]	1.24	1159.42	60.77
N1C1W1_T.BPP	[100.0, 82.0]	1.29	1309.04	72.89
N1W4B1R0.BPP	[783.0, 782.0]	1.82	5479	189.5
N1W4B1R1.BPP	[800.0, 799.0]	2.93	5596	227.17
N1W4B1R2.BPP	[788.0, 788.0]	4.47	5516	231.06
N1W4B1R3.BPP	[789.0, 788.0]	7.39	5517	238.52
N1W4B1R4.BPP	[793.0, 793.0]	5.8	5551	234.65
N1W4B1R5.BPP	[779.0, 778.0]	2.49	5452	237.61
N1W4B1R6.BPP	[784.0, 783.0]	2.18	5484	235.82
N1W4B1R7.BPP	[795.0, 794.0]	2.53	5559	234.66
N1W4B1R8.BPP	[768.0, 768.0]	5.79	5376	235.78
N1W4B1R9.BPP	[774.0, 773.0]	2.02	5413	225.85
N1W4B2R0.BPP	[816.0, 815.0]	2.73	5711	222.81
N1W4B2R1.BPP	[722.0, 722.0]	108.4	5054	201.77
N1W4B2R2.BPP	[763.0, 763.0]	3.64	5341	204.34
N1W4B2R3.BPP	[816.0, 815.0]	2.02	5709	238.16
N1W4B2R4.BPP	[802.0, 801.0]	2.59	5612	237.93
N1W4B2R5.BPP	[844.0, 843.0]	3.87	5906	251.62
N1W4B2R6.BPP	[795.0, 794.0]	4.38	5564	236.13
N1W4B2R7.BPP	[742.0, 741.0]	3.81	5188	217.31
N1W4B2R8.BPP	[855.0, 854.0]	2.5	5980	238.39
N1W4B2R9.BPP	[755.0, 754.0]	3.29	5284	231.69
N1W4B3R0.BPP	[742.0, 741.0]	1.67	5191	186.94
N1W4B3R1.BPP	[791.0, 790.0]	1.16	5536	239.28
N1W4B3R2.BPP	[790.0, 789.0]	2.44	6313	259.51
N1W4B3R3.BPP	[771.0, 770.0]	1.89	5396	232.22
N1W4B3R4.BPP	[789.0, 788.0]	3.33	5520	226.4
N1W4B3R5.BPP	[758.0, 757.0]	2.93	6059	244.72

N1W4B3R6.BPP	[790.0, 790.0]	3	5530	224.89
N1W4B3R7.BPP	[756.0, 755.0]	1.01	5286	246.35
N1W4B3R8.BPP	[830.0, 829.0]	3.88	5805	208.64
N1W4B3R9.BPP	[832.0, 831.0]	1.97	5818	202.64

Tab. 6-5) Result of Model 6-4 (Way II) with (m^*+1) bins.

Instances id [37]	**[C_{max}, C_{min}] Model 6-4 (Way II)**	**CPU time Model 6-4 (Sec.)**	**EG Model 6-2**	**CPU time Model 6-2 (Sec.)**
N1C1W1_A.BPP	[99.0, 87.0]	1.53	1238.22	57.82
N1C1W1_B.BPP	[100.0, 74.0]	1.47	1352.96	76.69
N1C1W1_C.BPP	[92.0, 89.0]	1.54	1229.46	49.46
N1C1W1_D.BPP	[100.0, 76.0]	1.47	1230.92	61.81
N1C1W1_E.BPP	[91.0, 85.0]	1.33	1287.7	54.61
N1C1W1_F.BPP	[99.0, 82.0]	1.32	1312.62	59.49
N1C1W1_G.BPP	[100.0, 86.0]	6.93	1188.86	63.31
N1C1W1_H.BPP	[97.0, 76.0]	1.07	1371.96	68.94
N1C1W1_I.BPP	[95.0, 82.0]	2.29	1377.6	55.46
N1C1W1_J.BPP	[99.0, 82.0]	2.75	1256.82	66.52
N1C1W1_K.BPP	[96.0, 82.0]	1.53	1343.24	59.2
N1C1W1_L.BPP	[99.0, 64.0]	1.49	1437.14	73.51
N1C1W1_M.BPP	[100.0, 76.0]	2.03	1253.38	63.45
N1C1W1_N.BPP	[99.0, 86.0]	1.23	1189.3	54.86
N1C1W1_O.BPP	[100.0, 72.0]	1.31	1325.08	65.62
N1C1W1_P.BPP	[96.0, 80.0]	1.16	1434.4	59.06
N1C1W1_Q.BPP	[97.0, 76.0]	1.18	1483.8	62.43
N1C1W1_R.BPP	[95.0, 83.0]	1.12	1376.22	61.31
N1C1W1_S.BPP	[100.0, 84.0]	1.26	1226.26	64.61
N1C1W1_T.BPP	[100.0, 79.0]	1.26	1388.6	75.68
N1W4B1R0.BPP	[685.0, 684.0]	3.12	5479	203.28
N1W4B1R1.BPP	[700.0, 699.0]	2.21	5596	256.82
N1W4B1R2.BPP	[690.0, 689.0]	4.6	5516	249.25
N1W4B1R3.BPP	[690.0, 689.0]	3.19	5517	257.01
N1W4B1R4.BPP	[694.0, 693.0]	2.71	5551	253.74
N1W4B1R5.BPP	[682.0, 681.0]	8.72	5452	256.15
N1W4B1R6.BPP	[686.0, 685.0]	5.68	5484	259.41
N1W4B1R7.BPP	[695.0, 694.0]	4.54	5559	260.86
N1W4B1R8.BPP	[672.0, 672.0]	25.31	5376	252.5
N1W4B1R9.BPP	[677.0, 676.0]	3.73	5413	245

N1W4B2R0.BPP	[714.0, 713.0]	4.17	5711	241.68
N1W4B2R1.BPP	[632.0, 631.0]	3.26	5054	241.31
N1W4B2R2.BPP	[668.0, 667.0]	2.28	5341	246.38
N1W4B2R3.BPP	[714.0, 713.0]	2.18	5709	273.33
N1W4B2R4.BPP	[702.0, 701.0]	2	5612	259.65
N1W4B2R5.BPP	[739.0, 738.0]	2.33	5906	270.5
N1W4B2R6.BPP	[696.0, 695.0]	3.35	5564	253.3
N1W4B2R7.BPP	[649.0, 648.0]	3.8	5188	244.98
N1W4B2R8.BPP	[748.0, 747.0]	3	5980	257.37
N1W4B2R9.BPP	[661.0, 660.0]	3.49	5284	258.73
N1W4B3R0.BPP	[649.0, 648.0]	1.02	5191	219.31
N1W4B3R1.BPP	[692.0, 692.0]	12.94	5536	254.91
N1W4B3R2.BPP	[702.0, 701.0]	2.68	6313	284.66
N1W4B3R3.BPP	[675.0, 674.0]	2.14	5396	256.45
N1W4B3R4.BPP	[690.0, 690.0]	203.16	5520	243.06
N1W4B3R5.BPP	[674.0, 673.0]	2.06	6059	240.2
N1W4B3R6.BPP	[692.0, 691.0]	2.62	5530	252.32
N1W4B3R7.BPP	[661.0, 660.0]	1.37	5286	264.81
N1W4B3R8.BPP	[726.0, 725.0]	0.27	5805	222.82
N1W4B3R9.BPP	[728.0, 727.0]	1.67	5818	219.62

Tab. 6-6) Result of Model 6-4 (Way II) with (m^*+2) bins.

6.7.2 Analysis Results

After running our test for the 100 instances from [37] and for each number of bins $m \in [m^*, m^* + 5]$, we compared Model 6-3 and Model 6-4 with the EG obtained. The following table resumes the result giving the percentage of cases in which each model improved the EG of the initial solution provided by [37].

	m^*	$m^* + 1$	$m^* + 2$	$m^* + 3$	$m^* + 4$	$m^* + 5$
% Better EG via Model 6-3 (Way I)	32.98%	12%	12%	13.95%	30.55%	27.78%
% Better EG via Model 6-4 (Way II)	55.32%	28%	28%	30.23%	16.67%	16.67%
% Same EG in both models	11.7%	60%	60%	55.81%	52.78%	55.55%

Tab. 6-7) Comparison between Model 6-3 (Way I) and Model 6-4 (Way II) with EG.

To evaluate the Expected Profit (EP) in Model 6-2, we considered the effect of changing defect probability (p) and Profit Factor (PF) on the value of EP. The following figure shows the result obtained with instance N1C1W1_A from [37].

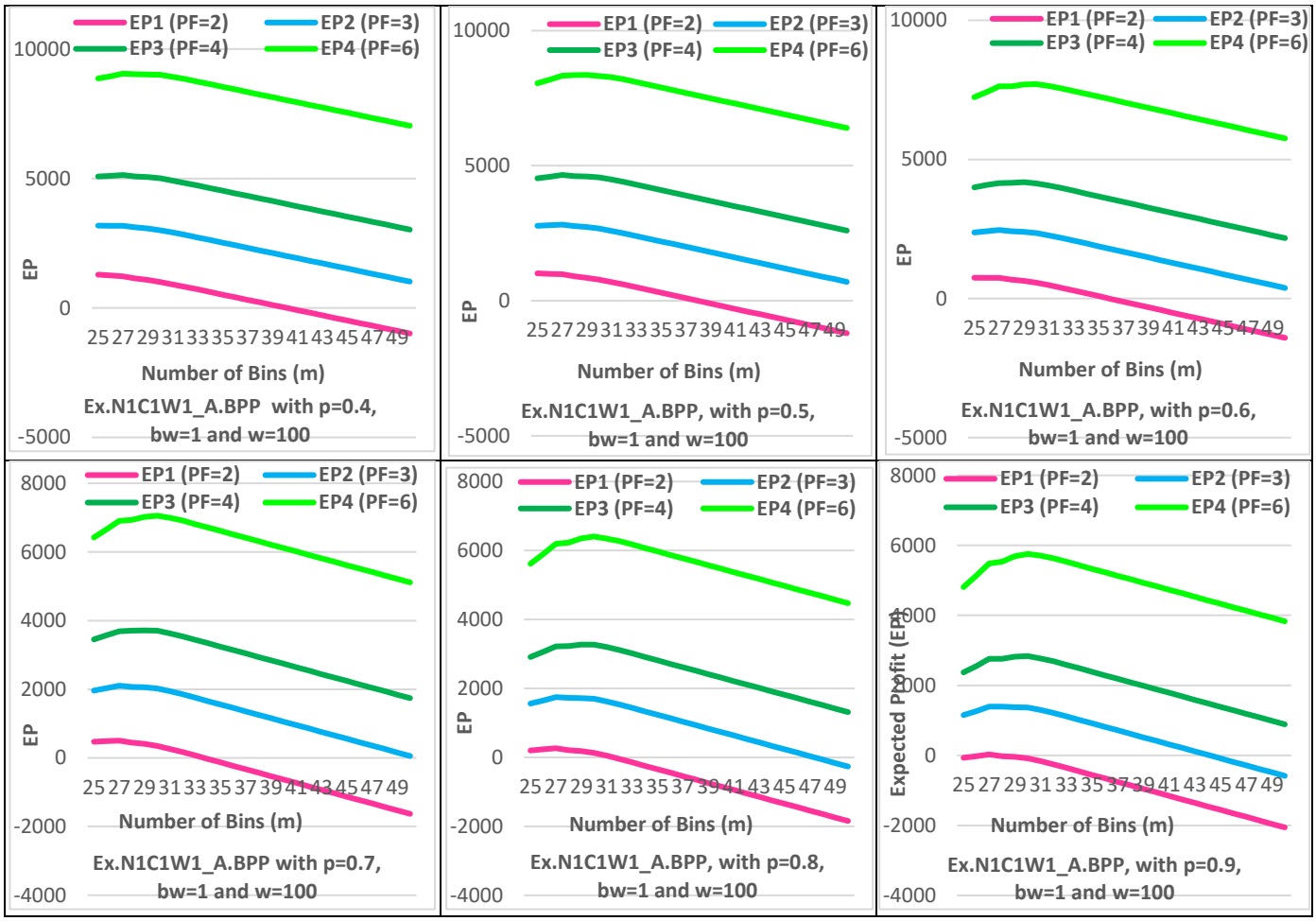


Chart. 6-1) The effect of p and PF on EP for instance N1C1W1_A.BPP from [37].

For this instance, the defect probability (p) was considered in the range of [0.4,0.9], with a constant amount of unit cost of raw material (bw), we drew the chart of expected profit (EP) (see (Eq. 6-19)) based on the number of bins (m), for different amount of profit factors (PF). The charts with dark pink color for $PF = 2$, with light blue color for $PF = 3$, with green color for $PF = 4$ and with light green color are for $PF = 6$.

In this case, $m^* = 25$, was the optimal number of bins but one can see, with a little bit increasing the number of bins and with different amount of profit factor, also with high probability of defect, expected profit (EP) is increased. For example, in the last chart for p=0.9, with $PF = 2$ (dark pink), the pick of chart is with $m^* = 27$, for $PF = 3$ (light blue) $m^* = 28$ is the maximum point and for $PF = 4\ and\ 6$

(green and light green respectively) in $m^* = 30$, we have maximum EP for this instance. So, with a good compromise among parameters on expected profit can ensure improved results.

6.8 Conclusions and future research

The topic of developing robust cutting patterns and cutting plans was addressed, with the robustness of a pattern defined as the probability that all its elements can be cut to avoid uniformly distributed point-shaped faults. To find a robust plan that minimizes the impact of point-shaped faults occurring in bins, we devised two efficient heuristic approaches based on the solution of a MULTIPROCESSOR SCHEDULING PROBLEM via integer linear programming. The range of applicability of the proposed approaches was tested via an extensive numerical test using literature instances of BIN PACKING.

Various research branches can be generated from our study. First, we assumed that multiple faults in a bin is a rare event, therefore a natural extension would be to relax this hypothesis. Second, our model assumes point-shaped failures, which may not be the case in industrial applications: our results are easily extended to faults of known size ϵ, however considering faulty size as a random variable introduces a new research issue. Third, we only looked at one-dimensional cutting operations, so leaving important industrial applications out of our scope. Ultimately, the complexity of the problems under consideration necessitates further development, such as approximation algorithms or exact problem formulations.

7 Bibliography

[1] M. Hazewinkel, Encyclopaedia of Mathematics, Dordrecht , Netherlands: Springer Science+Business Media, 1995.

[2] D. Bertsimas and J. N. Tsitsiklis, "Introduction to Linear Optimization", Belmont, Massachusetts: Athena Scientific, 1997.

[3] A. Sultan, "Linear programming : an introduction with applications", New York: Academic Press, INC., 1993.

[4] E. Balas, "Integer Programming and Combinatorial Optimization", Copenhagen, Denmark: Springer, 1995.

[5] H. A. Taha, "Integer Programming: Theory, Applications, and Computations", Academic Press, 1975.

[6] J. P. Vielma, "Mixed Integer Linear Programming Formulation Techniques", SIAM Review, 2015.

[7] "Mixed-Integer Programming (MIP) Basics | Gurobi. URL: http://www.gurobi . com / resources / getting - started / mip - basics".

[8] H. N. Psaraftis, "A dynamic programming approach to the single-vehicle, many-to-many immediate request dial-a-ride problem," *Transportation Science,* vol. 2, no. 14, pp. 130-154, 1980.

[9] H. N. Psaraftis, "An exact algorithm for the single-vehicle many-to-many diala-ride problem with time windows," *Transportation Science,* vol. 3, no. 17, pp. 351-357, 1983.

[10] J. Desrosiers, Y. Dumas and F. Soumis, "A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows," *American Journal of Mathematical and Management Sciences,* vol. 3, no. 6, pp. 301-325, 1986.

[11] J. Odoni and H. N. Psaraftis, "A heuristic algorithm for the multivehicle advance request Dial-a-ride problem with time windows," *Transport Resarch Part B: Methodological,* vol. 3, no. 20, pp. 243-257, 1986.

[12] T. Sexton and L. D. Bodin, "Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling," *Transportation Science,* vol. 4, no. 19, pp. 378-410, 1985.

[13] T. Sexton and L. D. Bodin, "Optimizing single vehicle many-to-many operations with desired

delivery times: II. Routing," *Transportation Science,* vol. 4, no. 19, pp. 411-435, 1985.

[14] P. Toth and D. Vigo, "Heuristic algorithms for the handicapped persons transportation problem," *Transportation Science,* vol. 1, no. 31, pp. 60-71, 1997.

[15] R. Borndörfer, M. Grötschel, F. Klostermeier, C. Küttner, "Telebus Berlin: Vehicle Scheduling in a Dial-a-Ride System," *Computer-Aided Transit Scheduling,* pp. 391-422, 1997.

[16] J. F. Cordeau and G. Laporte, "The dial-a-ride problem: models and algorithms," *Annals of Operations Research,* vol. 1, no. 153, pp. 29-46, 2007.

[17] R. Bent and P. V. Hentenryck, "A Two-Stage Hybrid Algorithm for Pickup and Delivery Vehicle Routing Problems with Time Windows," *Computers and Operations Research,* vol. 4, no. 33, pp. 875-893, 2006.

[18] S. Ropke and D. Pisinger, "A general heuristic for vehicle routing problems," *Computers and Operations Research,* vol. 8, no. 34, pp. 2403-2435, 2007.

[19] J. F. Cordeau, "A Branch-and-Cut Algorithm for the Dial-a-Ride Problem," *Operations Research,* vol. 3, no. 54, pp. 573-586, 2006.

[20] S. Ropke, J. F. Cordeau and G. Laporte, "Models and Branch-and-Cut Algorithms for Pickup and Delivery Problems with Time Windows," *Networks,* vol. 4, no. 42, pp. 258-272, 2007.

[21] S. Ropke and J. F. Cordeau, "Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows," *Transportation Science,* vol. 3, no. 43, pp. 267-286, 2009.

[22] E. Bartolini, "Algorithms for Network Design and Routing Problems," 2009.

[23] D. Gale and L. S. Shapley, "College Admissions and the Stability of Marriage," *The American Mathematical Monthly,* vol. 1, no. 69, pp. 9-15, 1962.

[24] C. Arbib, O. E. Karasan and M. Ç. Pınar , "On envy-free perfect matching," *Discrete Applied Mathematics,* no. 261, pp. 21-27, 2019.

[25] F. Rossi and S. Smriglio, "A Set Packing Model for the Ground-Holding Problem in Congested Networks," *European J. of Operational Research,* vol. 2, no. 131, pp. 400-416, 2001.

[26] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly,* vol. 2, no. 2, pp. 83-97, 1955.

[27] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics,* vol. 1, no. 5, pp. 32-38, 1957.

[28] J. Edmonds and R. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the Association for Computing Machinery,* no. 19, pp. 248-264, 1972.

[29] N. Tomizawa, "On Some Techniques Useful for Solution of Transportation Network Problems," *Networks,* vol. 1, no. 2, p. 173–94, 1971.

[30] H. W. Kuhn, A. W. Tucker, G. B. Dantzig, L. R. Ford and D. R. Fulkerson, "A Primal-Dual Algorithm for Linear Programs," *Linear Inequalities and Related Systems,* no. 387, pp. 171-182, 1957.

[31] R. S. Garfinkel, "An Improved Algorithm for the Bottleneck Assignment Problem," *Operations Research,* vol. 7, no. 19, pp. 1747-1751, 1971.

[32] P. S. Pundir, S. K. Porwal and B. P. Singh, "A New Algorithm for Solving Linear Bottleneck Assignment Problem," *Journal of Institute of Science and Technology,* vol. 2, no. 15, pp. 101-102, 2015.

[33] L. Gurobi Optimization, "Gurobi Optimizer Reference Manual," *Available at "https://www.gurobi.com",* 2021.

[34] A. S. Distribution, "Anaconda Documentation, Versione 1.7.2," *Anaconda Inc. Retrieved from https://docs.anaconda.com/,* 2020.

[35] T. Kluyver, "Jupyter Notebooks – a publishing format for reproducible computational workflows," *Positioning and Power in Academic Publishing: Players, Agents and Agendas,* pp. 87-90, 2016.

[36] P. S. Foundation, "Python Language Reference, Version 3.7.6," *Available at http://www.python.org,* 2020.

[37] A. Scholl A, R. Klein and Ch. Juergensen, "A fast hybrid procedure for exactly solving the one-dimensional bin packing problem," *Computers and Operations,* no. 24, pp. 627-645, 1997.

[38] R. Aboudi and P. B. Determining, "cutting stock patterns when defects are present," *Annals of Operations Research,* no. 82, pp. 343-354, 1998.

La borsa di dottorato è stata cofinanziata con risorse del Programma Operativo Nazionale 2014-2020 (CCI 2014IT16M2OP005), Fondo Sociale Europeo, Azione I.1 “Dottorati Innovativi con caratterizzazione industriale”