UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dottorato di Ricerca in
Ingegneria dell'Innovazione Industriale

Ciclo XXXIII

Decision support system based on Operations Research for pharmaceutical logistic problems

Nilson Felipe Matos Mendes

Il Coordinatore
Prof. Franco Zambonelli

Il Tutor
Prof. Manuel Iori

A.A. 2020–2021

Contents

Acknowledgments

First of all, I would like to thank my tutor Manuel Iori for being a patient tutor during these three years of Ph.D. studies, for teaching me a lot about Operations Research and for all the support given in the academic and non-academic field. Thanks also for all the opportunities provided.

I would like to thank all the components of the group of Operational Research of the Department of Sciences and Methods for Engineering of the University of Modena and Reggio Emilia, in particular, Stefano Novellani, Arthur Kramer, Raphael Kramer and Dario Vezzali for the great moments and the interesting discussions that we had. I would like to thank in a special way Alberto Locatelli, that invited me to visit his home in Friuli once and was a good friend during the pandemic.

Many thanks also to my co-authors, who helped me in writing the reports that form part of this thesis: Dr. Maxence Delorme from the Tilburg University and Prof. Jean-François Cordeau from the HEC Montreal,

Special thanks to Renata Barbosa for all the support during this journey.

Finally, I would like to thank my family for the endless support and encouragement. I am also very grateful to the bachelor and master students that worked close to me during these three years, for the good moments we have had together: Andrea Campani, Nicolas Campana, Francesco Gallesi, Hang Dong, Giovanni Vitorassi, Beste Gökçe, Maíza Biazon, Jean Queres, Rafael Oliveira, Lucas Burahem, Javiera Brañes and Luca Gangini.

I dedicate this thesis to my mother Rita, that beat a cancer, and my father Nilson, that beat the covid-19, while I was writing my thesis and riding my bike in the Padana Plain.

Reggio Emilia, April 15, 2021

Nilson F.M. Mendes

List of Figures

List of Tables

Abstract

Healthcare services are strongly dependent on the availability of equipment and medicines, as shortages can lead to treatments interruptions, reduced capacity, or undesirable delays. In the last decades, centralized group purchasing organizations, coupled with an outsourced pharmaceutical logistic, have replaced traditional approaches to avoid shortages. To make centralization strategy works, however, a good integration between warehouses and delivery infrastructure is fundamental. This means taking many decisions at all managerial levels. As these decisions are hard to be evaluated by hand, a computational tool becomes essential.

In this thesis, we present a decision support system for a pharmaceutical logistic company. In the first chapter, the software conception and implementation are presented. In the second chapter, the transportation part of the system is presented, with a focus on the computational approach to solve two closely related problems, a rich vehicle routing problem and a truck and driver scheduling problem. In the third chapter, we present a storage allocation problem that has special constraints associated with the pharmaceutical logistic, and an Iterated Local Search (ILS) based algorithm to solve it.

Additionally, the appendix contains two chapters that describe the results obtained in parallel researches developed by the author.

The first appendix presents an Adaptative Large Neighbourhood Search heuristic combined with a Set Partitioning model to solve a multiobjective dial-a-flight problem. In this problem, a heterogeneous fleet of airplanes must be routed to carry passengers to a destination. The objective is to minimize user inconvenience and costs. Each airplane has different speed, fuel consumption, capacity, and costs. The problem contains some hard-operational constraints such as airplane maximum weight, fuel unavailability in some airports and time windows.

The second appendix proposes four mathematical models and an ILS based heuristic to optimize a scheduling problem with position-dependent deterioration and maintenance activities. In this problem, a set of jobs must be scheduled on a set of parallel unrelated machines in order to minimize the makespan. Each job has an individual runtime and causes a deterioration on the machine that makes the runtimes of the next jobs rise. Maintenances, which have significant runtimes, can be scheduled between two jobs, making the machine recover its full performance.

Chapter 1

Introduction

The access to affordable healthcare is crucial to people wellness and is mentioned as one of the universal human rights (The United Nations, 1948). It is fundamental not only for the individuals, but to the whole society, influencing countries economy, instruction and even personal freedom. Without a regular and efficient access to healthcare it is not possible to people have a fully right to life and neither a real safe environment to develop any activity.

Unfortunately, it is a challenge providing a broad and good health coverage to the population. The costs of equipment, structure, and professional are huge, the management is complex hard, and the results are not easy to evaluate. Furthermore, we could cite other problems like the unequal distribution of health budget (70 times more per capita on high income countries than in low-income countries), lack of capillarity, overall corruption, private political interests, widespread lobbying and poor operational planning or execution. All those things together drain the resources and make whose that need more the service unattended.

To illustrate the scenario described above, we can start analysing the 2017 global spending on health, that was US$ 7.8 trillion, or about 10% of GDP and US$ 1.080 per capita, but 70 times more per capita on high income countries than in low income countries (WHO, 2019). It bring us around 5 million deaths could be avoided in 2016 (when the global spending on health was US$ 7.6 trillion) if the patients could receive a good-quality care and 3.6 million deaths were caused simply due to non-utilisation of health care (Kruk et al., 2018). Only in the United State, almost 45000 deaths per year are associated to the lack of a health insurance (Tanne, 2008), a reality of 27 million persons in the country at 2018 (Berchick et al., 2018).

As we can see, the effects of lack of healthcare access are somewhat common and known, but they can get worse quickly. As the infrastructure of public/private health systems are defined based on previous data, the arrival of a significant and unexpected event usually causes more overloads on services and force them to be re-dimensioned. In emergence occasions - like a pandemic, earthquake, hurricanes, etc… - the installed structure is often so stressed that becomes unable to deal with all the requests. Without discussing which could be the best method to manage a national healthcare system (public or private, insured or on demand,

etc...), it is straightforward to notice that in these cases, a rational and correct strategy to distribute and (re)allocate professionals, medicament and equipment is strongly recommended to avoid even more catastrophic situations, like the explosion of avoidable deaths, permanent damages, and finally the burden of all care workers involved.

A dramatic and most recent example of this situation was seen in the beginning of the covid-19 pandemic in March of 2020, when the disease spread quickly over all the world, mainly on Europe and North America. As the disease was still new and highly unpredictable, one of the unique recommended methods to slow down the spread of virus was the use of facial mask. However, most of countries governments were terrified with the mask shortage (a fast and cheap to produce item) and with the fact that half of world production was concentrated on China (Wu et al., 2020). To solve this problem, some heterodox measures were taken, as mask importing bans (OEC, 2020), artisan production or even cargo confiscation during transportation in international airports (Tarquini, 2020)(BBC, 2020). The lack of material was so critical that some doctors were asked to reuse disposable masks, a practice that would be never accepted in normal conditions (Pengilley, 2020) and other health workers got infected or even died because they were not able to follow the safety procedures (Hong, 2020). Furthermore, the lack of professionals to deal with all the patients lead make some countries, like Italy, to put medicine students to work on hospitals without taking final exams (Coleman, 2020).

It is evident that the mask shortage described could be better solved if the mask production was less centralised, or the global supply chain was not broken due several lockdowns or the local authorities have a contingence plan to answer an unexpected demand of materials, like the guidelines proposed by the American Centers for Disease Control and Prevention (CDC, 2020). A responsive supply chain redesign could allow hospitals or pharmacies storage rebalance and then to protect more people and probably saving lives. However, the number of decisions that must be taken in these scenarios is incredible high and the effects of these decisions can be worse than doing nothing. Finally, as we can see on (e.g. Caunhye et al. 2012, Galindo and Batta 2013, Acar and Kaya 2019, Özdamar and Ertem 2015 and Wang and Chen 2020), the decisions that need be taken are strongly dependent on damage and demand forecast.

Fortunately, huge events like the covid-19 pandemic are somewhat rare. Even in an average emergency, it does not take more than some weeks to be solved and neither causes long crashes in the supply network (except in case of wars). As in other kind of services, health care services have regular demands, with seasonal variations already known by the decision takers and authorities. The regularity of health care demand evidently helps to create strategies to solve some local and occasional problems, but it does not guarantee however that the service will run in an efficient or flexible way. In any case, the problem complexity remains considerable, requiring collection and analyse of large amount of data just to figure out what

are the problems or priorities. In the last decades, the use of information systems to execute these tasks are becoming more frequent every day. As they can delivery forecasts and insights hard to be delivered by skilled professionals, creating and updating these systems have a direct impact in improving the quality of service.

In this context, artificial intelligence and operations research techniques are already consolidated tools to solve problems related with the management of healthcare services, as can be seen on (e.g. Aboueljinane et al. 2013, Rais and Viana 2011, Aringhieri et al. 2017 and Acampora et al. 2013). Among these problems we can cite: hospital personnel allocation and scheduling (e.g. Burke et al. 2004 and Erhard et al. 2018), ambulance allocation and dispatching (see Bélanger et al. 2019, Bélanger et al. 2012 and Zaffar et al. 2016), health care facility location (e.g. Ahmadi-Javid et al. 2017, Güneş et al. 2019, Moeini et al. 2015, Afshari and Peng 2014, Sharma et al. 2019 and Dogan et al. 2019), patient to staff/facilities schedule and assignment (e.g. Ogulata et al. 2008, Condotta and Shakhlevich 2014 and Schimmelpfeng et al. 2012), operating room planning (e.g. Zhu et al. 2019, Aringhieri et al. 2015, Zhang et al. 2009 and Dios et al. 2015), etc...

Following the mask shortage thread, previously mentioned, we can introduce another issue that is one of most important, but usually less remembered, on health care services, the storage and distribution of medication and personal protective equipment (PPE). While the number of doctors, nurses, ambulances or some expensive machine are always cited when the quality of service is evaluated, the medicines supply is commonly forgot, especially in countries where they are not provided by the government/hospital. However, there are situations where a nurse and a bed can be useless to a patient if there is not gauze or a tetanus vaccine available for the first aid, or some surgery cannot be done by the lack of anaesthesia.

The poor management of medicine/PPE storage and distribution causes significant losses to hospitals, distributors, pharmacies and finally to patients. Not only the items shortage can be harmful, but an incorrect storage or transportation can lead to accidental poisoning, PPE contamination, reduced therapeutic effect and finally disposal by expiration or degradation. Medicaments disposal is particularly undesirable because it represents a waste of economic resources that could have other destinations and thus an increase of average cost of healthcare.

There is not much data about the size of economic impact caused by unappropriated storage on medical centres, however the available information can show how important this to their internal budget. For example, a study conducted at hospitals in the Capital Region of Copenhagen, Denmark, estimated savings of 1,5 million euros in one year in the region only by correcting the storage of refrigerated drugs (Colberg et al., 2017). In another study is presented an estimate of Trueman et al. (2010) showing that each year between £100-£800 million worth of dispensed medicines go unused and are ultimately discarded by United Kingdom National Health Service (NHS), which represents around 0.3% of the budget in that time. In this case, however, the unused/discard of medicines are not only caused by tactic

or operational management of supply, but also by problems related with the interruption of medicament use by patients (due the cure, death or side effects).

An efficient supply chain is one of the alternatives to reduce this kind of problem. As it is possible to reduce the time and complexity required to receive the material needed, redistribute non used medicines and predict how the materials and medicines will be requested during a time horizon, it is also possible to reduce the complexity of keeping these items stored and thus, the possibility of loss due to incorrect storage practices.

In other worlds, one of the strategies to solve this problem is to make any part of the supply chain work only (or mostly) with its speciality. A hospital should be concerned mainly with how to take care of patients, not how to manage a warehouse. In the other hand, a distributor should not be aware about the doctor scheduling and its influence on the demand of gloves and masks. A doctor should not have to check if there is a medicine in some shelve in a unknown place before prescribe it to a patient and neither the patient should not care about to take all the pills he received only to avoid waste. These decisions are in different environments and all a high-level healthcare manager should worry about is to keep the process working in harmony to avoid conflicts, questionings, and shortages.

In an ideal working environment, a hospital would keep only a small storage to some operation days and receive the material needed in only few days or hours. Those materials would be kept in a proper larger warehouse most of time and distributed to many customers as soon as possible.

This strategy is already adopted by public administrators in some Italian regions to keep a fast and regular flow of medicines and PPEs to the hospitals and pharmacies they manage. It is based in the outsourcing of storage and distribution services, in which a public administration firms a contract with a single transportation company and make it responsible by the delivery of products required in their installations. In this structure the company does not buys or negotiates any product with the providers - that have already they own contract with government - but receives, organise and stores the products, and after transport it to the costumers when they require it.

In this operation, as it was mentioned, the hospitals and pharmacies do not need to manage a large storage installation by themselves, because it is fast and quick to ask for a load of materials every time they want. Furthermore, the hospital/pharmacy manager does not deal with a poll of providers to keep its services working, do a market survey every time they need an item or to worry about buying large amounts of products to get better prices or a quick delivery. All these operations are done through an interface represented by the government contracted distributor, that tries to make the task of demanding one ton of medicines as simple as asking for a pizza, or almost it.

To the government, this approach gives the advantage of buying a large volume of products and then getting better prices and conditions in the contracts. It allows either to take medium-

long term decisions, that are not affected by government crisis or changes or maybe by short sighted decisions (if we are optimistic when we think the current government can make good decisions). The public administration also does not need to worry about how and when those products will be delivered and how those products will be individually used by each customer, as those decisions are decentralised and the costumers have relative autonomy to request necessary items.

Finally, to the distribution company this layout is interesting because it allows them to have a long and stable operation, reducing risks, cost, and complexity. It also makes possible to create better installations to store the products (as it will be used in a long period), reducing the eventual losses of medicines caused by incorrect or low quality storage that could exist in costumer's installations if they need to keep large volumes of products.

To this operation works it needs to be economically viable for all stakeholders, mainly to the distributor, that has not an "infinite" source of money to spend and needs to have some profit to avoid its bankruptcy. In this context, the work described in this thesis is focused on a pharmaceutical product distributor operation, with the objective of improving its efficiency and reduce costs. The text reports the proposal and development of an analytics software that works as a decision support system, where combinatorial optimization methods are the main tool used. The motivation of the project was the rising complexity involved in the management of product storage, worker shifts, vehicle loads, delivery routes, warehouse dimensioning, etc... that made necessary a more scientific, automatic and precise evaluation of costs and alternatives.

All the activities and system functionalities developed were done under the company supervision and following its guidelines and operations. The focus was to support the current business operation, but not proposing structural changes or a new way of work, following a philosophy of fitting the software in the company, not the company in the software.

The discussion in the text will be limited to the algorithms proposed and strategies to recover, process and analyse the data already available. The text will not present deeper considerations about the company operations or if they are intrinsically reasonable or not, unless when it directly affects the way the algorithms or functionalities were developed.

Considering the situation described above we can specify the main and secondary objectives of the study reported in this thesis as follows:

- **Main objective** Create a decision support system based on Operational Research algorithms for improving the decision-making process in a pharmaceutical logistic company and evaluate the company strategies and operation.

- **Secondary objectives**

  - Evaluate the suitability of Operational Research algorithms to solve real logistic problems in a controlled and automated company operation.

- Study the relevance of keeping a consistent data flow to get best results in a warehouse management.

- Study and report the software process implementation in the company as well the changes in the project required after the first contact with the final users.

The structure of the text will be the following: In Chapter 2, an overall vision of the system will be presented, showing the main functionalities, technologies used to implement it, the data flow and storage, the deployment procedure and a small description of use. In Chapter 3 is described the routing problem, and the strategy to solve the truck and driver allocation problem. In Chapter 4 is presented the storage allocation module, in which we detail the data input, the data processing to let the information ready to be used in the algorithm, and the heuristic used to solve the problem. Finally, we present the conclusions, with an overview of possible system improvements and directions in research.

Additionally, the appendix contains two chapters that describe the results obtained in parallel researches developed by the author. The first appendix presents an Adaptive Large Neighbourhood Search heuristic combined with a Set Partitioning model to solve a multi-objective dial-a-flight problem. The second proposes a set of mathematical models and an ILS based heuristic to optimise a scheduling problem with position-dependent deterioration and maintenance activities.

1.1 Bibliography

Coronavirus: Us accused of 'piracy' over mask 'confiscation'. *BBC*, 2020. URL: http://www.oecd.org/coronavirus/policy-responses/the-face-mask-global-value-chain-in-the-covid-19-outbreak-evidence-and-policy-lessons-a4df866d/

The face mask global value chain in the covid-19 outbreak: Evidence and policy lessons, May 4, 2020. URL: http://www.oecd.org/coronavirus/policy-responses/the-face-mask-global-value-chain-in-the-covid-19-outbreak-evidence-and-policy-lessons-a4df866d/.

Optimizing personal protective equipment (ppe) supplies, 2020. URL: https://www.cdc.gov/coronavirus/2019-ncov/hcp/ppe-strategy/index.html.

L. Aboueljinane, E. Sahin, Z. Jemai. A review on simulation models applied to emergency medical service operations. *Computers & Industrial Engineering*, 66(4):734 – 750, 2013.

G. Acampora, D. J. Cook, P. Rashidi, A. V. Vasilakos. A survey on ambient intelligence in healthcare. *Proceedings of the IEEE*, 101(12):2470–2494, 2013.

M. Acar, O. Kaya. A healthcare network design model with mobile hospitals for disaster preparedness: A case study for istanbul earthquake. *Transportation Research Part E: Logistics and Transportation Review*, 130:273 – 292, 2019.

H. Afshari, Q. Peng. Challenges and solutions for location of healthcare facilities. *Industrial Engineering and Management*, 3(2):1–12, 2014.

A. Ahmadi-Javid, P. Seyedi, S. S. Syam. A survey of healthcare facility location. *Computers & Operations Research*, 79:223 – 263, 2017.

R. Aringhieri, M. Bruni, S. Khodaparasti, J. van Essen. Emergency medical services and beyond: Addressing new challenges through a wide literature review. *Computers & Operations Research*, 78:349 – 368, 2017.

R. Aringhieri, P. Landa, P. Soriano, E. Tànfani, A. Testi. A two level metaheuristic for the operating room scheduling and assignment problem. *Computers & Operations Research*, 54:21 – 34, 2015.

V. Bélanger, A. Ruiz, P. Soriano. Déploiement et redéploiement des véhicules ambulanciers dans la gestion d'un service préhospitalier d'urgence. *INFOR: Information Systems and Operational Research*, 50(1):1–30, 2012.

V. Bélanger, A. Ruiz, P. Soriano. Recent optimization models and trends in location, relocation, and dispatching of emergency medical vehicles. *European Journal of Operational Research*, 272(1):1 – 23, 2019.

E. R. Berchick, E. Hood, J. C. Barnett. Health insurance coverage in the united states: 2017. *Current population reports. Washington DC: US Government Printing Office*, 2018.

E. K. Burke, P. De Causmaecker, G. V. Berghe, H. Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.

A. M. Caunhye, X. Nie, S. Pokharel. Optimization models in emergency logistics: A literature review. *Socio-Economic Planning Sciences*, 46(1):4 – 13, 2012. Special Issue: Disaster Planning and Logistics: Part 1.

L. Colberg, L. Schmidt-Petersen, M. K. Hansen, B. S. Larsen, S. Otnes. Incorrect storage of medicines and potential for cost savings. *European Journal of Hospital Pharmacy*, 24(3):167–169, 2017.

J. Coleman, Italy will rush 10,000 student doctors into service, scrapping final exams,March 18, 2020. URL: https://thehill.com/policy/healthcare/public-global-health/488191-italy-will-rush-10000-student-doctors-into-service.

A. Condotta, N. Shakhlevich. Scheduling patient appointments via multilevel template: A case study in chemotherapy. *Operations Research for Health Care*, 3(3):129 – 144, 2014.

M. Dios, J. M. Molina-Pariente, V. Fernandez-Viagas, J. L. Andrade-Pineda, J. M. Framinan. A decision support system for operating room scheduling. *Computers & Industrial Engineering*, 88:430 – 443, 2015.

K. Dogan, M. Karatas, E. Yakici. A model for locating preventive health care facilities. *Central European Journal of Operations Research*:1–31, 2019.

M. Erhard, J. Schoenfelder, A. Fügener, J. O. Brunner. State of the art in physician scheduling. *European Journal of Operational Research*, 265(1):1 – 18, 2018.

G. Galindo, R. Batta. Review of recent developments in OR/MS research in disaster operations management. *European Journal of Operational Research*, 230(2):201 – 211, 2013.

E. D. Güneş, T. Melo, S. Nickel, Location problems in healthcare, in: Location science, Springer, 2019, pp. 657–686.

N. Hong. 3 hospital workers gave out masks. Weeks later, they all were dead. *The New York Times*, April 5 2020. URL: https://www.nytimes.com/2020/05/04/nyregion/coronavirus-ny-hospital-workers.html

M. E. Kruk, A. D. Gage, N. T. Joseph, G. Danaei, S. García-Saisó, J. A. Salomon. Mortality due to low-quality health systems in the universal health coverage era: a systematic analysis of amenable deaths in 137 countries. *The Lancet*, 392(10160):2203 – 2212, 2018.

H. L. Wu, J. Huang, C. J. Zhang, Z. He, W.-K. Ming. Facemask shortage and the novel coronavirus disease (covid-19) outbreak: Reflections on public health measures. *EClinicalMedicine*, 21:100329, 2020.

M. Moeini, Z. Jemai, E. Sahin. Location and relocation problems in the context of the emergency medical service systems: a case study. *Central European Journal of Operations Research*, 23(3):641–658, 2015.

S. N. Ogulata, M. Koyuncu, E. Karakas. Personnel and patient scheduling in the high demanded hospital services: a case study in the physiotherapy service. *Journal of Medical Systems*, 32(3):221–228, 2008.

World Health Organization, Global spending on health: a world in transition. *Technical Report*. World Health Organization, 2019.

L. Özdamar, M. A. Ertem. Models, solutions and enabling technologies in humanitarian logistics. *European Journal of Operational Research*, 244(1):55 – 65, 2015.

V. Pengilley, Sydney doctors claim they are being asked to reuse coronavirus face masks, sparking safety fears, 2020. URL: https://www.abc.net.au/news/2020-03-30/sydney-doctors-asked-to-reuse-face-masks-in-coronavirus-shortage/12100952.

A. Rais, A. Viana. Operations research in healthcare: a survey. *International Transactions in Operational Research*, 18(1):1–31, 2011.

K. Schimmelpfeng, S. Helber, S. Kasper. Decision support for rehabilitation hospital scheduling. *OR Spectrum*, 34(2):461–489, 2012.

B. Sharma, M. Ramkumar, N. Subramanian, B. Malhotra. Dynamic temporary blood facility location-allocation during and post-disaster periods. *Annals of Operations Research*, 283(1):705–736, 2019.

J. H. Tanne. More than 26 000 americans die each year because of lack of health insurance. *BMJ: British Medical Journal*, 336(7649):855, 2008.

A. Tarquini, Coronavirus, mascherine per l'Italia sequestrate dalla Repubblica Ceca. L'ambasciata italiana: "Praga si è impegnata a inviarci un numero uguale", 2020. URL: https://www.repubblica.it/esteri/2020/03/21/news/coronavirus_cosi_la_repubblica_ceca_ha_sequestrato_680_mila_mascherine_inviate_dalla_cina_all_italia-251883320/.

The United Nations. *Universal Declaration of Human Rights*. 1948.

P. Trueman, D. Taylor, K. Lowson, A. Bligh, A. Meszaros, D. Wright, J. Glanville, J. Newbould, M. Bury, N. Barber, et al. *Evaluation of the scale, causes and costs of waste medicines. report of dh funded national project.*, 2010.

C. Wang, S. Chen. A distributionally robust optimization for blood supply network considering disasters. *Transportation Research Part E: Logistics and Transportation Review*, 134:101840, 2020.

M. A. Zaffar, H. K. Rajagopalan, C. Saydam, M. Mayorga, E. Sharer. Coverage, survivability or response time: A comparative study of performance statistics used in ambulance location models via simulation-optimization. *Operations Research for Health Care*, 11:1 – 12, 2016.

B. Zhang, P. Murali, M. Dessouky, D. Belson. A mixed integer programming approach for allocating operating room capacity. *Journal of the Operational Research Society*, 60(5):663–673, 2009.

S. Zhu, W. Fan, S. Yang, J. Pei, P. M. Pardalos. Operating room planning and surgical case scheduling: a review of literature. *Journal of Combinatorial Optimization*, 37(3):757–805, 2019.

Chapter 2

Creating a decision support system to a healthcare logistic operator

Developing a software for a company and inserting it in the everyday company operation is often a hard task, because it involves understanding problems, defeating internal resistance and changing business practices. In this chapter, we present the company operation and the development process of the decision support system described in this thesis. The first part of the chapter gives an overview of health care supply chain management and how our partner company operates on it. In the second part, we present how was the software conception, but without discussion about the algorithms used to solve the optimisation problems.

2.1 The health care supply chain management

Healthcare services are one of the most essential services in the modern society, comparable only with security and food/water supply. It is composed by a pool of activities like first aid services, psychological aid, chronic disease treatments, medicines distribution, clinical examination, disease prevention, support to pregnant, elder or disabled persons, physiotherapy, etc... These services are useful not only to keep people alive or bring them immediate relieve, but either to increase the well-being through the reduction of considerations about availability and quality of the healthcare. It is also important to highlight the economic impacts of a good healthcare system when it avoids the loss of workforce caused by premature deaths, incapacitating diseases, temporary licenses and also the burden of relatives and professionals dedicated in caring the patients.

The importance of healthcare to the society and economy have attracted the attention of governments and private entities. They, concerned in keeping people healthy with a limited budget,have increased the efforts to provide a higher quality of service and make this service accessible to a broader public. Among the measures adopted to reach this objective, we highlight:

- Increasing the number of professionals in different areas (doctors, psychologists, nurses, physiotherapists, paramedics, biochemists, etc...).
- Building new hospitals, pharmacies and other health facilities while keeping and recovering the old ones.
- Expanding existent hospital and number of available beds.
- Health orientation to disease prevention.
- Improvement on medical equipment and medicine distribution.

The increase in number of professionals can be observed around the world. It is also possible to notice a rise in number of hospitals and available beds, making possible to receive more patients simultaneously inside a structure.

However, healthcare service needs more than people and physical infra structure to work well. The service is highly dependent on availability of materials like medicines, equipment, and office materials. As the health service providers are aware of the impossibility of predicting exactly when each item will be used, they need to keep the items stored in hospitals and pharmacies until when they are required or discarded by medical personnel or patients. Defining how these things will be stored, distributed, and replenished is a complex task that requires a high knowledge about items shelf life, storage conditions, turnover rate, etc and also a pool of professionals to allocate, retrieve and ordering items, control inventory levels and items integrity, etc... aiming to avoid any level of shortage. These tasks create an overload on health staff, that ideally should be concentrated on caring the patients.

Keeping material flow in a facility that provides a product or a service to the final customer is a challenge not only to hospitals and pharmacies. Instead, most of companies that execute activities significantly dependent on manufactured materials have similar problems. In most of them, it is almost impossible to control the flow of the material needed, from the production until the use, with a reasonable cost and time. Thus, to continue operating and satisfying customer demands, the companies are constrained to insert themselves in a supply chain.

A supply chain, as defined in Aldrighetti et al. (2019) is a network of organisations and processes wherein enterprises (suppliers, manufacturers, distributors and retailers) collaborate to acquire raw material, convert it in intermediary products and then in final products to finally deliver them to customers. Supply chains are part of an evolution on manufacturing systems, once the responsibility of doing well each step is divided among several parts, that can have a more focused and efficient operation.

While a good supply chain reduces a company operation complexity, it also requires the ability of maintaining all the links working in a harmonic way. For instance, an industry responsible to create final products needs to track its suppliers and have a plan to mitigate the effects of missing or delayed deliveries. At the same time, it needs to check if their

distributors (internal or outsourced) are always able to deliver the production to the next link, which can be a distribution centre, a customer or even other facilities owned by the company. The effort of coordinating all the entities in this context is called supply chain management.

In some cases, a failure of a supplier in delivering a product does not represent a relevant problem, due to the availability of companies that can replace it quickly and without significant losses. Unfortunately, in the medical or pharmaceutical supply chain this is not true. Unlike consumer products, where the customer can delay a purchase or buy an alternative product, in a health care service maybe there is not alternative treatment or time for waiting a new supplier (Mustaffa and Potter, 2009). With a globally dispersed and consecutive production/distribution steps, it is not unusual the supply chain cycle time to be 100-300 days, making hard to address expansion on the demand, as we already mentioned in the introduction (Shah (2004) and Friemann and Schönsleben (2016)).

Due the high unpredictability of demand in short term and to lower the risks that can be caused by an eventual break in the flow, many hospitals adopt the strategy of keeping their inventory level close to 100% to avoid eventual shortages (Uthayakumar and Priyan (2013) and Aldrighetti et al. (2019)). Although this strategy is efficient in avoiding problems in the service, it increases operational costs. Estimates say that 10% to 18% of hospital net revenues are spent only in inventory costs (Volland et al. (2017), Nicholson et al. (2004) and Jarrett (1998)). High inventories also can lead to waste of medicines and other sensible materials, due to problems in the storage or short shelf lives.

Another option to solve the problem is making the supply chain more robust, especially on delivering medicines to hospitals and pharmacies at the right time and then reducing their managerial burden in weaker nodes, the costumers. This solution requires a higher level of management and a deep knowledge of the actors involved in the network, that most of times have an active participation of governments, hospitals and pharmacies centralised groups.

To clarify this approach, we need first to clarify the parts of the process, as well their roles and connections. A typical pharmaceutical supply chain will consist of a subset of the following nodes: primary manufacturing, secondary manufacturing, market warehouses, distribution centres, wholesalers, group purchasing organisations, retailers and hospitals (Shah (2004) and Uthayakumar and Priyan (2013)).

Primary manufacturers process raw material and produce the medicine active ingredients, that will be sent to the secondary manufacturers, where the ingredients will be put together and mixed with other materials in order to create a formula and so the final product (Shah, 2004). Wholesalers and group purchasing organisations (GPO) are companies or group of companies that buys large quantities of materials from the producers to a posterior distribution to retailers, that can be done directly or through warehouses/distribution centres. The main difference between them is that a GPO is formed by customers looking for the advan-

tages of buying large amounts of the same products, like lower prices, continuous supply, or better delivery conditions. By its turn, wholesalers are individual companies that deal with producers to get the products and after selling them to retailers in large amounts.

In this network, wholesalers and GPO have been increasing in importance as the main actors in the process of materials acquisition in hospitals and pharmacies. This move started when hospital and pharmacies managers observed the advantages of outsourcing part of their logistics, specially the centralisation of requests and storage, a lower number of invoices to process, faster deliveries and reduction on the need of huge storages on customers facilities.

This strategy, however, must be planned better than the previous, due its higher operation complexity. It is far easier to correct minor problems in the inventory of a single hospital than solve a problem caused by a centralised warehouse. Once all the orders are managed by just one company, all the issues that could happen individually in each customer can be also centred in this company. Thus, to work well, this approach needs to guarantee a high reliability of the warehouse and distributors and a good communication between the customers and logistic operators.

In Beaulieu et al. (2018) is presented an example of how problematic this strategy can be in the worst case. In this study, a GPO terminated a contract with a logistic operator approximately one year after the begin of services, mostly because of several mistakes on deliveries and explosion of costs to the operator.

In other hand, a highly successful example of this kind of organisation can be seen in Italy, especially in North and Northeast regions, where in the last 15 years, healthcare has been changing from a traditional decentralised system to a modern centralised one (Aldrighetti et al., 2019).

In Reggio Emilia, for example, there is the Area Vasta Emilia Nord (AVEN), one of three area vasta consortia based on Emilia-Romagna today, created in 2004 to serve healthcare customers in the Reggio Emilia province and adjacencies (Aldrighetti et al., 2019). In these consortia, purchases are centralized and negotiated by high governmental entities, that ask suppliers to deliver all the required products in a set of large warehouses. These warehouses are managed by a third part logistic company that receives and delivers the orders to hospitals and pharmacies.

Similarly, there are *area vasta* like structures in Lombardy, Veneto, Friuli-Venezia-Giulia and Tuscany. In this context, our partner company entered in this market saw this new organization of health care services as an opportunity to expand its business, by offering logistic services during a long period to a stable and trustworthy client. Nowadays, they are responsible for warehouses management, product storage and distribution in several *area vastas* like organizations.

In the next section, we will explain in detail how the company operate in these healthcare centralized organizations. Following, how our project was conceived and implemented inside

the company culture.

2.2 The company operation on centralised healthcare logistic

The company operation in the context of healthcare supply chain is focused on product storage and distribution, according with the specifications defined in a contract with government entities and the requirements of hospitals and pharmacies in the covered zone.

An important characteristic of this operation is the isolation between the services provided to different areas. Even when two customers are geographically close, the structure of one region cannot be used for helping to serve another. For example, the trucks used to deliver products to one region must be used only in that region, without occasional re-allocations. In the same way, a product can not be picked from one regional warehouse and transported to another, even if this move is controlled and the product is replaced posteriorly.

There are several reasons to the company proceed in this way. The first one is the contract, in which is described that warehouses are built or rent to be used by an specific region, not being a property of the company. We can also cite the different budgets each warehouse set in one region have for working, the risks of eventual contamination, loss of track-ability or undesired lot swaps and finally managerial problems caused by eventual miscommunications about resource sharing.

Internally, the warehouse operation is more like a regular structure. When the products arrive, they pass by an inspection phase and if the result of this inspection is positive, they are allocated in an empty space inside their dedicated zone. These zones are defined following the ABC class criteria and the type of product (i.e. refrigerated, flammable, toxic, controlled, corrosive).

By its turn, product distribution is triggered by an order, sent to the warehouse by a hospital or pharmacy served in the area. After that, a team of pharmacists validate the orders according with the delivery capacity and items availability. Once the order is validated, it enters in a queue of orders to be recovered in shelves and then accumulated in an expedition point. The items are then put in boxes, following their specificities and destination, and travel by rolls to a zone where they will be organised in pallets to be loaded in trucks.

Once the trucks are loaded, they can delivery the products directly to the clients or to intermediary warehouses, where the pallets can be stored a single day. The delivery routes are created respecting the vehicle characteristics, the compatibility with customer structure to receive it and finally driver related constraints, as will be detailed in the next chapter.

2.3 Software introduction

The use of electronic computers started in the first half of 20th century and were initially restricted to some military and scientific researches, in which long and hard mathematical

calculus, previously executed by hand, must be done quicker. The world "computer" itself comes from the Latin *computare*, that means execute calculus and the first electronic computers where called computer machines, as can be seen in the name of a pioneer computing society, the Association of Computer Machinery (ACM).

Early in the computer machinery development some persons saw as interesting the possibility of using those automatic calculators in commercial and industrial environment, even if it was done in a limited range of activities due to the high costs of maintaining these machines. This interest become evident and was boosted with the development of COBOL programming language, in 1959, that become so popular that still is used in current day softwares (even with a high effort to replace it). During the sixty years that separates the rise of COBOL and the current days, the development of computers changed completely our lives, creating situations that someone could describe as impossible 20 years ago. This process was nominated with several names, as third industrial revolution, digital revolution, industry 4.0, etc...but all of them with the objective of expressing the same idea: the replacement and empowering of human capabilities by some kind of automatic or artificial intelligence.

One of scientific areas that were highly benefited by the computer invention and improvement was Operations Research. With a name that poorly describes its scope, the Operations Research have born as a military discipline and science to study how to translate human activities or decisions in a mathematical representation and then optimise some metric through the choose of one (or some) of these possible decisions. With this particular way of describing the world, Operations Research was always strongly dependent of the computation speed, even before the popularisation of electronic devices, mainly when it used the popular method to optimise linear problems, the simplex algorithm, that basically consist in finding several solutions to a linear equation system using techniques of linear algebra.

As the success (or failure) of a company is strongly related with their member's decisions, and the manual evaluation of these decisions is often not possible, the Operations Research arose as an important tool in commercial, industrial, and financial world. Even if the area is not known and recognised as other fields, like Artificial Intelligence, Game Theory or Statistics, is hard to see a large company that does not use, at least indirectly, its techniques to get better results.

Operations Research has a central role in the project described in this thesis, once it is the kernel of all the decision process improved by the system. As the company must respect strict quantitative e qualitative contract requirements and still have profit, balancing the level of service and the costs becomes fundamental. Providing an automated tool to analyse several complex scenarios is, in this context, particularly important to get good results without compromising a huge volume of workforce that could be used in most hard to automatise tasks.

In the sequence of chapter is presented the software project and how it was planned, from

the requirement analysis until the main major changes asked by the company after the first contact with the system. Also, the development process will be described, showing the system architecture and the justification to the design chosen.

2.4 Software conception

The project presented in this thesis started with an agreement between the UNIMORE Department of Sciences and Methods for Engineering (DISMI) and a logistic company in 2017.

More specifically, the system project started with the company demand for a computational tool for helping to solve organisational problems related with the pharmaceutical products distribution. These problems would be mostly related with warehouse organisation and delivery routing, being focused on day-to-day operations and some tactical decisions.

A relevant amount of time was spent to understand the optimisation problems to be solved, their scope and how the company would like to input the data and visualise the optimisation output. This step was conducted with several interviews and a constant communication done through questions and answers sent by email. The objective at this point was to create a software requirements specification, a document in which is described what the proposed software should do without describing how the software will do it. This document is particularly useful to create an agreement and a common understanding between developers, project managers and the contracting company, in a way that an formal path can be followed without surprises (Jalote, 2012).

Defining software requirements is a well-known issue in the software engineering industry and either on Operations Research. Although this phase is not commonly cited on Operations Research literature, it can change completely the solving approach due to the variations on the problem understanding it can lead. Once the software requirement specification is a very wide topic with a rich literature, we will not discuss in detail the methods used in industry (also in academia) to create it. Nevertheless, when it is needed, it will be reported how the specification guided the software development and eventually how specification changes have impacted on development flow.

Considering the optimisation problem specification, the points that needed to be elucidated with the company were: (a) the metrics which should be optimised; (b) the decisions allowed to be made (and consequently the ones that must remain unchanged); and (c) decision limitations. These points were used to define, respectively, the problem objective function, the decision variables, and the constraints, as well to confirm the problems were typically mixed linear-integer problems (MILP), thus within the limits of our expertise.

It is worth to notice that a formal definition of problem decision variables, bounds and constraints does not mean, in any sense, a definition of how the problem will be solved using

a model, exactly, approximately or even heuristically. This is only to clarify, to both parts, the understanding of the problem, to avoid conflicts in posterior phases of project.

Although these points look easy to understand and define in the first glance, on real applications they represent a painful part of work. Most of times contractors are not able to give direct answers to the questions presented or even have never thought about it. In other situations, if they can exactly answer what they need, they are not able to provide or retrieve the data that would be necessary to validate their assumptions or to evaluate constraints or objectives.

An example of issue on the input specification of an optimisation software is to define if an input represents or not a feasible scenario. This is a well-known problem, but not a trivial one, as can be seen be seen in (Chinneck, 2007). While both commercial and open source solvers usually can define if the input model is feasible, it is not possible to yield that this check will be done quickly. Even when it is done quickly, many times the reason of the infeasibility cannot be easily determined, mainly if the unique available information are the variable and constraint sets that cause the infeasibility, like is presented in MILP-solvers. Finally, both cases described before are based in the fact that is possible, or practical, to represent the problem as a model (only to check the feasibility), instead of checking it using a custom implementation.

2.4.1 Problems definition and main system functionalities

A set of four optimisation problems were fixed to be put in the software. They are shortly described below:

- **Products storage allocation:** Defines how to organise the SKUs in a warehouse with the objective of optimising the process of picking, assembly, and delivery the SKUs required by the costumers. Starting from a set of SKUs, a set of orders and a warehouse description, the algorithm needs to decide in which cell and level (if the cell is vertically divided) a SKU must be stored. The objective function chosen to be minimised is the total distance to pick all the products and bring them to the dispatch point. The position of each SKU must be unique, as must be also unique the SKU in each position. Some SKUs cannot be stored in some areas on the warehouse and some SKU families need to be put in separated areas, without being close to other SKU families.

- **Picking personnel scheduling and rostering:** Defines when each picker will start and stop to work during a time horizon aiming to reduce the costs without delaying the picking process. In this case problem is also desirable to avoid high variations on the number of workers in shot time periods, do not allow overtimes or short daily shifts.

- **Vehicle and driver scheduling:** Once the products are already prepared to be dispatched (or almost prepared), the vehicles and its respective drivers must be already

ready to depart. Thus, we need to define which vehicle will be assigned to each delivery route as well who will drive it. The objective is to reduce the number of vehicles and drivers necessary to deliver all the orders without delays and without disrespecting the drivers maximum working hours. Due to operational constraints, some vehicles/drivers cannot visit some customers and neither be assigned to routes where these customers are visited.

- **Vehicle routing:** Define a set of routes to deliver all the orders in a specific time horizon aiming to reduce the transportation cost. Given a set of vehicles, grouped by capacity and average speed, should visit the clients, and deliver them the orders in the day and time required (or maybe in the previous day). The number of orders delivered by each vehicle is defined by the vehicle capacity that must be greater or equals to the sum of orders "size". Intermediary warehouses can be used, when are available, to store for one day products that must be delivered in hard to reach customers.

It is not hard to see how these four problems are closely related inside the warehouse operation. The products only arrive on time at the costumer's facilities (hospitals and pharmacies) if it is possible to depart early enough from the warehouse. However, it is not possible an early departure if the products are not loaded into the truck at the schedule time. To put all the required products into the trucks a group of pickers needs to pick all the products and other group needs to prepare them to the expedition. The size of this group and how many hours they need work to finish this task, the work shift of each one of them and the costs of working during different periods of the day. Furthermore, the number of hours a given set of pickers needs to pick all the products depends on how the products are organised in the warehouse, and the orders that need to be processed. Finally, to define what time would be early enough, we need to know the route the vehicle will run and how much stops it will do, that depends on how much vehicles are available, which one can visit each customer, how many drivers are available and how long can be their shifts.

One of the main challenges of this project is exactly this interdependence. The coverage of its decisional chain involves different levels of management and planning, from an operational work decided by some low-level supervisor, until strategic ones, where only high executives are involved. These parts need to work separately but with some degree of harmony, to provide flexibility but without breaking desirable confidentiality.

Defining the trucks and drivers that will be assigned by each route, for example, is a typical operational decision, taken by someone that visualise how the issues that can happen with one specific client or vehicle and has direct contact with drivers and their demands. This person does not need to know details about previous orders, cost of vehicles or the suppliers of each product to take a decision. In another hand, the product allocation in the warehouse is a tactical decision, that requires a good analytical evaluation, access to sensible information about product prices, shelf time and demands as well as a vision of the possible effects of a

right or wrong decision, requirements that are commonly fulfilled only by managers or low level directors. Furthermore, defining the number of vehicles, drivers, the size and location of a warehouse, the approximate number of employees, etc...are strategic decisions that works as inputs of other problems, but are mostly decided by top executives, responsible to guide the company in one or other direction.

Considering all the information presented in the previous sections and paragraphs and a high volume of sensible information that cannot be presented in this thesis, we could define the main set of functionalities, presented below:

- Create new problem scenarios
- Load a problem scenario data from *.xlsx* files
- During the data load, check possible inconsistencies and exhibit a log with them to the user
- Keep isolation between scenarios and its optimisation
- Enable the creation of empty scenarios to be fulfilled through the GUI
- Delete a scenario with its respective solution if it exists
- Start the selected scenario optimisation with one click
- Detect infeasibilities in the scenario and warn the user about them
- Allow the user to export the scenario optimisation solution
- Allow the user to export the scenario information in the same format the system loaded
- Create solution data dashboards and show them in the screen
- Duplicate scenario
- Implement big data algorithms on demand
- Create and remover new users
- Control the access to parts of software according to user type
- Divide the system in modules according to the problem
- Present, to each module, individual data load and visualisation screens
- Import in the module, as an input, the solution of another module
- Allow information edition through file import

- Allow insertion of data through file import

The system division in modules, as can be seen, is one of the main features of the system. As described above, each module will host the functionalities required to handle problem data and solutions, as well interact with another module when this is useful. This division also facilitate the insertion of new problems in the system in the future without impact in the handling of legacy models.

Another important concept used to define the functionalities is scenario. A scenario represents a problem instance, as well its detected inconsistencies and solutions. The data of a scenario is most of times independent of any data of others scenarios, so all editions, insertions and deletions made in one scenario will be not mirrored in others. Once the scenario data is loaded in the system, it remains on it until be actively deleted by the user, in this way, it is possible to track all the already loaded and optimised scenarios quickly, what enables comparisons and simulations almost automatically.

Details about how these functionalities were implemented, the problems found, and the solutions obtained will be described in the sequence of the text.

2.5 System architecture and implementation

The proposed system was conceived to be a web application running in a Microsoft Windows Server. We decided to create a web application due the advantage of enabling the user to access the system remotely in most of computers without concerns about installation, setting or compatibility issues. This also reduces the stress of keeping the system working each user machine individually, reducing maintenance and user support. The main disadvantages are related with handling concurrent accesses, security issues and the cost of keeping a server. However, as the number of concurrent users will be low and the company provided a structure to isolate the system operation, most of the possible problems are basically mitigated.

Before explaining how the system is implemented, we will present some software engineering concepts and their relevance on software development and functioning. These concepts are basically related with the organisation of responsibilities and functionalities inside the code (or system architecture) and are well known in development community, thus they will be not explained in deep details in this text.

A modern web application is commonly divided in two parts, the front-end and the back end. The front-end corresponds to the rendering, input and output processes that run directly on the browser, without accessing any remote machine, database, or file systems. The back end corresponds to processes that run in a remote machine or set of machines, called indistinctly “server”, that has a high computational power and access to databases, files, and other resources the user cannot access directly or keep in its all computer. In short, the first is responsible to computer-user interaction and information visualisation in the client

computer, the second is responsible by managing data storage, request and recovering, heavy data processing, data flow and security.

Inside the back/front end organisation is possible to create many software architectures, depending on factors like scalability, security, data volume, quantity of parallel users, interface complexity, integration with other software, etc. For example, thanks to the user's higher computational power, new web browser JavaScript motors, improvements on HTML, CSS and JavaScript languages, and the grow of web development libraries, it is possible to create complex web applications that run almost entirely on user computers, without any special plugin or software (like Adobe Flash), including games, compilers and complex text editors. By its turn, with the fall of hardware costs, expansion of the Internet network and improvements on parallel processing and redundancy, it is also possible to run very heavy applications on personal computers or mobile phones, like statistical analysis, a web search, simulations or data analysis, etc... without concerns about the user hardware and keeping a nice and easy to use interface.

In both cases described the back- and front-end separation remains. However, in the first case almost all tasks performed by the front end, while in the second, most tasks are executed by the back end. In specific cases, some of the functionalities can be even provided by another applications, called web services, that work like independent and callable parts of the system, kept by other development team or company.

One of the most popular software architecture patterns to implement a back/front end organisation, and our choice in the project, is the Model-View-Controller (or MVC). In this pattern the software code is basically divided in three: the model, representing the data managed in the system, as well the logic to communicate with database, thus to create, read, update or delete data (i.e. CRUD operations); the view, that renders the information and layout in the screen; and the controller, that manages the actions in the system and connects models with views (Aniche et al. (2018), Leff and Rayfield (2001) and Freeman (2015)). The main advantage of this pattern is the possibility of creating several different visualisations based on the same data, without dependence of how this data is represented or recovered in the database, once the controller will take care of interpreting the different kinds of requests and deliver what the interface needs to work.

But while MVC pattern is already supported and implemented in relevant web development frameworks, like CodeIgniter (http://codeigniter.com/), ASP.NET (https://dotnet.microsoft.com/apps/aspnet), Django (https://www.djangoproject.com/), Ruby on Rails (https://rubyonrails.org/) and Struts (https://struts.apache.org/) , being a consolidated way of building web applications, it does not fit perfectly on back and front separation. This is due the difficulties found in defining in which side (client or server) controllers and models will be allocated, once the view is obviously defined in the client computer (Leff and Rayfield, 2001).

In our project, part of these decisions had unequivocal answers. As the optimisation algorithms require big amounts of computational resources, all the code related with them were allocated on server. Following this same idea, no scenario validation or feasibility test was designed to run in the user machine. All the model part, as the data should be shared among the users, was allocated also on server. Events listening and handling, in other hand, were put all as a JavaScript responsibility on the user browser, except when those events were clicks in links or very general buttons, like delete or edit.

To put all these things peacefully together a pool of technologies was used. On the base of the application we used the ASP.NET Core framework (https://docs.microsoft.com/en-us/aspnet/core/) , an allegedly cross-platform version of ASP.NET framework, already mentioned. However, to avoid any kind of portability issues, we decided to develop the system all to Windows Server operational system, using the standard C# language.

Over the ASP.NET Core, we use Bootstrap (https://getbootstrap.com/) and Vue.js (https://vuejs.org/) libraries. The first uses a set of standard CSS classes to create responsive web pages in an easy way. The second provides the possibility of creating interactive and standardized behaviours on page to simplify input management, aggregation between different parts of page, partial reloads, etc..., being the first layer to create the interface functionalities. Moreover, to compose the dashboards were the libraries Google Charts (https://developers.google.com/chart) and D3.js (https://d3js.org/), as well the Google Maps JavaScript API (https://developers.google.com/maps/documentation/javascript) to draw maps and point relevant information on it.

The data input and output are performed using Microsoft Excel files, due to the high level of familiarity of users with this software and its intuitive way of structuring data. As the information in Excel files are not in plain text format or any easy to convert binary format, to read or write files we use the .NET library NPOI (https://github.com/dotnetcore/NPOI). The reading/writing processes are performed in the server and does not require the user any Microsoft Office installation in both user and server machine. It worth to notice that it is possible to insert information manually, but as the system require large amounts of data in each scenario, it is not recommended input all scenario data using this method.

The data storage, by its turn, is done using Microsoft SQL Server Express, the free version of the Database Management System (DBMS). While it has not all the functionalities and the same performance of the paid version, it is fast and robust enough to support our system in its initial usage.

To access and manage the data in the database we used the Entity Framework object-database mapper (https://docs.microsoft.com/en-us/ef/), that enables LINQ queries, change tracks, update and schema migrations, and works in other databases, like PostgreSQL, MySQL and Azure Cosmos DB. This framework also supports the coding-first approach, where the data definition is used as source to create a relational database, instead of adapting

to a previously created database.

Regarding the optimization algorithm's codes, the unique external and specific library used was JuMP (Dunning et al., 2017). This library implements a domain-specific modelling language for mathematic optimization embedded in Julia programming language, being especially useful to implement solver independent mathematical models, due to the interfaces provided by the MathOptInterface package (https://github.com/jump-dev/MathOptInterface.jl), an optimisation library also available for Julia.

2.5.1 Interface and data flow

As mentioned before, the system was developed as a web application, to provide a greater portability and lower user hardware requirements. More specifically, the interface was implemented using the libraries Bootstrap and Vue.js over a backend created with .NET Core framework. In this section, we make a short presentation of this interface and how the data flows on it, using as example the Routing and Business Intelligence modules.

The main objective of the interface was providing an easy and quick access to the main system functionalities, as data load and visualisation, solver call, solution visualisation and data export, as well allow an agile alternation between the modules.

As can be seen in the Figure 2.1, in the system main screen we have, in the left navigation bar, the list of optimisation modules, a link to the Business Intelligence Module, a link to special tools. In the centre, there are three shortcuts to access important or recently modified data. On the top navigation bar there are some information links and account control.

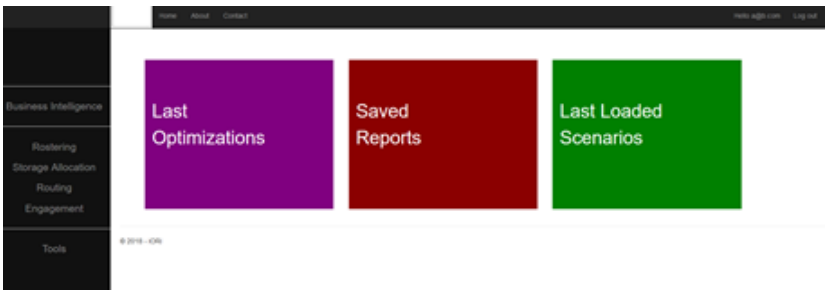


Figure 2.1: System main screen

It is important to notice that some modules are referred by a short name for aesthetic reasons. In this way, the picker scheduling and rostering module is refered by the link Rostering and the truck and driver scheduling is referred as Engagement.

Once the user click in one module link the system shows a submenu, in which is possible to access the import data screen, the input data screen or the solution screen. In the Business Intelligence link it is showed the list of analysis available (that will be not discussed in this thesis because its is not relevant to the main subject).

In the Figure 2.2 we can see the import data screen of the module Routing. As in the

other modules, it shows a list of available scenarios. When one of them is selected, the inferior box is showed and then is possible to chose the files that will be used to load the data.

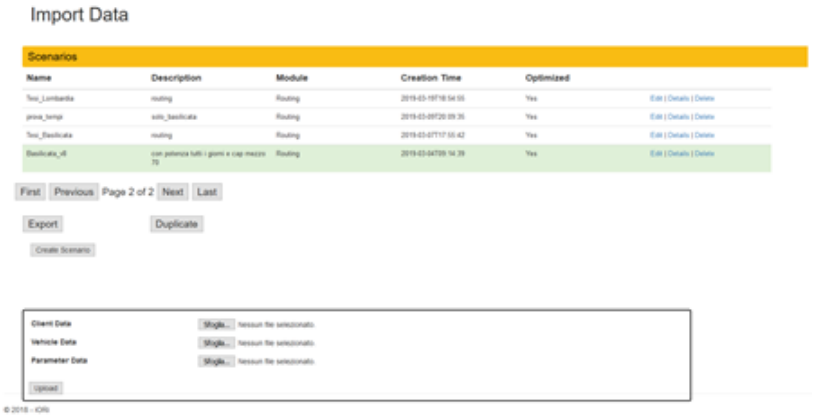


Figure 2.2: Data import screen

As can be seen, in this screen is also possible to create new scenarios, export scenario data or duplicate a scenario. This last function is particularly useful to compare different variations of a same master scenario (changing the number of available vehicles or some time windows, for example) without having to reload all the files.

The data loaded in the system can be seen in a view like the presented in the Figure 2.3. As in the import data screen, there is a table with the scenarios, and once a scenario is selected, its data is showed to the user. As the number of tables can be large and makes the visualisation of some information difficult, it is possible to collapse the exhibition of some tables to let others more evident. From this screen is also possible to edit scenario data and call the optimisation.

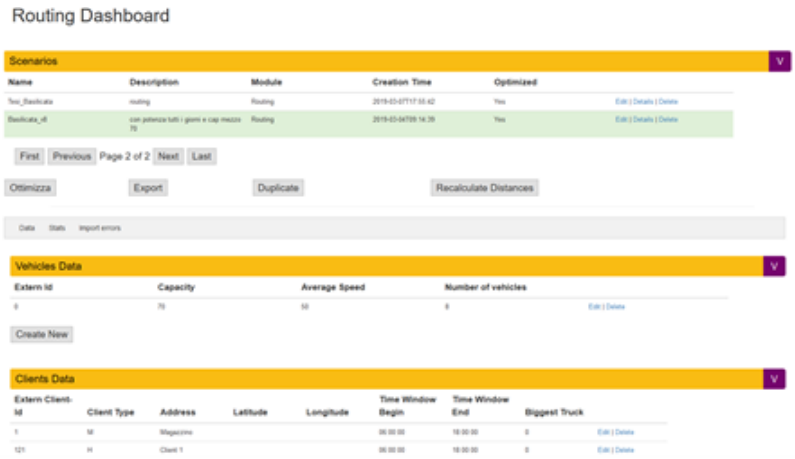


Figure 2.3: Routing dashboard tables

In the same screen described in the previous paragraph is possible to alternate to a chart

view, in which is possible to have an overview of scenario data directed to users interested in having the main information about the input (as managers or directors) without controlling or editing smaller details, like a vehicle capacity or the size of a demand. An example of this view is presented in the Figure 2.4, where there are three small charts of the Routing module.

Figure 2.4: Routing dashboard charts

One information that can be quickly obtained in observing the charts of Figure 2.4, for example, is the low availability of the clients at afternoon or night. It is also possible to notice that are not demands to be satisfied on weekend and the different capacities of each vehicle type.

Other than having insights about the input data, these charts permits to control some constraints of the problem without having to manage directly the data tables or proceeding complex calculations by hand or using an auxiliary software.

When the user finishes the data analyse or edition (when this is done), the optimisation can be called through a simple button click. The optimisation process, from this point, is totally independent of user action, and cannot be interrupted, cancelled, or restarted due to security reasons. It is not required to the user set any further parameter, like solver path, port, or number of threads, once it is fully managed by the application.

The optimisation process start with the writing of an input file, where the information loaded by the user and stored in the data base will be passed in a convenient format, easier to be read by the solver but less friendly than a worksheet. After that, the solver process is called and receive the input file as a parameter. While the solver optimise the scenario, some logs can be generated, but they are not presented to the user, first to reduce the data traffic and second because it most of time does not provide any clue about when the optimisation will finish, that is the most relevant information to the user. However, in solver code, there are some mechanisms to stop the optimisation after very long-time executions or eventual errors. In this case, this information is presented in the interface to inform the problem.

If the solver process finishes successfully, a file with the solution is generated and then loaded in the database. The scenario then is set as optimised, what warns the user about

the end of the optimisation (if the application was closed in the meantime) and permits the optimisation report to be showed.

The optimisation report, as the input, can be seen in tables or charts. In Figure 2.5 it is possible to see the report of the Routing module, where is possible to visualise the generated routes in a map, the total distance and the number of clients visited, etc...

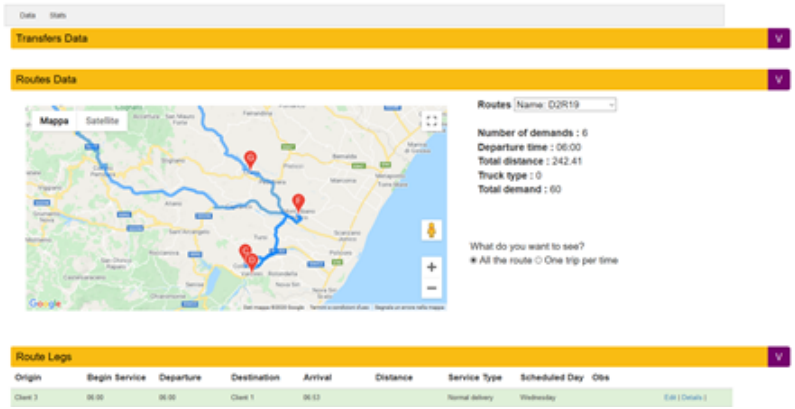


Figure 2.5: Routing solution overview

Similarly, in the chart view (vide Figure 2.6), some statistics about the solution can be viewed, like the total distance travelled, the distance by route and the utilisation of vehicle capacity in each route.

Figure 2.6: Routing solution charts

A human-friendly written optimisation report, without the charts, can be exported to spreadsheet files with a single click. This report is destinated to the personnel responsible to execute the resulting plan and to be distributed, analysed, or consulted without directly accessing the system. It can also allow the information to be reloaded in a third part software if it is interesting to the company.

As data visualisation represented a relevant concern in the software, it was not restricted

only to the optimisation modules or to represent complex data sets. In the Business Intelligence module this resource was utilised in a deeper way, with more complex chart layouts. In the ABC Analysis report, for example, we implemented a tree map chart to represent the accumulated number of requests in each class (vide Figure 2.7), as well the number of requests of each product, as can be seen on Figure 2.8.

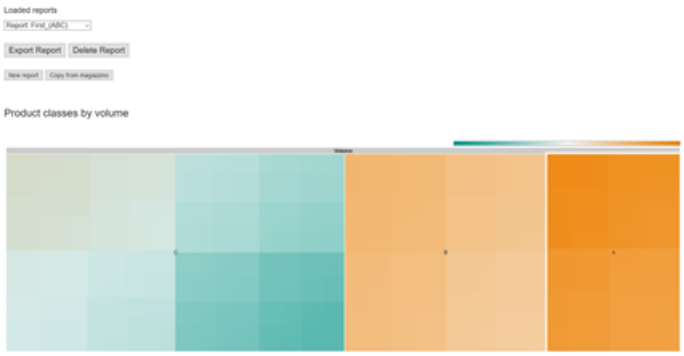


Figure 2.7: ABC analysis report (chart visualisation)

In the Figure 2.8 bellow, the products that are in A class (i.e. those more required in the warehouse) are individualised as a detail of the "item A" in the Figure 2.7. The number of details levels is not limited, so we could use this chart also to represent the prevalence of requests in each pavilion or shelf of a warehouse, producing a quick to navigate visualisation of the orders.

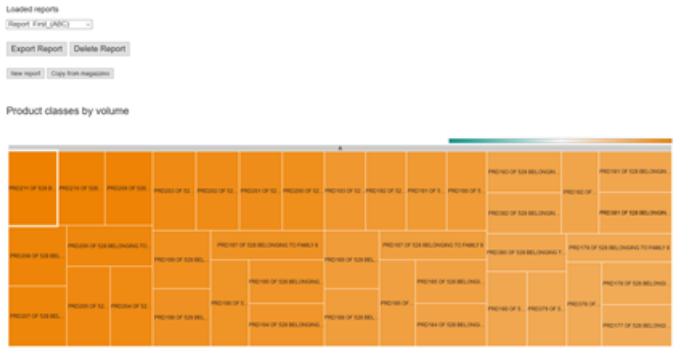


Figure 2.8: ABC analysis report (chart visualisation detailed)

Still in the context of orders statistics visualisation, we implemented a chord dependence diagram to represent the products that are most frequently required in the same order, as a part of the Basket Analysis report (vide Figure 9). In this chart, wider arcs represent stronger connections, while thinner represent weaker. The colours, by its turn, indicates which product has more requests in the connection.

Complex charts like these, more than just an aesthetic resource, are useful to highlight information that is not easily visible in a data table or in regular charts, like a histogram or

a pie chart. We aimed with these advanced tools to provide features that could allow the company understand better how itself works and in this way evaluate better its current state and the effects of its decisions.

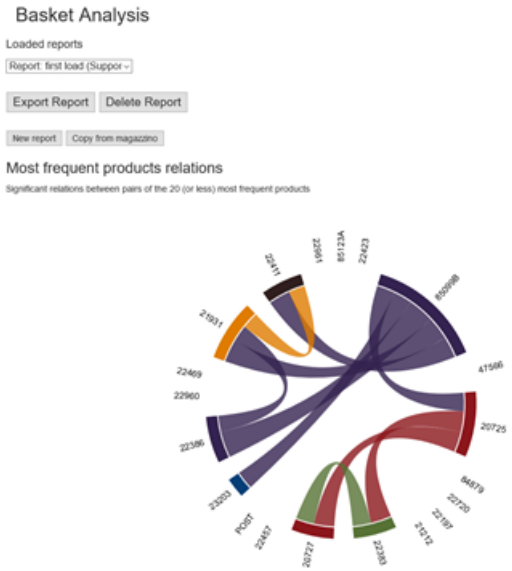


Figure 2.9: Basket analysis correlation chart

2.6 Conclusion

In this chapter we presented a general view of the company context and operation inside a healthcare supply chain and how the software was conceived and implemented to solve operational, tactical, and strategic problems related with the warehouse operation.

The system was surprisingly well received in the company, considering the fact it was not developed in a regular software house and its level of complexity. Some problem with the usability and performance were reported, mainly regarding large run times, input errors report and infeasible solutions. These problems were partially solved, enabling the system to be used in the day-to-day operations.

As future research, the objective is to improve the system reliability and robustness, to full accomplish the objective of using the system in all managerial levels without support of a specialist team. Other aspect is to provide functionalities in the dashboards to allow personalised reports, that could be saved or exported with a few clicks.

Finally, considering the algorithms, one desired new functionality would be the control of heuristic and model parameters, to provide more flexibility to the decision taker to prioritise some metrics or get quick solutions to the problems.

2.7 Bibliography

R. Aldrighetti, I. Zennaro, S. Finco, D. Battini. Healthcare supply chain simulation with disruption considerations: a case study from Northern Italy. *Global Journal of Flexible Systems Management*, 20(1):81–102, 2019.

M. Aniche, G. Bavota, C. Treude, M. A. Gerosa, A. van Deursen. Code smells for model-view-controller architectures. *Empirical Software Engineering*, 23(4):2121–2157, 2018.

M. Beaulieu, J. Roy, S. Landry. Logistics outsourcing in the healthcare sector: Lessons from a Canadian experience. *Canadian Journal of Administrative Sciences/Revue Canadienne des Sciences de l'Administration*, 35(4):635–648, 2018.

J. W. Chinneck. *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, volume 118. Springer Science & Business Media, 2007.

I. Dunning, J. Huchette, M. Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

A. Freeman, The Model/View/Controller Pattern, Apress, Berkeley, CA, 2015, pp. 527–552. doi:10.1007/978-1-4842-0394-1_27.

F. Friemann, P. Schönsleben. Reducing global supply chain risk exposure of pharmaceutical companies by further incorporating warehouse capacity planning into the strategic supply chain planning process. *Journal of Pharmaceutical Innovation*, 11(2):162–176, 2016.

P. Jalote. *An integrated approach to software engineering*. Springer Science & Business Media, 2012.

P. G. Jarrett. Logistics in the health care industry. *International Journal of Physical Distribution & Logistics Management*, 1998.

A. Leff, J. T. Rayfield, Web-application development using the model/view/controller design pattern, in: Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference, 2001, pp. 118–127. doi:10.1109/EDOC.2001.950428.

N. H. Mustaffa, A. Potter. Healthcare supply chain management in Malaysia: a case study. *Supply Chain Management: An International Journal*, 2009.

L. Nicholson, A. J. Vakharia, S. Selcuk Erenguc. Outsourcing inventory management decisions in healthcare: Models and application. *European Journal of Operational Research*, 154(1):271 – 290, 2004.

N. Shah. Pharmaceutical supply chains: key issues and strategies for optimisation. *Computers & Chemical Engineering*, 28(6):929 – 941, 2004. FOCAPO 2003 Special issue.

R. Uthayakumar, S. Priyan. Pharmaceutical supply chain and inventory management strategies: Optimization for a pharmaceutical company and a hospital. *Operations Research for Health Care*, 2(3):52 – 64, 2013.

J. Volland, A. Fügener, J. Schoenfelder, J. O. Brunner. Material logistics in hospitals: A literature review. *Omega*, 69:82 – 101, 2017.

Chapter 3

The Multi-Trip Rich Vehicle Routing Problem with Truck and Driver Scheduling

[1]

In this chapter, we present two modules of the decision support system which deal with a Multi-Trip VRP characterized by additional constraints and Truck and Driver Scheduling. The problem is solved in the software with a two-phase algorithm: the first phase consists of an Iterated Local Search metaheuristic that defines the vehicle routes, whereas the second phase invokes a mathematical model to assign trucks and drivers to the routes. The software allows, between the two phases, changes in the solution to better fit the company requirements. Computational results prove the effectiveness of the proposed method.

3.1 Introduction

Vehicle Routing Problems (VRP) are a traditional and well studied topic in Operations Research and Management Science. The process of defining efficient and convenient routes is one of the main concerns for a large number of companies, and it can lead to significant losses when it is not performed accurately. In such field, developing a good Decision Support System (DSS), capable of satisfying all requirements and covering all relevant characteristics of the real problem at hand, is not an easy task. Despite the vast literature dealing with different VRP types, finding a variant that exactly describes the operations of a company is also difficult. Most of the times, even state-of-the-art algorithms are not satisfactory in delivering good real routes, due to unrealistic assumptions, lack of information, too optimistic or de-

[1]The results of this chapter appears in: NFM. Mendes and M. Iori, A Decision Support System for a Multi-trip Vehicle Routing Problem with Trucks and Drivers Scheduling, *22nd International Conference on Enterprise Information Systems (ICEIS)*, 1(2020), pp. 339-349, 2020. (Mendes and Iori., 2020).

terministic approaches or simply because they are not flexible enough to provide alternatives when the best solution is not approved by the decision maker. Conscious of this situation, companies are always looking for alternatives that could give them a better (or more robust) approximation, of even hiring professionals or software houses to developed custom softwares that exactly fit their needs.

In this chapter, we describe a branch of a DSS that we are developing for an Italian company specialized in the storage and distribution of pharmaceutical products. This branch provides a routing plan and a driver scheduling and assignment for each route. Furthermore, it provides a set of quantitative reports that allow one to better analyze the quality and fitness of the solution obtained. The problem we solve is a Rich Multi-Trip VRP with Driver and Vehicle Scheduling, where the term "rich" implies that the problem contains a number of additional constraints with respect to the basic VRP. The problem derives indeed from the union of a VRP with Time Windows (VRPTW), see, e.g., (Solomon, 1984), and a Multi-Trip Vehicle Routing and Scheduling Problem (MTVRSP), see, e.g., (Brandao and Mercer, 1997), but also includes additional constraints defined by the distribution company. In short, the aim is to create a minimum-cost one-week routing plan to deliver products to a set of clients by using a heterogeneous fleet of vehicles based at a central depot, while satisfying clients service time windows, vehicle and drivers incompatibilities, driving regulation, and presence of intermediate depots.

The contributions of this work are multiple: we clearly describe an optimization problem derived from a real-world distribution activity; we solve the problem by means of a two-phase algorithm; we present a software architecture that allows for an intuitive and quick man-in-the-middle approach to make the algorithm fully usable within a DSS; we present a large computational evaluation on a set of realistic instances; and we discuss how the approach can be replicated to solve other difficult VRP with operational constraints.

The remainder of the chapter is organized as follows. In Section 3.2, a short review of the related literature is provided. In Section 4.3, a formal description of the problem and the company vision of it are given. Next, in Section 3.4, we describe how the system works and how the user participates to the solution process. In Section 3.5, we describe the solution approach and then, in Section 3.6, we computationally evaluate it on a case study.

3.2 Literature review

As aforementioned, the problem that we study in this chapter is a union of a VRPTW, see, e.g., Solomon (1984), and a MTVRSP, see, e.g., Brandao and Mercer (1997). The VRPTW has a broad and well-studied literature, and it is included in many MTVRSP problems. Consequently, we focus our short review on the MTVRSP literature. We refer the reader interested in VRPTW and other VRP variants to the books by Golden et al. (2008) and Toth

and Vigo (2014).

Despite the fact that they well model scenarios where short routes are common or desirable, MTVRSP are relatively new in the literature. While the traditional VRP was introduced sixty years ago by Dantzig and Ramser (1959), the multi-trip version appeared only thirty years after with Fleischmann (1990). In such article, the term used was not even"multi-trip", but "VRP with multiple vehicle uses". Other well known characteristics were also studied in that article, like time windows and heterogeneous fleet. The number of vehicles was also limited, making the use of the same vehicle several times more desirable.

Six years later Taillard et al. (1996) proposed a three phase heuristic for the problem that started by generating a pool of reasonable routes (first phase) from which a subset was selected (second phase) and then assigned to feasible workdays according to a bin packing-like procedure (third phase).

The first explicit mention to the term "truck drivers" as a part of the problem name occurred, to the best of our knowledge, in Brandao and Mercer (1997). Such paper considered the driver maximum working hours in a day as well the needed breaks during a route. Other features, like unloading time, vehicle incompatibilities with some clients and the possibility to hire additional vehicles if the available ones were not enough, were also taken into account. To solve the resulting problem, they proposed a tabu search heuristic. They found good results for instances with approximately 20 vehicles, divided into two types, and 70 clients. The work was then pursued by the same authors in Brandão and Mercer (1998), where they discussed a simplified version of an algorithm used in a real-world application, and they reported reductions of about 5% on the delivery costs.

Campbell and Savelsbergh (2004) proposed an efficient insertion heuristic to the basic Vehicle Routing and Scheduling Problem (VRSP), a MTVRSP without multi-trips, having polynomial complexity. Zäpfel and Bögl (2008) presented a heuristic method to solve an MTVRSP with possibility of drivers outsourcing. Cattaruzza et al. (2014a) proposed a memetic algorithm and an adaptation of the split procedure, see, e.g., Duhamel et al. (2011), to segment a chromosome into a MTVRP solutions. Azi et al. (2014) used an Adaptive Large Neighborhood Search (ALNS) algorithm to solve a variant of the MTVRSP problem where visits to some clients could be avoided.

Just a few papers in the literature developed exact methods to solve the MTVRSP. The first was Azi et al. (2010), that made use of a column generation approach embedded with a branch-and-cut algorithm. A similar approach was also employed by Macedo et al. (2011). A more robust method was provided by Mingozzi et al. (2013), where the authors presented two set-partitioning formulations for the problem, one having a binary variable for each route and the other a binary variable for each schedule. Their solution procedure used bounding methods to reduced the original set of routes and schedules. Hernandez et al. (2016) also used a similar strategy, but with less refinements in the space solution. In Tang et al. (2015),

a pickup and delivery problem inside an airport was modeled as a trip-chain-oriented set-partitioning, and then solved directly using CPLEX.

In 2006, a new regulation about truck drivers was created in Europe, aiming at giving better job conditions to the workers and rising the safety on the transportation activity. The EC Nº 561/2001 defined a large set of rules about drivers work regime like maximum duration of daily shifts, mandatory breaks, overtimes, minimum rest time between two shifts and weekly rests. Among the new rules we can highlight the minimum rest of 45 minutes after 4.5 hours of uninterrupted activity (that can be replaced by a 15 minutes break after a 30 minutes break within the 4.5 hours), daily driving time of nine hours (that can be extended to 10 hours two times per week), at least 11 hours of daily rest and a maximum of 56 hours of weekly driving time. This regulation entered in force in April 2007, and right after that digital tacographs became mandatory in the European Union, allowing the authorities to check, a posteriori, the drivers working times. This new scenario motivated several studies where the driver was individually considered.

The first study considering these new constraints was presented, to the best of our knowledge, by Goel (2009). In this paper, a Large Neighborhood Search (LNS) is used to create a weekly schedule where all the rest and break rules are respected. To reduce the explored solution space, the authors did not considered overtimes and the 15 minutes more 30 minutes break possibility.

Kok et al. (2010), on the other hand, was the first to consider all the rules presented in the EC Nº 561/2001 regulation and in the Directive 2002/15/EC. To solve an MTVRP incorporating such features, they proposed a restricted dynamic programing framework where clients were sequentially added to the end of partial vehicle routes. Feasibility of such additions was controlled by extra state dimensions.

A few years later, Drexl et al. (2013) proposed a two-stage method to solve a MTVRSP with the presence of relay stations where the drivers are allowed to change a vehicle. The first stage consists in solving a pickup-and-delivery problem, see, e.g., Battarra et al. (2014) and Doerner and Salazar-González (2014) with time windows and relay stations, whereas the second consists in soling a VRPTW with multiple depots.

Two papers deal with a multi-commodity MTVRSP variant where some commodities cannot be transported together. The former is by Battarra et al. (2009), who introduce the problem and proposes an adaptive guidance heuristic to solve it, whereas the latter is by Cattaruzza et al. (2014b), who describe an ILS heuristic.

Vehicle routing and scheduling problems are studied in the context of home healthcare planning by Algethami et al. (2017), that compares some operators in a genetic algorithm. In the same path Algethami et al. (2019) proposes an adaptive multiple crossover to the problem.

Some most recent studies in the MTVRSP area are those by Masmoudi et al. (2016) and

Benkebir et al. (2019). The former tests a set of metaheuristics to solve a multi-trip dial-a-ride problem, and the second proposes a hybrid method composed by a genetic algorithm and a local search for the MTVRSP. Others relevant study are He and Li (2019), where is described a two-echelon multi-trip VRP in a context of crop harvesting and Babaee Tirkolaee et al. (2019) that describes a case study of a multi-trip VRP applied to an urban waste collection. Further information about the MTVRSP can be found in Cattaruzza et al. (2016). We also refer to Lahyani et al. (2015) as a recent review of the literature on rich VRP.

3.3 Problem description

The Multi-Trip Rich Vehicle Routing Problem with Truck and Driver Scheduling, henceforth refereed to as MTRVRPTDS, describes a one-week products delivery operation. In this problem, trucks and drivers are individually assigned to each route, considering driver regulation and operational constraints. The routes are created trying to minimize the distances, and must respect the client time windows. Deliveries can be anticipated (changing from the required day to the previous day) if it is allowed by the client and useful to improve the costs of the overall plan.

Formally, let D be a set of days in a week, F a set of storage facilities, C a set of clients, T a set of trucks and W a set of drivers. Each client $c \in C$ has a demand of O_{cd} pallets, on day $d \in D$, that must be delivered inside a time window $[\alpha_c, \beta_c]$. A service time of S_{cd} minutes is required to unload the delivery to client c in day d, independently of the truck internal organization of the cargo. Each vehicle $t \in T$ is driven by a unique driver $w \in W$. Vehicles are grouped in types, on the basis of their speed V_t and capacity Q_t. Drivers have different contracts and skills. They are allowed to work at most h hours by day, at most H^*_w hours per week, and they may drive only a subset $T_w \subseteq T$ of the available vehicles, for each $w \in W$.

A unique depot is given. This is due to the fact that the company centralizes all the deliveries of a region to that depot. The depot is supplied of pharmaceutical products by a regional entity, and it is expected that all the products are available in the moment in which they are required by the clients. Each route departs from the unique depot, passes through a subset of clients, and then returns to the depot. An agreement between drivers and the company limits the maximum duration of a route in 8 hours, less than the maximum daily shift allowed by the ECD 56/2006 which is 9 hours. It is also imposed that the total weekly working time cannot exceed 40 hours, and that the daily working extra time is limited to at most 20% of the maximum daily shift.

Some vehicles cannot be sent to visit some clients (because they are too big for the road or they do not have enough power to climb a hill). This is taken into account by considering a set R_t of clients that can be reached by vehicle $t \in T$.

Each vehicle and driver can be assigned to more than one route per day. This is possible if the next route starting time is after the first route ending time plus the time required to reload the truck or the driver have a break.

Additionally to the main depots there are, in some regions, intermediate warehouses that can be supplied by exclusive routes coming from the main depots. Those intermediate warehouses receive loads that will be delivered in the next day to the clients, working like a buffer. The use of an intermediate warehouse has a cost I_f, with $f \in F$ being an intermediate warehouse and F being the set of all intermediate warehouses.

A missing point about driver regulation that is not included in the problem definition concerns the mandatory breaks that drivers should do during the day and the rests between the days. They were not considered in the software specification, because they were considered too operational and quite unpredictable. As reported by one of the employees, as delays or changes in the routes can happen, it could be hard to stop at the moment defined by the specific sequence provided by the DSS, because the driver could be in a non-safe place. Beside this, it could trigger some drivers resistance due to the fact that some of them have already their preferences on where to stop in each place they use to visit. In any case, we discuss how these constraints could be included in our approach below. Regarding the rest between days, no route should start before 5 AM. If a driver arrives late at night, the responsible manager would change the scheduling to avoid this driver to pick a too early route in the next day. This type of on-the-fly changes are usual when using DSS to solve optimization problems.

3.4 Software data flow

The usability is a very important feature considered by us to develop this software. It is commonly pointed out as one of the main factors for the success of a software in a company. A poor usability can make a DSS with advanced algorithms and analytic tools unused due to the resistance of the decision makers to learn and deal with the system complexity.

To ensure that our DSS is used by the company, we decided to create a friendly web interface to deal with the process of loading, visualizing and control of optimization inputs. More than only providing an easy-to-learn tool to run an algorithm, this interface was thought to allow the decision maker to be part of the solving process and have a clear overview of solution quality. The application described in this chapter is a part inside a larger software belonging to the same project. This module is divided into two parts, following the strategy proposed by us to solve the problem.

The flow starts with the rich VRP input load. This input consists in three files, one describing the clients, with all the information about time windows, demands, service times and vehicle restrictions; the second describing the type of vehicles available with its respective average speeds, capacity and number of units available, and finally the third with the

parameters.

The distances between the clients can be informed by the user in the client file or evaluated in the system using the LibOSM (https://github.com/Marcussacapuces91/LibOsm/) that is part of the Open Street Maps ecosystem and Lemon (https://lemon.cs.elte.hu/trac/lemon), from COIN-OR. These libraries together make possible to calculate real road distance between any pair of points that corresponds a valid address in a given region using only a local geo-spatial database. In this way, it is possible to get all the information needed without the use of Internet or accessing external web services.

After all the data are loaded (or calculated, if we consider distances), the user can start the optimization with a click. To solve this part of the problem we invoke a heuristic algorithm that takes approximately 1 to 3 minutes to converge with instances involving around 200 clients. Such heuristic is described in detail in Kramer et al. (2019), and is sketched below in the next section. The results obtained are a set of product transfers from a depot to an intermediate warehouse and a set of delivery routes departing from the main depot. Each route or transfer has a departure time and receives a vehicle type assigned to it, but not information about the driver.

Before proceeding to the next step, the decision maker can adjust the solution obtained by changing the departure time, the vehicle type assigned, the clients in the route and the visit order. After each change, a solution automatic validation is performed. If a change leads to an infeasible solution, a rollback procedure is done and the user is informed with a pop-up.

To plan an individual driver and truck to each route the user can proceed in two ways. The first way is to load route information using Excel files like in the previous phase. This method is useful when some other persons or systems created the routes, so it is possible to get some improvement by better using the resources available. The second way is to load the first phase solution as input to the second part, and this can be done with just one click. Next, the user just needs to insert the information about the drivers and the details about each vehicle.

The driver and truck assignment is calculated by solving a mathematical model using a Mixed Integer Linear Programming (MILP) solver. As in the first phase, changes in the solution provided by the system are allowed and checked, as already described. Once all these steps are finished, the solution remains stored in the software database and can be visualized in a dashboard or downloaded in an Excel file.

A last important point to highlight is the tolerance of the DSS with respect to infeasible solutions. In real problems, not satisfying all constraints is common. However, it is not interesting for a decision maker to simply have no answer after it called a solver. Considering this, we show any solution obtained at the end of optimization, treating the typical sources of infeasibility as penalizations in the objective functions, but warning the user about that. A typical situation of this type occurs when not enough trucks or drivers are available for the

daily deliveries. In this case, the model incurs in a penalty, the solution is however returned, and the decision maker knows which routes can be directly performed by the company and which ones should be postponed to the next day or given to a third-party logistic operator. The tolerance with infeasible solution is not extended to infeasibilities inserted by the user through solution edition. As aforementioned, no valid modifications are undone to avoid deteriorate a feasible or almost feasible solution.

3.5 Solution Approach

To solve the MTRVRPTDS, we use a two-phase approach. In the first phase, we use the Multi-Start Iterated Local Search (MS-ILS) metaheuristic developed by Kramer et al. (2019). This MS-ILS was originally developed to solve the same VRP we face in this chapter, but without considering multiple-trips neither the presence of a limited umber of trucks and drivers. We thus modified this algorithm in the way described below in Section 3.5.1 to fit with the new characteristics. In the second phase, the routes created in the previous phase are given in input to a new mathematical model, described in Section 3.5.2, that assigns drivers and trucks to the routes and defines the effective departing time of each route.

3.5.1 Multi-start ILS Heuristic

The subproblem solved in the first phase of the MTRVRPTDS defines a set of delivery routes and product transfers from main depots to intermediate warehouses. In this phase, trucks and drivers are not individualized, but we just define how many vehicles of each capacity and average speed are used. The deliveries must be done by satisfying the client time windows and some of them can be anticipated to the previous day at the expenses of opening an auxiliary depot. To solve this problem, we used an adapted version of the MS-ILS by Kramer et al. (2019), where a penalty is added in the cost function whenever the number of vehicles of a certain type used is greater than the number of available vehicles.

The algorithm can be briefly described as follows. At each iteration, a constructive method creates an initial route connecting the depot to a hospital (that are the clients with larger demands in our instances) and associate the largest allowed vehicle to the route. After this is done for all the hospitals, the remaining clients are inserted in the routes created following a lowest-cost-increase criterion. Time windows violations are accepted, but penalized.

After creating the initial solution, the algorithm starts the ILS loop. In this loop, a Randomized Variable Neighrborhood Descent (RVND) algorithm is used as local search procedure. The RVND selects, at each iteration, an inter-route neighborhood (from a list of four) and executes it. If the solution is not improved, then the neighborhood is removed from the list and the algorithm tries another one. Otherwise, the list is reinitialized and the algorithm tries to improve the current solution using one of three possible intra-route

neighborhoods. When the list of available inter-routes neighborhood becomes empty (i.e., after four not improvement iterations) the method stops. After the local search has been performed, a perturbation phase is invoked. In this step, a local search is chosen randomly (from a list of three) to randomly modify the solution. The ILS method is iterated until a given number of iterations without improvements is reached (in our settings, 20 iterations). The multi-start executes the ILS 20 times.

3.5.2 Mathematical Model

The subproblem solved in the second phase of the MTRVRPTDS defines which truck and driver must execute a route and defines the departure and arrival times at each client, considering the truck average speed, the service start time and the time windows. In some cases, no truck and driver are assigned to a route due to limited resources. In this case, a penalty is applied in the objective function. The penalty roughly corresponds to the cost of assigning the route to an external distribution company. We also apply penalties when a driver works more than her maximum daily working hours or compatibilities are not satisfied. The maximum overtime is modeled as a hard constraint, as well the weekly maximum working hours.

The MILP model uses the following parameters:

- DC_w - Cost of driver $w \in W$
- $DOTC_w$ - Overtime hour cost of driver $w \in W$
- PR - Penalty for route not assigned
- Γ - Total daily time (1440)
- S_c^r - Service time of client $c \in C$ in the route $r \in R$
- Δ_{ab}^r - Total distance between points a and b in the route $r \in R$
- Δ_r - Total distance on route $r \in R$
- S_r^* - Total service time on route $r \in R$
- Ω_t - Truck $t \in T$ average speed
- (α_c, β_c) - Limits of time windows of client $c \in C$
- E_r - Set of routes segments in the route $r \in R$
- C_r - Set of clients in the route $r \in R$
- D_r - Day in which the route $r \in R$ is executed

- L - Truck loading time
- R_d - Routes in the day $d \in D$
- H_w - Daily maximum working hours of driver $w \in W$
- H_w^* - Weekly maximum working hours of driver $w \in W$
- Ξ'_{wc} - Equals to 1 if it is strictly forbidden to assign the driver $w \in W$ to routes containing the client c, equals to 0 otherwise
- Ξ^+_{tc} - Equals to 1 if it is strictly forbidden to assign the truck $t \in T$ to routes containing the client c, equals to 0 otherwise
- Ξ^*_{tw} - Equals to 1 if it is strictly forbidden to assign the driver $w \in W$ and the truck $t \in T$ to the same route, equals to 0 otherwise
- Φ'_{wc} - Equals to 1 if it is not desirable to assign the driver $w \in W$ to routes containing the client c, equals to 0 otherwise
- Φ^+_{tc} - Equals to 1 if it is not desirable to assign the truck $t \in T$ to routes containing the client c, equals to 0 otherwise
- Φ^*_{tw} - Equals to 1 if it is not desirable to assign the driver $w \in W$ and the truck $t \in T$ to the same route, equals to 0 otherwise
- dep - Depot
- $f, l \in C$ - First and last clients of a route

The equations (3.1) to (3.33) defines the model proposed. All the variables that represents time instants or intervals as well the temporal parameters are expressed in minutes. Every time the characters f and l appear as a client-index in the model, they represent the first and last clients of the route.

$$Min \sum_{w \in W} \sum_{d \in D} (\zeta_{wd} * DC_w + DOTC_w * \varrho_{wd}) + PR * \mu_r + penComp \tag{3.1}$$

Subject to

$$\sum_{w \in W} x_{wr} + \mu_r = 1 \quad r \in R \tag{3.2}$$

$$\sum_{i \in T} y_{ir} + \mu_r = 1 \quad r \in R \tag{3.3}$$

$$max(\alpha_c, \theta_{cr}) \leq \eta_{ir} \quad \forall r \in R \quad c \in C_r \tag{3.4}$$

$$\eta_{cr} + S^r_c \leq \phi_{cr} + \mu_r * \Gamma \quad \forall r \in R, \quad c \in C_r \tag{3.5}$$

$$\phi_{c1,r} + \sum_{i \in T} y_{ir} * \Delta^r_{c1,c2}/\Omega_i \leq \theta_{c2,r} \quad \forall r \in R, \quad (c1, c2) \in E_r \tag{3.6}$$

$$\phi_{dep,r} - (\eta_{ls} + S^s_l + \sum_{i \in T} y_{is} * \Delta^s_{l,dep}/\Omega_i) - L \geq (\gamma_{rs} - 1) * \Gamma \quad \forall r, s \in R | D_r \neq D_s \tag{3.7}$$

$$x_{wr} + x_{ws} \leq \gamma_{rs} + \gamma_{sr} + 1 \quad \forall r, s \in R, \ w \in W \tag{3.8}$$

$$y_{ir} + y_{is} \leq \gamma_{rs} + \gamma_{sr} + 1 \quad \forall r, s \in R, \ i \in T \tag{3.9}$$

$$\sum_{r \in R_d} x_{wr} \leq |R| * \zeta_{wd} \quad \forall w \in W, \quad d \in D \tag{3.10}$$

$$\pi_{wd} \leq \phi_{fr} + \Gamma * (1 - x_{wr}) \quad \forall w \in W, \quad d \in D \ \ , r \in R_d \tag{3.11}$$

$$\epsilon_{wd} + \Gamma * (1 - x_{wr}) \geq \phi_{lr} + \sum_{i \in T} y_{ir} * \Delta^r_{l,dep}/\Omega_i \quad \forall w \in W, \forall \ \ d \in D \ \ , r \in R_d \tag{3.12}$$

$$\epsilon_{wd} - \pi_{wd} \leq H_w + \varrho_{wd} \quad \forall w \in W, \quad d \in D \tag{3.13}$$

$$\sum_{d \in D} (\epsilon_{wd} - \pi_{wd}) \leq H^*_w \quad \forall w \in W \tag{3.14}$$

$$x_{wr} \leq (1 - \Xi'_{wc}) + (1 - \Xi'_{wc}) * p'_{wc} - \Phi'_{wc} \quad \forall w \in W, \quad c \in C_r \tag{3.15}$$

$$y_{ir} \leq (1 - \Xi^+_{ic}) + (1 - \Xi^+_{ic}) * p^+_{ic} - \Phi^+_{ic} \quad \forall i \in T \ \ , c \in C_r \tag{3.16}$$

$$x_{wr} + y_{ir} \leq (2 - xi^*_{iw}) + (2 - \Xi^*_{iw}) * p^*_{iw} - 2 * \Phi^*_{iw} \quad \forall w \in W, i \in T, r \in R \tag{3.17}$$

$$penComp = M * (\sum_{i \in T} \sum_{c \in C} p^+_{ic} + \sum_{w \in W} \sum_{c \in C} p'_{wc} + \sum_{w \in W} \sum_{i \in T} p^*_{iw}) \tag{3.18}$$

$$\phi_{dep,r} + \sum_{i \in T} y_{ir} * (\Delta_r - \Delta^r_{l,dep})/\Omega_i \quad + S^*_r \leq \theta_{lr} + S^r_l + \Gamma * \mu_r \;\; \forall r \in R \tag{3.19}$$

$$\phi_{dep,r} + \sum_{i \in T} y_{ir} * (\Delta^r_{dep,f})/\Omega_r \quad \geq \alpha_f - \Gamma * \mu_r \quad \forall r \in R \tag{3.20}$$

$$x_{wr} \in 0, 1 \;\; \forall w \in W, r \in R \tag{3.21}$$

$$y_{tr} \in 0, 1 \;\; \forall t \in T, r \in R \tag{3.22}$$

$$\mu_r \in 0, 1 \;\; \forall r \in R \tag{3.23}$$

$$0 \leq \eta_{cr} \leq \beta_c - S^r_c \quad \forall r \in R, c \in C_r \tag{3.24}$$

$$0 \leq \theta_{cr} \leq \beta_c - S^r_c \quad \forall r \in R, c \in C_r \tag{3.25}$$

$$0 \leq \phi_{cr} \leq \beta_c \quad \forall r \in R, c \in C_r \tag{3.26}$$

$$\pi_{wd}, \epsilon_{wd} \geq 0 \quad \forall w \in W, d \in D \tag{3.27}$$

$$0 \leq \varrho_{wd} \leq 0.2 * H_w \quad \forall w \in W, d \in D \tag{3.28}$$

$$p'_{wc} \geq 0 \quad \forall w \in W, c \in C \tag{3.29}$$

$$p^+_{tc} \geq 0 \quad \forall t \in T, c \in C \tag{3.30}$$

$$p^*_{tw} \geq 0 \quad \forall t \in T, w \in W \tag{3.31}$$

$$\zeta_{wd} \in 0, 1 \quad \forall w \in W, d \in D \tag{3.32}$$

$$\gamma_{sr} \in 0, 1 \quad \forall r, s \in R \tag{3.33}$$

The binary variable x_{wr} defines if the driver $w \in W$ is assigned to the route $r \in R$.

Similarly, y_{ir} is a binary variable that defines if the truck $i \in T$ is assigned to the route $r \in R$. In the model implementation, no y variable is created when the truck capacity is lower than the total demand in the route. The variables θ_{cr}, η_{cr} and ϕ_{cr} define, respectively, the arrival time, service begin time and departure time for client $c \in C$ and route $r \in R$. The binary variable γ_{rs} defines if the routes $r, s \in R$ can be assigned to the same driver and truck. Routes in different days have no restrictions of this kind. The variables π_{wd} and ϵ_{wd} represent the first departure and last arrival time for driver $w \in W$ in day $d \in D$. The variables ζ_{wd} define that the driver $w \in W$ was assigned to a route in the day d.

Some variables are defined to describe situations where penalties must be applied. The variable μ_r represents a non-executed route, ϱ_{wd} represents instead the overtime of driver $w \in W$ in day $d \in D$. The variables, p'_{wc}, p^{+}_{ic} and p^{*}_{tw} are, respectively, binary variables that represent non-desirable driver/client, truck/client and truck/driver assignments. Finally, *penComp* simply sums up all the non-desirable assignment penalties.

Constraints (3.2) and (3.3) define that, in order to execute a route, we should assign a driver and a truck, otherwise the μ_r variable corresponding to that route would be activated. Constraints (3.4) to (3.6) define the minimum begin service time, client departure time and client arrival time, respectively. In (3.7), we verify if it is possible to assign the same route to the same pair driver/client. The two following constraints avoid or permit it, according to the value of variables γ_{rs} and γ_{sr}. Constraints (3.10) check if a driver is used in the day. In constraints (3.11) to (3.14), the driver working hours (including eventual pauses) are calculated and limited. Constraints (3.15) to (3.18) check the assignment incompatibilities. Finally, (3.19) defines a lower bound to arrive in the last client and (3.20) defines an upper bound for the departure of a route. The remaining constraints ensure variable domains.

3.6 Case study

In this section, we present the computational results that we obtained on a case study. The aim of the tests we performed was to evaluate the DSS performance in finding good solutions. We used a PC equipped with a processor Intel Core i5-7200 with 2.5 GHz, Windows 10 and 8GB of RAM. The heuristic was implemented using C++ and the model using JuMP library of Julia language. To solve the model, we used the IBM MILP solver CPLEX 12.8.

The instances we used are taken from a realistic scenario originating in the Italian region of Basilicata. All the instances have some common characteristics, such as 187 clients, 2 types of vehicles, truck average speeds (40 km/h for the larger vehicle type and 60 km/h for the smaller vehicle type), the daily demands, the assignment resctrictions, 8 hours of maximum shift duration, and 60 minutes between two consecutive routes assigned to the same driver/truck. In Table 3.1 we report some details on number of customers and daily demands.

Table 3.1: Number of clients and total demands per day

Day	Mon	Tue	Wed	Thu	Fri	Sat
N. of clients	37	37	44	38	51	2
Total demand	530	490	560	540	630	60

We created instances by attempting variations in the number of vehicles of each type, capacity of vehicles, time window size and maximum number of clients per route. All those variations generated a total of 40 instances, divided in 5 blocks of 8 instances each. All tests were executed like in a standard use of the software, as described in Section 3.4. To give a user perspective of the results we limited the maximum run time of the MILP solver to 30 minutes.

Table 3.2 summarizes the main results we obtained. The table reports the identification number of each instance (ID), the truck capacity (TC), the time window size (TW), the maximum number of clients per route (MCR) and the number of trucks per type (N. of trucks).

For what the concerns the results obtained by the optimization method, we highlight in column NR the number of routes generated by the MS-ILS heuristic adopted in the first phase of our algorithm, and in column Km the total distance of such routes. The first phase required between one and two minutes to solve any of the instances in the table. We could not find a clear correlation between instance configuration and run time of the algorithm. Regarding the solution quality, we observe that the number of routes generated by the first phase algorithm does not have a significant correlation with number of vehicles used and total distance (correlations -0.008 and 0.03, respectively). Even in instances with a total of 10 trucks the number of routes does not change significantly. On the other hand, fleet total capacity creates a larger variation on number of routes as well as the total distance run (correlations -0.73 and -0.71, respectively). As we can see, the number of routes and total distance grow almost equally as the fleet capacity reduce.

Average distance by route is 156 km, with a small standard deviation of only 4.7 km. This means all routes can be traveled in less than four hours, even with the slowest vehicle. This is an advantage in small-sized time window scenarios and makes the problem regarding breaks along the day less relevant.

In the second phase, the model was able to find assignments to all routes for about 30% of the instances. In another 42% of instances, only one or two routes were not assigned. In a real life operation this kind of solutions is not a major concern if visualized in advance. The decision maker can improve these solutions by contacting an external truck and driver, or delaying some deliveries.

However, cases where a higher number of non-assigned routes (as for instances 1 and 2 in

Table 3.2: Instance variable parameters and main data about obtained solutions. Abbreviations : TC - Truck Capacity, TW - Time Window, MCR - Maximum clients per route, NR - Number of routes, Time - Model solving run time, UB - Model objective function, LB - Lower bound, NAR - Non-assigned routes

	Parameters				Results						
ID	TC	TW	MCR	N. of trucks	NR	Km	Time	UB	LB	Gap	NAR
1A	70/60	6-18	6	3 - 5	48	7635	1845	15435	15084	2.32	3
2A				2 - 6	47	7610	1857	15469	15133	2.2	3
3A				4 - 4	48	7587	1837	15435	15084	2.3	3
4A				5 - 3	49	7699	1813	5522	5176	6.7	1
5A				6 - 2	49	7551	1857	10522	10084	4.3	2
6A				7 - 3	49	7615	1839	15380	15107	1.8	3
7A				3 - 7	49	7632	1901	10380	10084	2.9	2
8A				5 - 5	49	7632	1833	10380	10084	2.9	2
1B	70/40	6-18	6	3 - 5	58	9197	1834	1198	751	5.9	0
2B				2 - 6	60	9388	1907	11251	5684	97.9	2
3B				4 - 4	58	9282	1833	5618	5212	7.8	1
4B				5 - 3	59	9163	1831	5671	5196	9.1	1
5B				6 - 2	56	9109	1814	671	212	216.5	0
6B				7 - 3	58	9512	1849	476	210	126.6	0
7B				3 - 7	58	9167	1839	1676	1374	21.2	0
8B				5 - 5	58	9237	1832	5444	5105	6.6	1
1C	50/40	6-18	6	3 - 5	69	10179	1814	1337	814	64.2	0
2C				2 - 6	69	10254	1812	6337	684	826.4	1
3C				4 - 4	68	10131	1812	1241	127	87.7	0
4C				5 - 3	70	10365	1813	1337	120	101.4	0
5C				6 - 2	70	10365	1813	1390	117	108.8	0
6C				7 - 3	69	10239	1812	440	84	423.8	0
7C				3 - 7	69	10402	1813	551	110	441.0	0
8C				5 - 5	69	10383	1838	508	84	504.7	0
1D	70/60	7-17	8	5 - 3	46	7409	1848	11284	10748	5.0	2
2D				2 - 6	47	7314	1833	11284	10748	5.0	2
3D				4 - 4	47	7359	115	11284	11284	0	2
4D				5 - 3	46	7402	106	16251	16251	0	3
5D				6 - 2	47	7338	1840	16347	15148	7.9	3
6D				7 - 3	46	7442	355	10476	10476	0	2
7D				3 - 7	47	7466	133	11076	11076	0	2
8D				5 - 5	47	7454	194	5476	5476	0	1
1E	70/60	7 - 13 and 11 - 18	8	5 - 3	46	7307	48	31205	31205	0	6
2E				2 - 6	47	7456	54	36453	34653	0	7
3E				4 - 4	46	7336	54	16559	16559	0	3
4E				5 - 3	47	7520	56	16400	16400	0	2
5E				6 - 2	46	7476	67	11443	11443	0	2
6E				7 - 3	47	7465	67	15637	15637	0	3
7E				3 - 7	47	7444	70	21301	21301	0	4
8E				5 - 5	46	7635	69	733	733	0	0

block E) are more critical. Those cases could be caused by problems like deliveries imbalance, non appropriated fleet size or worse, a bad warehouse location. On the other hand, it could represent a lack o efficiency of the algorithm in building routes and schedules with the available resources, which can be verified with a deeper solution analysis.

Table 3.2 also highlights a low number of instances solved to optimality (14 out of 40) and some large gaps. The gap increases when the fleet capacity is reduced and the time windows get tighter. As the gaps were not directly connected with the quality of the solutions, we looked for another factors that could be interfering in the convergence of model solving. To investigate changes that could provide a better performance in the proposed method, we tested a modified version of the model in which variable ζ_{wd} as well constraints 3.10 were removed.

The results that we obtained with this simplified model are shown in Table 3.3. The changes we made on the model were useful in improving the solution convergence, aa all the instances were solved to the proven optimality. All but one of instances were solved in less than one minute, and in many cases the gap between the objective functions found on regular and modified versions were lower than 10%. The numbers of non-assigned routes in this model version were the same as those found with the original model. This makes us conclude that the simplified model is a good compromise between the representation of the real-world problem and the need fo a quick and effective solution convergence.

Table 3.3: Results obtained solving the model without drivers fixed costs. Abbreviations: Time - Model solving run time in seconds, UB - Model objective function, LB - Lower bound, NAR - Non-assigned routes

	Results				
ID	Time	UB	LB	Gap	NAR
A1	16	15000	15000	0	3
A2	15	15000	15000	0	3
A3	13	15000	15000	0	3
A4	15	5000	5000	0	1
A5	14	10000	10000	0	2
A6	15	15000	15000	0	3
A7	14	10000	10000	0	2
A8	15	10000	10000	0	2
B1	20	600	600	0	0
B2	20	10600	10600	0	2
B3	16	5000	5000	0	1
B4	23	5000	5000	0	1
B5	16	600	600	0	0
B6	22	0	0	0	0
B7	19	1200	1200	0	0
B8	14	5000	5000	0	1
C1	21	600	600	0	0
C2	35	5600	5600	0	1
C3	32	0	0	0	0
C4	32	0	0	0	0
C5	211	0	0	0	0
C6	18	0	0	0	0
C7	17	0	0	0	0
C8	22	0	0	0	0
D1	20	10600	10600	0	2
D2	17	10600	10600	0	2
D3	28	10600	10600	0	2
D4	37	15600	15600	0	3
D5	17	15600	15600	0	3
D6	16	10000	10000	0	2
D7	22	10600	10600	0	2
D8	25	5000	5000	0	1
E1	15	30600	30600	0	6
E2	25	35600	35000	0	7
E3	26	15600	15600	0	3
E4	27	15600	15600	0	2
E5	72	10600	10600	0	2
E6	18	15000	15000	0	3
E7	14	20600	20600	0	4
E8	15	0	0	0	0

3.7 Conclusions

In this chapter, we presented a decision support system to help decision makers in the solution of real cases of a Multi-Trip Rich Vehicle Routing Problem with Truck and Driver Scheduling, a problem where good delivery routes need to be created and then matched with the available trucks and drivers. We proposed a two-phase heuristic procedure, in which the first phase is an adaptation of a metaheuristic from the literature, and the second phase consists of a mathematical model.

Extensive computational experiments were performed on realistic instances. We could observe that the system had troubles in identifying good solutions in very restricted scenarios, but it could consistently produce good quality solutions in other reasonable scenarios. For such scenarios, we could also note that the algorithm had a regular performance behavior, and this is an important feature to make the user trust the software. The run time was most of mostly low, satisfying the requirements of the system without compromising the solution qualities.

Future research directions will be concentrated on adapting and testing the current approach in more flexible and general scenarios. For example, when deliveries can be done in the next day, vehicles or drivers are not available in some days of the week, or when different truck average speeds must be used depending on the fact that the vehicle is in an urban area or not. Considering the user experience, we plan to make visible to the decision maker data about road blocks, tolls, and information on client satisfaction, to help her in the evaluation of eventual route changes. A synchronization with the warehouse operation software is also being evaluated to improve the allocation of workers to recover products and load trucks. We also plan to replace the mathematical model with a quick and effective metaheuristic, so as to be able to provide in quick time good-quality problem solutions.

3.8 Bibliography

H. Algethami, D. Landa-Silva, A. Martinez-Gavara, Selecting genetic operators to maximise preference satisfaction in a workforce scheduling and routing problem, in: Proceedings of the 6th International Conference on Operations Research and Enterprise Systems, 2017, pp. 416–423.

H. Algethami, A. Martínez-Gavara, D. Landa-Silva. Adaptive multiple crossover genetic algorithm to solve workforce scheduling and routing problem. *Journal of Heuristics*, 25(4-5):753–792, 2019.

N. Azi, M. Gendreau, J.-Y. Potvin. An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research*, 202(3):756–763, 2010.

N. Azi, M. Gendreau, J.-Y. Potvin. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, 41:167–173, 2014.

E. Babaee Tirkolaee, P. Abbasian, M. Soltani, S. A. Ghaffarian. Developing an applied algorithm for multi-trip vehicle routing problem with time windows in urban waste collection: A case study. *Waste Management & Research*, 37(1_suppl):4–13, 2019.

M. Battarra, J.-F. Cordeau, M. Iori, Pickup-and-delivery problems for goods transportation, in: P. Toth, D. Vigo (Eds.), The Vehicle Routing Problem, Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, 2014, pp. 161–191.

M. Battarra, M. Monaci, D. Vigo. An adaptive guidance approach for the heuristic solution of a minimum multiple trip vehicle routing problem. *Computers & Operations Research*, 36(11):3041 – 3050, 2009.

N. Benkebir, M. L. Pouliquen, J.-F. Trévien, A. Bounceur, R. Euler, E. Pardiac, M. Sevaux. On a multi-trip vehicle routing problem with time windows integrating European and French driver regulations. *Journal on Vehicle Routing Algorithms*, 2(1):55–74, 2019.

J. Brandão, A. Mercer. A tabu search algorithm for the multi-trip vehicle routing and scheduling problem. *European Journal of Operational Research*, 100(1):180–191, 1997.

J. Brandao, A. Mercer. The multi-trip vehicle routing problem. *Journal of the Operational Research Society*, 49(8):799–805, 1998.

A. M. Campbell, M. Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38(3):369–378, 2004.

D. Cattaruzza, N. Absi, D. Feillet. Vehicle routing problems with multiple trips. *4OR*, 14(3):223–259, 2016.

D. Cattaruzza, N. Absi, D. Feillet, T. Vidal. A memetic algorithm for the multi trip vehicle routing problem. *European Journal of Operational Research*, 236(3):833–848, 2014a.

D. Cattaruzza, N. Absi, D. Feillet, D. Vigo. An iterated local search for the multi-commodity multi-trip vehicle routing problem with time windows. *Computers & Operations Research*, 51:257 – 267, 2014b.

G. B. Dantzig, J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

K. Doerner, J. Salazar-González, Pickup-and-delivery problems for people transportation, in: P. Toth, D. Vigo (Eds.), The Vehicle Routing Problem, Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, 2014, pp. 161–191.

M. Drexl, J. Rieck, T. Sigl, B. Press. Simultaneous vehicle and crew routing and scheduling for partial- and full-load long-distance road transport. *Business Research*, 6(2):242–264, 2013.

C. Duhamel, P. Lacomme, C. Prodhon. Efficient frameworks for greedy split and new depth first search split procedures for routing problems. *Computers & Operations Research*, 38(4):723–739, 2011.

B. Fleischmann. The vehicle routing problem with multiple use of vehicles. *Fachbereich Wirtschaftswissenschaften, Universität Hamburg*, 1990.

A. Goel. Vehicle scheduling and routing with drivers' working hours. *Transportation Science*, 43(1):17–26, 2009.

B. Golden, S. Raghavan, E. Wasil (Eds.). *The Vehicle Routing Problem: Latest Advances and New Challenges*, Operations Research/Computer Science Interfaces Series. Springer, 2008.

P. He, J. Li. The two-echelon multi-trip vehicle routing problem with dynamic satellites for crop harvesting and transportation. *Applied Soft Computing*, 77:387–398, 2019.

F. Hernandez, D. Feillet, R. Giroudeau, O. Naud. Branch-and-price algorithms for the solution of the multi-trip vehicle routing problem with time windows. *European Journal of Operational Research*, 249(2):551–559, 2016.

A. L. Kok, C. M. Meyer, H. Kopfer, J. M. J. Schutten. A dynamic programming heuristic for the vehicle routing problem with time windows and european community social legislation. *Transportation Science*, 44(4):442–454, 2010.

R. Kramer, J.-F. Cordeau, M. Iori. Rich vehicle routing with auxiliary depots and anticipated deliveries: An application to pharmaceutical distribution. *Transportation Research Part E: Logistics and Transportation Review*, 129:162–174, 2019.

R. Lahyani, M. Khemakhem, F. Semet. Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research*, 241(1):1–14, 2015.

R. Macedo, C. Alves, J. V. de Carvalho, F. Clautiaux, S. Hanafi. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. *European Journal of Operational Research*, 214(3):536 – 545, 2011.

M. A. Masmoudi, M. Hosny, K. Braekers, A. Dammak. Three effective metaheuristics to solve the multi-depot multi-trip heterogeneous dial-a-ride problem. *Transportation Research Part E: Logistics and Transportation Review*, 96:60 – 80, 2016.

N. F. M. Mendes, M. Iori., A decision support system for a multi-trip vehicle routing problem with trucks and drivers scheduling, in: Proceedings of the 22nd International Conference on Enterprise Information Systems - Volume 1: ICEIS„ INSTICC, SciTePress, 2020, pp. 339–349. doi:10.5220/0009364403390349.

A. Mingozzi, R. Roberti, P. Toth. An exact algorithm for the multitrip vehicle routing problem. *INFORMS Journal on Computing*, 25(2):193–207, 2013.

M. M. Solomon. *Vehicle routing and scheduling with time window constraints: Models and algorithms (heuristics)*, 1984.

É. D. Taillard, G. Laporte, M. Gendreau. Vehicle routeing with multiple use of vehicles. *Journal of the Operational Research Society*, 47(8):1065–1070, 1996.

J. Tang, Y. Yu, J. Li. An exact algorithm for the multi-trip vehicle routing and scheduling problem of pickup and delivery of customers to the airport. *Transportation Research Part E: Logistics and Transportation Review*, 73:114–132, 2015.

P. Toth, D. Vigo (Eds.). *Vehicle Routing: Problems, methods, and applications*. SIAM, Philadelphia, 2nd edition, 2014.

G. Zäpfel, M. Bögl. Multi-period vehicle routing and crew scheduling with outsourcing options. *International Journal of Production Economics*, 113(2):980–996, 2008.

Chapter 4

An Iterated Local Search for a Pharmaceutical Storage Location Assignment Problem with Prohibition and Isolation Constraints

In healthcare supply chain, centralised warehouses are used to store large amounts of products close to hospitals and pharmacies in order to avoid shortages and reduce storage costs. To reach these objective, the warehouses need to have efficient order retrieval and dispatch procedures, as well as a storage allocation policy that allows to keep the items safe. Considering this scenario, we present a Storage Location Assignment Problem with Prohibition and Isolation Constraints, that models the targets and restrictions of a storage policy in a pharmaceutical warehouse. In this problem, we look for the minimisation of the total distance travelled by the order pickers to recover all products required in a set of orders. We propose an Iterated Local Search algorithm to solve the problem, and present numerical experiments based on simulated data. The results show a relevant improvement with respect to a greedy full turnover procedure commonly adopted in real life operations.

4.1 Introduction

Healthcare services are strongly sensible to equipment or medicine shortages, as they could cause attendance delays and interruptions and consequently put patient lives at risk. Traditionally, the main approach to avoid this problem was the use of high inventory levels and constant item replenishment Uthayakumar and Priyan (2013) Aldrighetti et al. (2019).

However, this solution has been often considered expensive and hard to be managed, as it requires large dedicated spaces into facilities and workload of healthcare personnel Volland et al. (2017).

Nowadays, more efficient approaches have been adopted, as acquisitions through *Group Purchasing Organisations* (GPO) and wholesalers. One of the most successful approaches is the sharing of centralised warehouses to store together products to be distributed to different customers located in the same geographical area. Centralised warehouses are particularly useful because they allow healthcare facilities to share a common structure to store a large volume of products and receive them quickly when they are needed. This enables a constant material flow, a reduction in the personnel costs, a reduced storage space in the customer facility and a lower work burden over healthcare workers.

The advantages described above are highly dependent on the warehouse reliability and capability of delivering the ordered products in the short terms defined by the customers. This reliability is, in turn, a direct result of an efficient warehouse internal organisation, which requires a good *storage location policy*.

A storage location policy is a general strategy to assign *Stock Keeping Units* (SKU) to storage positions inside a warehouse. It aims to optimising a metric (e.g. total time or distance travelled to store and retrieve SKUs, congestion, space utilisation, pickers ergonomic), while considering issues like product re-allocations efforts, demand oscillation, picking precedence and storage restrictions.

The metric commonly adopted to evaluate the quality of these policies is the distance travelled by the pickers to retrieve all products in a set of orders. This metric is particularly relevant because picking operations accounts for around 35% of the total warehouse operational costs (Wang et al., 2020) and the time/energy spent to reach a product is a waste of resources that must be minimised. In other contexts, the evaluation can also consider issues like congestion, picker ergonomic, product storage conditions and total space utilisation, that can also lower the warehouse operation efficiency.

In this chapter, we describe a problem originated from a real life operation of a pharmaceutical product distributor. It consists in the optimisation of a dedicated storage allocation policy in a picker-to-parts warehouse, i.e. a warehouse where each product has a fixed/dedicated position and pickers travel until product location to retrieve order items. Most specifically, we deal with a *Storage Location Assignment Problem with Prohibition and Isolation Constraints* (SLAP-PIC). In this problem, some products can not be assigned to some locations due to reasons like refrigeration or ventilation, and some other products need to be isolated in specific areas due to toxicity or contamination concerns. Furthermore, the problem considers a flexible warehouse layout configuration, with shelves not necessarily grouped on blocks or in a unique pavilion.

This study provides two main contributions. The first consists in defining a framework to

represent general warehouses and to extract the relevant information from this representation. The second is an *Iterated Local Search* (ILS) algorithm, that takes a set of orders and a warehouse layout in input and returns a list of assignments of products to locations that minimises the total distance travelled to fulfill the orders.

The remainder of the chapter is organised as follows: in Section 4.2, a broad literature review is presented; Section 4.3 provides a detailed problem description; Section 4.4 presents all the data processing done before the optimisation starts; Section 4.5 describes the ILS algorithm; Section 4.6 describes the numeric experiments carried out. In Section 4.7 the results are presented and then the conclusions are drawn in Section 4.8.

4.2 Literature Review

The *Storage Location Assignment Problem* (SLAP) is a generalisation of the well know Assignment Problem in which the objective function is a complex function that usually depends of several factors, like warehouse layout, picking policy, picker routing policy and order batching (Dijkstra and Roodbergen, 2017). It aims to optimise some warehouse metrics like shipping time, equipment downtime, on time delivery, delivery accuracy, product damage, storage cost, labour costs, throughput, turnover and picking productivity (Staudt et al. (2015), Reyes et al. (2019)). The two most common metrics in the literature are picking travel time and travel distance (Reyes et al., 2019), which both require to solve special cases of either the *Travelling Salesman Problem* (TSP) or the *Vehicle Routing Problem (VRP)*, according to the presence of picker capacity constraints or to the simultaneous use of multiple pickers. Using the picker travel distance has as advantage an easier evaluation, as it does not requires to deal with issues like congestion and items handling/sorting. Additionally, there are no concerns about picker average speed or searching and handling delays.

In this context, several methods to optimise order picking routes have been proposed, both inside SLAP variants studies or in independent researches. As pointed out in Dijkstra and Roodbergen (2017), routing problems in warehouses can be seen as special case of the Steiner Travelling Salesman Problem, that in some layouts can be solved to optimality (Ratliff and Rosenthal (1983), Lu et al. (2016) and Scholz et al. (2016), Cambazard and Catusse (2018)) but in general layouts is mostly solved using heuristics (De Santis et al. (2018), Chen et al. (2019), Roodbergen and Koster (2001), Theys et al. (2010)). The exact methods cited, however, are used to solve problems with already defined warehouse allocations (as can be seen also in Gu et al. (2007) and Reyes et al. (2019)) and they are mostly algorithms based on a graph theoretic algorithm for single-block warehouses Lu et al. (2016). A noticeable exception for a joint storage assignment and routing exact optimisation strategy is presented in Bolaños Zuñiga et al. (2020), but the proposed model reaches optimality only on small instances.

Nonetheless, it is known that in real life operations pickers tend to deviate from optimal or non-intuitive routes (Elbert et al., 2017). To simulate this behaviour, many studies consider simple heuristics for picker routing as return point method, largest gap method, returning point or S-shape routing (De Koster et al. (2007), Elbert et al. (2017), Kim et al. (2020)). These heuristics also simplify order picking evaluation as they quickly allow the computation of travelling distances in deterministic problems or the evaluation expected travelling distances in stochastic problems (Dijkstra and Roodbergen, 2017). For the interested reader, an extensive analysis of these algorithms is presented in De Koster et al. (2007).

It is important to notice that the routing method efficiency is influenced by the warehouse layout. In both SLAP and picker routing problems, the warehouses can have a single block or multiple blocks. A block is defined as a set of parallel shelves with tight corridors among them (*aisles*), from where it is possible to pick products located in the two shelves at their borders. The connection between these corridors (and consequently between different pairs of shelves) is called *cross-aisle*.

In the same sense, SLAP literature cites frequently two types of picking policies: picker-to-parts and parts-to-picker. In a picker-to-parts warehouse, as those considered in SLAP-PIC, the picker receives an item list and visits each item position and transport the items to an accumulation/expedition point. Conversely, in a parts-to-picker system, an automated storage and retrieve system, composed by one or more automated guided vehicles, picks ordered items and delivers them in the place where they will be prepared to be dispatched. We can also highlight the existence of the pick-and-pass system (also called progressive zoning system Pan et al. (2015)), in which each picker is responsible to retrieve a specific subset of products in an order and then deliver the incomplete order to the next picker, until all the products to be put together and dispatched.

Once the evaluation method is defined and the parameters above are set, the main decision in SLAP is the storage policy. Among the most relevant policies, we can cite: *random storage*, *dedicated storage* and *group based storage* (Wang et al. (2020), Žulj et al. (2018)).

A random storage policy allocates products in empty positions inside the warehouse using a random criteria (e.g. closest open location), without including any further complexity in the decision process. In opposite way, dedicated storage ranks the products according to some criteria - popularity, turnorver, *Cube per Order Index* (COI) - putting the best ranked products close to the accumulation/expedition locations. Finally, class based storage separates products in groups and assigns the most interesting groups to places close to the accumulation/expedition positions, without fixing a specific position for each product in the set. The SLAP-PIC uses a dedicated storage policy where the metric is the total travelled distance.

Several studies report that random storage policies lead to a better space utilisation, due to frequent reuse of storage positions, but rise the travelled distances to pick the products

(Muppani and Adil, 2008b) and require higher searching times or control over product locations (Quintanilla et al., 2015). The distance, on other hand, is successfully reduced in dedicated storage policy, but this policy rises re-allocations costs (because of demand fluctuations) and space utilisation, as empty spaces can be reserved to products not currently available. It is stated that class based policies (or zoning) is a balance between random and dedicated storage strategies, but it requires more strategic efforts to define the number of groups and their positions on the warehouse.

Random storage is seldomly studied in the literature, being more an operational and practical approach used as a benchmark to evaluate other methods Pan et al. (2015). One noticeable exception is Quintanilla et al. (2015), which proposes a heuristic to dynamically allocate pallets in a storage area considering stacking constraints in order to reduce the total area used. Pallet stacking is also discussed in Öztürkoğlu (2020), which proposes a bi-objective mathematical model and a constructive algorithm.

A dedicated storage policy is considered in Dijkstra and Roodbergen (2017), in which exact distance evaluations are used to define the product assignment. Guerriero et al. (2013) proposes a non-linear model and an ILS to address a storage allocation problem in a multi-level warehouse considering the compatibility between product classes. Wang et al. (2020) departs from an S-shape routing policy and multi-level storage to create a two-phase algorithm to assign the items to locations, and use a multi-criteria approximation to evaluate the solutions. Other notable works regarding dedicated storage are Battini et al. (2015), which proposes a storage assignment and travel distance estimation to design and evaluate a manual picker to part system and Bolaños Zuñiga et al. (2020), which proposes a model to describe a storage assignment problem, solving it to proven optimality for small instances.

Class based policies are studied in Rao and Adil (2013), Muppani and Adil (2008b) and Muppani and Adil (2008a). In Rao and Adil (2013), class boundaries are defined based on the picking travel distance in a two-block and low-level warehouse where returning routing policy is used. The second uses a Simulated Annealing algorithm to define classes and assign locations to them inside a warehouse considering simultaneously space and picking costs. The work in Muppani and Adil (2008a) extends Muppani and Adil (2008b) by using a branch-and-bound algorithm instead of a heuristic, and by including space use reduction in the considered metrics.

In Pang and Chan (2017), the authors propose a data-mining based algorithm that uses association rules to define product dedicated positions in order to minimise the travelled distance in a picker-to-parts warehouse. Similar approaches are presented in Ming-Huang Chiang et al. (2014) for class based policy, Chuang et al. (2012) for cluster assignment, Fontana and Nepomuceno (2017) for product classification and storage, and Kim et al. (2020) for frequent item set grouping and slot allocation. In Li et al. (2016), an ABC classification and product affinity method is used to define the best positions for the products.

Picker-to-parts systems are addressed in Pan and Wu (2009), Pan et al. (2012) and Pan et al. (2015). The first models the warehouse operation as a Markov chain in order to calculate the expected distance travelled by the picker in three different zoning cases. The second proposes a heuristic for a case where congestion effects are considered. The last uses a genetic algorithm to define the best workload balance among the pickers.

More complex problems are surveyed in van Gils et al. (2018). In Van Gils et al. (2018), batching, routing and zoning are combined to optimise the warehouse operation. A discussion on storage allocation with product picking precedence can be found in Žulj et al. (2018), whereas Thanos et al. (2019) defines a problem where allocation and routing must be decided together in order to avoid congestion during the picking.

4.3 Problem description

The SLAP can be shortly described as follows: given a set P of products to be stored, a set O of orders, in which an order is a subset of products to be picked up by a picker in a single route, and a set L of locations in a warehouse, define an assignment $g : P \rightarrow L$ in which the evaluation function $v(g, O) = z$ is minimised. In other words, given the evaluation function v that maps a value to each ordered pair consisting of a set of orders and an assignment of products to locations, define the assignment that minimises v.

In the SLAP-PIC, the SLAP variant described here, g is an injective function, i.e. each product is assigned to a single location and each location receives a single product. Additionally, $v(g, O)$ is defined as the minimum travelled distance to pick all the products in each order (designated by $D(g, O)$), plus the penalties for non desirable or missing allocations, (designated by $\Phi(g)$). Following the company operational rules, it is assumed that the warehouse uses a picker-to-parts picking policy (the picker visits the products locations) and orders splitting/batching are not allowed, making each order an individual and independent route. These assumptions make it possible to decompose the distance $D(g, O)$ as the sum of the minimal travelling distances to pick the products in each order $o \in O$, designated by $d(g, o)$. With these considerations, the SLAP-PIC objective function can be described as:

$$Min \quad z = v(g, O) = \sum_{o \in O} d(g, o) + \Phi(g) \tag{4.1}$$

It can be noticed that to evaluate $d(g, o)$ it is necessary to solve another optimisation problem, more specifically a variant of the TSP. Namely, if there is a single accumulation/expedition point, each product is assigned to a unique location and $P_o = \{p_o^1, ..., p_o^{|P_o|}\} \subseteq P$ is the set of products requested in an order, then $d(g, o)$ is the minimum distance to depart from the accumulation/expedition point, visit all the locations $(g(p_o^1), ..., g(p_o^{|P_o|}))$ in the best possible order and then come back. Conversely, in the SLAP-PIC we allow the presence of more than one accumulation/expedition point, so the picker can depart from any of these

points and return to another if this operation reduces the total distance travelled $d(g, o)$. The algorithms can be easily adapted to deal with the case in which the expedition points are, instead, fixed for each product.

Defining an optimal picker routing in the scenario above is relatively simple if the warehouse is organised in blocks of identical and parallel shelves. However, in the SLAP-PIC, the shelves can have different sizes, cell quantities, orientations and positioning (as will be detailed in Section 4.4) and also be located in different pavilions. To deal with this setting, a regular distance matrix containing the distances between each pair of locations is considered as the input of the distance minimisation method, disregarding any further information about the warehouse organisation.

The second part of the objective function, the penalty value $\Phi(g)$, is evaluated based on two factors: number of products not assigned to any location and number of undesired allocations. In this sense, the possible configurations of the function g are limited by a set F of assignment prohibitions and a set I of isolation constraints. Each assignment prohibition $f \in F$ is a hard constraint (i.e. it must be strictly respected) composed by a tuple of three values (p, n, c) representing a product p, the nature n of the prohibited location (cell, shelf or pavilion) and the code c of the prohibited location, respectively. For instance, the prohibition $(p_1, \text{“}shelf\text{”}, k_1)$ defines that product p_1 cannot be allocated on shelf k_1.

An isolation constraint is based on the product classification. Given a set T of types, representing the most relevant product characteristic to the storage (toxic, radioactive, humid, etc...), an isolation constraint $\iota \in I$ is a tuple of three values (t, n, s) specifying that products of type $t \in T$ should be allocated in an isolated $n \in \{\text{“}cell\text{”}, \text{“}shelf\text{”}, \text{“}pavilion\text{”}\}$ with an enforcement $s \in \{\text{“}weak\text{”}, \text{“}strong\text{”}\}$. The enforcement s defines if the isolation is a hard constraint or can be relaxed with a penalty.

4.4 Input data processing

A common assumption in studies about SLAP is the regularity of the warehouse layout. Most of the times, the warehouse is represented by sets of parallel and identical shelves that can be accessed through aisles between them and cross-aisles that allow moving from one aisle to another. These sets are called blocks, and most of the literature about SLAP or picker routing considers the presence of one or more blocks in the warehouse.

However, for several reasons, this assumption creates problems when a proposed algorithm needs to be released to production environment. In many facilities, for example, physical or operational barriers are present (like build columns and machines), so layouts cannot be represented as simple blocks.

In this section, we describe the format we created to represent general warehouse layouts, aiming at allowing the use of our algorithm to a large number of warehouses. Furthermore, we

present the data processing performed to get the distance matrix of all the relevant positions in the warehouse, and thus to make it possible to use TSP algorithms to evaluate the total distance travelled by pickers to retrieve all the products in a set of orders.

4.4.1 Input format

The main objective in proposing a new format to describe the warehouse layout is to allow a quick description of different types of layouts. More specifically, a good format should provide: *(a)* easiness of transcription in a spreadsheet or text file; *(b)* readability; *(c)* direct representation on relational databases; *(d)* a simple and robust representation of internal valid paths; *(e)* possibility of defining pavilions and connections between them; *(f)* possibility of defining multiple accumulation/expedition points; *(g)* possibility of defining heterogeneous shelves and cells.

We defined points *(a)*, *(b)* and *(c)* aiming at a future implementation in a decision support system and also at an easy utilisation of the system, as most users are used to text files or spreadsheets and most developers are comfortable in using relational databases.

The robustness to represent paths (point (d)) was considered due to the existence of several operational rules that limit the traffic inside warehouses, mainly when it involves large vehicles or robots. In the proposed format, it is possible to define rectilinear corridors (with a start position, length, direction and sense) and rectilinear segments (with start and end position) to connect two corridors (see Figure 4.1). Non rectilinear corridors or connections were left out due to the variable number of points needed to define them and also because of the difficult evaluation of their length. We also defined that corridors must be parallel to one of the Cartesian axes, not allowing in this way oblique corridors.

Points *(e)* and *(f)* were adopted to take into consideration larger warehouses, in which internal divisions are common and operations can be less centralised. Notwithstanding, the pavilions must be always represented as rectangles, as allowing other formats would significantly increase the representation complexity.

Point *(g)* is modelled to allow describing warehouses with a very diversified set of stored items, from large machines that are positioned on pallets to small tools that can be stored in cabinet drawers.

The resulting format is a union of seven tables (Pavilions, Shelves, Cells, Corridors, Curves and Pavilion Exits, Accumulation/Expedition Points), as presented in Figure 4.2. Each pavilion is composed of a code, a point (with coordinates x and y over a Cartesian plane) representing the bottom-left extremity, width and length (the height is not important to our problem, but can be easily included if needed). By its turn, each shelf has either a code, a bottom-left point, the number of rows and columns, the size of the cells (width, length and height) and the block code indicating where it is located. Each cell has a code, width, length, row and column at the shelf, a reference to the shelf and a number of vertical levels (1 or

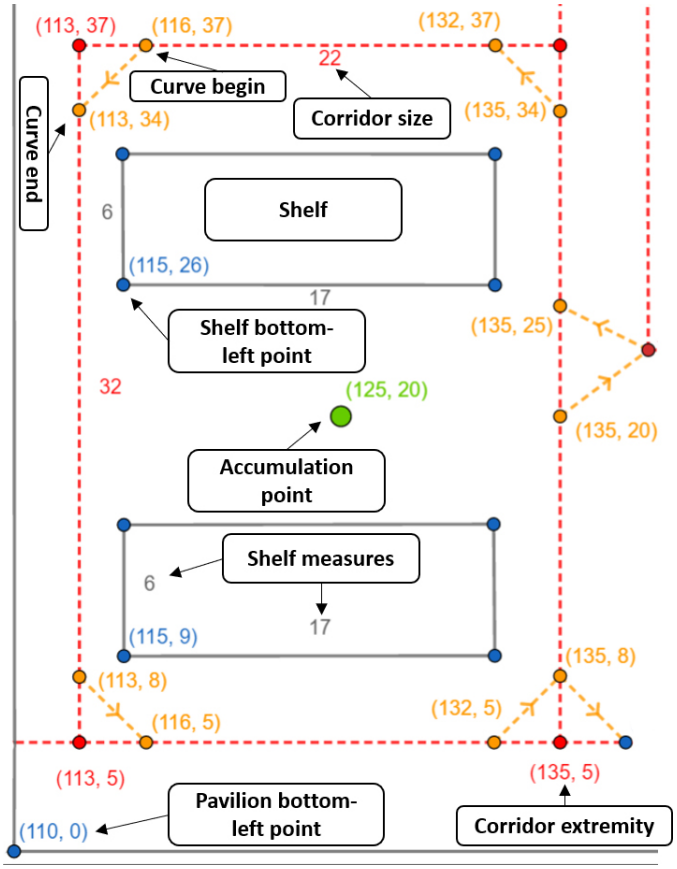


Figure 4.1: Partial warehouse representation with main elements information. The dashed lines are corridors, and the dashed arrows are curves

more). Corridors have a code, an initial point, a direction (horizontal or vertical), a sense (up-down, bottom-up, left-to-right, right-to-left, both), a length and a reference to the block where it is located. Each curve has a code, a reference to both corridors it connects, an initial point and a final point (in this case the length is calculated by the application). Each block exit contains a point, a width, a code and a reference to the blocks it connects. Finally, accumulation/expedition points are composed of a code and a bi-dimensional coordinate.

Products and orders use similar structures, but with less data. Each products is represented by a tuple containing code, description and type (size and weight are not considered in this problem) and each order with a tuple containing a code and a deadline. A list of items of items is assigned to each order, where each item contains a product code and a quantity.

4.4.2 Distance matrix extraction

Once all input information has been loaded, it is necessary to process it to get useful information for the algorithm. Basically, from a warehouse layout description we must extract a distance matrix containing the distances from each delivery point or storage location to all

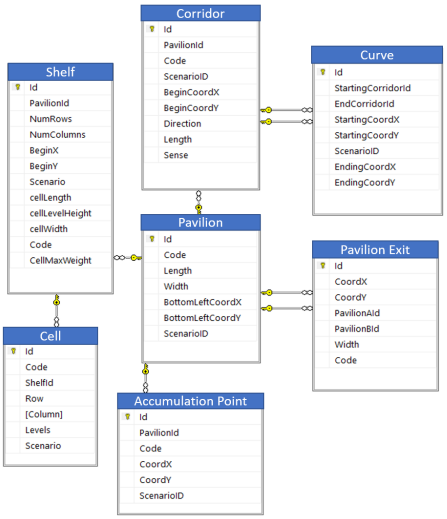


Figure 4.2: Warehouse database

the others, and a compact representation of cells, shelves and blocks, to check if the allocation prohibition and isolation constraints are being respected. The representation of warehouse structures (i.e. cells, shelves and blocks) was done simply using standard object-oriented classes, with no relevant improvements, so it is not detailed in the text.

To build the distance matrix we follow three steps: *(1)* transform the warehouse layout in a directed graph; *(2)* separate the vertices that represent storage locations and accumulation/expedition points from those that represent the paths in the warehouse; and *(3)* run iteratively a shortest path algorithm to determine distances between vertices.

The first step, the conversion of the warehouse in a graph, starts by creating vertices to represent accumulation/expedition points and inferior levels of the cells (i.e. those located closer to the ground, that are the connection points between shelves and corridors). The vertices are positioned in the central point of the cell level. After that, vertices to represent high levels in cells are created, always in the level central point. In detail, when there is more than one level in a cell, each vertex representing a cell level is connected by two arcs, one in each sense, to the level immediately above and below. In other words, when a cell is divided in five vertical levels, the third level is connected with the fourth and the second levels, but not with the first or fifth levels. The edge length correspond to the vertical distance between the level centres, that is the height of the cell divided by the number of cells (i.e. the height of a single level). In the instances of our study, the level height is fixed in 1.5 units.

It is important to notice that in this approach vertices representing external adjacent cells in a same shelf are not directly connected by an edge, except in some special cases that are described later. This is based on the fact that, in many cases, a picker needs to do a non-negligible backward movement to go from a cell to another. The length of this backward movement, however, is as small as the distance from the closest corridor path to the shelf, so

it depends on the input and can be adjusted by the user.

Once connections between the vertices in the shelves are finished, the process of connecting these shelves to the corridors begins. We consider that each shelf can be accessed only by its longest sides (except in squared shelves, that can be accessed by any side). In case of a vertical shelf (i.e. a shelf with the larger side parallel to the y-axis), we consider that a path may reach the shelf through its lateral (left/right) extremities, whereas for a horizontal shelf we consider bottom/up extremities. When the shelf is squared, four corridors are selected, following the same logic. A corridor can not be selected if it does not go through all the shelf side or if it is located in a different pavilion. If there is not a corridor in one of the sides, all the inferior cell levels in that side are connected to the adjacent cell levels in the opposite side, to become accessible from the remaining corridor. As it can be noticed, this is one of the special cases mentioned in the previous paragraph. If there are no corridors in the laterals of the shelf, this shelf is considered unreachable.

After the adjacent corridors be defined, each inferior cell level vertex is connected by a pair of edges (one in each sense) to a vertex created over the corridor exactly in front of the cell vertex.

The new vertices created on corridors in the previous step are then merged with the vertices that represent the extremities of the curves (that are always over corridors) to define all the valid paths in the warehouse. Two consecutive vertices on a corridor are connected by two edges in a two-way corridor (designated by the word ”both” in the field ”sense” on corridor input data) and one edge otherwise. To avoid confusions, we defined that there is not a curve on corridor interception points, thus it is not right to suppose that it is possible to move from a corridor to another through these points unless a curve passing on it is defined.

Following the procedure, each vertex representing an accumulation/expedition point is connected to the closest non-storage vertex in the pavilion in which it is located by two edges, one in each sense. This connection can be a traversal one if there is no possibility of establishing a horizontal or vertical one.

To connect two different pavilions, we first create a vertex to represent each pavilion exit and then connect it to the closest vertex in each pavilion. The connection with pavilions vertices are done in both senses, with the same rules adopted to connect the accumulation/expedition points described in the previous paragraph.

After all these steps, we obtain a graph that represents the connections between all the relevant points in the warehouse. During the graph building process, all vertices are associated with the location they represent, so it is possible to associate each edge with the distance between its incident vertices in the warehouse. Furthermore, each vertex receives a label that is used to define if it is a storage vertex or a vertex representing an accumulation/expedition point, thus being relevant in the distance matrix. The graph built is then passed to an algorithm that calculates the distance matrix.

The algorithm that calculates the distance matrix is a loop, where at each iteration a Dijkstra shortest path algorithm is executed departing from one storage or accumulation/expedition vertex. In this way, all the graph vertices are considered in the shortest path evaluation.

4.5 Iterated Local Search

In this section, we describe the algorithm we propose to solve the SLAP-PIC. This algorithm is an ILS heuristic in which a constructive greedy algorithm generates an initial solution and three different neighbourhood structures are used to improve it.

The constructive algorithm is divided into two phases (see Algorithm 1). In the first phase (lines from 6 to 23), it assigns locations to products without isolation constraints (i.e. products that are not present in any tuple $\iota \in I$) or with isolation constraints containing an enforcement $s =$ "*weak*". In the second phase (lines from 25 to 40), it assigns locations to isolated products, according to their types.

Initially, all the products are sorted by descending order of popularity, i.e. the products more frequently required are put first, and those less frequently after. Similarly, the storage locations are sorted by distance from the closest accumulation/expedition point. Then, the most popular product is assigned to the closest empty location, if this assignment is allowed. When an assignment is forbidden, the algorithm tries iteratively the next location, until an allowed position is found or the loop reaches the last position. If a product belongs to a strongly isolated type, it is not assigned during this phase.

In the second phase of the constructive algorithm, all the products belonging to a strongly isolated type are divided by type (line 25 in Algorithm 1) and, similarly, the warehouse locations that are isolated are grouped by level (block, shelf or cell), in line 26. The allocation is done according to the structure size, starting from the isolated blocks and finishing with the isolated cells. At each step, the product types that have the corresponding isolation level are selected and sorted by decreasing order of frequency. Then, each type is assigned to the isolated area where it is possible to maximise the total frequency, without worrying with the internal assignment optimisation. After each step, the list of available positions and products is updated. When all the available isolated spaces are occupied or all the products are assigned, the algorithm ends.

It is important to notice that it may be impossible to assign all the products to the locations in the warehouse due to prohibition and isolation constraints. This situation can happen even when the number of available locations is higher than the number of products. Furthermore, the constructive algorithm, as a heuristic method, may be not able to find an initial valid assignment even when it exists. In both the cases mentioned, the solution evaluation procedure penalises the objective function according to the number of products

Algorithm 1 `Greedy algorithm`

```
1: g ← ∅                                                  ▷ Start with an empty assignment
2: L* ← distanceSort(L)                                   ▷ Locations ordered by distance
3: A ← L*                                                 ▷ Available locations
4: P ← sortByFrequence(P)
5:                                                        ▷ First part of greedy algorithm
6: for each p ∈ P do                                      ▷ For each product in P
7:     if isStronglyIsolated(p) then
8:         continue
9:     end if
10:    l ← firstAvailable(A)
11:    while true do
12:        if l == null then
13:            break
14:        end if
15:        if isForbidden(l, p) then
16:            l ← nextAvailable(A)
17:            continue
18:        end if
19:        g ← g ∪ (p, l)
20:        A ← A\{l}
21:        break
22:    end while
23: end for
24:                                                       ▷ Second part of greedy algorithm
25: Λ ← stronglyIsolatedProductsByType(P)
26: Ψ ← availableIsolatedStructures(A)
27: μ ← allocateOnIsolatedBlock(Y, Ψ)
28:                                  ▷ First allocation. Assigns products isolated by block
29: g ← g ∪ μ
30: Ψ ← updateAvailableIsolateStructures(Ψ, μ)
31: Λ ← updatestronglyIsolatedProductsByType(μ, P)
32:                                  ▷ Second allocation. Assigns products isolated by shelf
33: μ ← allocateOnIsolatedShelf(Y, Ψ)
34: g ← g ∪ μ
35: Ψ ← updateAvailableIsolateStructures(Ψ, μ)
36: Λ ← updatestronglyIsolatedProductsByType(μ, P)
37:                                  ▷ Third allocation. Assigns products isolated by cells
38: μ ← allocateOnIsolatedCell(Y, Ψ)
39: g ← g ∪ μ
40: Ψ ← updateAvailableIsolateStructures(Ψ, μ)
```

not assigned.

A valid solution, however, can still present one of the side effects of allocating groups of isolated products together in the warehouse. The first is assigning relatively good positions to several products with a low number of requests due to the existence of some very popular products in the set, that are responsible by a skewed popularity of that group. The second effect is approximately the opposite, i.e. very popular products can be allocated in bad positions due to the fact that average popularity is low for their set. Both these problems are

similar to those that are reported in class or zone based storage location problems (see Rao and Adil (2013), Muppani and Adil (2008b)).

After the greedy algorithm creates an initial solution, its objective function value is calculated according to the procedure described in Section 4.5.1. The ILS heuristic enters then in a loop that explores the solution space (lines 4 to 25 of Algorithm 2). This loop is composed by three neighbourhood structures that are combined as a single local search, and a perturbation method.

Algorithm 2 `ILS algorithm`

1: $g^* \leftarrow initialSolution(L, P)$ ▷ Get greedy assignment
2: $g \leftarrow g^*$
3: $nonImprovingIter \leftarrow 0$
4: **while** $nonImprovingIterations < IWI$ **do**
5: $g^w \leftarrow g$ ▷ Initialize the best solution on loop
6: $g' \leftarrow mostFrequentLocalNeighbourhood(g)$
7: **if** $\frac{v(g^w,O)-v(g',O)}{v(g^w,O)} \geq \delta$ **then**
8: $g^w \leftarrow g'$
9: **end if**
10: $g' \leftarrow insideShelfLocalNeighbourhood(g)$
11: **if** $\frac{v(g^w,O)-v(g',O)}{v(g^w,O)} \geq \delta$ **then**
12: $g^w \leftarrow g'$
13: **end if**
14: $g' \leftarrow insidePavilionNeighbourhood(g)$
15: **if** $\frac{v(g^w,O)-v(g',O)}{v(g^w,O)} \geq \delta$ **then**
16: $g^w \leftarrow g'$
17: **end if**
18: $nonImprovingIter \leftarrow nonImprovingIter + 1$
19: **if** $\frac{v(g^*,O)-v(g^w,O)}{v(g^*,O)} \geq \delta$ **then**
20: $g^* \leftarrow g_w$
21: $nonImprovingIter \leftarrow 0$
22: **end if**
23: $g \leftarrow perturbation(g^*)$
24: **end while**

In a loop iteration, each neighbourhood structure evaluates a small set of neighbours of the current solution (denoted by g) and picks the one with lowest objective function value (thus using a best improvement criteria). The best solution in the loop, denoted by g^w, is then compared with the best global solution, g^*, and every time g^w is better, it is assigned to g^* and the number of iterations without improvement (line 22) is reset. Finally, the iteration closes with a perturbation of g^*. The loop stops if the number of iterations without improvement reaches the value of the input parameter *iterations without improvements* (IWI).

To avoid non significant improvements, we assume that an assignment g_1 is better than an assignment g_2 if and only if the objective function value of g_1 is at least 0.1% lower than that of g_2 (i.e. the ratio of between difference of solution values, $v(g_1, O) - v(g_2, O)$, and the old solution value $v(g_2, O)$ must be smaller than $\delta = -0.001$).

All the neighbourhood structures used in the local search are simple location swaps between two products. The first neighbourhood (*mostFrequentLocalNeighbourhood*) consists in swapping the assignments of two products belonging to the subset of 20% most required products. The second (*insideShelfNeighbourhood*) consists of swapping the assignments of two products assigned to locations in the same shelf, working as an intensification of the search. The third neighbourhood (*insidePavilionNeighbourhood*) is a wider search, in which pairs of products assigned to the same pavilion have their locations swapped.

The neighbourhood structures work following the same steps: randomly choose two products, check if the swap between these products is valid, evaluate the change in objective function and store the best solution found.

To check the swap validity, three rules are used: (1) all swaps that assign a product to a forbidden position are not allowed; (2) no swaps are allowed between a product belonging to a strongly isolated type and a product not belonging to a strongly isolated type; (3) no swaps between products of different types are allowed. It is important to notice that validity does not mean feasibility. In fact, while the two first rules were created to avoid infeasible solutions in the search, the third was created to filter the moves in order to avoid recalculating the penalties related to weak isolated types. It can be noticed that the second and third rules are complementary (the third makes the second redundant). In the numeric tests we tested two algorithm versions, one with rules (1) and (2), and the other with rules (1) and (3).

To evaluate the solution after a swap, we reevaluate the distances of the routes affected by that swap and the total of products with weak isolation constraints allocated in altered areas. As the number of swaps performed during the algorithm execution is huge and the number of orders can easily reach some thousands, the procedures to evaluate them must have a strong performance.

After all neighbourhoods have been explored and the global best solution has possibly been updated, a perturbation is performed over the best global solution. It consists in $\|P\|/20$ unconstrained and valid swaps, chosen randomly and applied in a loop. The resulting solution is then used as the initial solution in the next ILS iteration.

4.5.1 Solution evaluation

To evaluate the routing distance, we propose a combination of two ideas: the first is using different TSP algorithms according to the size of the instance and the second is to keep a mapping of routes that passes from each position to recalculate only picking distances of routes containing products involved in the swap. In the last case, initially the algorithm retrieves all the routes affected and controls if a route passes from both locations where the products are currently allocated. If the route has both locations on it, the reevaluation is useless (because the set of locations to be visited does not change), otherwise it is evaluated with the new location replacing the old one.

The evaluation is controlled by the parameter $TSP(\alpha, \beta)$. In this parameter, the constants α and β are two integers representing the order size thresholds used to choose each algorithm method for estimating the minimum distance to visit the product locations in a route. Le $|o|$ be the number of items in an order $o \in O$, if $|o| \leq \alpha$ an exhaustive search is run (i.e. all the possibilities are tested and then the distance found is optimal). Otherwise, if $\alpha < |o| \leq \beta$, a closest neighbour algorithm is used to initialise the route and a subsequent quick local search is performed. In this local search, $(|o| - 1)$ swaps among two consecutive locations are tested in $(|o| - 1)$ iterations (always departing from the first to the last product) and the current solution is updated when the swap reduces the smaller distance. Finally, if $|o| > \beta$, just the closest neighbour heuristic is used.

The route evaluation method described above presents a quadratic complexity, as the exponential time approach is limited according to the number of visited points. Although it does not guarantee an optimal route, it is better suited to our purpose of using non block-based warehouse layouts than algorithms based on the method proposed in Ratliff and Rosenthal (1983).

For explaining how the evaluation of penalties by breaking weak isolation constraints is done, we use Algorithm 3 below. This pseudo code shows the process for shelves, but it is practically identical for cells and pavilions.

Algorithm 3 `Isolation penalty evaluation after a swap`

1: $totalPenalty \leftarrow 0$
2: $S \leftarrow allShelves(L)$
3: $T^w \leftarrow WeakIsolationTypes()$
4: $P_s \leftarrow productsAllocated(g, s) \quad \forall s \in S$
5: **for each** $s \in S$ **do**
6: $P_i \leftarrow \{p \mid p \in P_s, type(p) \in T^w\}$
7: $P_f \leftarrow \{p \mid p \in P_s, type(p) \notin T^w\}$
8: $T_s \leftarrow \{type(p) \mid p \in P_s\}$
9: ▷ Group products with isolation constraints by type
10: $H_t \leftarrow \{p \in P_i | type(p) = t\} \quad \forall t \in T^w$
11: **if** $|P_i| = 0$ **or** $|distinct(T_s)| = 1$ **then**
12: continue
13: **end if**
14: $penalty \leftarrow 0$
15: $x \leftarrow max_{t \in T^w}(|H_t|)$ ▷ Type with max cardinality
16: $r \leftarrow x$
17: **if** $|P_f| \geq |P_i|$ **then**
18: $penalty \leftarrow W_{pen} * |P_i|^2/|P_s|$
19: **else**
20: $penalty \leftarrow W_{pen} * (|P_f|^2 + r)/|P_s|$
21: **end if**
22: $totalPenalty \leftarrow totalPenalty + penalty$
23: **end for**

First of all, the algorithm gets all products allocated and groups them by shelves (line

4). For each shelf, products are grouped by type (line 9) and then the algorithm counts the number of different product types assigned to the shelf. If there is only one type assigned to the shelf or if no assigned product has an isolation constraint $\iota \in I | n = "shelf"$ (see isolation constraint definition on Section 4.3), no penalty is applied (lines 11 to 13). Otherwise, the actual penalty evaluation is performed (lines 14 to 22).

The penalty strategy is based on the division of the products assigned to a structure (which can be a shelf, a cell or a pavilion) into two groups, one with isolation constraints and another without. One of these groups is defined as the minority (resp. majority) if it is the group with less (resp. more) assignments in a structure.

Departing from the *minority* (*majority*) definition, the method tries to push the search through the predominant configuration by penalising minority groups. For example, if the shelf is mostly occupied by products belonging to types without isolation constraints, it penalises the products with isolation constraints (lines 17 and 18) that are the minority. Similarly, it penalises products belonging to types without isolation constraints if they are the minority in the shelf (lines 19 and 20). Furthermore, to differentiate among similar assignments, we consider the proportion of products belonging to the minority/majority over the total number of products, instead of simply counting the number of products. This decision was taken due to the fact that only counting the products was causing no changes in the total penalty after a swap.

4.6 Instance set

In this section, we describe the numeric experiments performed to test the efficiency of the proposed ILS algorithm. In these experiments, we observed the algorithm performance on different instances, the average run time, the local search capacity of improving the initial solution, the parameters' value influence on the final solution and the solution quality.

We created three warehouse layouts with significant differences between them, not only regarding the number of positions, but also regarding how these positions are distributed in the area. A short description of the warehouses is provided in Table 4.1.

Table 4.1: Warehouse layout overview

ID	pavilion	shelves	cells	accum/disp points
W1	1	10	200	3
W2	1	10	240	3
W3	2	12	260	3

Visual representations of the warehouses are provided in Figures 4.3, 4.4 and 4.5. The shelves are defined by the rectangles with circles on the corners. The dashed lines are the

corridors and the dashed arrows are the connections between the corridors. The accumulation/expedition points are represented by isolated circles, as well as the connection between two pavilions is represented by a triangle (see the detail in the centre of Figure 4.5). In these figures, it is possible to notice some of the aforementioned characteristics of these generalised warehouses, as the presence of heterogeneous shelves or blocks, the mandatory moving sense and the multiple accumulation/expedition points.

The experiments were divided into two parts. The first part was aimed to analysing the algorithm performance considering only the travelled distance and thus its suitability to deal with directly calculated distances. The second part tested the performance when considering the prohibition and isolation constraints, in order to check if these constraints were well handled in the algorithm.

We tested the insertion of two sets of products in the warehouses, one with 100 and the other with 200 products. For each product set, we tested scenarios with 500, 1000 and 5000 orders. For each combination of warehouse, number of products and number of orders, five instances were created, for a total of $3 * 2 * 3 * 5 = 90$ instances. In all these instances, we used cells with only one position/level, as showed in Table 4.1.

Regarding the algorithm parameters, we deeply investigated two values: the maximum number of IWI, used as stopping criteria, and the TSP thresholds described in Section 4.5. Three values of maximum number of IWI were tested, and three different TSP thresholds. Additionally, we tested the local search with and without prohibition of swaps between products of different types. In this way, 18 algorithm parameter combinations were experimented.

After that, a set of 25 instances with higher number of products (400, 800, 1600) and orders (2000,5000, 10000) were used to observe the algorithm behaviour and try to establish a practical limit for its use. In this instance set, we used a single warehouse ($W3$), but changed the number of levels in each cell in order to allow the storage of all the products. In this way, instances with 400, 800 and 1600 products were respectively associated with warehouse variants with 465, 833 and 1694 storage positions. It is important to highlight that the warehouse configuration was identical in instances with the same number of products.

To test the algorithm capabilities in managing prohibition and isolation constraints, we used a subset of the initial instances, using 3 variations to each combination of warehouse, number of products and number of orders, for a total of $3*2*3*3 = 54$ instances. For each of these 54 instances, we tested 5 prohibition and isolation constraints, for a total of 270 tests.

In all the instances mentioned above, the number of products by order was determined following a Poisson distribution with average number of events equal to 6. The products inside each order were defined following a uniform distribution, but without allowing the same product to be requested twice in the same order.

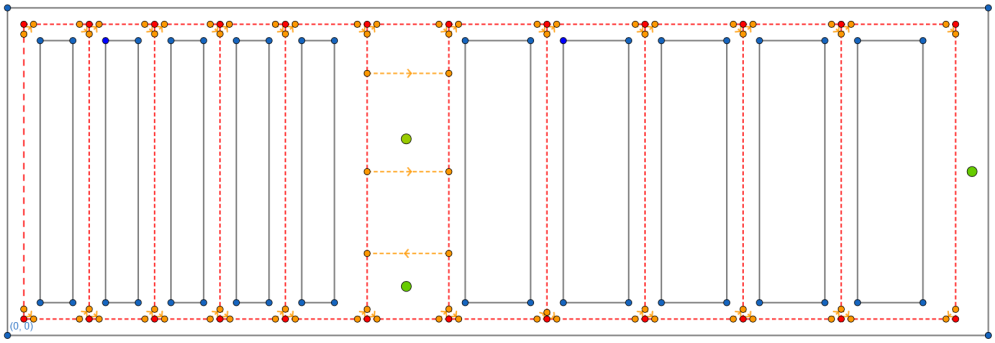


Figure 4.3: Layout of the warehouse W1

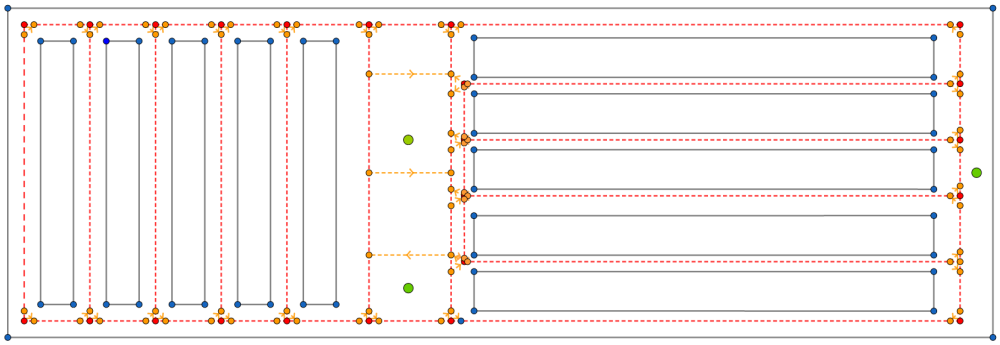


Figure 4.4: Layout of the warehouse W2

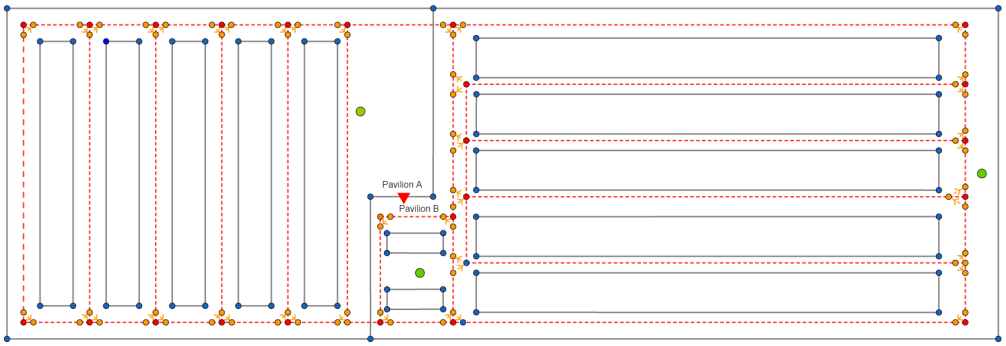


Figure 4.5: Layout of the warehouse W3

4.7 Results

As we described in the previous section, the tests were performed in three steps, the first with a set of 90 instances without prohibition or isolation constraints, a second with a set of 25 instances with higher number of products and orders, and finally the third with a set of 270 instances, based on the first group of instances, but with prohibition and isolation constraints. The algorithm was implemented in C++17 language, with the source code being compiled

with -O3 flag. The tests were performed on a Dell Precision Tower 3620 computer with a 3.50GHz Intel Xeon E3-1245 processor and 32GB of RAM memory, running the Ubuntu 16.04 LTS operational system. All the executions were performed on a single thread, without any significant concurrent process. In the next three sections, we present the results of the three experiments.

4.7.1 Parameters evaluation on basic instances

The results obtained for the first instance set are presented in Tables 4.2 and 4.3. More detailed results are provided in the appendix. Each line in this table represents the average of the computational results with a specific algorithm parameter setting. In the detailed results, each line represent the sum of five individual instances, except those in column “gain”, which are the average of individual gains. In other words, in Tables 4.2 and 4.3 the column ”gain” is an average of the average gains in each group of five instances, and the remaining columns are an average of the sum of results in a group of five instances. In Table II, the results are grouped by the TSP evaluation parameters, whereas in Table III they are grouped by IWI value.

We start our analysis from the effect of the TSP evaluation parameters on the solution value. As the constructive algorithm is not affected by these parameters, it provides always the same initial solution to a given instance, but with a different objective function value, due to the difference in the solution evaluation.

As expected, using exact evaluations (and better heuristics) in more routes leads to a decrease in the objective function values. However, considering the initial solution value, this variation is inferior to 5% in total from the most precise to the less precise evaluation, which suggests that simple heuristics are sufficiently efficient to check the quality of a storage assignment. This evidence is in accordance with what is stated in the literature and practised in real scenarios, mainly by the pickers, that tend to follow the most intuitive and sub-optimal routes (De Santis et al. (2018), Elbert et al. (2017)). However, the improvement on evaluation precision is unequivocally counterbalanced by a significant run time increase if the whole algorithm is considered. The total run time using the TSP(7,11) configuration is over the double of the total run time using the TSP(5,9) configuration.

An interesting effect of the TSP evaluation parameter in the solution is the negative correlation between the evaluation precision and the improvement obtained by the local search. In other words, if we rise the number of routes that are evaluated by an exact method (or better heuristics), we get less improvement over the greedy solution. A possible cause of this result is the higher probability of a travel distance over-estimation when evaluating good quality solutions if the evaluation precision is low. This could conduct the search to a region with bad solutions, increasing the convergence time without improving the solution quality.

In the same sense, it is noticeable that an increase in the number of products and orders

in the instances also reduces the gains over the initial solution. This is an expected result, as it is harder to balance all picker route distances if there are more routes to balance and more points to visit among the different routes.

When comparing the results of scenarios with the same evaluation parameters but different IWI values, it is possible to notice that a higher IWI only slightly improves the average gains over the initial solution (on average less than 0.7 percentage points increase for each two more IWI), even with significantly higher run times. Among the hypotheses to explain this behaviour, we can cite a bad performance of the perturbation method to escape from local optima, a too quick convergence of the method to values close to an average local optimum, the existence of too many local optima, or the existence of several regions of the solution space with very similar characteristics causing repetitive searches.

It is interesting to notice that when comparing "type-free swap results" with "same-type swap" results, the latter on average gets better solutions with a higher run time. We expect that a more restricted swap could lead to a quick conversion and thus a worse solution. As the differences between solutions are relatively small, while between run times are relevant, we can suppose that the method, when using a more restricted local search, performs more but smaller improvements. This explanation is compatible with concerns about meta-heuristic parameter calibration, in which a developer tries to balance the size of intensification and diversification steps in order to find a better algorithm performance.

Table 4.2: Computational results grouped by TSP evaluation parameters

Swap policy	IWI	TSP(5,9)				TSP(6,10)				TSP(7,11)			
		initial (m)	final (m)	time (s)	gain (%)	initial (m)	final (m)	time (s)	gain (%)	initial (m)	final (m)	time (s)	gain (%)
Same-type	10	8377170	5037321	5194.14	43.34	8223513	5212871	5634.41	39.70	8020050	5654668	10394.04	32.10
	8	8377170	5081859	4153.90	42.70	8223513	5238769	4956.94	39.19	8020050	5688999	8549.11	31.47
	6	8377170	5127175	3414.91	41.88	8223513	5296696	3939.77	38.27	8020050	5745588	7598.40	30.85
Type-free	10	8377170	5122995	4329.97	42.26	8223513	5275617	5452.76	38.56	8020050	5716626	9292.56	31.20
	8	8377170	5150287	3546.09	41.85	8223513	5297885	4465.60	38.21	8020050	5720474	7629.36	30.75
	6	8377170	5179081	2944.77	41.39	8223513	5328058	3613.45	37.59	8020050	5766739	6124.49	30.18

Table 4.3: Computational results grouped by *Iterations Without Improvement* (IWI)

Swap policy	TSP	10 IWI				8 IWI				6 IWI			
		initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)
Same-type	(5, 9)	8377170	5037321	5194.14	43.34	8377170	5081859	4153.90	42.70	8377170	5127175	3414.91	41.88
	(6, 10)	8223513	5212871	5634.41	39.70	8223513	5238769	4956.94	39.19	8223513	5296696	3939.77	38.27
	(7, 11)	8020050	5654668	10394.04	32.10	8020050	5688999	8549.11	31.47	8020050	5745588	7598.40	30.85
Type-free	(5, 9)	8377170	5122995	4329.97	42.26	8377170	5150287	3546.09	41.85	8377170	5179081	2944.77	41.39
	(6, 10)	8223513	5275617	5452.76	38.56	8223513	5297885	4465.60	38.21	8223513	5328058	3613.45	37.59
	(7, 11)	8020050	5716626	9292.56	31.20	8020050	5720474	7629.36	30.75	8020050	5766739	6124.49	30.18

4.7.2 Large instances

To analyse the algorithm run time growth and also to check its efficiency in a large warehouse, we performed the experiments summarised in Table 4.4. In this table, "initial", "final" and "time" represent the sum of values for five different instances, and "gain" is an

average of five instance gains. In these instances, there are no isolated types or prohibited allocations and the warehouse is an expanded version of the warehouse $W3$, as described in Section 4.6.

Table 4.4: Computational results on large instances. Parameters used: 8 IWI, $TPS(7, 11)$
.

ID	$\|P\|$	$\|O\|$	initial	final	gain (%)	time(s)
1	400	5000	20750075	17193737	17.14	33636.85
2	800	2000	8688385	7336701	15.55	35179.62
3	800	10000	44726480	37982957	15.08	223500.85
4	1600	2000	8613652	7592868	11.85	115812.40
5	1600	10000	45126739	38745007	14.14	831373.62

In these instances, the gain over the initial solution obtained by the search is between 11% and 18%. This interval is far inferior to those observed in smaller instances, as expected, but still relevant in a real life operation. The run time rises approximately in a linear way with respect to the number of orders (6.37 times for instances with 800 products and 7.18 times for instances with 1600 products), but very fast with respect to the number of products to be allocated, getting to an average of 48 hours for instances with 1600 products.

An interesting result can be noticed when observing the initial solution value. While this seems to be directly proportional to the number of orders (as more orders mean more routes), it does not change with the number of products. This can be explained by the proportional growth of warehouse capacity per area, which makes the average picking density to remain similar, causing a similar route distances.

4.7.3 Instances with isolation and prohibition constraints

In Tables 4.5 and 4.6, we show the algorithm results for instances with isolation (both tables) and allocation prohibition (second table) constraints. In Table 4.5, the block *Isolation 1* refers to instances containing one type of weak isolation constraints, the block *Isolation 2* refers to instances containing one type of weak isolation and one type of strong isolation constraints, and finally block *Isolation 3* has the same isolated types of block *Isolation 2* but with interchanged enforcement. In turn, in Table 4.6 the blocks *Isolation 1* and *Isolation 2* have the same isolation constraints of Table 4.5, but with allocation prohibitions.

Considering the results in Table 4.5, we can see that the penalty contribution in the initial solution value on *Isolation 1* block is less than 16% for all the instance groups, with an average of 5.5%. In the other two blocks, however, the average contribution are 47.8% and 70.8%.

The first thing we can observe is a satisfactory improvement over the initial solution obtained by the local search, although smaller than that observed in the previous results. In the *Isolation 3* block, this improvement is on average 7.1% and in the worst cases (where penalties

are more relevant in initial solution) any improvement is observed. Nevertheless in instances without hard isolation constraints, this metric rises to 19.4% and 15.1% in blocks *Isolation 1* and *Isolation 2*, respectively. These results may suggest that if the constructive algorithm does not handle well isolation constraints, then local search has problems in improving the solution.

We notice that the local search reduces proportionally less the penalty value than the overall objective function value, except in instances with 200 products in warehouse W3 (lines 16 to 18 of Table 4.5), suggesting that the method could be giving more relevance to travel distance.

The average run time was smaller than one hour (or accumulated 10800 seconds in three executions) for all the instance settings, except on setting on line 15 and *Isolation 1*. It suggests that the method converges in an acceptable time even in more realistic and complex mid-size instances.

Looking at the results in Table 4.6, we can notice how the initial solution value is similar for the instances without prohibition constraints. Even with 15% to 30% of products presenting some allocation prohibition in the shelf level the initial solutions values for these instances were less than 2% higher or even lower than the values for instances without prohibition constraints.

We can notice in this part of experiments that a solution probably will not change too much in low to moderate levels of allocation prohibitions. This can happen for two main reasons: products are assigned to locations close to that where it is forbidden to be, or the differences between product demands is not too relevant and small perturbations are not significant.

Table 4.5: Computational results for instances with isolation constraints but without allocation prohibitions. Abbreviations: I.P. = sum of initial penalties, F.P. = sum of final penalties, G% = average percentage gain over initial objective function, P.G.% = average percentage gain over initial penalties. Parameters used: 8 IWI, $TPS(7, 11)$

Id	Isolation 1							Isolation 2							Isolation 3						
	init	I.P.	end	F.P.	G%	P.G.%	time (s)	init	I.P.	end	F.P.	G%	P.G.%	time (s)	init	I.P.	ind	F. P.	G%	P.G.%	time (s)
1	1203.2	140.1	651.7	96.3	45.8	31.0	905.3	1896.2	1032.7	1518.0	956.2	19.9	7.4	675.9	1883.9	1021.5	1536.5	950.1	18.5	7.0	512.4
2	2244.6	146.1	1331.0	123.0	40.7	15.1	1418.0	3564.7	1866.8	2916.2	1757.1	18.1	5.9	1560.0	3523.8	1815.7	2898.0	1723.7	17.8	5.1	1900.0
3	10672.8	145.2	6475.2	134.6	39.3	6.7	10084.3	16979.5	8388.1	14599.4	8360.5	14.0	0.3	5536.0	16922.1	8324.3	14387.5	8282.3	15.0	0.5	6539.9
4	1138.8	150.3	712.8	114.6	37.4	23.2	762.7	1820.9	1026.2	1562.7	986.3	14.2	3.9	717.8	1833.8	1039.8	1511.8	944.4	17.5	9.1	1203.9
5	2108.9	151.8	1400.3	118.4	33.6	21.8	2364.8	3438.8	1861.6	3022.9	1804.2	12.1	3.1	1252.3	3413.5	1831.6	2990.6	1727.2	12.4	5.7	1270.1
6	10001.6	166.0	6803.1	124.7	32.0	24.0	8425.0	16396.6	8396.5	14557.5	8349.1	11.2	0.6	6643.4	16331.5	8329.0	14547.2	8299.2	10.9	0.4	5918.2
7	1111.4	173.3	758.6	102.8	31.8	40.3	848.0	1708.8	1005.7	15115.7	954.7	11.5	5.1	1160.8	1723.2	1021.6	1523.6	946.6	11.6	7.4	965.6
8	2020.9	149.8	1514.2	115.4	25.1	22.0	1834.7	3244.8	1826.6	2968.8	1793.8	8.5	1.8	1453.5	3228.0	1811.3	2934.7	1731.6	9.1	4.4	2857.9
9	9661.7	174.0	7324.4	140.4	24.2	17.4	10952.3	15646.7	8380.0	14435.6	8354.0	7.7	0.3	9748.0	15621.0	8323.3	14367.3	8294.1	8.0	0.4	7730.5
10	1312.4	80.0	960.6	68.0	26.7	14.7	688.3	2179.6	1084.3	1701.9	868.3	22.0	20.1	700.2	3293.7	2868.2	3215.2	2810.6	2.4	2.0	75.9
11	2546.0	84.0	1905.6	70.0	25.1	16.0	1792.3	3670.7	1509.7	3071.6	1230.0	16.3	18.5	1825.2	6325.4	5469.0	6277.2	5423.2	0.8	0.8	143.3
12	12571.4	79.0	9296.7	76.0	26.0	3.3	9691.1	16241.7	5155.5	13580.0	5113.4	16.4	0.8	8602.0	31158.0	26838.0	31158.0	26838.0	0.0	0.0	293.9
13	1271.2	86.0	890.7	647.3	29.9	22.3	1238.5	2137.1	1087.9	1613.0	814.9	24.5	25.1	929.9	3259.9	2869.7	3213.9	2826.1	1.4	1.5	65.6
14	2440.0	81.4	1879.6	717.7	23.0	11.6	1662.9	3630.5	1546.4	2906.2	1246.8	20.0	19.4	2385.5	6277.5	5478.7	6252.9	5458.8	0.4	0.4	161.9
15	12124.6	80.9	8826.2	735.6	27.2	9.1	11249.2	15839.9	5216.2	13220.4	4990.1	16.5	4.3	10725.5	30824.2	26830.7	30824.2	26830.7	0.0	0.0	392.7
16	1302.8	105.6	1072.9	69.6	17.6	33.4	729.1	2134.0	1080.1	1827.0	884.9	14.3	18.0	697.6	3184.8	2856.5	3162.5	2831.9	0.7	0.9	67.7
17	2502.3	94.0	2076.6	70.8	17.0	23.3	1919.7	3627.5	1515.8	3161.0	1288.1	12.8	14.8	1843.3	6152.5	5476.0	6108.3	5430.1	0.7	0.8	284.0
18	12494.3	90.7	10200.0	75.6	18.4	14.9	10426.4	16047.1	5201.4	14278.5	4928.0	11.0	5.3	11685.1	30328.6	26848.6	30317.0	26835.6	0.0	0.1	569.8
	88728.8	2178.3	64079.7	1710.2	28.9	19.4	76993	130205.2	57181.7	112452.2	54680.4	15.1	8.6	68142	185285.4	139053.6	177226.6	138184.3	7.1	2.6	30953

Table 4.6: Computational results for instances with isolation and prohibition constraints. Abbreviations: I.P. = sum of initial penalties, F.P. = sum of final penalties, G% = average percentage gain over initial objective function, P.G.% = average percentage gain over initial penalties. Indexes refer to input parameters like on Table 4.7 to Table 4.18. Parameters used: 8 IWI, $TPS(7, 11)$

Id	W	$\|P\|$	$\|O\|$	Isolation 1							Isolation 2						
				init	I.P.	end	F.P.	G%	PG%	time (s)	init	I.P.	end	F.P.	G%	P.G.%	time (s)
1			500	1215.9	153.1	675.4	112.8	44.32	25.38	829.45	1896.1	1032.2	1510.1	951.3	20.36	7.84	864.47
2		w1	1000	2249.4	149.6	1290.2	95.6	42.63	35.43	2141.19	3563.4	1863.8	2932.8	1771.2	17.64	4.96	1498.90
3			5000	10675.9	147.7	6580.5	157.1	38.36	-6.53	8519.80	16979.2	8385.1	14538.6	8360.0	14.37	0.30	6359.05
4			500	1151.9	162.8	721.7	128.3	37.33	20.93	783.01	1839.7	1059.7	1603.6	1025.6	12.82	3.15	609.02
5	100	w2	1000	2112.3	155.1	1409.5	126.8	33.28	17.90	2041.65	3486.3	1932.1	3076.9	1857.5	11.77	3.90	1469.15
6			5000	9998.9	164.3	6829.0	157.9	31.70	2.89	9016.58	16665.4	8801.5	14881.2	8735.2	10.71	0.76	7166.63
7			500	1113.6	175.4	755.2	109.8	32.19	36.99	1002.66	1725.8	1039.2	1554.1	1003.8	9.94	3.39	1207.71
8		w3	1000	2020.9	149.8	1514.2	115.4	25.07	21.97	1833.82	3290.8	1897.1	3004.5	1865.1	8.70	1.73	1557.57
9			5000	9663.8	176.1	7334.3	140.1	24.10	19.00	10559.08	15888.2	8785.0	14723.9	8747.2	7.32	0.43	8692.53
10			500	1314.7	83.0	979.6	79.0	25.48	4.87	845.45	2185.1	1089.8	1728.0	893.8	20.98	18.17	835.60
11		w1	1000	2562.3	98.0	1899.8	72.0	25.83	26.33	2300.68	3670.8	1509.2	3066.6	1239.0	16.44	17.91	1867.37
12			5000	12625.6	154.6	9436.4	178.6	25.26	-25.94	8473.10	16241.8	5155.5	13499.5	5020.0	16.88	2.62	9276.24
13			500	1276.2	90.9	919.2	75.2	27.97	14.48	764.10	2135.8	1087.9	1639.2	833.1	23.24	23.43	847.04
14	200	w2	1000	2442.3	84.4	1868.8	71.6	23.47	15.01	2281.61	3631.7	1546.4	2981.6	1341.4	17.88	13.18	2036.78
15			5000	12134.7	88.1	8840.5	88.8	27.15	-0.59	12758.23	15836.8	5216.2	13157.2	4889.3	16.92	6.28	10916.91
16			500	1293.7	98.1	1069.1	70.2	17.36	28.40	1322.95	2134.0	1080.1	1835.0	892.9	13.98	17.25	688.72
17		w3	1000	2501.5	93.1	2068.9	73.0	17.29	21.15	2326.59	3625.4	1513.6	3167.8	1280.9	12.61	15.21	1597.07
18			5000	12497.3	91.4	10209.5	93.1	18.31	-2.27	10228.83	16047.1	5201.4	14284.9	4935.6	10.98	5.11	11593.53
Totals				88850.9	2315.8	64401.7	1945.4	28.7	14.2	78029	130843.2	58196.0	113185.7	55642.8	14.6	8.1	69084

4.8 Conclusion

In this chapter, we studied the Storage Location Assignment Problem with Prohibition and Isolation Constraints, a generalisation of the classic Storage Location Problem in which some products may need to be allocated in reserved positions and some other products can not be assigned to specific locations. This problem was proposed in the context of a pharmaceutical logistic operator aiming at improving its performance and providing more flexibility in defining warehouse layouts. Additionally, the work also described how to process the warehouse input data, which can be useful in several other studies that, as this one, need to deal with non-conventional warehouse layouts or for situations where design changes also need to be evaluated.

We proposed an ILS method to solve the problem and we tested it on a large set of instances to demonstrate its suitability in providing good solutions within an acceptable run time. We could also notice that variations on the stopping criteria caused relevant changes in the run time, but just slight changes in the solution quality. On the other hand, the variation in the parameters related to the TSP solutions (to determine the pickers' routes) proved to be very influential in both run time and solution quality. Large instances were solved well by the ILS algorithm, however with higher run times, with this rise strongly related to the number of products to be allocated and less to the number of orders considered. Instances with isolation and prohibition constraints presented lower improvements in the local search

phase, but still relevant for commercial purposes.

For future researches, we suggest to investigate more deeply the influence of initial allocation of strongly isolated types in the overall performance of the optimization method, as well the possibility of guiding this allocation through user manual input. Investigating the suitability of the method to more dynamic situations is also relevant, mainly when different information are available for the orders. It could be interesting to create stronger strategies to avoid route re-evaluation or discard non improving swaps, as this could reduce significantly the run time and allow a broader search.

4.9 Bibliography

R. Aldrighetti, I. Zennaro, S. Finco, D. Battini. Healthcare supply chain simulation with disruption considerations: a case study from northern italy. *Global Journal of Flexible Systems Management*, 20(1):81–102, 2019.

D. Battini, M. Calzavara, A. Persona, F. Sgarbossa. Order picking system design: the storage assignment and travel distance estimation (sa&tde) joint method. *International Journal of Production Research*, 53(4):1077–1093, 2015.

J. Bolaños Zuñiga, J. A. Saucedo Martínez, T. E. Salais Fierro, J. A. Marmolejo Saucedo. Optimization of the storage location assignment and the picker-routing problem by using mathematical programming. *Applied Sciences*, 10(2):534, 2020.

H. Cambazard, N. Catusse. Fixed-parameter algorithms for rectilinear steiner tree and rectilinear traveling salesman problem in the plane. *European Journal of Operational Research*, 270(2):419–429, 2018.

F. Chen, G. Xu, Y. Wei. Heuristic routing methods in multiple-block warehouses with ultra-narrow aisles and access restriction. *International Journal of Production Research*, 57(1):228–249, 2019.

Y.-F. Chuang, H.-T. Lee, Y.-C. Lai. Item-associated cluster assignment model on storage allocation problems. *Computers & Industrial Engineering*, 63(4):1171–1177, 2012.

R. De Koster, T. Le-Duc, K. J. Roodbergen. Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2):481–501, 2007.

R. De Santis, R. Montanari, G. Vignali, E. Bottani. An adapted ant colony optimization algorithm for the minimization of the travel distance of pickers in manual warehouses. *European Journal of Operational Research*, 267(1):120–137, 2018.

A. S. Dijkstra, K. J. Roodbergen. Exact route-length formulas and a storage location assignment heuristic for picker-to-parts warehouses. *Transportation Research Part E: Logistics and Transportation Review*, 102:38–59, 2017.

R. M. Elbert, T. Franzke, C. H. Glock, E. H. Grosse. The effects of human behavior on the efficiency of routing policies in order picking: The case of route deviations. *Computers & Industrial Engineering*, 111:537–551, 2017.

M. E. Fontana, V. S. Nepomuceno. Multi-criteria approach for products classification and their storage location assignment. *The International Journal of Advanced Manufacturing Technology*, 88(9-12):3205–3216, 2017.

J. Gu, M. Goetschalckx, L. F. McGinnis. Research on warehouse operation: A comprehensive review. *European Journal of Operational Research*, 177(1):1–21, 2007.

F. Guerriero, R. Musmanno, O. Pisacane, F. Rende. A mathematical model for the multi-levels product allocation problem in a warehouse with compatibility constraints. *Applied Mathematical Modelling*, 37(6):4385–4398, 2013.

J. Kim, F. Méndez, J. Jimenez. Storage location assignment heuristics based on slot selection and frequent itemset grouping for large distribution centers. *IEEE Access*, 8:189025–189035, 2020.

J. Li, M. Moghaddam, S. Y. Nof. Dynamic storage assignment with product affinity and ABC classification—a case study. *The International Journal of Advanced Manufacturing Technology*, 84(9-12):2179–2194, 2016.

W. Lu, D. McFarlane, V. Giannikas, Q. Zhang. An algorithm for dynamic order-picking in warehouse operations. *European Journal of Operational Research*, 248(1):107–122, 2016.

D. Ming-Huang Chiang, C.-P. Lin, M.-C. Chen. Data mining based storage assignment heuristics for travel distance reduction. *Expert Systems*, 31(1):81–90, 2014.

V. R. Muppani, G. K. Adil. A branch and bound algorithm for class based storage location assignment. *European Journal of Operational Research*, 189(2):492–507, 2008a.

V. R. Muppani, G. K. Adil. Efficient formation of storage classes for warehouse storage location assignment: a simulated annealing approach. *Omega*, 36(4):609–618, 2008b.

Ö. Öztürkoğlu. A bi-objective mathematical model for product allocation in block stacking warehouses. *International Transactions in Operational Research*, 27(4):2184–2210, 2020.

J. C.-H. Pan, P.-H. Shih, M.-H. Wu. Storage assignment problem with travel distance and blocking considerations for a picker-to-part order picking system. *Computers & Industrial Engineering*, 62(2):527–535, 2012.

J. C.-H. Pan, P.-H. Shih, M.-H. Wu, J.-H. Lin. A storage assignment heuristic method based on genetic algorithm for a pick-and-pass warehousing system. *Computers & Industrial Engineering*, 81:1–13, 2015.

J. C.-H. Pan, M.-H. Wu. A study of storage assignment problem for an order picking line in a pick-and-pass warehousing system. *Computers & Industrial Engineering*, 57(1):261–268, 2009.

K.-W. Pang, H.-L. Chan. Data mining-based algorithm for storage location assignment in a randomised warehouse. *International Journal of Production Research*, 55(14):4035–4052, 2017.

S. Quintanilla, Á. Pérez, F. Ballestín, P. Lino. Heuristic algorithms for a storage location assignment problem in a chaotic warehouse. *Engineering Optimization*, 47(10):1405–1422, 2015.

S. S. Rao, G. K. Adil. Optimal class boundaries, number of aisles, and pick list size for low-level order picking systems. *IIE Transactions*, 45(12):1309–1321, 2013.

H. D. Ratliff, A. S. Rosenthal. Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. *Operations Research*, 31(3):507–521, 1983.

J. Reyes, E. Solano-Charris, J. Montoya-Torres. The storage location assignment problem: A literature review. *International Journal of Industrial Engineering Computations*, 10(2):199–224, 2019.

K. J. Roodbergen, R. Koster. Routing methods for warehouses with multiple cross aisles. *International Journal of Production Research*, 39(9):1865–1883, 2001.

A. Scholz, S. Henn, M. Stuhlmann, G. Wäscher. A new mathematical programming formulation for the single-picker routing problem. *European Journal of Operational Research*, 253(1):68–84, 2016.

F. H. Staudt, G. Alpan, M. Di Mascolo, C. M. T. Rodriguez. Warehouse performance measurement: a literature review. *International Journal of Production Research*, 53(18):5524–5544, 2015.

E. Thanos, T. Wauters, G. Vanden Berghe. Dispatch and conflict-free routing of capacitated vehicles with storage stack allocation. *Journal of the Operational Research Society*:1–14, 2019.

C. Theys, O. Bräysy, W. Dullaert, B. Raa. Using a TSP heuristic for routing order pickers in warehouses. *European Journal of Operational Research*, 200(3):755–763, 2010.

R. Uthayakumar, S. Priyan. Pharmaceutical supply chain and inventory management strategies: Optimization for a pharmaceutical company and a hospital. *Operations Research for Health Care*, 2(3):52–64, 2013.

T. Van Gils, K. Ramaekers, K. Braekers, B. Depaire, A. Caris. Increasing order picking efficiency by integrating storage, batching, zone picking, and routing policy decisions. *International Journal of Production Economics*, 197:243–261, 2018.

T. van Gils, K. Ramaekers, A. Caris, R. B. de Koster. Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. *European Journal of Operational Research*, 267(1):1–15, 2018.

J. Volland, A. Fügener, J. Schoenfelder, J. O. Brunner. Material logistics in hospitals: A literature review. *Omega*, 69:82 – 101, 2017.

M. Wang, R.-Q. Zhang, K. Fan. Improving order-picking operation through efficient storage location assignment: A new approach. *Computers & Industrial Engineering*, 139:106186, 2020.

I. Žulj, C. H. Glock, E. H. Grosse, M. Schneider. Picker routing and storage-assignment strategies for precedence-constrained order picking. *Computers & Industrial Engineering*, 123:338–347, 2018.

Appendices

Detailed results of instances without prohibition and isolation constraints

In this appendix, we provide the detailed results of the computational tests that we performed. Tables 4.7 - 4.12 give the results that can be found in aggregated form in Table 4.2 of the paper. Similarly, Tables 4.13 - 4.18 give the results that can found in aggregated form in Table 4.3.

Table 4.7: Computational results with 10 iterations without improvement and same type swaps

Id	$\|P\|$	W	$\|O\|$	TSP(5,9)				TSP(6,10)				TSP(7,11)			
				initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)
1	100	W1	500	1863122	643795	1416.91	65.42	1820373	759141	1232.90	58.26	1773048	875664	2334.21	50.57
2			1000	3619817	1787229	2152.33	50.51	3568509	1745007	3053.97	51.02	3490044	1964363	4578.46	43.62
3			5000	18264608	9093720	11529.61	50.21	17995734	9348413	9653.54	48.05	17546196	10418730	21562.78	40.62
4		W2	500	1738176	749821	1105.09	56.84	1698025	786442	1572.88	53.64	1649763	943287	2016.83	42.77
5			1000	3418249	1767552	2781.71	48.27	3352255	1858347	2994.79	44.54	3264205	2044617	5535.53	37.35
6			5000	17188079	9400950	10876.40	45.30	16867999	9704003	14055.28	42.47	16407861	10773500	27788.25	34.34
7		W3	500	1653207	863097	1332.10	47.80	1611516	916877	1981.55	43.10	1564028	1045144	2970.31	33.17
8			1000	3264462	2067380	2171.73	36.65	3198054	2103654	3013.66	34.20	3119300	2224897	6118.19	28.66
9			5000	16610035	10728080	13407.73	35.41	16266313	11176857	11322.93	31.29	15821210	11950470	22883.67	24.47
10	200	W1	500	2156216	1112431	1569.49	48.36	2121769	1183152	1439.20	44.16	2068296	1341081	2284.14	35.10
11			1000	4282403	2529056	2332.02	40.93	4206423	2611997	3125.52	37.85	4109578	2901679	4503.44	29.36
12			5000	21684669	13510348	11255.04	37.69	21332262	13882116	13439.19	34.92	20850055	15135223	21140.16	27.41
13		W2	500	2083847	1095096	1415.01	47.42	2036449	1160075	1879.71	43.01	1982033	1316481	2639.49	33.57
14			1000	4116914	2594961	2651.52	36.94	4033488	2664212	2762.17	33.93	3942651	2846162	5246.66	27.80
15			5000	20931512	13175500	11779.78	37.05	20552869	13633392	11590.95	33.66	20060073	14452568	23990.97	27.95
16		W3	500	2106403	1346994	1689.98	36.02	2060127	1451049	1622.29	29.54	2006669	1552207	3005.51	22.63
17			1000	4197221	2940923	2735.93	29.92	4109572	3060037	3389.02	25.51	4022267	3236041	4859.28	19.54
18			5000	21610121	15264839	11292.08	29.36	21191498	15786908	13289.82	25.50	20683618	16761904	23634.83	18.96
AVG(10 IWI)				8377170	5037321	5194.14	43.34	8223513	5212871	5634.41	39.70	8020050	5654668	10394.04	32.10

Table 4.8: Computational results with 8 iterations without improvement and same type swaps

Id	$\|P\|$	W	$\|O\|$	TSP(5,9)				TSP(6,10)				TSP(7,11)			
				initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)
1	100	W1	500	1863122	658629	1121.38	64.63	1820373	761493	1144.76	58.13	1773048	904412	1593.26	48.95
2			1000	3619817	1787229	2062.77	50.51	3568509	1768931	2464.99	50.33	3490044	1975038	3834.99	43.31
3			5000	18264608	9156846	9124.42	49.86	17995734	9416788	8735.32	47.67	17546196	10545356	14915.26	39.90
4		W2	500	1738176	755739	963.26	56.49	1698025	822714	1047.40	51.52	1649763	968547	1382.24	41.26
5			1000	3418249	1784835	2377.88	47.77	3352255	1878072	2437.76	43.95	3264205	2069681	4040.82	36.59
6			5000	17188079	9548894	8343.91	44.44	16867999	9773424	12441.96	42.05	16407861	10907010	21758.31	33.52
7		W3	500	1653207	864876	1164.64	47.69	1611516	926984	1716.35	42.48	1564028	1050926	2400.94	32.80
8			1000	3264462	2069549	2010.25	36.58	3198054	2108461	2753.09	34.04	3119300	2234950	4917.18	28.33
9			5000	16610035	10840263	9452.44	34.74	16266313	11189512	10213.07	31.21	15821210	11980105	20399.12	24.28
10	200	W1	500	2156216	1154577	994.26	46.41	2121769	1197067	1201.78	43.50	2068296	1344773	2067.05	34.92
11			1000	4282403	2537949	2118.47	40.72	4206423	2611997	3263.97	37.85	4109578	2918427	3880.86	28.95
12			5000	21684669	13665592	8564.34	36.98	21332262	13897061	12357.61	34.85	20850055	15167752	18726.45	27.25
13		W2	500	2083847	1132826	1010.73	45.61	2036449	1175915	1476.57	42.24	1982033	1336864	2039.75	32.53
14			1000	4116914	2626841	2118.94	36.17	4033488	2675243	2459.03	33.66	3942651	2869019	4236.23	27.21
15			5000	20931512	13232150	10193.32	36.78	20552869	13711921	9467.74	33.28	20060073	14452568	23376.70	27.95
16		W3	500	2106403	1367616	1224.51	35.05	2060127	1470465	1257.29	28.59	2006669	1587418	1908.28	20.86
17			1000	4197221	2976128	2163.52	29.08	4109572	3092862	2552.12	24.72	4022267	3244747	3971.77	19.32
18			5000	21610121	15312919	9761.18	29.14	21191498	15818931	12234.08	25.35	20683618	16844392	18434.85	18.56
AVG(8 IWI)				8377170	5081859	4153.90	42.70	8223513	5238769	4956.94	39.19	8020050	5688999	8549.11	31.47

Table 4.9: Computational results with 6 iterations without improvement and same type swaps

Id	$\|P\|$	W	$\|O\|$	TSP(5,9)				TSP(6,10)				TSP(7,11)			
				initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)
1	100	W1	500	1863122	699210	714.78	62.45	1820373	779973	965.56	57.12	1773048	904412	1541.92	48.95
2			1000	3619817	1817946	1507.86	49.67	3568509	1777854	2183.66	50.08	3490044	1985514	3280.76	43.01
3			5000	18264608	9385337	5650.03	48.61	17995734	9453478	7527.82	47.46	17546196	10722394	11195.59	38.89
4		W2	500	1738176	774898	656.12	55.38	1698025	832506	869.10	50.95	1649763	975617	1229.57	40.83
5			1000	3418249	1803243	1916.47	47.23	3352255	1927597	1643.34	42.48	3264205	2105117	3103.03	35.50
6			5000	17188079	9615727	7623.22	44.05	16867999	9994437	8158.42	40.74	16407861	11057203	35739.39	32.61
7		W3	500	1653207	891332	791.83	46.09	1611516	963603	1043.42	40.20	1564028	1060652	1998.30	32.18
8			1000	3264462	2092301	1609.67	35.89	3198054	2130693	2133.27	33.35	3119300	2250560	3919.20	27.83
9			5000	16610035	10886623	8008.95	34.46	16266313	11319295	7146.49	30.41	15821210	12061439	14672.06	23.76
10	200	W1	500	2156216	1183044	752.91	45.10	2121769	1210797	985.81	42.85	2068296	1349210	1897.03	34.70
11			1000	4282403	2615873	1560.78	38.90	4206423	2706474	4511.81	35.61	4109578	2963492	2701.63	27.86
12			5000	21684669	13712575	7633.00	36.76	21332262	14138717	8103.87	33.72	20850055	15377782	12455.90	26.24
13		W2	500	2083847	1143118	908.24	45.12	2036449	1199252	1142.82	41.08	1982033	1365380	1407.08	31.10
14			1000	4116914	2657124	1689.36	35.42	4033488	2678389	2327.08	33.58	3942651	2873013	3906.89	27.11
15			5000	20931512	13279742	8853.02	36.56	20552869	13760341	8176.46	33.05	20060073	14627528	15227.61	27.08
16		W3	500	2106403	1376644	1094.63	34.62	2060127	1494706	931.68	27.42	2006669	1597795	3550.68	20.35
17			1000	4197221	2999593	1824.05	28.52	4109572	3141990	1713.22	23.51	4022267	3256739	3476.91	19.02
18			5000	21610121	15354824	8673.41	28.95	21191498	15830433	11352.04	25.30	20683618	16886730	15467.72	18.36
AVG(6 IWI)				8377170	5127175	3414.91	41.88	8223513	5296696	3939.77	38.27	8020050	5745588	7598.40	30.85

Table 4.10: Computational results with TSP(5,9) and same type swaps

Id	$\|P\|$	W	$\|O\|$	10 IWI				8 IWI				6 IWI			
				initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)
1	100	W1	500	1863122	643795	1416.91	65.42	1863122	658629	1121.38	64.63	1863122	699210	714.78	62.45
2			1000	3619817	1787229	2152.33	50.51	3619817	1787229	2062.77	50.51	3619817	1817946	1507.86	49.67
3			5000	18264608	9093720	11529.61	50.21	18264608	9156846	9124.42	49.86	18264608	9385337	5650.03	48.61
4		W2	500	1738176	749821	1105.09	56.84	1738176	755739	963.26	56.49	1738176	774898	656.12	55.38
5			1000	3418249	1767552	2781.71	48.27	3418249	1784835	2377.88	47.77	3418249	1803243	1916.47	47.23
6			5000	17188079	9400950	10876.40	45.30	17188079	9548894	8343.91	44.44	17188079	9615727	7623.22	44.05
7		W3	500	1653207	863097	1332.10	47.80	1653207	864876	1164.64	47.69	1653207	891332	791.83	46.09
8			1000	3264462	2067380	2171.73	36.65	3264462	2069549	2010.25	36.58	3264462	2092301	1609.67	35.89
9			5000	16602803	10728080	13407.73	35.41	16610035	10840263	9452.44	34.74	16610035	10886623	8008.95	34.46
10	200	W1	500	2156216	1112431	1569.49	48.36	2156216	1154577	994.26	46.41	2156216	1183044	752.91	45.10
11			1000	4282403	2529056	2332.02	40.93	4282403	2537949	2118.47	40.72	4282403	2615873	1560.78	38.90
12			5000	21684669	13510348	11255.04	37.69	21684669	13665592	8564.34	36.98	21684669	13712575	7633.00	36.76
13		W2	500	2083847	1095096	1415.01	47.42	2083847	1132826	1010.73	45.61	2083847	1143118	908.24	45.12
14			1000	4116914	2594961	2651.52	36.94	4116914	2626841	2118.94	36.17	4116914	2657124	1689.36	35.42
15			5000	20931512	13175500	11779.78	37.05	20931512	13232150	10193.32	36.78	20931512	13279742	8853.02	36.56
16		W3	500	2106403	1346994	1689.98	36.02	2106403	1367616	1224.51	35.05	2106403	1376644	1094.63	34.62
17			1000	4197221	2940923	2735.93	29.92	4197221	2976128	2163.52	29.08	4197221	2999593	1824.05	28.52
18			5000	21610121	15264839	11292.08	29.36	21610121	15312919	9761.18	29.14	21610121	15354824	8673.41	28.95
AVG(TSP(5,9))				8377170	5037321	5194.14	43.34	8377170	5081859	4153.90	42.70	8377170	5127175	3414.91	41.88

Table 4.11: Computational results with TSP(6,10) and same type swaps

Id	$\|P\|$	W	$\|O\|$	10 IWI				8 IWI				6 IWI			
				initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)
1	100	W1	500	1820373	759141	1232.90	58.26	1820373	761493	1144.76	58.13	1820373	779973	965.56	57.12
2			1000	3568509	1745007	3053.97	51.02	3568509	1768931	2464.99	50.33	3568509	1777854	2183.66	50.08
3			5000	17995734	9348413	9653.54	48.05	17995734	9416788	8735.32	47.67	17995734	9453478	7527.82	47.46
4		W2	500	1698025	786442	1572.88	53.64	1698025	822714	1047.40	51.52	1698025	832506	869.10	50.95
5			1000	3352255	1858347	2994.79	44.54	3352255	1878072	2437.76	43.95	3352255	1927597	1643.34	42.48
6			5000	16867999	9704003	14055.28	42.47	16867999	9773424	12441.96	42.05	16867999	9994437	8158.42	40.74
7		W3	500	1611516	916877	1981.55	43.10	1611516	926984	1716.35	42.48	1611516	963603	1043.42	40.20
8			1000	3198054	2103654	3013.66	34.20	3198054	2108461	2753.09	34.04	3198054	2130693	2133.27	33.35
9			5000	16266313	11176857	11322.93	31.29	16266313	11189512	10213.07	31.21	16266313	11319295	7146.49	30.41
10	200	W1	500	2121769	1183152	1439.20	44.16	2121769	1197067	1201.78	43.50	2121769	1210797	985.81	42.85
11			1000	4206423	2611997	3125.52	37.85	4206423	2611997	3263.97	37.85	4206423	2706474	4511.81	35.61
12			5000	21332262	13882116	13439.19	34.92	21332262	13897061	12357.61	34.85	21332262	14138717	8103.87	33.72
13		W2	500	2036449	1160075	1879.71	43.01	2036449	1175915	1476.57	42.24	2036449	1199252	1142.82	41.08
14			1000	4033488	2664212	2762.17	33.93	4033488	2675243	2459.03	33.66	4033488	2678389	2327.08	33.58
15			5000	20552869	13633392	11590.95	33.66	20552869	13711921	9467.74	33.28	20552869	13760341	8176.46	33.05
16		W3	500	2060127	1451049	1622.29	29.54	2060127	1470465	1257.29	28.59	2060127	1494706	931.68	27.42
17			1000	4109572	3060037	3389.02	25.51	4109572	3092862	2552.12	24.72	4109572	3141990	1713.22	23.51
18			5000	21191498	15786908	13289.82	25.50	21191498	15818931	12234.08	25.35	21191498	15830433	11352.04	25.30
AVG(TSP(6,10))				8223513	5212871	5634.41	39.70	8223513	5238769	4956.94	39.19	8223513	5296696	3939.77	38.27

Table 4.12: Computational results with TSP(7,11) and same type swaps

Id	$\|P\|$	W	$\|O\|$	10 IWI				8 IWI				6 IWI			
				initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)
1	100	W1	500	1773048	875664	2334.21	50.57	1773048	904412	1593.26	48.95	1773048	904412	1541.92	48.95
2			1000	3490044	1964363	4578.46	43.62	3490044	1975038	3834.99	43.31	3490044	1985514	3280.76	43.01
3			5000	17546196	10418730	21562.78	40.62	17546196	10545356	14915.26	39.90	17546196	10722394	11195.59	38.89
4		W2	500	1649763	943287	2016.83	42.77	1649763	968547	1382.24	41.26	1649763	975617	1229.57	40.83
5			1000	3264205	2044617	5535.53	37.35	3264205	2069681	4040.82	36.59	3264205	2105117	3103.03	35.50
6			5000	16407861	10773500	27788.25	34.34	16407861	10907010	21758.31	33.52	16407861	11057203	35739.39	32.61
7		W3	500	1564028	1045144	2970.31	33.17	1564028	1050926	2400.94	32.80	1564028	1060652	1998.30	32.18
8			1000	3119300	2224897	6118.19	28.66	3119300	2234950	4917.18	28.33	3119300	2250560	3919.20	27.83
9			5000	15821210	11950470	22883.67	24.47	15821210	11980105	20399.12	24.28	15821210	12061439	14672.06	23.76
10	200	W1	500	2068296	1341081	2284.14	35.10	2068296	1344773	2067.05	34.92	2068296	1349210	1897.03	34.70
11			1000	4109578	2901679	4503.44	29.36	4109578	2918427	3880.86	28.95	4109578	2963492	2701.63	27.86
12			5000	20850055	15135223	21140.16	27.41	20850055	15167752	18726.45	27.25	20850055	15377782	12455.90	26.24
13		W2	500	1982033	1316481	2639.49	33.57	1982033	1336864	2039.75	32.53	1982033	1365380	1407.08	31.10
14			1000	3942651	2846162	5246.66	27.80	3942651	2869019	4236.23	27.21	3942651	2873013	3906.89	27.11
15			5000	20060073	14452568	23990.97	27.95	20060073	14452568	23376.70	27.95	20060073	14627528	15227.61	27.08
16		W3	500	2006669	1552207	3005.51	22.63	2006669	1587418	1908.28	20.86	2006669	1597795	3550.68	20.35
17			1000	4022267	3236041	4859.28	19.54	4022267	3244747	3971.77	19.32	4022267	3256739	3476.91	19.02
18			5000	20683618	16761904	23634.83	18.96	20683618	16844392	18434.85	18.56	20683618	16886730	15467.72	18.36
AVG(TSP(7,11))				8020050	5654668	10394.04	32.10	8020050	5688999	8549.11	31.47	8020050	5745588	7598.40	30.85

Table 4.13: Computational results with 10 iterations without improvement and type-free swaps

Id	$\|P\|$	W	$\|O\|$	TSP(5,9)				TSP(6,10)				TSP(7,11)			
				initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)
1	100	W1	500	1863122	688460	1049.17	63.01	1820373	779120	1285.88	57.16	1773048	910452	2120.97	48.61
2			1000	3619817	1791275	2058.80	50.41	3568509	1830061	2286.49	48.64	3490044	1995489	4621.97	42.73
3			5000	18264608	9419606	8949.95	48.43	17995734	9521883	10266.15	47.08	17546196	10637039	17390.73	39.37
4		W2	500	1738176	761293	1182.55	56.15	1698025	833373	1123.95	50.89	1649763	965330	1967.86	41.45
5			1000	3418249	1821434	2292.24	46.70	3352255	1864768	3125.35	44.35	3264205	2068291	5273.02	36.63
6			5000	17188079	9703878	8064.67	43.54	16867999	10005262	10982.35	40.68	16407861	11023260	19733.38	32.81
7		W3	500	1653207	878658	1273.18	46.85	1611516	955374	1456.05	40.71	1564028	1050139	3185.57	32.85
8			1000	3264462	2051631	2597.06	37.13	3198054	2122762	2581.12	33.60	3119300	2235194	5954.34	28.32
9			5000	16610035	10919314	9764.56	34.26	16266313	11206128	13856.15	31.11	15821210	12033240	22204.77	23.94
10	200	W1	500	2156216	1163102	1126.40	46.00	2121769	1225737	1209.77	42.12	2068296	1363992	2068.95	33.98
11			1000	4282403	2554383	1918.18	40.34	4206423	2678907	2511.89	36.26	4109578	2919459	4903.96	28.93
12			5000	21684669	13678088	9767.57	36.92	21332262	14047837	11584.34	34.14	20850055	15215309	20532.95	27.02
13		W2	500	2083847	1119589	1235.36	46.25	2036449	1194563	1814.90	41.31	1982033	1348216	2074.76	31.96
14			1000	4116914	2632998	1944.55	36.01	4033488	2678879	3312.09	33.56	3942651	2881494	4646.65	26.90
15			5000	20931512	13343452	9830.26	36.25	20552869	13555181	14102.51	34.05	20060073	14611072	19638.07	27.16
16		W3	500	2106403	1384837	1464.09	34.22	2060127	1458611	1472.43	29.17	2006669	1585810	2082.30	20.94
17			1000	4197221	2970750	2209.75	29.20	4109572	3110259	2214.39	24.29	4022267	3251472	4276.77	19.14
18			5000	21610121	15331165	11211.12	29.06	21191498	15892401	12963.94	25.01	20683618	16804009	24589.13	18.76
AVG(10 IWI)				8377170	5122995	4329.97	42.26	8223513	5275617	5452.76	38.56	8020050	5716626	9292.56	31.20

Table 4.14: Computational results with 8 iterations without improvement and type-free swaps

Id	$\|P\|$	W	$\|O\|$	TSP(5,9)				TSP(6,10)				TSP(7,11)			
				initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)
1	100	W1	500	1863122	703451	753.66	62.21	1820373	795519	963.65	56.26	1773048	917213	1546.85	48.24
2			1000	3619817	1801145	1794.57	50.14	3568509	1830061	2199.38	48.64	3490044	2009532	3751.49	42.33
3			5000	18264608	9500498	6447.41	47.98	17995734	9569267	8104.84	46.82	17546196	10647035	15741.41	39.32
4		W2	500	1738176	773024	1035.01	55.47	1698025	835486	1013.45	50.77	1649763	983667	1313.70	40.34
5			1000	3418249	1823832	2058.66	46.63	3352255	1903942	2249.08	43.18	3264205	2100007	3692.80	35.66
6			5000	17188079	9716818	7313.14	43.46	16867999	10071013	8304.96	40.29	16407861	11114358	14207.07	32.26
7		W3	500	1653207	901207	932.26	45.49	1611516	959071	1115.10	40.48	1564028	1076160	1811.56	31.19
8			1000	3264462	2074976	1972.30	36.42	3198054	2123482	2475.20	33.58	3119300	2263067	4261.11	27.43
9			5000	16610035	10973259	7614.82	33.93	16266313	11231579	9946.34	30.95	15821210	12042125	20140.67	23.89
10	200	W1	500	2156216	1175430	914.97	45.42	2121769	1234783	1012.59	41.70	2068296	1377972	1619.87	33.30
11			1000	4282403	2554383	1855.29	40.34	4206423	2686891	2252.90	36.07	4109578	2960831	3249.35	27.93
12			5000	21684669	13833241	7361.23	36.21	21332262	14118268	9570.55	33.81	20850055	15215309	19704.13	27.02
13		W2	500	2083847	1124095	1134.20	46.03	2036449	1208866	1381.51	40.61	1982033	1353051	4041.15	31.72
14			1000	4116914	2632998	1865.37	36.01	4033488	2716272	2272.37	32.64	3942651	2888965	4256.31	26.71
15			5000	20931512	13394442	7716.01	36.01	20552869	13575695	13123.20	33.95	20060073	14654508	16789.39	26.95
16		W3	500	2106403	1397642	1163.73	33.61	2060127	1459944	1377.38	29.10	2006669	1600855	1601.69	20.20
17			1000	4197221	2979038	1914.61	29.00	4109572	3117649	1943.11	24.11	4022267	3268721	2720.52	18.72
18			5000	21610121	15345678	9982.30	28.99	21191498	15924142	11075.14	24.86	20683618	16495153	16879.43	20.25
AVG(8 IWI)				8377170	5150287	3546.09	41.85	8223513	5297885	4465.60	38.21	8020050	5720474	7629.36	30.75

Table 4.15: Computational results with 6 iterations without improvement and type-free swaps

Id	$\|P\|$	W	$\|O\|$	TSP(5,9)				TSP(6,10)				TSP(7,11)			
				initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)
1	100	W1	500	1863122	710041	615.43	61.86	1820373	802493	798.98	55.88	1773048	921284	1381.87	48.01
2			1000	3619817	1827270	1354.14	49.42	3568509	1846916	1695.58	48.16	3490044	2051297	2310.25	41.14
3			5000	18264608	9500498	6145.85	47.98	17995734	9574910	7382.33	46.79	17546196	10647035	15088.98	39.32
4		W2	500	1738176	782645	802.19	54.92	1698025	837863	918.01	50.63	1649763	983667	1250.36	40.34
5			1000	3418249	1838643	1585.54	46.19	3352255	1929867	1776.16	42.41	3264205	2108767	3025.33	35.39
6			5000	17188079	9805776	5626.22	42.95	16867999	10075126	7496.81	40.27	16407861	11126325	12614.12	32.18
7		W3	500	1653207	904109	810.99	45.31	1611516	959071	1047.57	40.48	1564028	1086747	1309.46	30.51
8			1000	3264462	2096514	1685.32	35.76	3198054	2151665	1872.93	32.70	3119300	2275998	3340.04	27.02
9			5000	16610035	11071035	6152.08	33.35	16266313	11327960	7705.81	30.36	15821210	12114603	13785.32	23.43
10	200	W1	500	2156216	1201820	616.52	44.20	2121769	1266760	610.72	40.18	2068296	1397783	1161.11	32.34
11			1000	4282403	2568280	1559.54	40.01	4206423	2734677	1580.81	34.95	4109578	2982607	2353.65	27.40
12			5000	21684669	13910569	6404.63	35.85	21332262	14133159	8863.95	33.75	20850055	15310350	15055.44	26.57
13		W2	500	2083847	1136748	880.90	45.42	2036449	1276315	960.40	37.30	1982033	1371078	1379.46	30.81
14			1000	4116914	2654296	1548.63	35.49	4033488	2721778	2029.19	32.50	3942651	2953971	2873.39	25.06
15			5000	20931512	13394442	7321.53	36.01	20552869	13697250	8960.95	33.35	20060073	14691942	14202.45	26.76
16		W3	500	2106403	1412356	963.22	32.92	2060127	1480661	1045.72	28.10	2006669	1606961	1345.33	19.89
17			1000	4197221	2989301	1616.41	28.76	4109572	3117649	1837.96	24.11	4022267	3269457	2422.04	18.70
18			5000	21610121	15419114	7316.72	28.65	21191498	15970927	8458.13	24.64	20683618	16901426	15342.22	18.29
AVG(6 IWI)				8377170	5179081	2944.77	41.39	8223513	5328058	3613.45	37.59	8020050	5766739	6124.49	30.18

Table 4.16: Computational results with TSP(5,9) and type-free swaps

Id	$\|P\|$	W	$\|O\|$	10 IWI				8 IWI				6 IWI			
				initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)
1	100	W1	500	1863122	688460	1049.17	63.01	1863122	703451	753.66	62.21	1863122	710041	615.43	61.86
2			1000	3619817	1791275	2058.80	50.41	3619817	1801145	1794.57	50.14	3619817	1827270	1354.14	49.42
3			5000	18264608	9419606	8949.95	48.43	18264608	9500498	6447.41	47.98	18264608	9500498	6145.85	47.98
4		W2	500	1738176	761293	1182.55	56.15	1738176	773024	1035.01	55.47	1738176	782645	802.19	54.92
5			1000	3418249	1821434	2292.24	46.70	3418249	1823832	2058.66	46.63	3418249	1838643	1585.54	46.19
6			5000	17188079	9703878	8064.67	43.54	17188079	9716818	7313.14	43.46	17188079	9805776	5626.22	42.95
7		W3	500	1653207	878658	1273.18	46.85	1653207	901207	932.26	45.49	1653207	904109	810.99	45.31
8			1000	3264462	2051631	2597.06	37.13	3264462	2074976	1972.30	36.42	3264462	2096514	1685.32	35.76
9			5000	16610035	10919314	9764.56	34.26	16610035	10973259	7614.82	33.93	16610035	11071035	6152.08	33.35
10	200	W1	500	2156216	1163102	1126.40	46.00	2156216	1175430	914.97	45.42	2156216	1201820	616.52	44.20
11			1000	4282403	2554383	1918.18	40.34	4282403	2554383	1855.29	40.34	4282403	2568280	1559.54	40.01
12			5000	21684669	13678088	9767.57	36.92	21684669	13833241	7361.23	36.21	21684669	13910569	6404.63	35.85
13		W2	500	2083847	1119589	1235.36	46.25	2083847	1124095	1134.20	46.03	2083847	1136748	880.90	45.42
14			1000	4116914	2632998	1944.55	36.01	4116914	2632998	1865.37	36.01	4116914	2654296	1548.63	35.49
15			5000	20931512	13343452	9830.26	36.25	20931512	13394442	7716.01	36.01	20931512	13394442	7321.53	36.01
16		W3	500	2106403	1384837	1464.09	34.22	2106403	1397642	1163.73	33.61	2106403	1412356	963.22	32.92
17			1000	4197221	2970750	2209.75	29.20	4197221	2979038	1914.61	29.00	4197221	2989301	1616.41	28.76
18			5000	21610121	15331165	11211.12	29.06	21610121	15345678	9982.30	28.99	21610121	15419114	7316.72	28.65
AVG(TSP(5,9))				8377170	5122995	4329.97	42.26	8377170	5150287	3546.09	41.85	8377170	5179081	2944.77	41.39

Table 4.17: Computational results with TSP(6,10) and type-free swaps

Id	$\|P\|$	W	$\|O\|$	10 IWI				8 IWI				6 IWI			
				initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)	initial	final	time (s)	gain (%)
1	100	W1	500	1820373	779120	1285.88	57.16	1820373	795519	963.65	56.26	1820373	802493	798.98	55.88
2			1000	3568509	1830061	2286.49	48.64	3568509	1830061	2199.38	48.64	3568509	1846916	1695.58	48.16
3			5000	17995734	9521883	10266.15	47.08	17995734	9569267	8104.84	46.82	17995734	9574910	7382.33	46.79
4		W2	500	1698025	833373	1123.95	50.89	1698025	835486	1013.45	50.77	1698025	837863	918.01	50.63
5			1000	3352255	1864768	3125.35	44.35	3352255	1903942	2249.08	43.18	3352255	1929867	1776.16	42.41
6			5000	16867999	10005262	10982.35	40.68	16867999	10071013	8304.96	40.29	16867999	10075126	7496.81	40.27
7		W3	500	1611516	955374	1456.05	40.71	1611516	959071	1115.10	40.48	1611516	959071	1047.57	40.48
8			1000	3198054	2122762	2581.12	33.60	3198054	2123482	2475.20	33.58	3198054	2151665	1872.93	32.70
9			5000	16266313	11206128	13856.15	31.11	16266313	11231579	9946.34	30.95	16266313	11327960	7705.81	30.36
10	200	W1	500	2121769	1225737	1209.77	42.12	2121769	1234783	1012.59	41.70	2121769	1266760	610.72	40.18
11			1000	4206423	2678907	2511.89	36.26	4206423	2686891	2252.90	36.07	4206423	2734677	1580.81	34.95
12			5000	21332262	14047837	11584.34	34.14	21332262	14118268	9570.55	33.81	21332262	14133159	8863.95	33.75
13		W2	500	2036449	1194563	1814.90	41.31	2036449	1208866	1381.51	40.61	2036449	1276315	960.40	37.30
14			1000	4033488	2678879	3312.09	33.56	4033488	2716272	2272.37	32.64	4033488	2721778	2029.19	32.50
15			5000	20552869	13555181	14102.51	34.05	20552869	13575695	13123.20	33.95	20552869	13697250	8960.95	33.35
16		W3	500	2060127	1458611	1472.43	29.17	2060127	1459944	1377.38	29.10	2060127	1480661	1045.72	28.10
17			1000	4109572	3110259	2214.39	24.29	4109572	3117649	1943.11	24.11	4109572	3117649	1837.96	24.11
18			5000	21191498	15892401	12963.94	25.01	21191498	15924142	11075.14	24.86	21191498	15970927	8458.13	24.64
AVG(TSP(6,10))				8223513	5275617	5452.76	38.56	8223513	5297885	4465.60	38.21	8223513	5328058	3613.45	37.59

Table 4.18: Computational results with TSP(7,11) and type-free swaps

Id	$\|P\|$	W	$\|O\|$	10 IWI				8 IWI				6 IWI			
				initial (m)	final (m)	time (s)	gain (%)	initial (m)	final (m)	time (s)	gain (%)	initial (m)	final (m)	time (s)	gain (%)
1	100	W1	500	1773048	910452	2120.97	48.61	1773048	917213	1546.85	48.24	1773048	921284	1381.87	48.01
2			1000	3490044	1995489	4621.97	42.73	3490044	2009532	3751.49	42.33	3490044	2051297	2310.25	41.14
3			5000	17546196	10637039	17390.73	39.37	17546196	10647035	15741.41	39.32	17546196	10647035	15088.98	39.32
4		W2	500	1649763	965330	1967.86	41.45	1649763	983667	1313.70	40.34	1649763	983667	1250.36	40.34
5			1000	3264205	2068291	5273.02	36.63	3264205	2100007	3692.80	35.66	3264205	2108767	3025.33	35.39
6			5000	16407861	11023260	19733.38	32.81	16407861	11114358	14207.07	32.26	16407861	11126325	12614.12	32.18
7		W3	500	1564028	1050139	3185.57	32.85	1564028	1076160	1811.56	31.19	1564028	1086747	1309.46	30.51
8			1000	3119300	2235194	5954.34	28.32	3119300	2263067	4261.11	27.43	3119300	2275998	3340.04	27.02
9			5000	15821210	12033240	22204.77	23.94	15821210	12042125	20140.67	23.89	15821210	12114603	13785.32	23.43
10	200	W1	500	2068296	1363992	2068.95	33.98	2068296	1377972	1619.87	33.30	2068296	1397783	1161.11	32.34
11			1000	4109578	2919459	4903.96	28.93	4109578	2960831	3249.35	27.93	4109578	2982607	2353.65	27.40
12			5000	20850055	15215309	20532.95	27.02	20850055	15215309	19704.13	27.02	20850055	15310350	15055.44	26.57
13		W2	500	1982033	1348216	2074.76	31.96	1982033	1353051	4041.15	31.72	1982033	1371078	1379.46	30.81
14			1000	3942651	2881494	4646.65	26.90	3942651	2888965	4256.31	26.71	3942651	2953971	2873.39	25.06
15			5000	20060073	14611072	19638.07	27.16	20060073	14654508	16789.39	26.95	20060073	14691942	14202.45	26.76
16		W3	500	2006669	1585810	2082.30	20.94	2006669	1600855	1601.69	20.20	2006669	1606961	1345.33	19.89
17			1000	4022267	3251472	4276.77	19.14	4022267	3268721	2720.52	18.72	4022267	3269457	2422.04	18.70
18			5000	20683618	16804009	24589.13	18.76	20683618	16495153	16879.43	20.25	20683618	16901426	15342.22	18.29
AVG(TSP(7,11))				8020050	5716626	9292.56	31.20	8020050	5720474	7629.36	30.75	8020050	5766739	6124.49	30.18

Conclusion

Planning the storage and distribution of pharmaceutical products in large scale is a hard task, even to the most experienced operators. In this thesis, we presented a decision support system to help improve this activity.

As we showed, the software was created following the guidelines provided by the company. We created a web application with a friendly interface and being accessible through a standard internet browser in which it is possible to input problem data manually or using spreadsheets, visualize and validate this input, call the optimization or analysis procedures and then visualize and export the results. To solve the delivery routing and the vehicle/driver assignment problems, we proposed a two-level method, in which the first part is a heuristic procedure and the second uses a mathematical model. By turn, to solve the storage assignment problem we proposed a new warehouse layout representation and processing, which was used as initial step of an ILS algorithm.

The system development and implementation in the company started with this thesis, but it will certainly be continued in the near future. Currently, vehicle routing and truck/driver assignment modules are being actively used in the company operation. The storage allocation module is in homologation phasis and the last module, that manages warehouse picker schedules, is close to be concluded.

In the next steps, the main objectives are: (a) improving the software integration to make it able to have an automatic communication with other systems already used in the company; (b) increasing robustness and scalability capacity; (c) inserting new optimization modules and allow more custom settings in the modules already in use and (d) creating more detailed and interative reports, to enable an even better solution evalution.

Among the scientific contributions that can emerge with advance of this work, we highlight the study of methods to solve storage allocation problem considering all the definitions and constraints presented in this thesis. This problem represents a major challenge to Operations Research, once the effects of warehouse layouts, demands flutuations and allocation or picker routing policies are complex to be evaluated and costly to be adopted or reversed in real life. We believe that a robust solution to this problem can bring a framework to several related problems described in the literature. Considering the other modules, we can suggest a closer investigation of the effects of labour legislation and workers performance on the results.

Appendices

Appendix A

Solution methods for scheduling problems with sequence-dependent deterioration and maintenance events

In this work, we study the problem of scheduling jobs and maintenance activities on a set of unrelated parallel machines, by considering that the processing time of a job increases according to a deterioration factor that depends both on the machine and on the set of jobs the machine has processed since its last maintenance. The objective we consider is to minimize the makespan. We introduce four mixed integer linear programming models, two of which using big-M constraints and the other two using an exponential number of variables. We also propose an iterated local search metaheuristic to tackle large size instances and we provide empirical evidence of the performance of the proposed approaches by means of extensive computational experiments.

A.1 Introduction

Fatigue and deterioration might severely affect human and machine performances, causing an increase in the number of errors they make, in the quantity of resources they waste, or in the processing time of the jobs they perform. In order to alleviate this issue, work stops or maintenance events can be scheduled, so as to recover the full productivity of the agent and improve the overall performance of the system.

In this paper, we deal with the problem of scheduling jobs and maintenance activities on a set of unrelated parallel machines, where the processing time of each job on a given machine depends on the initial processing time of the job on the machine (i.e., its processing time if

the machine was working at full performance) and on the deterioration level of the machine. The machines are not identical, which means that a job may have two different processing times and deterioration factors on two different machines. The duration of a maintenance also depends on the machine on which it is performed, but it is not impacted by the current deterioration level of the machine. The objective is to find a feasible schedule in which all jobs are processed and the makespan (i.e., the last completion time of a job) is minimized. This type of problems occurs, for example, in construction industry, where the difficulty of a given job impacts the workers' level of tiredness and thus increases the time they need to accomplish subsequent jobs, and in the cutting industry, where the hardness of a material deteriorates the cutting tools and increases the time required to cut other materials (see Ruiz-Torres et al. (2017)). In both cases, a short time in which the agent is not operating (such as a break for the worker or a maintenance operation for the cutting tools) is enough to restore full productivity.

In terms of contributions, we propose a linearization of the Mixed Integer Non-Linear Programming (MINLP) model originally proposed by Ruiz-Torres et al. (2017) and three additional Mixed Integer Linear Programming (MILP) models: an improvement of our first model that significantly reduces symmetry, a model based on the classical set covering formulation, and a model inspired by the arc flow formulation by Valério de Carvalho (1999). Our resulting models can be solved by invoking a standard MILP solver and obtain optimal solutions for small size instances. For larger instances, we propose a metaheuristic procedure based on the concept of *Iterated Local Search* (ILS). We show, trough extensive computational experiments, that we can obtain good quality solutions in a few seconds for a variety of instances, considering a large range of deterioration rates, processing times, and maintenance times.

The remainder of the paper is organised as follows. Section A.2 provides a literature review on scheduling problems with deterioration and maintenance activities. We formally describe our problem in Section A.3, and we introduce the mathematical models in Section A.4. The ILS is presented in Section A.5. The outcome of extensive computational experiments assessing the quality of our approaches is reported in Section A.6. Finally, we draw some conclusions and discuss interesting future research directions in Section A.7.

A.2 Literature review

The literature on scheduling problems with deterioration can be split into two main groups, according to how the deterioration is estimated. The first group focuses on job deterioration and uses the paradigm that the jobs processed at a later stage require an additional time with respect to the jobs processed at an early stage (e.g, due to some physical properties). In such a case, we adopt the term *time-dependent* deterioration. The second group focuses

on machine deterioration and adopts the paradigm that processing a job deteriorates some components of the machine, and this makes the processing time of subsequent jobs longer. In such a case, we adopt the term *sequence-dependent* deterioration.

Scheduling problems with time-dependent deterioration have been investigated since the late Eighties. To the best of our knowledge, the first study was presented by Gupta and Gupta (1988), and focused on a single machine problem in which the duration of a job depends on its starting time. The authors mentioned relevant applications in chemical and metallurgical processes, where the temperature of the material cools down if it is not used immediately, requiring a longer time to be processed. This seminal study was followed in the next years by Kunnathur and Gupta (1990), who proposed algorithms based on dynamic programming and branch-and-bound, by Browne and Yechiali (1990), who analyzed the effects of different deterioration schemes and derived optimal scheduling policies that minimize either the expected makespan or its variance, by Mosheiov (1991, 1994, 1996), who outlined theoretical properties on the optimal job sequence, and by Kubiak and van de Velde (1998), who studied the case in which the supplementary time caused by deterioration is bounded.

The problem where multiple parallel machines are available was studied by Mosheiov (1995, 1998), who proved that makespan minimization with linear deterioration is an $\mathcal{NP}$-hard problem if there are at least two machines. He also introduced several compact MILP formulations and heuristic algorithms. More recently, Ji and Cheng (2008) proposed a polynomial-time approximation scheme for the case in which the number of machines is fixed.

The problem where the deterioration function works through steps (i.e., the processing time of a job changes only if it is processed after a given deadline) was studied by Cheng and Ding (2001) for the single machine case, and by Leung et al. (2008) and Lalla-Ruiz and Voß (2016) for the multiple machine case.

Surveys on scheduling with time-dependent deterioration were proposed by Alidaee and Womer (1999) and Cheng et al. (2004). We also refer the reader to the recent survey proposed by Gawiejnowicz (2020) for an up-to-date overview of this class of problems.

To the best of our knowledge, the first paper on scheduling with sequence-dependent deterioration was proposed by Ruiz-Torres et al. (2013), who studied the unrelated parallel machine case with the objective of makespan minimization. They showed that the problem is $\mathcal{NP}$-hard and proposed an MINLP model and a simulated annealing algorithm. A few years later, Santos and Arroyo (2017) proposed an ILS and an ILS combined with a random variable neighborhood descent algorithm, and showed that their algorithms computationally outperformed the one proposed in Ruiz-Torres et al. (2013) on both small size instances (50 jobs and 10 machines) and large size instances (150 jobs and 20 machines). In the same period, Araújo et al. (2017) linearized the model proposed in Ruiz-Torres et al. (2013) and improved its performance so that it was able to solve exactly instances with up to 50 jobs and 10 machines. More recently, Ding et al. (2019) studied a similar problem in which the

deterioration level of a machine when processing a job depends on the deterioration factor of the job itself in addition to those of the jobs already processed. The authors studied the objectives of minimizing the makespan and minimizing the weighted completion time. They proposed an ejection chain algorithm and tested it on instances with up to 50 jobs and 10 machines.

Maintenance activities were also studied by Kuo and Yang (2008), Zhao and Tang (2010), Yang (2011), and Yang et al. (2012) in the context of scheduling with *position-dependent* deterioration. Position-dependent deterioration can be considered as a special case of sequence-dependent deterioration in which the additional time required to process a job depends only on the number of jobs that were processed since the last maintenance (and not on the type of jobs, as in the sequence-dependent case). While Kuo and Yang (2008) and Zhao and Tang (2010) focused on the single machine problem, Yang (2011) and Yang et al. (2012) dealt with the multiple machine case. In all these problems, as in ours, a maintenance activity restores the full productivity of a machine.

We note, in addition, that the term "maintenance activity" is widely used in the scheduling literature, sometimes with different meanings with respect to the one we adopt in our work. The term might refer, for example, to a mandatory operation (see, e.g., Nesello et al. (2018b)) or to a *rate modifying* (rm) operation that changes the processing time of subsequent jobs (either by speeding them up or slowing them down).

A large stream of the literature is dedicated to rm activities. To the best of our knowledge, the concept was introduced by Lee and Leon (2001) to describe a scheduling behavior in an electronic assembly line. In their single machine scheduling problem, each job had two possible processing times, depending on whether it was scheduled before or after the rm activity. The problem was then to decide the job ordering and the position of the rm activity, if necessary. This study was extended by Mosheiov and Sidney (2003), who addressed a similar problem with the addition of precedence constraints. They also studied the problem with the addition of a learning effect, which can be seen as the opposite of a position-dependent deterioration (i.e., processing a job shortens the duration of every subsequent job). Some recent studies, as Wang and Li (2017) and Lu et al. (2018), consider the case in which the rm activity has an execution time that linearly depends on its starting time. We refer the interested reader to the recent book by Strusevich and Rustogi (2017) for further details on rm activities.

The first work in which sequence-dependent deterioration and maintenance activities were studied together was proposed by Ruiz-Torres et al. (2017). In their paper, the deterioration level of a machine when processing a job depends only on the types of jobs that the machine has processed since its last maintenance. The authors studied the case of identical machines. They introduced an MINLP model to describe the problem and proposed some constructive heuristics to find good quality solutions. In our work, which extends the one by Ruiz-Torres et al. (2017), we propose new mathematical models and metaheuristic algorithms

for scheduling problems with sequence-dependent deterioration and maintenance events. All our techniques are valid for the general case of unrelated machines. A preliminary version of our work containing the model of Section A.4.2 and a prior ILS implementation was presented in Mendes and Iori (2019).

A.3 Problem description

Let $J = \{1, 2, \ldots, n\}$ be a set of independent jobs, all available at the beginning of the working horizon, to be processed on a set $M = \{1, 2, \ldots, m\}$ of unrelated parallel machines subject to deterioration. Let p_{ij} be the ideal processing time of job j ($j \in J$) on machine i ($i \in M$), that is, the processing time when the machine is fully operative, so either at the beginning of the time horizon or right after a maintenance event has occurred. Let also $d_{ij} \geq 1$ be the *delay factor* caused by the deterioration of machine i after processing job j. As the delay factors are multiplicative, let $\delta_{ij} \geq 1$ be the *accumulated delay factor* due to the deterioration of machine i because of all the jobs it has processed before j since the last maintenance. This sequence-dependent deterioration (which is also made clear by means of Example 1 at the end of this section) can be described as follows:

- the actual processing time of job j on machine i is equal to $p_{ij}\delta_{ij}$;
- the accumulated delay factor caused by the deterioration of machine i after processing job j is equal to $\delta_{ij}d_{ij}$.

It can be observed that the accumulated delay factor at the start of job j (i.e., δ_{ij}) is equal to the product of the delay factors of all jobs processed before j since the last maintenance.

Let t_i be the duration of a maintenance event that returns machine i to its fully operative state (i.e., $\delta_{ij} = 1$, for all j processed right after a maintenance activity). A single machine cannot process a job and perform a maintenance activity at the same time. In addition, preemption is not allowed, so a machine cannot be interrupted while it is processing a job to perform a maintenance activity. There is no theoretical limit on the number of maintenance activities that can be performed on a single machine (even if in practice every relevant schedule has at most $n - 1$ maintenance activities), nor on the number of maintenance activities that can be performed in parallel on all machines.

Each job must be assigned to a machine in such a way that the makespan (i.e., the maximum completion time among all jobs) is minimized. By summing the actual processing times of the jobs and the maintenance times, we can easily compute the completion time of each machine, and thus the makespan. The insertion of idle times on the machine schedules cannot decrease the makespan, because there are no limitation on the number of simultaneous maintenance activities occurring at a given time, all jobs are available at time 0, and there are no precedence constraints. Thus, there is always an optimal solution without idle times.

According to the three-field notation by Graham et al. (1979), this problem can be denoted as R|Sdd,mnt|C_{max}, where "R" stands for unrelated parallel machines, "Sdd" for sequence-dependent deterioration, "mnt" for maintenance, and "C_{max}" for makespan minimization. The R|Sdd,mnt|C_{max} is strongly $\mathcal{NP}-$hard because it generalizes the well-known R||C_{max}, already proven to be strongly $\mathcal{NP}$-hard in Pinedo (2016). In the following, we introduce an example that will be resumed in the next sections to outline the behavior of our models.

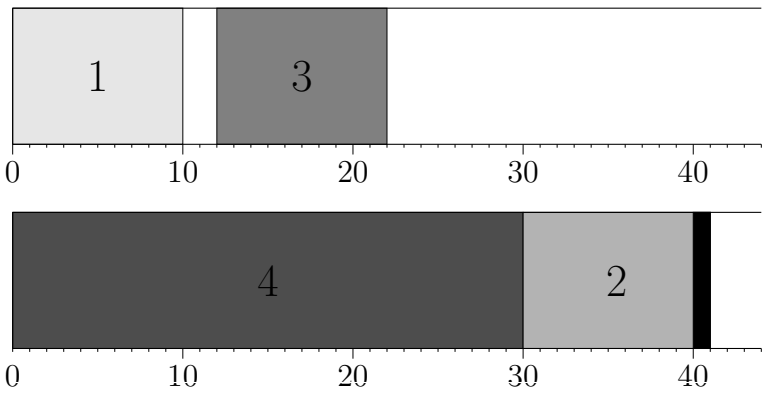
EXAMPLE 1 *Let us consider the following instance with two machines and four jobs, whose ideal processing times, delay factors, and maintenance durations are displayed in Table A.1.*

Table A.1: Ideal processing times, delay factors, and maintenance times for Example 1

machine	t_i	ideal processing time (p_{ij})				delay factor (d_{ij})			
		$j=1$	$j=2$	$j=3$	$j=4$	$j=1$	$j=2$	$j=3$	$j=4$
$i=1$	2	10	20	10	30	1.2	1.1	1.1	1.2
$i=2$	5	20	10	10	30	1.1	1.2	1.1	1.1

As shown in Figure A.1, jobs 1 and 3 are scheduled on machine 1 separated by a maintenance activity (represented in white in the figure). Job 4 is scheduled first on machine 2, followed by job 2. The actual processing time of job 2 is 11 instead of 10 because of the deterioration caused by job 4 (represented by hashed lines in the figure). The optimal makespan is 41. Note that we included a maintenance on machine 1 between job 1 and job 3 for descriptive purposes. One could also skip the maintenance and process job 3 first, followed by job 1, and obtain a load of 21 units on machine 1. In any case, the optimal makespan would remain 41 as the workload on machine 2 is unchanged.

Figure A.1: Optimal job scheduling for Example 1 (machine 1 on top, machine 2 at the bottom)



A.4 Mathematical models

In this section, we present the MINLP model by Ruiz-Torres et al. (2017), introduce its linearized version, and then propose three novel MILP models. To ease notation, if the meaning of a variable is the same in several models, then we maintain the same name.

A.4.1 Non-linear position-based formulation

The model of Ruiz-Torres et al. (2017) decomposes each machine into slots $H = \{1, 2, \ldots, |H|\}$. A slot defines the position of an activity (a job or a maintenance) on a machine schedule and has no pre-determined duration. Since we do not know a priori the number of slots required by each machine, we use a valid upper bound $|H| = 2n-1$ to estimate it. Indeed, in the worst case a single machine processes all the jobs and performs a maintenance activity between each pair of jobs. Because the number of slots per machine i is generally larger than the number of activities assigned to i, some of the slots on i may remain free (i.e., with no assigned activity). Such slots are herein called *empty slots.* Better upper bounds on H can be derived, using the form $|H| = 2(n-m')+1$, if we can prove that there is always an optimal solution in which m' machines perform at least one job, or, in other words, in which no single machine performs strictly more than $n - m' + 1$ jobs, so resulting in at most $n - m'$ maintenance operations. In particular, if the ideal processing time of each job is the same on every machine, then we have $m' = m$.

Let x_{ijh} be a binary variable taking the value 1 if job j is executed in slot h of machine i, and 0 otherwise. Similarly, let s_{ih} be a binary variable taking the value 1 if a maintenance is executed in slot h of machine i, and 0 otherwise, and let q_{ih} be a continuous variable that reports the current *performance ratio* of machine i in slot h. The performance ratio is simply the inverse of the accumulated delay factor δ_{ij}: it satisfies $0 < q_{ih} \leq 1$ and the actual processing time of job j on machine i in slot h is equal to $\frac{p_{ij}}{q_{ih}}$, while the performance ratio of machine i after processing job j in slot h is equal to $q_{ih}(1 - d'_{ij})$.

Parameter d'_{ij} is called *deterioration effect* and satisfies $0 \leq d'_{ij} < 1$. It is adopted by Ruiz-Torres et al. (2017) to replace delay factor d_{ij} by imposing the relation $d_{ij} = \frac{1}{1-d'_{ij}}$. For example, a job j whose deterioration effect on machine i is $d'_{ij} = 0.01$ has a delay factor on machine i of $d_{ij} = \frac{1}{0.99} \approx 1.0101$.

By using these variables and an additional continuous variable $\mathrm{C_{max}}$ indicating the value of the makespan, the R|Sdd,mnt|$\mathrm{C_{max}}$ can be modeled as:

$$\min \quad \mathrm{C_{max}} \tag{A.1}$$

$$\text{s.t.} \quad \sum_{j \in J} x_{ijh} + s_{ih} \leq 1 \qquad \forall i \in M, h \in H, \tag{A.2}$$

$$\sum_{i \in M} \sum_{h \in H} x_{ijh} = 1 \qquad \forall j \in J, \tag{A.3}$$

$$\sum_{j\in J}\sum_{h\in H}\frac{p_{ij}}{q_{ih}}x_{ijh}+\sum_{h\in H}t_i\ s_{ih}\leq \mathrm{C}_{\max} \qquad \forall i\in M, \tag{A.4}$$

$$x_{ijh}\leq \sum_{j'\in J}x_{ij',h-1}+s_{ih} \qquad \forall i\in M, j\in J, h\in H\backslash\{1\}, \tag{A.5}$$

$$q_{i,h-1}\sum_{j\in J}(1-d'_{ij})\ x_{ij,h-1}+s_{i,h-1}=q_{ih} \qquad \forall i\in M, h\in H\backslash\{1\}, \tag{A.6}$$

$$q_{i1}=1 \qquad \forall i\in M, \tag{A.7}$$

$$x_{ijh}\in\{0,1\} \qquad \forall i\in M, j\in J, h\in H, \tag{A.8}$$

$$s_{ih}\in\{0,1\} \qquad \forall i\in M, h\in H, \tag{A.9}$$

$$q_{ih}\geq 0 \qquad \forall i\in M, h\in H, \tag{A.10}$$

$$\mathrm{C}_{\max}\geq 0. \tag{A.11}$$

Objective function (A.1) imposes the minimization of the makespan. Constraints (A.2) state that each slot can either accommodate a job, a maintenance activity, or remain empty. Constraints (A.3) impose each job to be processed by a machine. Constraints (A.4) make sure that the maskespan is equal to or greater than the finishing time of each machine. Constraints (A.5) force the empty slots to be positioned at the end of the planning horizon. Constraints (A.6) compute the performance ratio of each machine in each slot through an inductive process based on the previous slot of the machine. Constraints (A.7) impose the machines to be fully operative at the beginning of the planning horizon. It can be noticed that constraints (A.4) and (A.6) are non-linear.

A.4.2 Linearized position-based formulation

The first MILP formulation that we propose is derived from the work by Araújo et al. (2017), who modeled R|Sdd|$\mathrm{C}_{\max}$ the version of our problem without maintenance activities. It uses an accumulated delay factor δ_{ih} instead of the performance ratio q_{ih}, and the delay factor d_{ij} instead of the deterioration effect d'_{ij}. It also uses a continuous variable a_{ijh} that indicates the actual processing time of job j on machine i at time slot h, and "big-M" constraints that force a_{ijh} to take (at least) value $p_{ij}\delta_{ih}$ when job j is assigned to slot h on machine i, and 0 otherwise. The linearized position-based formulation is as follows:

$$\min \quad \mathrm{C}_{\max} \tag{A.12}$$

$$\text{s.t.} \quad \sum_{j\in J}x_{ijh}+s_{ih}\leq 1 \qquad \forall i\in M, h\in H, \tag{A.13}$$

$$\sum_{i\in M}\sum_{h\in H}x_{ijh}=1 \qquad \forall j\in J, \tag{A.14}$$

$$a_{ijh}\geq p_{ij}\delta_{ih}-K^a_{ij}(1-x_{ijh}) \qquad \forall i\in M, j\in J, h\in H, \tag{A.15}$$

$$\sum_{j\in J}\sum_{h\in H} a_{ijh} + \sum_{h\in H} t_i s_{ih} \leq \mathrm{C}_{\max} \qquad \forall i \in M, \tag{A.16}$$

$$\sum_{j\in J} x_{ijh} + s_{ih} \leq \sum_{j\in J} x_{ij,h-1} + s_{i,h-1} \qquad \forall i \in M, h \in H\backslash\{1\}, \tag{A.17}$$

$$d_{ij}\delta_{i,h-1} - K^b_{ij}(s_{i,h-1} + 1 - x_{ij,h-1}) \leq \delta_{ih} \qquad \forall i \in M, j \in J, h \in H\backslash\{1\}, \tag{A.18}$$

$$\delta_{i1} = 1 \qquad \forall i \in M, \tag{A.19}$$

$$x_{ijh} \in \{0,1\} \qquad \forall i \in M, j \in J, h \in H, \tag{A.20}$$

$$s_{ih} \in \{0,1\} \qquad \forall i \in M, h \in H, \tag{A.21}$$

$$a_{ijh} \geq 0 \qquad \forall i \in M, j \in J, h \in H, \tag{A.22}$$

$$\delta_{ih} \geq 1 \qquad \forall i \in M, h \in H, \tag{A.23}$$

$$\mathrm{C}_{\max} \geq 0. \tag{A.24}$$

Objective function (A.12) and constraints (A.13) and (A.14) are identical to those reported in the MINLP model of Section A.4.1. Constraints (A.15) are used to define the actual processing time of job j in slot h of machine m. Coefficient K^a_{ij} is a large constant defined for any machine i and job j in such a way that a_{ijh} is allowed to take value 0 when $x_{ijh} = 0$ and forced to take at least value $p_{ij}\delta_{ih}$ when $x_{ijh} = 1$. Constraints (A.16) are the linearized version of constraints (A.4), and are used to calculate the makespan. Constraints (A.17) are the counterparts of constraints (A.5) and prevent a job or a maintenance activity to be scheduled after an empty slot (slot h on machine i is empty if $\sum_{j\in J} x_{ijh} + s_{ih} = 0$). In other words, all the empty slots of a given machine are forced to be positioned at the end of the planning horizon. Worth is mentioning that, while constraints (A.5) have the purpose of removing symmetrical solutions, constraints (A.17) are necessary for the correctness of the model, because inserting an empty slot in the middle of the planning horizon would be equivalent to performing an instantaneous maintenance activity. Constraints (A.18) are the counterpart of constraints (A.6) and compute the accumulated delay factor of each machine in each time slot through an inductive process. The coefficient K^b_{ij} is a sufficiently large constant defined for any machine i and job j, so that δ_{ih} is allowed to take the value 0 when $x_{ij,h-1} = 0$. Constraints (A.19) impose the machines to be fully operative at the beginning of the planning horizon, but are not necessary for the correctness of the model. Note that one could use an additional set of constraints to prohibit the scheduling of two consecutive maintenance activities on a machine. This would potentially reduce symmetry in the search space as a solution might have several "duplicates" with unnecessary maintenance events. For example, we described in Figure A.1 an optimal solution with makespan 41 that schedules a maintenance event on machine 1 between jobs 1 and 3. This solution has 9 duplicates with the same makespan in which γ maintenance events ($\gamma = 2, \ldots, 10$) are scheduled between the two jobs.

In the following, we provide a property about the optimal scheduling structure derived

from Lemma 1 in Ruiz-Torres et al. (2017), we explain how to compute good values for the big-M coefficients, and we outline the weaknesses of the model.

PROPERTY 1 *There is always an optimal solution in which a job j is scheduled in a position h of a machine i in such a way that $(\delta_{ih}-1)p_{ij} \leq t_i$ holds.*

Proof. Consider a solution S in which a job j is scheduled in slot h on machine i such that $(\delta_{ih}-1)p_{ij} > t_i$, and let Q be the time required by machine i to process all the activities until it has finished job j. The alternative solution S' in which machine i schedules a maintenance in slot h and processes job j in slot $h+1$ takes $Q' = Q - \delta_{ih}p_{ij} + t_i + p_{ij}$. It follows that $Q - Q' = (\delta_{ih}-1)p_{ij} - t_i$, which is strictly positive according to the initial condition. Finally, because all delay factors d_{ij} are greater than or equal to 1, we can simply show that the accumulated delay factor in the original solution S after processing job j in slot h, which is $\delta_{i,h+1} = \delta_{ih}d_{ij}$, cannot be smaller than the accumulated delay factor in solution S' after performing a maintenance in slot h and processing job j in slot $h+1$, which is $\delta_{i,h+2} = d_{ij}$. As a result, because the two solutions S and S' have the same job ordering but S' completes j sooner and with a lower accumulated delay factor, then S' is at least as good as S. □

An interesting implication of Property 1 is that there is always an optimal solution that satisfies $\delta_{ih} \leq 1 + \frac{t_i}{p_{ij}}$, $\forall i \in M, h \in H : s_{ih} = 0$. This allows us to define the big-M values as:

$$K^a_{ij} = p_{ij} \times \min\Big\{ \prod_{j' \in J} d_{ij'}, (1 + \frac{t_i}{\min_{j'}\{p_{ij'}\}}) \times \max_{j'}\{d_{ij'}\} \Big\} \quad i \in M, j \in J, \tag{A.25}$$

$$K^b_{ij} = d_{ij} \times \min\Big\{ \prod_{j' \in J} d_{ij'}, (1 + \frac{t_i}{\min_{j'}\{p_{ij'}\}}) \times \max_{j'}\{d_{ij'}\} \Big\} \quad i \in M, j \in J. \tag{A.26}$$

Indeed, we know that, in order to deactivate constraints (A.15) when variables x_{ijh} take value 0, we need coefficients K^a_{ij} to be greater than or equal to $p_{ij}\delta_{ih}$. An obvious upper bound on δ_{ih} is the product of the delay factors of all jobs. Thanks to Property 1, a refined upper bound of δ_{ih} when a job is scheduled in position h (i.e., when $s_{ih} = 0$) is $(1+\frac{t_i}{\min_{j'}\{p_{ij'}\}})$. By multiplying this value by $\max_{j'}\{d_{ij'}\}$, we obtain an upper bound which is also valid when $s_{ih} = 1$. A similar reasoning can be applied to determine coefficients K^b_{ij}.

The linearized position-based model involves a polynomial number $O(|M||J|^2)$ of variables and a polynomial number $O(|M||J|^2)$ of constraints. However, it also has a large amount of symmetry: indeed, a subset of jobs processed between two maintenance activities can be exchanged with another subset of jobs processed between two other maintenance activities on the same machine without affecting the makespan. In the following, we introduce an event-based formulation aimed at palliating this issue.

A.4.3 Event-based formulation

The event-based formulation uses the notion of a *block*. A block is a sequence of jobs that occurs either before the first maintenance activity, or between two consecutive maintenance activities, or after the last maintenance activity. The model decides on which machine a job is scheduled, and whether a job initiates a block or is positioned after another job. The decisions regarding the ordering of the blocks within the same machine do not have any impact on the makespan and are thus removed from the model.

We define j_0 as a dummy job of processing time 0 and delay factor 1 that initiates each block. We use a binary variable $x_{ijj'}$ that takes the value 1 if job j' is scheduled right after job j on machine i, and 0 otherwise. For each machine, we identify sequences of jobs that can be disregarded by the model (as they are not necessary to reach an optimal solution) and store them in set $\mathcal{S}$. Continuous variables δ_j and a_{ij} now define the accumulated delay factor before starting job j and the actual processing time of job j on machine i, respectively. The event-based formulation is as follows:

$$\min \quad \mathrm{C}_{\max} \tag{A.27}$$

$$\sum_{i\in M}\sum_{j\in J\cup j_0} x_{ijj'} = 1 \qquad \forall j' \in J, \tag{A.28}$$

$$\sum_{j\in J\cup j_0} x_{ijj'} \geq \sum_{j''\in J} x_{ij'j''} \qquad \forall j' \in J, i \in M, \tag{A.29}$$

$$a_{ij'} \geq p_{ij'}\delta_{j'} - K^a_{ij'}(1 - x_{ijj'}) \qquad \forall i \in M, j \in J, j' \in J, \tag{A.30}$$

$$a_{ij'} \geq p_{ij'}x_{ij_0j'} \qquad \forall i \in M, j' \in J, \tag{A.31}$$

$$\sum_{j\in J} a_{ij} - t_i + \sum_{j'\in J} t_i x_{ij_0j'} \leq \mathrm{C}_{\max} \qquad \forall i \in M, \tag{A.32}$$

$$\delta_{j'} \geq \delta_j d_{ij} - K^b_{ij}(1 - x_{ijj'}) \qquad \forall i \in M, j \in J, j' \in J, \tag{A.33}$$

$$x_{ijj'} = 0 \qquad \forall (i, j, j') \in \mathcal{S} \tag{A.34}$$

$$\delta_{j'} \geq 1 \qquad \forall j' \in J, \tag{A.35}$$

$$x_{ijj'} \in \{0, 1\} \qquad \forall i \in M, j \in J \cup j_0, j' \in J, \tag{A.36}$$

$$a_{ij'} \geq 0 \qquad \forall i \in M, j' \in J, \tag{A.37}$$

$$\mathrm{C}_{\max} \geq 0. \tag{A.38}$$

Constraints (A.28) force each job to be scheduled on a machine, either after another job or after a dummy job j_0. Constraints (A.29) allow a job j'' to be scheduled after job j' on machine i only if job j' was itself scheduled after another job j or after the dummy job j_0 on machine i. Constraints (A.30) and (A.31) are used to define the actual processing time of job j' on machine i. While the former constraints are activated if j' is scheduled after another job, the latter ones are activated if j' is scheduled after the dummy job j_0. Constraints (A.32) are

used to calculate the makespan. Note that the number of maintenance activities scheduled on a given machine is equal to the number of blocks scheduled on that machine (i.e., the number of times a job is scheduled after the dummy job j_0) minus one. Constraints (A.33) compute the accumulated delay factor before starting job j' and constraints (A.34) forbid certain jobs to be scheduled after some others on specific machines. Initially, we forbid each job to be scheduled right after itself on any machine (i.e., $\forall i \in M, j \in J, (i,j,j) \in \mathcal{S}$).

EXAMPLE 1 (resumed) *The values of the variables of the event-based formulation for Example 1 in the solution depicted in Figure 1 are as follows:*

- $x_{101} = 1, \delta_1 = 1, a_{11} = 10$
- $x_{103} = 1, \delta_3 = 1, a_{13} = 10$
- $x_{204} = 1, \delta_4 = 1, a_{24} = 30$
- $x_{242} = 1, \delta_2 = 1.1, a_{22} = 11$

And constraints (A.32) *are as follows:*

- $C_{max} \geq 10 + 10 + 2(2-1) = 22$ *(load on machine 1)*
- $C_{max} \geq 30 + 11 + 5(1-1) = 41$ *(load on machine 2)*

In the following, we provide a property on the optimal scheduling structure that allows further triplets (i,j,j') to be added to set $\mathcal{S}$, so reducing by nearly a half the number of binary variables in the model. This property is derived from Lemma 1 in Ding et al. (2019) and Lemma 1 from Ruiz-Torres et al. (2013), both devoted to the case without maintenance, and was used (but not proved) by Ruiz-Torres et al. (2017) to derive heuristics for the case with maintenance.

PROPERTY 2 *There is always an optimal solution in which two successive jobs j and j' scheduled in the same block on a machine i satisfy the condition*

$$\frac{p_{ij}}{d_{ij}-1} \geq \frac{p_{ij'}}{d_{ij'}-1} \quad \forall i \in M, j, j' \in J : x_{ijj'} = 1. \tag{A.39}$$

Proof. Consider a solution S in which job j is scheduled right after job j' on machine i, in the same block, in such a way that $\frac{p_{ij}}{d_{ij}-1} > \frac{p_{ij'}}{d_{ij'}-1}$ holds. Let Q be the time required by machine i to process all activities until job j is finished. Let us also consider an alternative solution S' in which job j is scheduled on machine i just before job j', and let Q' be the time required to process all activities until job j' is finished. By denoting k the time required to process all jobs before the beginning of job j' (resp., job j) in solution S (resp., solution S'), we can

define Q (resp., Q') as follows:

$$\begin{aligned} Q =& k+p_{ij'}+(\delta_{j'}-1)p_{ij'}+p_{ij}+(\delta_{j'}d_{ij'}-1)p_{ij}, \\ Q' =& k+p_{ij}+(\delta_{j}-1)p_{ij}+p_{ij'}+(\delta_{j}d_{ij}-1)p_{ij'}. \end{aligned}$$

As $\delta_{j'}$ only depends on the set of jobs performed before job j' by machine i since the last maintenance, we know that $\delta_{j'}$ in Q is equal to δ_j in Q'. To ease the notation, we set $\delta_j = \delta_{j'} = \delta$ in the rest of the proof. By computing the difference between the completion times of the two partial schedules, we obtain:

$$\begin{aligned} Q - Q' =& (\delta-1)p_{ij'} + (\delta d_{ij'}-1)p_{ij} - (\delta-1)p_{ij} - (\delta d_{ij}-1)p_{ij'}, \\ =& (\delta-\delta d_{ij})p_{ij'} + (\delta d_{ij'}-\delta)p_{ij}, \\ =& (1-d_{ij})\delta p_{ij'} + (d_{ij'}-1)\delta p_{ij}, \\ =& -(d_{ij}-1)\delta p_{ij'} + (d_{ij'}-1)\delta p_{ij}. \end{aligned}$$

Because of (A.39), we know that $\frac{p_{ij}}{d_{ij}-1} > \frac{p_{ij'}}{d_{ij'}-1}$, or, in other words, as $d_{ij}-1$ and $d_{ij'}-1$ are strictly positive, that $p_{ij}(d_{ij'}-1) > p_{ij'}(d_{ij}-1)$. Thus, as also δ is strictly positive, then $Q-Q'$ is strictly positive as well. One can then notice that the accumulated delay factor after processing job j in solution Q is the same as the accumulated delay factor after processing job j' in solution Q', which is $\delta d_{ij} d_{ij'}$. As a result, since the two solutions have processed the same set of jobs, but solution S' completed it sooner and with the same accumulated delay factor, solution S' is at least as good as solution S. □

Note that, for a given machine, Property 2 is a necessary but not sufficient condition for the optimality of a schedule. That is, there could be some job schedules satisfying condition (A.39) that do not minimize the makespan of the machine (an intuitive example is to schedule all the jobs in the same block). This differs from the version of the problem without maintenance in which, for a given machine, the job schedules minimizing the makespan of the machine always satisfy condition (A.39). We also outline that there may exist several optimal solutions that do not satisfy condition (A.39): indeed, as the makespan only depends on the schedule of the machine(s) with largest workload, it is possible that one or more blocks in machines with smaller workload do not fulfill (A.39). However, Property 2 states that there is always at least one solution in which the job ordering of each block satisfies this condition. As a corollary of Property 2, we obtain:

$$x_{ijj'} = 0 \quad \forall i \in M, \forall j, j' \in J : \frac{p_{ij}}{d_{ij}-1} < \frac{p_{ij'}}{d_{ij'}-1}, \tag{A.40}$$

which allows us to set nearly half of the variables to 0 through constraints (A.34).

The event-based model also involves a polynomial number $O(|M||J|^2)$ of variables and a

polynomial number $O(|M||J|^2)$ of constraints. It has less symmetry since the block ordering decision is removed from the model. However, its linear relaxation is weak due to the big-M constraints. In the following, we introduce a block-based formulation free of big-M constraints.

A.4.4 Block-based formulation

While the event-based formulation decides on which machine a job is scheduled, and whether a job initiates a block or is positioned after another job, the block-based formulation selects from a large set of feasible blocks the ones that minimize the makespan. The decisions regarding the job ordering within each block are determined before running the model in accordance with Property 2. This formulation is a natural extension of the set covering formulation of Gilmore and Gomory (1961, 1963) originally proposed for the cutting stock problem. Some formulations using the notion of blocks were already proposed for other scheduling problems (see, e.g., Pacheco et al. (2013) for a single-machine scheduling problem with set-up times), but not all of these formulations associate a variable to each of the feasible blocks as the model presented in this section does.

We borrow the set covering notation and use $\mathcal{B}_i$ to define both the set of blocks and the set of block indices that can be scheduled on machine i. We also use b to define both a block and its index. Note that sets $\mathcal{B}_i$ and $\mathcal{B}'_i (i, i' \in M, i \neq i')$ are not necessarily identical as shown in the example at the end of this section. The b^{th} block on machine i is described by its duration a^i_b and by an integer array $(\alpha^i_{1b}, \alpha^i_{2b}, \ldots, \alpha^i_{|J|b})$, where α^i_{jb} takes the value 1 if job j is included in the b^{th} feasible block of machine i, and 0 otherwise. We now use binary variables x^i_b that take the value 1 if the b^{th} feasible block of machine i is selected, and 0 otherwise. The block-based formulation is as follows:

$$\min \quad \mathrm{C}_{\max} \tag{A.41}$$

$$\sum_{i \in M} \sum_{b \in \mathcal{B}_i} \alpha^i_{jb} x^i_b = 1 \qquad \forall j \in J, \tag{A.42}$$

$$-t_i + \sum_{b \in \mathcal{B}_i} (a^i_b + t_i) x^i_b \leq \mathrm{C}_{\max} \qquad \forall i \in M, \tag{A.43}$$

$$x^i_b \in \{0, 1\} \qquad \forall i \in M, b \in \mathcal{B}_i, \tag{A.44}$$

$$\mathrm{C}_{\max} \geq 0. \tag{A.45}$$

Constraints (A.42) impose that each job is contained in exactly one of the selected blocks, and constraints (A.43) calculate the makespan, which is minimized in (A.41).

In Algorithm 4, we provide a pseudo-code to exhaustively enumerate every feasible *non-dominated* block. A block b is dominated if there exists a set of blocks S containing the same jobs as b that can be processed in a shorter time (taking into account the duration of the $|S|$ blocks and the $|S| - 1$ maintenance activities in-between blocks). Dominated blocks are not

Algorithm 4 `Create blocks`

1: **for each** $i \in M$ **do** ▷ For each machine i
2: $\mathcal{B}_i \leftarrow \emptyset$, $b = 0$ ▷ There are no blocks in $\mathcal{B}_i$
3: **for each** $j \in J$ **do**
4: $\alpha^i_{jb} \leftarrow 0$ ▷ Create the empty block
5: **end for**
6: $a^i_b \leftarrow 0, \delta^i_b \leftarrow 1$, $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{b\}, b \leftarrow b+1$ ▷ Add the empty block to $\mathcal{B}_i$
7: **for each** $j \in J$ ordered by non-increasing $\frac{p_{ij}}{d_{ij}-1}$ **do** ▷ For each job
8: $\bar{\mathcal{B}}_i \leftarrow \emptyset$ ▷ There are no new blocks
9: **for each** $b' \in \mathcal{B}_i$ **do** ▷ For each existing block in $\mathcal{B}_i$
10: **for each** $j' \in J$ **do**
11: $\alpha^i_{j'b} \leftarrow \alpha^i_{j'b'}$ ▷ Block b is a copy of the existing block b'
12: **end for**
13: $\alpha^i_{jb} \leftarrow 1$, $a^i_b \leftarrow a^i_{b'} + \delta^i_{b'}\ p_{ij}, \delta^i_b \leftarrow \delta^i_{b'}\ d_{ij}$ ▷ add job j to block b and perform a^i_b and δ^i_b
14: **if** `isNotDominated(b)` **then** ▷ if the block is not removed by our reduction criterion
15: $\bar{\mathcal{B}}_i \leftarrow \bar{\mathcal{B}}_i \cup \{b\}, b \leftarrow b+1$ ▷ add block b to the new blocks, and increment b
16: **end if**
17: **end for**
18: **for each** $\bar{b} \in \bar{\mathcal{B}}_i$ **do**
19: $\mathcal{B}_i \leftarrow \cup\{\bar{b}\}$, ▷ add the new blocks to $\mathcal{B}_i$
20: **end for**
21: **end for**
22: **end for**

useful to the model because there is always an optimal solution that does not contain any dominated block. Before describing the algorithm steps, we give its general idea: starting from $\mathcal{B}_i$ containing only the empty block, the algorithm temporarily duplicates every block in $\mathcal{B}_i$ and adds the job currently processed in each duplicate. It then discards the newly generated blocks that are dominated, inserts the remaining ones into $\mathcal{B}_i$ and moves on to the next job. As a result, $\mathcal{B}_i$ contains at most 2^j blocks at step j.

For each machine i, the algorithm sets the block counter to 0 (step 2), creates the dummy empty block of duration 0 and accumulated delay factor 1, and adds it to the set of blocks of machine i (steps 3-6). Then, for each job j (preliminary re-ordered by non-increasing $\frac{p_{ij}}{d_{ij}-1}$ values), it creates a temporary new set $\bar{\mathcal{B}}_i$ in which it stores the new blocks generated when adding job j (step 8). After this, each original block in $\mathcal{B}_i$ is copied, one at a time, into block b (steps 9-12). The algorithm adds job j to block b and computes its duration and its accumulated delay factor (step 13). Then, the algorithm determines if block b is dominated or not in the way described below. If block b is not dominated, then it is added to the new blocks (step 15) and the block counter is incremented. If, instead, block b is dominated, then it will simply be overwritten during the next loop. Finally, the algorithm adds every newly generated non-dominated block from $\bar{\mathcal{B}}_i$ into $\mathcal{B}_i$ (steps 18-20) and moves on to the next item j.

Assessing whether a block is dominated or not is not trivial. An easy criterion can be derived through Property 1, by checking that the duration of the new block a^i_b is strictly

shorter than the duration of the old block $a^i_{b'}$ plus the ideal processing time of job j plus the duration of a maintenance activity t_i. If it is not the case, then it is shorter to perform a maintenance activity before processing j and block b is dominated (and so, it does not need to be generated). The more powerful test that we implemented consists in trying, in turns, to insert a maintenance activity between each pair of consecutive jobs and comparing the sum of the duration of these two blocks plus the duration of a maintenance activity with the duration of the new block a^i_b. If a^i_b is longer, then it is more advantageous to use two blocks separated by a maintenance activity instead of block b, and thus it is not necessary to generate b as it is dominated. Note that this test was also used in the heuristic of Ruiz-Torres et al. (2017) in order to improve an incumbent solution.

EXAMPLE 1 (resumed) *The blocks generated by Algorithm 4 for Example 1 are presented in Table A.2. An optimal solution provided by an ILP solver selects block 6 in machine 1 and block 9 in machine 2 for a total makespan equal to 41. Note that there exists other optimal solutions with a makespan equal to 41, such as the one obtained by selecting blocks 1 and 2 in machine 1 and block 10 in machine 2.*

Table A.2: Blocks generated by Algorithm 4 for Example 1

Jobs	Id	Machine 1				Machine 2			
		order	a^i_b	δ^i_b	`isNotDominated`	order	a^i_b	δ^i_b	`isNotDominated`
{1}	1	1	10	1.2	true	1	20	1.1	true
{2}	2	2	20	1.1	true	2	10	1.2	true
{3}	3	3	10	1.1	true	3	10	1.1	true
{4}	4	4	30	1.2	true	4	30	1.1	true
{1,2}	5	2,1	31	1.32	true	1,2	31	1.32	true
{1,3}	6	3,1	21	1.32	true	1,3	31	1.21	true
{1,4}	7	4,1	42	1.44	false	4,1	52	1.21	true
{2,3}	8	2,3	31	1.21	true	3,2	21	1.32	true
{2,4}	9	2,4	53	1.32	false	4,2	41	1.32	true
{3,4}	10	4,3	42	1.32	false	4,3	41	1.21	true
{1,2,3}	11	2,3,1	43.1	1.452	false	1,3,2	43.1	1.452	true
{1,2,4}	12	2,4,1	66.2	1.584	false	4,1,2	64.1	1.452	true
{1,3,4}	13	4,3,1	55.2	1.584	false	4,1,3	64.1	1.331	true
{2,3,4}	14	2,4,3	66.2	1.452	false	4,3,2	53.1	1.452	true
{1,2,3,4}	15	2,4,3,1	80.72	1.7424	false	4,1,3,2	77.41	1.5972	true

The block-based model involves a polynomial number of constraints $O(|M| + |J|)$ and an exponential number of variables, which empirically grows very fast as the number of jobs increases (some instances with 100 jobs led to models with more than 100 000 000 variables in our tests). A possible option could be to only generate a subset of variables through column generation and embed the procedure in a branch-and-price algorithm. In the following, we introduce an arc flow formulation whose number of variables still grows exponentially with the number of jobs, but at a lesser extent as shown in the computational experiments section.

A.4.5 Arc flow-based formulation

The arc flow formulation was popularized by Valério de Carvalho (1999) for the CSP, and extensively studied afterwards in several application fields. Recently, it has been applied to cutting and packing (see, e.g., Clautiaux et al. (2018), Dell'Amico et al. (2019), and Delorme and Iori (2020)), production (see, e.g., Nesello et al. (2018a) and Ramos et al. (ming)), and vehicle routing (see, e.g., Clautiaux et al. (2017)). In the scheduling field, arc flow models were developed to solve problems on parallel machines, either to minimize weighted tardiness (Pessoa et al. (2010)) or weighted completion times (Kramer et al., 2019).

In the arc flow-based formulation we propose for the R|Sdd,mnt|C_{max}, the composition of a block is defined as a path in a graph where nodes are intermediary accumulated delay factors and arcs are jobs. Formally, for each machine i, let $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{A}_i)$ be a multigraph where the node set $\mathcal{N}_i$ includes every relevant intermediary accumulated delay factors. A trivial way to define $\mathcal{N}_i$ is to include every z satisfying:

$$z = \prod_{j \in J}((d_{ij} - 1)y_j + 1),$$
$$y_j \in \{0, 1\} \qquad \forall j \in J.$$

Example 1 (resumed) *All the possible intermediary accumulated delay factors for Example 1 are* $\mathcal{N}_1 = \{1, 1.1, 1.2, 1.21, 1.32, 1.44, 1.452, 1.584, 1.7424\}$ *and* $\mathcal{N}_2 = \{1, 1.1, 1.2, 1.21, 1.32, 1.331, 1.452, 1.5972\}$.

To correctly represent a block, each path must start at the source node $s = 1$. There is no restriction on the node in which a path must terminate. Arcs in $\mathcal{A}_i$ are in the form (e, f, j) and have a given duration τ necessary to calculate the makespan, where j ($j \in J$) is the job index, e ($e \in \mathcal{N}_i$) is the intermediary accumulated delay factor at the beginning of job j, f ($f \in \mathcal{N}_i, f = e\ d_{ij}$) is the intermediary accumulated delay factor at the end of job j, and τ is obtained by multiplying p_{ij} (the ideal processing time of job j on machine i) by e (the accumulated delay factor at the beginning of the arc). By using a binary variable x^i_{efj} that takes the value 1 if arc (e, f, j) is selected on machine i, and 0 otherwise, the arc flow-based formulation is as follows:

$$\min \quad C_{max} \tag{A.46}$$
$$\sum_{i \in M} \sum_{\substack{(e,f,j) \in \mathcal{A}_i \\ j = j'}} x^i_{efj} = 1 \qquad \forall j' \in J, \tag{A.47}$$
$$\sum_{\substack{(e,f,j) \in \mathcal{A}_i \\ e = k}} x^i_{efj} \leq \sum_{\substack{(e,f,j) \in \mathcal{A}_i \\ f = k}} x^i_{efj} \qquad \forall i \in M, \forall k \in \mathcal{N}_i \setminus \{s\}, \tag{A.48}$$

$$-t_i + t_i \sum_{\substack{(e,f,j)\in\mathcal{A}_i \\ e=s}} x^i_{efj} + \sum_{(e,f,j)\in\mathcal{A}_i} e\, x^i_{efj}\, p_{ij} \leq \mathrm{C}_{\max} \qquad \forall i \in M, \tag{A.49}$$

$$x^i_{efj} \in \{0,1\} \qquad \forall i \in M, (e,f,j) \in \mathcal{A}_i, \tag{A.50}$$

$$\mathrm{C}_{\max} \geq 0. \tag{A.51}$$

Constraints (A.47) force exactly one arc associated with each job to be selected. Constraints (A.48) ensure flow conservation, or, in other words, that the number of jobs starting with accumulated delay factor k ($k \in \mathcal{N}_i \setminus \{s\}$) is smaller than or equal to the number of jobs ending with accumulated delay factor k. Constraints (A.49) calculate the makespan by summing, for each machine, the duration of the selected arcs with the time required for the maintenance activities (if we count one maintenance for each selected arc originating from the source node, we should remove one extra maintenance). In Algorithm 5, we provide a procedure to generate all the feasible arcs.

Algorithm 5 `Create arcs`

for each $i \in M$ **do** ▷ For each machine i
  $\mathcal{A}_i \leftarrow \emptyset$, $\mathcal{N}_i \leftarrow s$ ▷ There are no arcs in $\mathcal{A}_i$ and only the source node in $\mathcal{N}_i$
  for each $j \in J$ ordered by non-increasing $\frac{p_{ij}}{d_{ij}-1}$ **do** ▷ For each job
    $\bar{\mathcal{N}}_i \leftarrow \emptyset$ ▷ There are no new nodes
    for each $k \in \mathcal{N}_i$ **do** ▷ For each existing node in $\mathcal{N}_i$
      if `isNotDominated`$(k, d_{ij}k, j)$ **then** ▷ if the arc is not removed by our reduction criterion
        $\mathcal{A}_i \leftarrow (k, d_{ij}k, j)$ ▷ Add the new arc to $\mathcal{A}_i$
        $\bar{\mathcal{N}}_i \leftarrow d_{ij}k$ ▷ Add the new node to $\bar{\mathcal{N}}_i$
      end if
    end for
    for each $\bar{k} \in \bar{\mathcal{N}}_i$ **do**
      $\mathcal{N}_i \leftarrow \cup\{\bar{k}\}$, ▷ add the new nodes to $\mathcal{N}_i$
    end for
  end for
end for

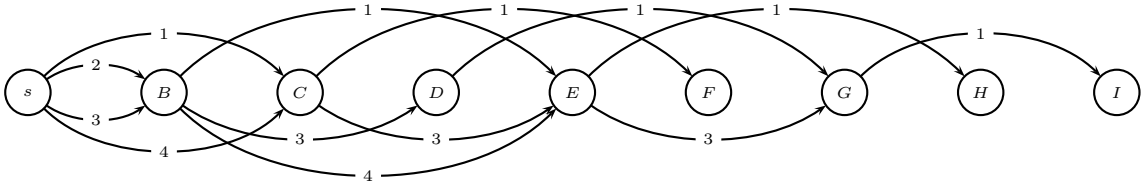
The arc generation algorithm is similar to the block generation algorithm. However, assessing whether an arc is dominated or not is more challenging. There is still an easy criterion which can be derived from Property 1, by checking that the duration of each new arc $(k, d_{ij}k, j)$ is strictly smaller than the ideal processing time of job j plus the duration of a maintenance activity (i.e., $p_{ij}k < p_{ij} + t_i$). If it is not the case, then it is shorter to perform a maintenance activity before processing j at intermediary accumulated delay factor k and arc $(k, d_{ij}k, j)$ does not need to be generated as it is dominated.

Example 1 (resumed) *The arcs generated by Algorithm 5 for Example 1 are displayed in Figures A.2 and A.3. The first set of nodes, $s, B, \ldots, G$, corresponds to each intermediary accumulated factor in $\mathcal{N}_1$ for machine 1, which are $1, 1.1, \ldots, 1.7424$. The second set of nodes, $s', B, \ldots, H'$, corresponds to each intermediary accumulated factor in $\mathcal{N}_2$ for machine 2, which*

are $1, 1.1, \ldots, 1.5972$.

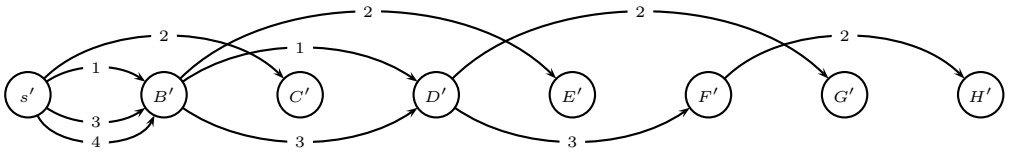
We now briefly explain how the graph in Figure 2 was obtained. Before processing the first item, the set of nodes $\mathcal{N}_1$ *only contains* s*, which corresponds to intermediary accumulated factor 1. After processing job 2, Algorithm 2 creates arc* $(s, B, 2)$*, where* B *corresponds to intermediary accumulated factor 1.1, and adds* B *to* $\mathcal{N}_1$*. After processing job 4, Algorithm 2 creates arcs* $(s, C, 4)$ *and* $(B, E, 4)$*, where* C *corresponds to intermediary accumulated factor 1.2 and* E *to 1.32 (1.1* $\times$ *1.2), and adds* C *and* E *to* $\mathcal{N}_1$*. After processing job 3, Algorithm 2 creates arcs* $(s, B, 3)$*,* $(B, D, 3)$*,* $(C, E, 3)$*, and* $(E, G, 3)$*, where* D *corresponds to intermediary accumulated factor 1.21 (1.1* $\times$ *1.1) and* G *to 1.452 (1.32* $\times$ *1.1), and adds* D *and* G *to* $\mathcal{N}_1$*. Note that nodes* E *and* B *were already contained in* $\mathcal{N}_1$ *and do not need to be added again. This gives an intuition of the reason why the number of variables grows at a lesser extent in the arc flow-based formulation than it does in the block-based formulation. Finally, after processing job 1, Algorithm 2 creates arcs* $(s, C, 1)$*,* $(B, E, 1)$*,* $(C, F, 1)$*,* $(D, G, 1)$*,* $(E, H, 1)$*, and* $(G, I, 1)$*, where* F *corresponds to intermediary accumulated factor 1.44 (1.2* $\times$ *1.2),* H *to 1.584 (1.32* $\times$ *1.2), and* I *to 1.7424 (1.452* $\times$ *1.2), and adds* F*,* H*, and* I *to* $\mathcal{N}_1$*.*

Figure A.2: Arcs generated for Machine 1 in Example 1 without reduction procedure



The block containing jobs 1 and 2 can be obtained on machine 1 by selecting the path containing arcs $(s, B, 2)$ *and* $(B, E, 1)$*, for a duration of* $20 \times 1 + 10 \times 1.1 = 31$*. The block containing jobs 2, 3, and 4 can be obtained on machine 1 by selecting the path containing arcs* $(s, B, 2)$*,* $(B, E, 4)$*, and* $(E, G, 3)$*, for a duration of* $20 \times 1 + 30 \times 1.1 + 10 \times 1.32 = 66.2$*. Note that the application of the easy reduction procedure would prevent arcs initiating at node* C *or at subsequent nodes to be generated.*

Figure A.3: Arcs generated for Machine 2 in Example 1 without reduction procedure



The block containing jobs 1 and 2 can be obtained on machine 2 by selecting the path containing arcs $(s', B', 1)$ *and* $(B', E', 2)$*, for a duration of* $20 \times 1 + 10 \times 1.1 = 31$*. The block containing jobs 2, 3, and 4 can be obtained on machine 2 by selecting the path containing arcs*

$(s', B', 4)$, $(B', D', 3)$, *and* $(D', G', 2)$, *for a duration of* $30 \times 1 + 10 \times 1.1 + 10 \times 1.21 = 53.1$. *Note that applying the easy reduction procedure would not prevent any arc to be generated.*

An optimal solution provided by an ILP solver selects arcs $(s, B, 3)$ *and* $(B, E, 1)$ *in machine 1 and arcs* $(s', B', 4)$ *and* $(B', E', 2)$ *in machine 2 for a total makespan of 41. There exists other optimal solutions with a makespan equal to 41, such as the one obtained by selecting arcs* $(s, B, 3)$ *and* $(s, C, 1)$ *in machine 1 and arcs* $(s', B', 4)$ *and* $(B', E', 2)$ *in machine 2.*

In the following, we propose a significantly more powerful reduction criterion by assigning an "expiration date" to each node. Before explaining in details the concept of expiration date, we first need to observe that, as $k \geq 1$,

$$(d_{ij} - 1) + k \leq (d_{ij} - 1)k + k = d_{ij}k,$$

or, in other words, that by approximating $d_{ij}k$ with $(d_{ij} - 1) + k$, we obtain a lower bound on the accumulated delay factor. This approximation has a nicer mathematical structure, because it removes the multiplicative aspect of the delay factors. We can now predict that, by processing a set S of jobs whose sum of ideal processing times is l (i.e., $\sum_{j \in S} p_{ij} = l$) after a set of jobs S' whose accumulated delay factor is k (i.e., $\prod_{j \in S'} d_{ij} = k$), then at least $l(k-1)$ additional time can be solely imputed to the machine deterioration caused by the jobs in S'. As a result, if $l(k-1)$ is greater than or equal to the time necessary to perform a maintenance activity t_i (i.e., if $l \geq \frac{t_i}{(k-1)}$), then it is better to schedule a maintenance activity between S' and S than to schedule the two sets of jobs successively one after the other.

The "natural" expiration date of a node on machine i is equal to $\frac{t_i}{(k-1)}$. When we create a new node $d_{ij}k$ through arc $(k, d_{ij}k, j)$, we compare its natural expiration date $\frac{t_i}{(d_{ij}k-1)}$ with the "real" expiration date of its predecessor (i.e., node k) minus the ideal processing time of job j and we keep the minimum among the two values. If node $d_{ij}k$ was already defined, then its expiration date becomes the maximum between its current expiration date and the expiration date it would have had if the node was just created.

Example 2 *Let us consider an instance with a single machine having maintenance time* $t_1 = 6$, *and seven identical jobs having ideal processing time* $p_{1j} = 100$ *and delay factor* $d_{1j} = 1.01$, *for* $(j = 1, \ldots, 7)$. *After creating the initial node s, we process each job in turn and obtain:*

- *node* $n_1 = 1.01$, *expiration date* $= \frac{6}{1.01-1} = 600$
- *node* $n_2 = 1.01^2$, *expiration date* $= \min\{600 - 100, \frac{6}{1.01^2-1}\} \approx 298.507463$
- *node* $n_3 = 1.01^3$, *expiration date* $= \min\{298.507463 - 100, \frac{6}{1.01^3-1}\} \approx 198,0132669$
- *node* $n_4 = 1.01^4$, *expiration date* $= \min\{198.0132669 - 100, \frac{6}{1.01^4-1}\} \approx 98.0132669$

No further nodes need to be created because the remaining jobs are longer than the expiration date. Note that if we only applied the reduction criterion from Property 1 we would have also created node 1.01^5*. An optimal solution schedules the first four jobs on the machine followed by a maintenance activity and then the last three jobs, for a total duration time of 715.0501. The selected arcs are* $(s, n_1, 1)$*,* $(n_1, n_2, 2)$*,* $(n_2, n_3, 3)$*,* $(n_3, n_4, 4)$*,* $(s, n_1, 5)$*,* $(n_1, n_2, 6)$*,* $(n_2, n_3, 7)$*.*

The arc flow-based model involves an exponential number of constraints $O(\sum_{i \in M} |\mathcal{N}_i|)$ and an exponential number of variables $O(\sum_{i \in M} |\mathcal{A}_i|)$. Empirically, these numbers grow at a reasonable pace as the number of jobs increases, because (i) the reduction criterion removes a significant number of arcs and nodes, and (ii) only a few supplementary nodes need to be created when generating the arcs for job j if the arcs of another job j' with similar delay factor (i.e., $d_{ij} = d_{ij'}$) were generated previously. Despite its scalability, the model becomes too large for instances with 10 machines and 200 jobs. In the next section, we introduce a metaheuristic able to find good quality solutions for large size instances.

A.5 Metaheuristic algorithm

In this section, we describe the ILS algorithm that we developed to look for high-quality solutions in quick time as the ILP formulations can be long to provide a feasible solution, especially for large size instances. The ILS uses a greedy algorithm to build an initial solution, and five local search procedures to improve it.

In the following, we call S a set of jobs, and (S, i) the block obtained on machine i if it had to process the jobs in S in the order determined by Property 2. The greedy procedure uses the notion of "temporary blocks", which are blocks that may be changed later in the algorithm, and the notion of "final blocks", which are blocks that are parts of the initial solution. The greedy procedure works as follows. First, we assign an empty set S_i of jobs to each machine $i \in M$. Then, for each job j, we determine the machine i whose temporary block is the best fitted to perform j, (i.e., the one for which $(S_i \cup \{j\}, i)$ has minimum duration), and add j to S_i. After that, we try to form a temporary block (S_i, i). If the block satisfies Property 1, we simply continue the job assignment. If, instead, the block does not satisfy the property, then (S_i, i) is split into two smaller blocks, (S'_i, i) and (S''_i, i), separated by a maintenance activity. We consider the first block (S'_i, i) as a final block and update job set S_i so that it now contains only the jobs in S''_i. Once all the jobs have been processed, we transform all the remaining temporary blocks S_i into final blocks.

The initial solution is improved by means of the following five local search procedures:

1. `Swap blocks:` Swap two blocks (S_1, i) and (S_2, i') to obtain (S_1, i') and (S_2, i). Machine i is the machine with the largest workload, and block (S_1, i) is the block of machine i with the largest duration. Block (S_2, i') is chosen randomly from another machine. The procedure performs the first swap that improves the makespan, if any, so working

with a *first improvement* policy. It tries to swap block (S_1, i) with at most $\bar{\Gamma}_1$ other blocks (S_2, i').

2. `Move job intra:` Select two blocks (S_1, i) and (S_2, i) on the same machine, and move one job j from the first block to the second one, so as to obtain blocks $(S_1 \setminus \{j\}, i)$ and $(S_2 \cup \{j\}, i)$. The procedure works with first improvement policy and tries to move at most $\bar{\Gamma}_2$ jobs j. At each iteration, machine i, job j, and the two blocks S_1 and S_2 are chosen randomly.

3. `Swap jobs intra:` Select two blocks (S_1, i) and (S_2, i) on the same machine, and swap two jobs $j \in S_1$ and $j' \in S_2$, so as to obtain blocks $(S_1 \cup \{j'\} \setminus \{j\}, i)$ and $(S_2 \cup \{j\} \setminus \{j'\}, i)$. The procedure works with first improvement policy and tries to swap at most $\bar{\Gamma}_3$ jobs j and j'. At each iteration, machine i, jobs j and j', and the two blocks S_1 and S_2 are chosen randomly.

4. `Move block:` Select the block with the largest duration from the machine with the largest workload (S_1, i), and move it to another machine i' so as to create block (S_1, i'). The procedure works in first improvement, and tries to move block S_1 to at most $\bar{\Gamma}_4$ machines. At each iteration, the machine i' with smallest workload (and that was not chosen in a previous iteration) is selected.

5. `Move job inter:` Select two blocks (S_1, i) and (S_2, i') on two distinct machines, and move one job j from the first set to the second one, so as to obtain blocks $(S_1 \setminus \{j\}, i)$ and $(S_2 \cup \{j\}, i')$. The procedure finishes after a given number of iterations $\bar{\Lambda}_5$. The procedure stops as soon as the makespan is improved and tries to move at most $\bar{\Gamma}_5$ jobs j. At each iteration, machines i and i', job j, and the two blocks S_1 and S_2 are chosen randomly.

The local search procedures are called one after the other. Each procedure i ($i = 1, \ldots, 5$) is called $\bar{\Lambda}_i$ times. Once the local search phase is terminated, we call the following perturbation procedure in order to diversify the solution:

- `Perturb:` Randomly remove 20% of the blocks and assign, one at a time, each of the removed jobs to the block with smallest accumulated delay factor in the machine with smallest workload.

In each of the aforementioned procedures, it is possible that a block has to be split into two smaller blocks if, after the job reordering from Property 2, the algorithm determines that a maintenance activity should be performed within the block because of Property 1. The metaheuristic stops once 20 local search/perturbation cycles were performed without any improvement in the incumbent solution. Preliminary experiments were used to determine the following parameters values: $\bar{\Gamma}_1 = 5$, $\bar{\Gamma}_2 = 10$, $\bar{\Gamma}_3 = 5$, $\bar{\Gamma}_4 = \frac{|M|}{2}$, $\bar{\Gamma}_5 = 10$, $\bar{\Lambda}_1 = 10$, $\bar{\Lambda}_2 = 30$,

$\bar{\Lambda}_3 = 30$, $\bar{\Lambda}_4 = 10$, and $\bar{\Lambda}_5 = 30$. This approach is different from the heuristics proposed by Ruiz-Torres et al. (2017), as (1) we use a perturbation component that prevents the objective function to get stuck in a local optimum, (2) our procedure includes a random component to avoid repeating a specific local search move in case the same solution is reached twice during the search procedure, and (3) our moves are suitable for the general case of non-identical machines, while the constructive heuristics of Ruiz-Torres et al. (2017) use the fact that the machines are identical.

A.6 Computational experiments

To test the performance of the methods we proposed for the R|Sdd,mnt|C_{max}, we generated a set of instances by adopting the parameters already used by Ruiz-Torres et al. (2017), namely: number of machines m taking value 2, 5, 10, or 20; ratio of jobs per machine n/m taking value 10, 15, or 20; delay factor d_{ij} uniformly distributed in the range $\{1.01, 1.02, \ldots, 1.06\}$ or in the range $\{1.05, 1.06, \ldots, 1.10\}$; integer duration t_i of a maintenance activity i uniformly distributed in either $[1, 3]$ or in $[1, 9]$. For each of the $4 \times 3 \times 2 \times 2 = 48$ configurations, we generated 10 instances resulting in 480 instances in total. In each of the 480 instances, the (integer) ideal processing time of a job p_{ij} was uniformly distributed in range $[1, 100]$. In our experiments, the ideal processing time of a job was the same on every machine (i.e., $p_{ij} = p_{i'j}$, $\forall i, i' \in M, j \in J$), but, unlike Ruiz-Torres et al. (2017), the durations of the maintenance activities and of the delay factors were machine-dependent. This decision was taken in order to make the machines unrelated, while preventing trivial decision making (e.g., if $p_{11} = 1$ and $p_{21} = 100$, then it is unlikely that job 1 gets assigned to machine 2). All our instances can be downloaded at https://github.com/mdelorme2/Scheduling_Sequence_Dependent_Deterioration_Maintenance_Events_Data. All our algorithms were coded in `C++`. The experiments were run on an Intel Xeon E5-2680W v3, 2.50GHz with 192GB of memory, running under Scientific Linux 7.5, and Gurobi 7.5.2 was used to solve the MILP models. Each instance was run using a single core with a time limit of 3600 seconds. We do not compare our models and metaheuristic with other approaches from the literature, because previous works did not consider maintenance (as in Santos and Arroyo Santos and Arroyo (2017) or Ding et al. (2019)) or were designed for identical machines (as in Ruiz-Torres et al. (2017)). We mention, however, that an adaptation of the approach in Ruiz-Torres et al. (2017) developed to handle non-identical machines was investigated in a preliminary version of this work (see Mendes and Iori (2019)), but turned out to be computationally outperformed by a preliminary version of our metaheuristic.

A.6.1 Computational results on small size instances

We first tested each of the four MILP models and the metaheuristic on the 120 randomly generated instances with 2 machines, and we provide detailed results in Tables A.3–A.6. In each table, column "Method" identifies the approach used, column "# opt." gives the number of proven optimal solutions found by the model, column "TT" gives the average execution time (in seconds) required to solve an instance (including those terminated by the time limit), column "TP" indicates the average time (in seconds) required to build the model, columns "LB" and "UB" indicate, respectively, the average lower and upper bounds produced, columns "nb. var.", "nb. cons.", and "nb. nzs" give, respectively, the average number of variables, constraints, and non-zero elements in the MILP models, column "LP" gives the average LP-relaxation of the model, and column "gap" gives the average relative gap computed as $100 \times \frac{UB-LB}{UB}$, where LB is the lower bound provided by the model and UB is the best solution found by the model.

Table A.3: Evaluation of the proposed approaches on instances with 2 machines. Arc flow I is without expiration data. Arc flow II is with expiration data

method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs	LP	gap
Slot	0	3597.9	0	756.37	795.84	7602	7685	44 448	0	4.4
Event	32	2791	0	786.57	791.49	2084	4079	17 582	0	0.6
Block	90	1084.6	1.3	791.05	791.09	1 393 975	32	12 154 388	790.9196	0.0
Arc flow I	119	109.3	0	791.08	791.09	24 889	11 094	99 493	790.9193	0.0
Arc flow II	120	60.3	0	791.09	791.09	6044	2865	24 113	790.9193	0.0
ILS	-	0.5	-	-	797.75	-	-	-	-	-

We observe that the MILP models that use big-M constraints displayed poor performance. We opted to keep the two models in our computational experiments because (1) they are natural extensions of other models proposed in the literature for the R|Sdd,mnt|C_{max} without maintenance, and (2) they outline how computationally difficult the problem is in practice. The position-based model could not solve a single instance to optimality within an hour (relative gap at 4.4% on average), and the event-based model only solved 32 instances in total (relative gap at 0.6% on average). This can be explained by the very poor quality of their continuous relaxation (having value 0), which is due to the disjunctive constraints. The block-based formulation obtained relatively good results, as it could solve 90 instances (relative gap at 0.005% on average), but the large number of variables it involved (more than a million on average), indicates that it is not a viable option for solving larger instances. The arc flow-based formulation solved all the instances in one minute on average, requiring only 6000 variables on average, and thus appearing as a good candidate for solving larger instances. The continuous relaxation of both the arc flow-based and the block-based formulations are very good, but are almost never equal to an optimal solution. The tiny difference between the continuous relaxation of the two models can be explained by the fact that a block cannot

contain twice the same job in the block-based formulation, while this can happen in the arc flow-based formulation. A similar behavior was already noticed for the bin packing problem (see Delorme and Iori (2020)).

For comparison purposes, we also ran the arc flow-based formulation without expiration dates. Even though the results we obtained were competitive (119 instances solved to proven optimality with an average running time of 2 minutes), using expiration dates is strictly better as it reduces by approximately 75% the model size. We thus adopted the expiration dates in all subsequent tests.

The metaheuristic was extremely fast (less than a second on average), and produced good quality solutions as the absolute gap with respect to the optimal solution was less than 7 on average. In the following, we provide more detailed results in which the instances are grouped by ratio of jobs per machine (Table A.4), delay factor range (Table A.5), and maintenance activity duration range (Table A.6).

Table A.4: Evaluation on instances with 2 machines, results grouped by n/m ratio

n/m	method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs
	Position	0	3597.6	0	499.1	516.4	3109	3162	17 888
	Event	20	2135.3	0	513.2	514.8	901	1742	7322
10	Block	40	138.0	0	514.8	514.8	16 415	22	108 220
	Arc flow	40	3.2	0	514.8	514.8	1475	801	5857
	ILS	-	0.4	-	-	520.4	-	-	-
	Position	0	3596.7	0	791.0	819.4	7069	7152	41 248
	Event	9	2853.9	0	811.5	816.1	1951	3812	16 382
15	Block	29	1189.4	0.4	815.7	815.8	461 919	32	3 846 625
	Arc flow	40	48.0	0	815.7	815.7	4958	2556	19 768
	ILS	-	0.5	-	-	822.1	-	-	-
	Position	0	3599.3	0	979.1	1051.7	12 629	12 742	74 208
	Event	3	3383.9	0	1035.0	1043.6	3401	6682	29 042
20	Block	21	1926.4	3.5	1042.7	1042.8	3 703 590	42	32 508 319
	Arc flow	40	129.7	0	1042.8	1042.8	11 699	5236	46 714
	ILS	-	0.7	-	-	1050.8	-	-	-

The results from Table A.4 indicate that instances become more difficult to solve as the ratio of jobs per machine n/m increases. This is particularly evident for the block-based formulation, which solved all the instances with $n/m = 10$ but only half of the instances with $n/m = 20$, and for the arc flow-based formulation, which solved all instances with $n/m = 10$ in 3 seconds on average but took more than 2 minutes on average to solve instances with $n/m = 20$. This is not surprising because the number of feasible blocks and the number of relevant intermediary accumulated factors increases with the number of jobs, resulting in an increase in the number of variables for both formulations. The big-M formulations too obtained better results for $n/m = 10$ (absolute gaps – computed as the difference between the upper bound and the lower bound found by the model – around 17 for the position-based and 1.5 for the event-based) than for $n/m = 20$ (absolute gaps around 70 for the position-based

and 8.5 for the event-based). The metaheuristic was not particularly impacted by the ratio of jobs per machine, as it obtained an absolute gap with respect to the optimal solution around 6 for $n/m = 10$ and equal to 8 for $n/m = 20$.

Table A.5: Evaluation on instances with 2 machines, results grouped by delay factor range

d_{ij} range	method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs
[1.01,1.06]	Position	0	3597.4	0	740.49	784.51	7602	7685	44 448
	Event	6	3369.1	0	774.05	780.42	2084	4079	17 582
	Block	35	1797.7	2.6	779.90	779.98	2 744 176	32	24 036 514
	Arc flow	60	110.4	0	779.96	779.96	11 023	5196	44 032
	ILS	-	0.5	-	-	788.71	-	-	-
[1.05,1.10]	Position	0	3598.4	0	772.26	807.16	7602	7685	44 448
	Event	26	2213.0	0	799.09	802.57	2084	4079	17 582
	Block	55	371.5	0	802.20	802.21	43 773	32	272 262
	Arc flow	60	10.2	0	802.21	802.21	1064	533	4194
	ILS	-	0.6	-	-	806.79	-	-	-

The results displayed in Table A.5 show that instances are more difficult to solve when the jobs have smaller delay factors d_{ij}. This is true for all models (e.g., the block-based formulation could solve 35 instances with small delay factors, while it could solve 55 instances with large delay factor) and also for the metaheuristic (as the absolute gap with respect to the optimal solution is around 9 for low delay factors while it is around 4.5 for large delay factors). This behaviour is expected for the arc flow-based and the block-based formulations, because the number of jobs per block decreases as the delay factors of the jobs increase, thus resulting in models in fewer variables.

We observe in Table A.6 that instances are more difficult to solve when the maintenance activities t_i have longer duration. This is evident for all models (e.g., the arc flow-based formulation could solve all instances with shorter maintenance duration in 14.1 seconds on average, while it took 40 seconds on average to solve instances with longer maintenance duration) and the metaheuristic. This can be explained by the fact that the number of jobs per block increases as the maintenance activity duration increases, resulting in additional variables for the arc flow-based and the block-based formulations.

A.6.2 Computational results on medium and large size instances

We tested the arc flow-based formulation and the ILS on the remaining instances having 5, 10, and 20 machines. We display the results we obtained in Tables A.7–A.9. In each table, column "group" indicates the criterion used to aggregate the instances into groups, and column "# inst." indicates the number of aggregated instances per group. The other columns are identical to those displayed in the previous tables. We opted not to report the relative gaps in this section as the quality of the lower bound obtained by the arc flow-based model was inconsistent and could even reach 0 sometimes when the model was not able to solve the

Table A.6: Evaluation on instances with 2 machines, results grouped by maintenance duration t_i

t_i range	method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs
[1,3]	Position	0	3598.7	0	765.25	786.91	7602	7685	44 448
	Event	21	2416.9	0	782.16	783.60	2084	4079	17 582
	Block	49	915.3	0.1	783.45	783.46	92 834	32	641 860
	Arc flow	60	14.1	0	783.46	783.46	1637	895	6486
	ILS	-	0.6	-	-	787.76	-	-	-
[1,9]	Position	0	3597.1	0	747.50	804.77	7602	7685	44 448
	Event	11	3165.2	0	790.98	799.38	2084	4079	17 582
	Block	41	1253.9	2.6	798.65	798.73	2 695 115	32	23 666 916
	Arc flow	60	106.5	0	798.71	798.71	10 451	4834	41 740
	ILS	-	0.5	-	-	807.74	-	-	-

linear relaxation in an hour. We focus instead on the quality of the upper bounds provided by the two tested approaches.

Table A.7: Evaluation on instances with 5 machines

group	# inst.	method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs
$n/m = 10$	40	Arc flow	1	3600	0.1	515.99	516.24	71 613	29 763	286 201
		ILS	-	2.1	-	-	524.16	-	-	-
$n/m = 15$	40	Arc flow	0	3600	0.3	777.55	777.75	224 008	84 430	895 660
		ILS	-	3.0	-	-	787.20	-	-	-
$n/m = 20$	40	Arc flow	0	3600	0.6	1038.26	1038.46	478 514	164 100	1 913 558
		ILS	-	3.8	-	-	1049.72	-	-	-
$d_{ij} \in [1, 6]$	60	Arc flow	0	3600	0.6	766.14	766.43	491 081	176 263	1 963 948
		ILS	-	2.6	-	-	778.50	-	-	-
$d_{ij} \in [5, 10]$	60	Arc flow	1	3600	0	788.40	788.53	25 010	9266	99 664
		ILS	-	3.3	-	-	795.55	-	-	-
$t_i \in [1, 3]$	60	Arc flow	0	3600	0	772.77	772.91	37 908	16 377	151 257
		ILS	-	3.3	-	-	779.42	-	-	-
$t_i \in [1, 9]$	60	Arc flow	1	3600	0.6	781.77	782.06	478 182	169 152	1 912 356
		ILS	-	2.7	-	-	794.63	-	-	-
Overall	120	Arc flow	1	3600	0.3	777.27	777.48	258 045	92 764	1 031 806
		ILS	-	3.0	-	-	787.03	-	-	-

We observe in Table A.7 that the arc flow-based formulation could only solve one instance to optimality among the 120 instances with 5 machines. However, the average absolute gap between the upper and lower bounds obtained is very small (around 0.2 on average). The metaheuristic was very fast but the solutions it obtained were around 10 units longer on average than those found by the arc flow-based formulation. Interestingly, we only observe a minor impact of the tested parameters on the absolute gaps obtained by the arc flow-based formulation:

- The absolute gap is around 0.2 for each ratio n/m;
- The absolute gap for short delay factors is around 0.3, while it is around 0.1 for long

delay factors;

- The absolute gap for short maintenance duration is around 0.1, while it is around 0.3 for long maintenance duration.

Similarly to what was noticed for instances with 2 machines, we observe that the metaheuristic tends to find better quality solutions for instances with long delay factors and short maintenance duration, while it is not particularly impacted by the average ratio n/m.

Table A.8: Evaluation on instances with 10 machines

group	# inst.	method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs
$n/m = 10$	40	arc flow	0	3600	1.2	516.75	517.51	904 542	314 498	3 617 176
		ILS	-	8.2	-	-	541.67	-	-	-
$n/m = 15$	40	Arc flow	0	3600	2.4	748.31	774.14	2 033 046	657 109	8 130 689
		ILS	-	12.1	-	-	778.64	-	-	-
$n/m = 20$	40	Arc flow	0	3600	6.5	938.12	1122.26	5 040 395	1 450 953	20 159 585
		ILS	-	17.0	-	-	1050.35	-	-	-
$d_{ij} \in [1, 6]$	60	Arc flow	0	3600	6.6	686.32	826.40	5 151 331	1 566 014	20 603 831
		ILS	-	11.1	-	-	778.61	-	-	-
$d_{ij} \in [5, 10]$	60	Arc flow	0	3600	0.1	782.47	782.87	167 324	49 026	667 801
		ILS	-	13.8	-	-	801.83	-	-	-
$t_i \in [1, 3]$	60	Arc flow	0	3600	0.4	769.61	770.04	340 493	127 799	1 360 478
		ILS	-	13.4	-	-	777.77	-	-	-
$t_i \in [1, 9]$	60	Arc flow	0	3600	6.4	699.17	839.23	4 978 162	1 487 241	19 911 155
		ILS	-	11.5	-	-	802.67	-	-	-
Overall	120	Arc flow	0	3600	3.4	734.39	804.63	2 659 328	807 520	10 635 816
		ILS	-	12.5	-	-	790.22	-	-	-

In Table A.8, we focus on the 120 instances with 10 machines. We notice that the arc flow-based formulation could not solve any instance to proven optimality. The average gap between the upper and lower bounds vary significantly depending on the tested parameters: for instances with short maintenance duration, long delay factors, and small n/m ratio, the absolute gap was less than 0.8 on average, while it was significantly higher for the other instances, even reaching 140 on average for instances with high maintenance duration. The behavior of the metaheuristic is relatively similar to what was observed for instances with 2 and 5 machines. We notice here for the first time that the metaheuristic found better solutions (upper bound equal to 790.22 on average) than the arc flow-based formulation (upper bound equal to 804.63 on average). By studying the distribution of parameter Δ, which we define as the difference between the upper bound obtained by the ILS and the upper bound obtained by the arc flow-based model (i.e., $\Delta = UB_{ILS} - UB_{AF}$), we report that Δ is equal to -14.41 on average, but its median is equal to 8.87. This indicates that $UB_{AF} < UB_{ILS}$ for a majority of instances, but in the rare cases in which $UB_{AF} > UB_{ILS}$, the difference is significant (up to approximatively 930). Further analysis outlined that the upper bound obtained by the arc flow-based model is particularly bad for instances in which most of the computation time is spent on computing the lower bound.

Table A.9: Evaluation on instances with 20 machines

group	# inst.	method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs
$n/m = 10$	40	Arc flow	0	3600	12.9	428.16	630.39	8 771 690	2 584 382	35 082 775
		ILS	-	35.2	-	-	535.29	-	-	-
$n/m = 15$	40	Arc flow	0	3600	43.9	587.08	991.11	28 817 601	7 058 348	115 264 422
		ILS	-	53.2	-	-	789.44	-	-	-
$n/m = 20$	40	Arc flow	0	3600	94.5	655.55	1283.72	38 341 270	8 760 478	153 577 723
		ILS	-	83.1	-	-	1062.14	-	-	-
$d_{ij} \in [1, 6]$	60	Arc flow	0	3600	99.9	328.39	1150.58	49 490 269	12 000 939	198 102 178
		ILS	-	50.5	-	-	787.31	-	-	-
$d_{ij} \in [5, 10]$	60	Arc flow	0	3600	1	785.47	786.23	1 130 105	267 866	4 514 435
		ILS	-	63.9	-	-	803.93	-	-	-
$t_i \in [1, 3]$	60	Arc flow	0	3600	4.2	693.53	851.68	3 555 787	1 134 663	14 217 164
		ILS	-	59.9	-	-	795.08	-	-	-
$t_i \in [1, 9]$	60	Arc flow	0	3600	96.7	420.33	1085.14	47 064 587	11 134 141	188 399 449
		ILS	-	54.4	-	-	796.16	-	-	-
Overall	120	Arc flow	0	3600	50.5	556.93	968.41	25 310 187	6 134 402	101 308 307
		ILS	-	57.2	-	-	795.62	-	-	-

Unsurprisingly, we observe in Table A.9 that for instances with 20 machines the arc flow-based formulation also obtains better results for instances with short maintenance duration, long delay factors, and small ratio n/m. The behavior of the metaheuristic is very satisfactory, as it now outperforms the average solution provided by the arc flow-based formulation on all types of instances, with a single exception on those with large delay factors. The average computational effort of the ILS is around one minute, and ranges between 35 seconds for $n/m = 10$, to 83 seconds for $n/m = 20$, on average.

A.7 Conclusion

We studied the problem of processing a set of jobs on a set of unrelated parallel machines by considering sequence-dependent deterioration and the option of restoring a machine to its full operational speed by performing a maintenance activity. We reviewed an integer non-linear programming formulation from the literature, and introduced four novel mixed integer linear programming formulations. We derived two properties from the literature that allowed us to improve the performance of the models. In addition, we developed a new metaheuristic approach, based on the concept of iterated local search, to provide good quality solutions for large size instances.

We tested all our approaches with an extensive set of computational experiments. Among the mathematical models, we observed that the arc flow-based formulation was the one providing the best results on average: it could solve to proven optimality all instances with 2 machines, and obtained good quality solutions for most instances with 5, 10, and 20 machines. We also noticed that, due to the large number of variables it requires, the arc flow-based formulation could have a large optimality gap, and thus be outperformed by the metaheuristic

in terms of running time and solution value. This happened mostly on instances with 20 machines. We also outlined specific parameters that made the instances easier to solve by our approaches (namely, small n/m ratio, short maintenance time, and long delay factors).

Interesting future research directions concern the development of a branch-and-price algorithm for the block-based formulation and the investigation of maintenance scheduling problems with sequence-dependent deterioration with alternative objective functions, such as weighted completion time, weighted earliness or weighted tardiness functions.

A.8 Bibliography

B. Alidaee, N. Womer. Scheduling with time dependent processing times: Review and extensions. *Journal of the Operational Research Society*, 50:711–720, 1999.

O. Araújo, G. Dhein, M. Fampa, Minimizing the makespan on parallel machines with sequence dependent deteriorating effects, in: XLIX SBPO, Símposio Brasileiro de Pesquisa Operacional, 2017. URL: http://www.sbpo2017.iltc.br/pdf/168932.pdf.

S. Browne, U. Yechiali. Scheduling deteriorating jobs on a single processor. *Operations Research*, 38(3):495–498, 1990.

T. Cheng, Q. Ding. Single machine scheduling with step-deteriorating processing times. *European Journal of Operational Research*, 134(3):623–630, 2001.

T. Cheng, Q. Ding, B. Lin. A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research*, 152(1):1–13, 2004.

F. Clautiaux, S. Hanafi, R. Macedo, M.-E. Voge, C. Alves. Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints. *European Journal of Operational Research*, 258(2):467–477, 2017.

F. Clautiaux, R. Sadykov, F. Vanderbeck, Q. Viaud. Combining dynamic programming with filtering to solve a four-stage two-dimensional guillotine-cut bounded knapsack problem. *Discrete Optimization*, 29:18–44, 2018.

M. Dell'Amico, M. Delorme, M. Iori, S. Martello. Mathematical models and decomposition methods for the multiple knapsack problem. *European Journal of Operational Research*, 274(3):886–899, 2019.

M. Delorme, M. Iori. Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing*, 32(1):101–119, 2020.

J. Ding, L. Shen, Z. Lü, B. Peng. Parallel machine scheduling with completion-time-based criteria and sequence-dependent deterioration. *Computers & Operations Research*, 103:35–45, 2019.

S. Gawiejnowicz. A review of four decades of time-dependent scheduling: main results, new topics, and open problems. *Journal of Scheduling*, 23:3–47, 2020.

P. Gilmore, R. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

P. Gilmore, R. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 11(6):863–888, 1963.

R. Graham, E. Lawler, J. Lenstra, A. R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, in: P. Hammer, E. Johnson, B. Korte (Eds.), Discrete Optimization II, volume 5 of *Annals of Discrete Mathematics*, Elsevier, 1979, pp. 287–326. doi:https://doi.org/10.1016/S0167-5060(08)70356-X.

J. Gupta, S. Gupta. Single facility scheduling with nonlinear processing times. *Computers & Industrial Engineering*, 14(4):387–393, 1988.

M. Ji, T. Cheng. Parallel-machine scheduling with simple linear deterioration to minimize total completion time. *European Journal of Operational Research*, 188(2):342–347, 2008.

A. Kramer, M. Dell'Amico, M. Iori. Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines. *European Journal of Operational Research*, 275(1):67–79, 2019.

W. Kubiak, S. van de Velde. Scheduling deteriorating jobs to minimize makespan. *Naval Research Logistics*, 45(5):511–523, 1998.

A. Kunnathur, S. Gupta. Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. *European Journal of Operational Research*, 47(1):56–64, 1990.

W.-H. Kuo, D.-L. Yang. Minimizing the makespan in a single-machine scheduling problem with the cyclic process of an aging effect. *Journal of the Operational Research Society*, 59(3):416–420, 2008.

E. Lalla-Ruiz, S. Voß. Modeling the parallel machine scheduling problem with step deteriorating jobs. *European Journal of Operational Research*, 255(1):21–33, 2016.

C.-Y. Lee, V. Leon. Machine scheduling with a rate-modifying activity. *European Journal of Operational Research*, 128(1):119–128, 2001.

J.-T. Leung, C. Ng, T. Cheng. Minimizing sum of completion times for batch scheduling of jobs with deteriorating processing times. *European Journal of Operational Research*, 187(3):1090–1099, 2008.

S. Lu, X. Liu, J. Pei, M. Thai, P. Pardalas. A hybrid ABC-TS algorithm for the unrelated parallel-batching machines scheduling problem with deteriorating jobs and maintenance activity. *Applied Soft Computing*, 66:168–182, 2018.

N.F.M. Mendes, M. Iori, A mathematical model and metaheuristic for a job and maintenance machine scheduling problem with sequence dependent deterioration, in: XLIX SBPO, Símposio Brasileiro de Pesquisa Operacional, volume 2, 2019.

G. Mosheiov. V-shaped policies for scheduling deteriorating jobs. *Operations Research*, 39(6):979–991, 1991.

G. Mosheiov. Scheduling jobs under simple linear deterioration. *Computers & Operations Research*, 21(6):653–659, 1994.

G. Mosheiov. Scheduling jobs with step-deterioration; minimizing makespan on a single- and multi-machine. *Computers & Industrial Engineering*, 28(4):869–879, 1995.

G. Mosheiov. Λ-shaped policies to schedule deteriorating jobs. *Journal of the Operational Research Society*, 47(9):1184–1191, 1996.

G. Mosheiov. Multi-machine scheduling with linear deterioration. *INFOR: Information Systems and Operational Research*, 36(4):205–214, 1998.

G. Mosheiov, J. Sidney. New results on sequencing with rate modification. *INFOR: Information Systems and Operational Research*, 41(2):155–163, 2003.

V. Nesello, M. Delorme, M. Iori, A. Subramanian. Mathematical models and decomposition algorithms for makespan minimization in plastic rolls production. *Journal of the Operational Research Society*, 69:326–339, 2018a.

V. Nesello, A. Subramanian, M. Battarra, G. Laporte. Exact solution of the single-machine scheduling problem with periodic maintenances and sequence-dependent setup times. *European Journal of Operational Research*, 266(2):498–507, 2018b.

J. Pacheco, F. Àngel Bello, A. Álvarez. A multi-start tabu search method for a single-machine scheduling problem with periodic maintenance and sequence-dependent set-up times. *Journal of Scheduling*, 16:661–673, 2013.

A. Pessoa, E. Uchoa, M. Poggi, R. Rodrigues. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, 2:259–290, 2010.

M. Pinedo. *Scheduling: theory, algorithms and systems development*. Springer, New York, 2016. Fifth Edition.

B. Ramos, C. Alves, J. Valério de Carvalho. An arc flow formulation to the multitrip production, inventory, distribution, and routing problem with time windows. *International Transactions in Operational Research*, 2020 (forthcoming).

A. Ruiz-Torres, G. Paletta, R. M'Hallah. Makespan minimisation with sequence-dependent machine deterioration and maintenance events. *International Journal of Production Research*, 55(2):462–479, 2017.

A. Ruiz-Torres, G. Paletta, E. Pérez. Parallel machine scheduling to minimize the makespan with sequence dependent deteriorating effects. *Computers & Operations Research*, 40(8):2051–2061, 2013.

V. Santos, J. Arroyo. Iterated greedy with random variable neighborhood descent for scheduling jobs on parallel machines with deterioration effect. *Electronic Notes in Discrete Mathematics*, 58:55–62, 2017.

V. Strusevich, K. Rustogi. *Scheduling with time-changing effects and rate-modifying activities*. Springer, Berlin, 2017.

J. Valério de Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86:629–659, 1999.

J.-B. Wang, L. Li. Machine scheduling with deteriorating jobs and modifying maintenance activities. *The Computer Journal*, 61(1):47–53, 2017.

D.-L. Yang, T. Cheng, S.-J. Yang, C.-J. Hsu. Unrelated parallel-machine scheduling with aging effects and multi-maintenance activities. *Computers & Operations Research*, 39(7):1458–1464, 2012.

S.-J. Yang. Parallel machines scheduling with simultaneous considerations of position-dependent deterioration effects and maintenance activities. *Journal of the Chinese Institute of Industrial Engineers*, 28(4):270–280, 2011.

C. Zhao, H. Tang. Single machine scheduling with general job-dependent aging effect and maintenance activities to minimize makespan. *Applied Mathematical Modelling*, 34(3):837–841, 2010.

Appendix B

Minimizing User Inconvenience and Operational Costs in a Dial-a-Flight Problem for Air Safaris

We study a Dial-a-Flight Problem faced by one of the major safari airline companies in Tanzania. Given a set of daily passenger requests and a fleet of heterogeneous airplanes, the problem requires to determine the best set of itineraries to transport the passengers from their origins to the requested destinations within specific time windows, while satisfying a number of operational constraints. The aim is to minimize user inconvenience, measured by delays with respect to the pre-defined time windows and by the number of intermediate stops in the itineraries, and operational cost. The problem is complicated by the high number of daily requests in peak touristic periods, and by the fact that refueling is possible only at a limited number of airstrips. We solve the problem by means of an adaptive large neighborhood search, which we enrich with local search operators and a set partitioning model. Extensive computational tests on real-world instances prove the effectiveness of the proposed algorithm, which can improve the solutions found by the company both in terms of operational cost and user inconvenience, in reasonable computational time.

B.1 Introduction

Tanzania is an Eastern African country with many tourist attractions,as national parks, conservation areas, reserves and marine parks. These include popular tourist destinations, such as the island of Zanzibar and the UNESCO World Heritage site of Mount Kilimanjaro. These attractions make tourism the largest foreign source of income for the country, contributing with an average of 2 billion U.S. dollars per year since 2012, which is roughly equivalent to 25% of all foreign exchange earnings. Tourism also contributes to more than

17% of the national gross domestic product, creating more than 1.5 million jobs and being the fastest growing sector (National Bureau of Statistics, Tanzania, 2018).

Many pioneering entrepreneurs were quick in identifying tourism as the true national vocation of Tanzania. In early years, tourism gave life to a great number of vehicle-based safari companies, which opened the country to a new surge of visitors. It is around 30 years ago that some companies recognized the opportunity to develop an airline safari network capable of accessing the most remote parts of the country, offering more agile and comfortable services to the visitors.

Currently, these companies operate on a network composed by more than 100 airstrips, connecting not only the main cities and tourist sites in the country, but also far-away destinations located in the middle of the park areas (see Figure B.1 for a condensed view of Tanzania's airstrip network). Flights between these airstrips are performed by means of fleets of small airplanes, each transporting around a dozen passengers. For most companies, flights are organized on a daily basis, combining in the best possible way the travel bookings received. The transport is done under tight constraints imposed by hard operational conditions, including the lack of refueling options in many of the airstrips, and the need to provide a high level service to the customers.

Figure B.1: Main airstrip network in Tanzania (image taken from http://www.coastal.co.tz/)

This article studies the operations of one of the major safari airline companies in Tanzania. The company has a fleet of around 20 airplanes comprising two models, the high-wing braced cabin monoplane Cessna Caravan-208B and the single-engine turboprop Pilatus PC12. The two models have a similar capacity in terms of passengers transported, but significant differences in speed, fuel consumption, cabin comfort, and maximum cargo weight. Each airplane

starts from a given airstrip and may end its daily sequence of flights at a different airstrip.

Transport requests for a certain day are collected until the previous day and then processed, at the company headquarters, so as to form the best set of flights. Each request consists of an itinerary from a given origin airstrip to a given destination airstrip, to be performed within a specific time window both at the pickup and at the drop-off sites. Possible violations of the time windows are accepted but penalized (in other words, these are *soft* time windows). All clients should be able to reach their destination with a maximum of three intermediate stops. The number of intermediate stops is not only a constraint, but also a crucial parameter to be taken into account in the evaluation of the user inconvenience. Passengers may also select two different, and not exclusive, additional classes of service: a passenger in *fast class* is guaranteed to reach his destination without any intermediate stop; a passenger in *extra-luggage class* has the right to transport an additional piece of luggage on board, resulting in a maximum of 30 kg instead of 15 kg. At most airstrips, takeoffs and landings must be performed in daylight as there is no artificial lighting system. This corresponds to a *hard* time window that cannot be violated. Fuel is available only at certain airstrips. In addition, for safety reasons, a minimum quantity of fuel after landing and a maximum weight before takeoff are imposed.

The aim of the company is to combine the daily requests into a set of flights, in such a way that: (i) all requests are fulfilled; (ii) all operative constraints are satisfied; (iii) user inconvenience, measured by delays in the time windows and number of intermediate stops, is minimized; and (iv) operational costs as well are minimized. Operational costs (which are discussed in detail in Section B.2 below) are caused by daily fees for the use of an airplane, number of kilometers traveled, refueling, and landing fees at airstrips. In the following, we refer to this problem as the *Dial-a-Flight Problem for Air Safaris* (DAFPAS).

The DAFPAS belongs to the category of *Dial-A-Flight Problems* (DAFP). This class of problems has received increasing attention in recent years in the contexts of taxi transportation (Espinoza et al., 2008a,b) and of air safari planning (Fügenschuh et al., 2013). The work by Fügenschuh et al. (2013) is the one that most resembles ours, but they study a problem with a different objective function, different constraints, and smaller instance sizes (as discussed in detail in Section B.3 below). The DAFP is one of the key problems arising in air passenger transportation, and differs from other classical airline scheduling problems (see, e.g., Klabjan 2005) because the planning changes on a daily basis instead of making use of structured medium-term or long-term schedules. The DAFP is more similar to the well-known *Dial-a-Ride Problem* (DARP), which is usually studied in the context of ground transportation vehicles (Cordeau and Laporte, 2007; Doerner and Salazar-González, 2014), and to other transportation on demand problems (Cordeau et al., 2007), with which it shares the need of identifying a user inconvenience function. All such problems are not only NP-hard, but also very difficult in practice, and instances of large size cannot be solved exactly

within limited computing times.

The aim of this paper is to propose a methodology to quickly obtain good-quality solutions for the real-world DAFP that we describe. To this aim, we develop an *Adaptive Large Neighborhood Search* (ALNS) metaheuristic, which relies on several destroy and repair mechanisms. We also embed into the ALNS a set of local search procedures to explore more intensively the neighborhoods around promising solutions, and adopt a *Set Partitioning* (SP) model to iteratively post-optimize the pool of routes that has been built during the search. Recently, ALNS methods have obtained very good results on a large number of vehicle routing problems (see, e.g., Ropke and Pisinger 2006 and Pisinger and Ropke 2010). Still in the field of routing, the combination of ALNS with local search have led to good results (as in, e.g., Dell'Amico et al. 2016), as well as the use of SP models as post-optimization tools (as in Subramanian et al. 2013). Convincing results have also been obtained by similar approaches on the closely related DARP, by local search based metaheuristics (Parragh et al., 2010; Masmoudi et al., 2017) and by ALNS algorithms (Masson et al., 2013; Gschwind and Drexl, 2019). In our study, the combined use of ALNS with local search and an SP model led to prominent computational results, achieving good-quality solutions with limited computational effort.

The main contributions of this paper are as follows:

- We describe in detail a real-world transportation on demand problem and we contrast it with the existing literature. The interest derives not only from the particular application at hand, but also from the fact that the problem is very general and may well represent several other situations arising in passenger transportation.

- We perform a deep study of the user inconvenience function, which is measured as the violation of pickup and drop-off time windows and the number of intermediate stops. An economic interpretation of the user inconvenience has been defined in agreement with the company.

- We design a new metaheuristic based on the ALNS paradigm. Some operators have been adapted from the existing literature, whereas others have been newly designed on the basis of the specific characteristics of the problem at hand.

- We use an SP model to attempt recombining the routes explored during the ALNS search. The model is used in an iterative manner, with the aim of determining the best balance between ALNS and SP computational efforts.

- We present extensive computational tests on a set of real-world instances. The outcome shows that the presented algorithm is effective in improving the solutions found by the company, achieving lower cost and lower user inconvenience, on average, within limited computational times, and hence can be considered as a good solution tool for the problem.

The remainder of this paper is organized as follows. Section B.2 provides a detailed description of the DAFPAS. Section B.3 reviews the related scientific literature. A formal mathematical formulation of the problem is provided in Section B.4. Section B.5 contains the details of the metaheuristic algorithm that we developed. The result and analysis of extensive computational experiments that we performed on a set of real-world instances are given in Section B.6. Some final conclusions and future research avenues are drawn in Section B.7.

B.2 Problem Description

The DAFPAS lies in the class of DAFP, but contains a number of specific characteristics induced by its application context. In this section, we describe the problem characteristics in detail and present some assumptions that we made to produce a good model.

B.2.1 Airstrips and Network

We are given a set of 21 airstrips, that represent the vertices of a complete undirected graph. Each airstrip is characterized by the fact that it can be used for refueling or not. Each airplane landing at an airstrip without refueling should always have in its tank a minimum quantity of fuel, imposed by security rules. The path to be followed for flying from an airstrip to another is known, and so is the distance to be traveled and the expected flight time and fuel consumption.

For each airstrip, a maximum allowed weight at take off for each airplane (see also Section B.2.3) and a landing fee are imposed. A minimum time in which an airplane is required to remain on the ground between a landing and the next takeoff is also imposed. This time, called *ground time* (and being around 20–30 minutes in our real-world application), depends on the airstrip and comprises the operational time for alighting/boarding passengers, the possible need for refueling, and a break time for the pilot.

Operations at an airstrip are allowed only within a daily time window. Indeed, for most of the airstrips, takeoffs and landings must happen in daylight. We thus set the operating time window to [6:00 am, 6:30 pm] in our tests. For some other airstrips, namely, Dar es Salaam and Arusha (see Figure B.1), earlier departures starting from 5:30 are allowed because of the presence of artificial light and an air control tower.

B.2.2 Requests

We are given a set of a transportation requests, each with given origin and destination airstrips. The majority of requests is associated with a single passenger. Under this assumption, it is possible that passengers that booked the air safary as a group be split into different flights. This is allowed by our methodology, and by the company as well, but does not occur frequently because optimization tends to group passengers with the same origin, destination

and time windows on the same flights. Some requests (around 14% of the total) are composed by more than one passenger. These correspond to families with children, who cannot be separated from a parent. Each passenger is characterized by a type (male, female, or child) and a class (standard, fast-class, extra-luggage, or combined fast-class and extra-luggage). This information is used to compute the expected weight of a request and the maximum number of intermediate stops between pickup and drop-off.

Each request has a specific time window both at the pickup and at the drop-off airstrips. At the time of booking, target pickup and drop-off times have been defined. The time windows are simply intervals set around these target times. In our case study, the intervals last one hour (half an hour before and half an hour after the target time) and correspond to a discrepancy with respect to the target that is supposed to be accepted without inconvenience by the clients. Early or late arrivals with respect to these time windows are still accepted, but penalized (see Section B.2.5 for details). In such cases, passengers are required to be at the pickup locations at the new arranged pickup time, and ground transportation services that will take care of the passengers after their landing should be at the drop-off locations at the new arranged arrival time. Both times are communicated the day before. Note that it is not expected that an airplane wait in case of early arrival. We simply suppose that passengers and ground services will be able to reach the new airstrip within the new arranged time, as this was communicated in advance. Waiting is allowed in case it helps minimize the total user inconvenience of a route (and this is indeed at the basis of one or our solution methods in Section B.5 below). Considering the size of the instances we tackle, we observe that the number of requests per day may vary from about 100 requests in the months of lowest demand, to about 350 requests in the months of highest demand.

B.2.3 Airplanes

We are given a heterogeneous fleet of airplanes. Each airplane has a certain cruise speed, which determines the traveling time between two airstrips, and a certain fuel consumption, discussed in detail in Section B.2.4. An airplane is also characterized by a number of seats for passengers, a maximum allowed total weight for taking off, and a maximum fuel capacity. In real applications, the total weight for taking off depends not only on the airplane, but also on the airstrip, because airstrips with higher altitude offer lower air lift. As the differences between the airstrips are very small in our case study, we assumed that each airplane has a unique value of maximum weight for taking off.

Note that the weight of an airplane depends on both the transported passengers and the quantity of loaded fuel, and this represents a key decision variable when building routes. This happens in any air transportation problem (Cordeau et al., 2007), but is particularly relevant in our study because refueling is not available in most of the airstrips and the airplanes have a small size. We can decide to load more passengers at the expense of a lower fuel tank level,

or, vice versa, to cover longer distances at the expense of a reduced number of transported passengers. Note also that, after landing at an airstrip, an airplane must have at least the minimum quantity of fuel required to reach the closest airstrip (in case landing, for any reason, cannot be performed at the planned airstrip). This is imposed on each intermediate stop and also on the last stop of a day, to avoid having an airplane stuck at an airstrip without sufficient fuel.

Each airplane is also associated with a home location. At the beginning of a day, the airplane is available at the airstrip at which it ended the last flight on the previous day. At the end of the day, it can be located at any airstrip (provided it has sufficient fuel). Each airplane should be back to its home location at least once every three days. This is imposed to perform a required periodic maintenance on the airplane. In our study, we relaxed this assumption, supposing that each day is independent from the others. However, going back to the home location can be obtained, in our solution method, by imposing this location as the last arrival of the day for an airplane. For considering, instead, the original constraint as in the real-world application, one should model the problem as a dynamic multi-period problem, to be solved with a rolling horizon approach. The dynamic component is caused by the fact that requests for day $t+1$ would be known only when the flights for day t are already being operated. The resulting problem is left as an interesting future research direction (see Section B.7).

Concerning the size of the problem, the fleet owned by the company is composed by around 20 airplanes. In our instances, the number of available airplanes varies between 10, in the months with lowest demand, to 15, in the months with highest demand. The remaining airplanes are either unused and parked at their home locations, or in maintenance, or rented for private safari flights. The use of private flights is quite common for groups aiming at a higher service level (at a higher cost). It is implicitly assumed that the number of airplanes is large enough to satisfy all requests, as capacity is taken into account when accepting the bookings for a given day. Should there be an excess of requests or a shortage of airplanes, other airplanes could be rented on the market.

B.2.4 Costs

As reported by Cordeau et al. (2007), most studies on transportation on demand problems fall under two categories: minimizing costs subject to full demand satisfaction and side constraints; or maximizing satisfied demand subject to vehicle availability and side constraints. Our problem belongs to the first category. The costs we incur for serving the requests are the cost of daily use of each airplane, the cost of the mileage traveled, the fuel consumption, and the cost for the landings. Let us explain them in more detail.

It is quite common for airline companies with seasonal demand to lease the airplanes for long periods. In case the number of transportation requests increases, new airplanes are

rented. As far as we are concerned, we deal with a problem where the company has at its disposal the entire fleet of airplanes. In the short-term, this means that the fixed cost of each airplane is only composed by its flight tax. Every day, and for each used airplane, the company pays a mandatory fee to the *Tanzania Civil Aviation Authority* (TCAA). Clearly, any used airplane has a certain fixed cost whose value does not affect the cost of a route. Nevertheless, the fixed costs gain in importance in the long-term view, especially for determining the economical sustainability of the activity. For this reason, in our computational tests (see Section B.6 below) we evaluated two scenarios, one in which the daily cost simply amounts to its mandatory fee, and another in which it also includes the daily operating cost composed of staff salary and amortization.

It is common for most of the companies operating routing services with fleets of vehicles to impose a cost for each kilometer traveled. This also happens for our case study, where this cost is considered independent from the type of airplane and takes into account the direct expenses and maintenance.

Refueling may take place at a limited set of airstrips. The cost of the gasoline changes from one location to another, although very slightly. In our model, we assumed it to be equal for all airstrips, so as to simplify the refueling evaluation. The fuel consumption cost is then determined as the cost per liter multiplied by the number of liters consumed. The determination of the fuel consumption for an airplane is a complex task that would require to consider flight conditions (e.g., weather) and the path between departure and arrival airstrips (e.g., acceleration, deceleration, turns, difference in altitudes). We decided to adopt the same simplified criterion adopted by the company for the evaluation of the fuel consumption. We simply use a given linear consumption for flights that last an hour or less. By multiplying the number of traveled minutes by this parameter, we obtain the total liters consumed in a flight. For longer flights, the consumption in the first hour is computed using the first parameter, and the remaining consumption is obtained by multiplying the remaining time by a second, smaller, fuel consumption parameter. The second parameter is smaller than the first, because in shorter flights fuel consumed during landing and takeoff has a higher impact on the overall fuel consumed and also because the airplane does not reach a high altitude, thus encountering a higher air resistance.

Any time an airplane lands, a corresponding landing fee should be paid to the TCAA. This fee is independent from the airstrip at which the landing occurred.

B.2.5 User inconvenience

User inconvenience is a measure of the dissatisfaction of a passenger, and should be minimized together with the operational costs. In our case, we opted to measure the number of intermediate stops, and the amount of violation of the time windows, both at the pickup and at the drop-off locations. The intermediate stops are already limited to a maximum of three

for a standard passenger and to one for a fast-class one, but on top of that they should also be minimized, as highly disliked. The time window violation is considered both for early arrivals (as the difference between the earliest time and the arrival time if the arrival occurs before the earliest time) and for late arrivals (as the difference between the arrival time and the latest time if the arrival occurs after the latest). The total time window violation, expressed in minutes, is multiplied by a first penalization parameter, and the total number of intermediate stops by a second parameter. The values of these parameters have been established on the basis of discussions with the company, but, in addition to that, we performed extensive tests to assess their impact on the solutions.

We note that the time window violation is related to total riding time, waiting time and duration, which are other measures adopted in the DARP context where a single time window is imposed on the pickup (Cordeau and Laporte, 2003). We also note that the company allows, in some cases, transshipment of passengers from an airplane to another in order to reach the required destinations. This is even more disliked by passengers, for obvious reasons. To look for low user-inconvenience solutions and obtain a simpler model, we disregarded the possibility of transshipment in our methodology. Fortunately, we were able to satisfy all requests on all instances even without transshipment. In Section B.6, in order to evaluate the cost of the company solutions, transshipments, if any, have been penalized doubly compared to the cost of an intermediate stop.

B.3 Literature Review

The DAFPAS lies in the class of *Pickup and Delivery Problems* (PDPs), where requests are characterized by a point in which they need to be collected and a second point where they have to be delivered (see Battarra et al. 2014 and Doerner and Salazar-González 2014 for recent surveys). Among PDPs, the closest problem to the DFAPAS is the DARP, which requires to meet pickup and delivery transport demands by using a fleet of ground vehicles while minimizing cost and user inconvenience. The number of papers devoted to the solution of practical DARP has risen consistently in recent years, as can be noticed in, e.g., Cordeau and Laporte (2007) and Ho et al. (2018). Important differences arise between the DARP and the DAFPAS, in the definition of the constraints (e.g., there is no fuel restriction for the DARP), of the costs (e.g., there is no landing fee for the DARP) and of the user inconvenience (which is more related to time spent on board for the DARP, and on number of intermediate stops for the DAFPAS).

We can include the DAFPAS in the areas of Transportation on Demand and Air Transportation. Transportation on demand concerns the relocation of passengers or goods between given origins and destinations, following specific requests by the users. Cordeau et al. (2007) give a description of this area, providing mathematical models for DARP services, urban

courier transportation, ambulance fleet management, as well as static and dynamic DAFP. Air transportation is a wide area of research characterized by a variety of optimization problems. We refer the interested reader to the surveys by Barnhart et al. (2003) and Lacasse-Guay et al. (2010), to the book by Wensveen (2016), and to the recent case studies by Cacchiani and Salazar-González (2020) and Parmentier and Meunier (2020).

A number of relevant works combine air transportation with transportation on demand. Desaulniers et al. (1997) solved the daily aircraft routing and scheduling problem, which consists in constructing daily schedules for a heterogeneous aircraft fleet, with the aim of minimizing the fixed cost for each aircraft, the cost of the fuel consumed and the salaries of the crew members. They proposed SP and time constrained network flow formulations, and obtained good results by employing column generation. Keskinocak and Tayur (1998) addressed the time-shared jet aircraft scheduling problem, which can be seen as a DAFP where each aircraft can serve only one customer at a time. They studied the problem complexity and proposed solution methods based on Dynamic Programming (DP) and Mixed Integer Linear Programming (MILP). Ronen (2000) developed a decision support system based on the use of an SP model for scheduling charter airplanes. They minimized an objective function that included a number of operational costs and penalties for violations of soft constraints.

Martin et al. (2003) considered on-demand aircraft schedules for the so-called fractional aircraft programs (FAP). In a FAP, fractional owners purchase portions of specific aircraft from a management company, based on the number of actual flight hours they need. They are guaranteed access to an aircraft whenever and wherever they need it, by booking their service in advance. The authors present a management system that includes a MILP model for scheduling the aircraft. Similar FAPs were later studied by Yao et al. (2008), who discuss strategic planning issues, such as aircraft maintenance, crew swapping, and methods to increase and differentiate demand, and by Yang et al. (2008), who propose a scheduling decision support tool based on exact and heuristic algorithms aimed at increasing aircraft utilization.

Fagerholt et al. (2009) consider an air taxi service in Norway. Air taxi is an on-demand service in which customers can book in advance seats on aircraft operating on small regional airports. They presented a strategic decision support tool that helps estimate the trade-off between fleet size and service by heuristically solving an underlying DAFP. A similar problem involving a Belgian company has been studied by Van der Zwan et al. (2011), who developed an SP model. Very recently, Munari and Alvarez (2019) considered a FAP in which the aim is to determine airplane routing and scheduling to fulfill a list of flight requests. They propose a compact MILP model that takes into account mandatory aircraft maintenance and possible flight upgrades.

A typical feature of the DAFPAS is the limited fuel capacity. Other optimization problems with this feature have been considered in the literature. That is the case, for instance, in the Green Vehicle Routing Problem, which concerns fleets composed by alternative fuel-powered

vehicles and helps in overcoming difficulties due to limited vehicle driving range in conjunction with limited refueling infrastructure (see, e.g., Bektaş et al. 2016). A restricted operational range caused by limited fuel capacity is a major concern also in military applications. Solution approaches in this field have investigated the use of aerial refueling, as in, e.g., Yamani et al. (1990), Yuan and Mehrez (1995) and Kannon et al. (2015).

A closely related transportation problem concerns on demand routing of helicopters. This topic has received a good amount of attention in recent years, especially for what concerns the transportation of rig crews in oil and gas offshore platforms, which was studied, among others, by Fiala Timlin and Pulleyblank (1992), Menezes et al. (2010), Qian et al. (2012), Hermeto et al. (2014) and de Alvarenga Rosa et al. (2016). As in our problem, using alternative vehicles, like vessels, is not an option because of low speed combined with long distances that need to be covered. The number of stops is also considered, not because of user inconvenience but for security reasons, as takeoffs and landings are dangerous on offshore platforms.

To the best of our knowledge, the term DAFP originates from the works of Espinoza et al. (2008a,b). Important differences arise between their problem and the one we face, as they can refuel at any airport, and they control user inconvenience by imposing hard constraints on maximum transit time and allowing at most one intermediate stop. In Espinoza et al. (2008a), the problem is modeled with a multicommodity network flow, having a direct flight for each pair of airports (a, b) and each departure time at a, and indirect flights (with one intermediate stop) for each triplet of airports (a, b, c) and each pair of departure times at a and b. The size of the network grows quickly with the number of airports, so they use aggregation techniques. They solve to proven optimality instances with up to six airplanes. In Espinoza et al. (2008b), they consider larger instances. They develop a parallel local search heuristic that invokes the multicommodity model for smaller instances containing a limited number of airplanes. Their approach is not practically replicable to our case study because it is based on the strict assumption that at most one intermediate stop occurs for each passenger. In related work, Engineer et al. (2011) introduce a column generation approach making use of a DP that operates on the time-expanded network underlying the previous multicommodity flow model. They use arc-based resource relaxation, forward and backward search, and a quick completion heuristic. They provide solutions for instances with up to 200 airplanes. This approach too depends on the assumption that at most one intermediate stop is allowed.

Another work that is related to our DAFP is the air travel routing and scheduling problem solved by Fügenschuh et al. (2013). The problem considers restrictions on earliest departure and latest arrival times, maximum load and flight time, and refueling only at a limited number of locations. The authors develop and test different problem formulations. The one that performs better is based on a relaxation of time window constraints, that are then re-inserted by means of branching. It solves instances having between 8 and 13 airports and between 10 and 23 requests. This problem is different from our DAFP as it does not include

user inconvenience and minimizes traveling costs.

B.4 Problem Notation and Mathematical Formulation

We are given a complete undirected graph $G = (V, E)$, where V is the set of airstrips and E the set of edges connecting all pairs of airstrips. Each airstrip $i \in V$ is associated with an operating time window $[e_i, l_i]$, and with a binary parameter r_i taking value 1 if and only if it is possible to refuel in i. With each edge $(i, j) \in E$ we associate a distance d_{ij}, a cost c_{ij}, a fuel consumption g_{ij}, and a traveling time t_{ij}.

We are also given a set S of n requests. Each request $s \in S$ has a pickup airstrip $s^+ \in V$, a delivery airstrip $s^- \in V$, a number of passengers π_s, a weight w_s, a maximum number of intermediate stops σ_s, and tentative pickup and delivery time windows $[e_{s^+}, l_{s^+}]$ and $[e_{s^-}, l_{s^-}]$, respectively. Requests are satisfied by a fleet F composed of m airplanes. Each airplane $f \in F$ is located at airstrip $f^+ \in V$ at the beginning of the day, and is associated with a maximum number of passengers Π_f, a maximum fuel capacity G_f, and a maximum weight capacity W_f. The weight capacity should not be exceeded by the weighted sum of both passengers and fuel.

Let Ω^f be the set of all routes of an airplane $f \in F$. For simplicity, let $r \in \Omega^f$ define both a route and the corresponding route index. A route is a sequence of airstrips $r = (r_1, r_2, \ldots, r_{|r|})$. The first airstrip r_1 corresponds to the starting depot f^+ of f. Let $V(r) = \{i \in V : i \in r\}$ denote the set of airstrips visited by r, and note that $|V(r)| \leq |r|$ because an airstrip might be visited multiple times by a route. Let also $E(r) = \{(r_1, r_2), (r_2, r_3), \ldots, (r_{|r|-1}, r_{|r|})\}$ be the set of edges traversed by the route. The landing fee at an airstrip is denoted by c_ℓ, and the daily flight fee for an airplane by c_φ. By defining the cost per kilometer as c_d and the cost per liter of fuel as c_g, the total cost of a route is consequently given by

$$c_r^f = c_\varphi + c_\ell(|r| - 1) + \sum\nolimits_{(i,j)\in E(r)} (c_d d_{ij} + c_g g_{ij}). \tag{B.1}$$

Let us also denote by $S(r) \subseteq S$ the set of requests that are serviced by r. Let $r(s^+)$ denote the index of the vertex of r at which the pickup of s occurs, and $r(s^-)$ the index of the vertex at which the delivery occurs. Let $\psi_s = r(s^-) - r(s^+) - 1$ be the number of intermediates stops for s. Let also $a(i)$ denote the time in which the airplane arrives at vertex $i \in r$, considering $a(1)$ as the time in which the airplane is ready for departing at depot r_1. Under this notation, $a(r(s^+))$ gives the pickup time of s and $a(r(s^-))$ its delivery time. We can thus compute $\tau_{s^+} = \max\{e_{s^+} - a(r(s^+)); 0\} + \max\{a(r(s^+)) - l_{s^+}; 0\}$ as the time window violation, if any, at the pickup point of s. The value of τ_{s^+} takes into account both earliness, in its first component, and lateness, in its second component. Similarly, let $\tau_{s^-} = \max\{e_{s^-} - a(r(s^-)); 0\} + \max\{a(r(s^-)) - l_{s^-}; 0\}$ define the time window violation, if any, at the delivery point, and $\tau_s = \tau_{s^+} + \tau_{s^-}$ be the overall violation. Let ρ_ψ and ρ_τ be, respectively, the penalization factors associated with intermediate stops and time window

violations. The total user inconvenience of route r is measured as

$$u_r^f = \sum\nolimits_{s \in S(r)} (\rho_\psi \psi_s + \rho_\tau \tau_s). \tag{B.2}$$

We can thus define the overall objective value associated with route r as

$$z_r^f = c_r^f + u_r^f. \tag{B.3}$$

To mathematically formulate the problem, we can represent each route as a column of an SP model. For airplane $f \in F$, and route $r \in \Omega^f$, let δ_{sr}^f be a binary parameter equal to 1 if request s is served by r, and 0 otherwise. Let y_r^f be a binary variable taking the value 1 if route r of airplane f is used, and 0 otherwise. The DAFPAS can be stated as

$$\text{(SP)} \quad \min \sum_{f \in F} \sum_{r \in \Omega^f} z_r^f y_r^f \tag{B.4}$$

$$\text{s.t.} \quad \sum_{f \in F} \sum_{r \in \Omega^f} \delta_{sr}^f y_r^f = 1 \qquad s \in S \tag{B.5}$$

$$\sum_{r \in \Omega^f} y_r^f \leq 1 \qquad f \in F \tag{B.6}$$

$$y_r^f \in \{0, 1\} \qquad f \in F, r \in \Omega^f. \tag{B.7}$$

Objective function (B.4) requires to minimize the sum of the route costs, computed using (B.3). Constraints (B.5) force each request to be served. Constraints (B.6) state that each airplane is used at most once, and constraints (B.7) give the variable domain.

For our instances, we find it convenient to consider airplane types instead of airplanes. Airplanes being of the same model and being located at the same airstrip at the beginning of the day are said to be of the same type. In other words, all airplanes of the same type can be interchanged as they can perform the same routes at the same cost. Now, we can re-consider the fleet F, originally composed by m airplanes, as a fleet F' composed by t airplane types, each having m^f airplanes, in such a way that $\sum_{f \in F'} m_f = m$. We can thus reformulate model (B.4)–(B.7) by replacing F with F' and substituting (B.6) with

$$\sum_{r \in \Omega^f} y_r^f \leq m_f \qquad f \in F'. \tag{B.8}$$

Despite this reduction, model SP remains very difficult to solve in practice because it contains an exponential number of columns. It can be used, however, in two different ways: (i) by solving the continuous relaxation of SP we can compute the reduced cost of a column, i.e., a route, and thus estimate how much this route could contribute to a complete solution; and (ii) by replacing the complete sets Ω^f of routes by smaller sets and solving the model to integer optimality, we can obtain a heuristic solution. Both approaches are employed in our

metaheuristic method, as outlined in the next section.

B.5 Solution Methodology

To solve the DAFPAS, we implemented a metaheuristic that is based on iterated executions of an ALNS that is enriched with local search operators and an SP model. The method, called *Iterated ALNS* in the following, is summarized in Algorithm 6. It starts by creating a solution x with a constructive heuristic and a set of local search operators. The routes of this solution, represented by routes(x) in the pseudocode, are used to initialize an overall pool P of routes, which is going to be used later by the SP model. Then, the algorithm performs $iter_{\max}$ calls to the inner ALNS procedure. The first $iter_{phase1}$ times, the ALNS is invoked with an acceptance criterion that favors diversification and a stopping criterion that allows to perform a large search. In the remaining iterations, the ALNS is invoked with a more strict stopping criterion and with an acceptance criterion that favors intensification. Details on the adopted criteria and parameter values are given in Section B.5.6 below.

Algorithm 6 Iterated Adaptive Large Neighborhood Search

```
1: procedure ITERATED ALNS
2:     x ← Constructive Heuristic
3:     x ← Local Search(x)
4:     P ← routes(x)                                   ▷ Pool of routes
5:     for iter := 1 to iter_max do
6:         if (iter ≤ iter_phase1) then
7:             x ← ALNS(x, P, acceptance_criterion_1, stopping_criterion_1)
8:         else
9:             x ← ALNS(x, P, acceptance_criterion_2, stopping_criterion_2)
10:        end if
11:        end-if
12:    end for
13:    end-do
14:    return (x)
15: end procedure
```

The core part of the solution method is the ALNS, whose pseudocode is provided in Algorithm 7. At step 1, the iteration counter t is set to 0 and some weight parameters to be used in the main ALNS loop are initialized. At step 2, it sets the current solution x_{curr} as the incumbent received in input. The main loop is performed until the stopping criterion received in input is met. It considers the current solution x_{curr} and modifies it by: (i) selecting destroy and repair operators according to the weights; (ii) applying these operators to perturb x_{curr}; and (iii) using local search to improve it. The destroy operator uses a degree of destruction d, randomly selected in a given interval. Any time a new solution is obtained, pool P is possibly enlarged with the new routes in the solution. The new solution obtained, x_{new}, is compared

to the current one according to the input acceptance criterion. If the decision is to accept it, then x_{curr} is set to x_{new}. In such a case, we also check if x_{new} improves the incumbent solution and possibly update it.

After the main loop has been completed, the pool P is used to populate an SP model, invoked at step 18. This corresponds to the model outlined in Section B.4, but invoked with the limited number of routes contained in P instead of all possible routes, and with a limited computational time. Before solving SP to integer optimality, with the aim of reducing the size of P and consequently the computational time required to solve SP, we first solve, at step 17, the linear relaxation of the model. We use the solution found to compute the reduced costs of all columns in the pool. The Θ columns of highest reduced cost, whose probability of entering the optimal SP solution is very low, are removed from P. The solution found by SP(P) is then returned to the Iterated ALNS of Algorithm 6, together with the updated pool of routes. In the remainder of this section, we describe the details of each algorithmic component.

Algorithm 7 Adaptive Large Neighborhood Search with local search and SP model

1: **procedure** ALNS(x_{best}, P, acceptance_criterion, stopping_criterion)
2: set $t \leftarrow 0$ and initialize weights w_{mt}
3: $x_{curr} \leftarrow x_{best}$
4: **while** (stopping_criterion not met) **do**
5: select destroy and repair method using weights w^d_{mt} and w^r_{mt}
6: generate a degree of destruction $d \in [d_{\min}, d_{\max}]$
7: $x_{new} \leftarrow$ Repair(Destroy(x_{curr}, d))
8: $P \leftarrow P \cup$ routes(x_{new})
9: $x_{new} \leftarrow$ Local Search(x_{new})
10: $P \leftarrow P \cup$ routes(x_{new})
11: **if** (Accept(x_{curr}, x_{new}, acceptance_criterion)) **then**
12: $x_{curr} \leftarrow x_{new}$
13: **if** ($z(x_{curr}) < z(x_{best})$) **then** $x_{best} \leftarrow x_{curr}$
14: **end if**
15: **end-if**
16: $t \leftarrow t + 1$
17: update weights w_{mt}
18: **end while**
19: **end-do**
20: solve $L(SP(P))$ and remove from P the Θ columns with higher reduced cost
21: $x_{new} \leftarrow$ SP(P) ▷ Set Partitioning model
22: **if** ($z(x_{new}) < z(x_{best})$) **then** $x_{best} \leftarrow x_{new}$
23: **return** (x_{best}, P)
24: **end procedure**

B.5.1 Constructive Heuristic

In the hope of quickly obtaining a first starting solution, we developed a constructive heuristic that is based on the concept of *cheapest insertion* and that builds routes in a parallel fashion. It opens m routes, one per airplane f, considering its departure airstrip f^+. Then, it attempts to extend the routes by inserting requests from S, one at a time. The requests are sorted in random order and then selected according to it. Their insertion in the routes is attempted by considering only a restricted set of positions. Let us consider a generic request s to be inserted in a route r. We define four types of insertions:

(i) s in inserted as first request in r. This insertion is attempted only if r is still empty. In such a case, r is expanded by including s^+ and s^-, one after the other, directly after f^+. The insertion of s^+ is skipped if $s^+ = f^+$. The starting time is computed so as to be exactly on time for the pickup in s^+;

(ii) s in inserted as last request in r, so s^+ and s^- are inserted at the end of r. The insertion of s^+ is skipped if it is equal to the last airstrip visited by r. The departure time at the beginning of r is not changed after the insertion, so cost and user inconvenience can be computed quickly;

(iii) s is inserted only if both s^+ and s^- are already contained in r. In this way, the airplane operating r needs no detour to pick up and drop off the additional passenger(s), but the feasibility of all constraints must still be checked;

(iv) s is inserted only if s^+ is already in the route, and in this case s^- is inserted as last airstrip visited by r.

Once a request has been selected, all routes are scanned with respect to the four defined types of insertion, and the one being feasible, if any, and having cheapest insertion cost is selected. The procedure is iterated until all requests have been served or there is no more space for further insertions. It is worth noting that no polynomial-time heuristic can guarantee to find a feasible solution for the DAFPAS, because this is a difficult task (indeed, just loading weights w_s into the airplanes by respecting capacities W_f is as hard as the classical bin packing problem). For this reason, we include the heuristic in a loop that is iterated, each time creating a new random order of requests, until a feasible solution is obtained. In our tests, we managed to obtain a feasible starting solution for each instance with at most two iterations of the loop. At the end of the loop, if a feasible solution using strictly less than $|F|$ routes is found, the remaining unused routes are removed and the associated airplanes are simply kept in their original locations.

B.5.2 Destroy Operators

We have implemented *Random-removal*, *Worst-removal*, *All-removal*, and *Service time and Distance oriented removal* operators, which are quite common in the ALNS literature (Pisinger and Ropke, 2010). All operators receive in input a solution x composed by $|\text{routes}(x)|$ routes and a percentage value d representing the degree of destruction. The value of d is randomly selected, at each ALNS iteration, in the interval $[d_{min}, d_{max}]$. In Random-removal, Worst-removal, and All-removal, d represents the percentage of the number of routes affected by the destruction. For *Service time and distance oriented removal*, d is the percentage of requests relocated. The output of a destroy operator is a partial solution where some requests and/or some airstrips have been removed from the preexisting routes. All routes obtained after destruction are elaborated in order to preserve feasibility. The removed requests will be reinserted by means of the repair operators.

Random Removal. It randomly selects, with uniform distribution, a route r in x. Then, it randomly selects a vertex r_i in r and removes it. All requests that depart from or arrive at r_i are removed as well from r. The route is then processed, so as to recompute costs and user inconvenience. The process is repeated $\lceil |\text{routes}(x)|d/100 \rceil$ times.

Worst Removal. Similarly to the previous operator, Worst Removal randomly selects a route r. In this case, however, the vertex r_i to be removed at each iteration is chosen as the worst vertex in the route, i.e., as the vertex whose removal would lead to the largest decrease in the route value. The decrease is computed in an approximated but quick way, as the saving that could be obtained by: (1) reducing the distance traveled by connecting directly r_{i-1} to r_{i+1}; (3) removing user inconvenience penalties associated with requests landing at or departing from r_i. The removal process is repeated $\lceil |\text{routes}(x)|d/100 \rceil$ times.

All Removal. It aims at a larger diversification with respect to the two previous operators. It selects a route r and then removes from r a certain number p of vertices, where p is a random number generated, with uniform distribution, between 1 and $|\text{routes}(x)|$. The process is repeated $\lceil |\text{routes}(x)|d/100 \rceil$ times.

Service time and Distance oriented Removal. A number of destroy operators in the literature, starting from Shaw (1997), attempt to remove requests that are close to one another, either in terms of distance, or time window, or both. The rationale behind that is to facilitate, later on, the work of the repair method. To this end, we define the *relatedness* of two requests, s and q, as $\delta(s, q)$, and compute it as the sum of the travel distances d_{s^+,q^+} and d_{s^-,q^-}, and of the time distances between the target pickup and delivery times of s and q. We start by selecting a route r at random. Then, we select the request s that has the highest

user inconvenience in r and remove it. Then, we compute the relatedness $\delta(s, q)$ of all other requests q in r. We remove all requests q for which $\delta(s, q) \leq \delta_{\min}$ holds, with $\delta_{\min}$ being a parameter defined with preliminary experiments. Once this is done, we iterate by selecting a new route and iterate until $\lceil nd/100 \rceil$ requests have been removed.

B.5.3 Repair Operators

We built four operators, which all attempt to reinsert the removed requests by considering all routes in a parallel fashion. In case a repair operator does not manage to reinsert all removed requests, the solution is simply disregarded and the ALNS continues the search from x_{curr}.

Best Insertion. It considers the requests in inverse order with respect to their removal. For each request s, it considers all possible insertion positions, in all routes, and checks whether the insertion would be feasible and how much it would increase the solution value. It then reinserts s in the position leading to the lowest cost increase. The process is repeated until all requests have been reinserted.

Two-Regret Insertion. It works as Best Insertion, but the requests are inserted in non-increasing order of two-regret value. In detail, the operator evaluates for each request the costs of the cheapest insertion position and of the second cheapest insertion position, and it computes the two-regret as the difference between these two costs. It then selects the request of maximum regret and inserts it in the cheapest position. It reiterates, recomputing all regret values at each iteration, until all requests have been reinserted.

Forbidden Insertion. It works as Best Insertion, but disregards the possibility of reinserting a request in the same route which it was removed from.

Perturbation Insertion. It works as Best Insertion, but, any time it computes the cost of inserting a request in a position, it multiplies the cost by a perturbation factor p randomly selected in the interval $[0.8, 1.2]$. The idea, inspired by Ropke and Pisinger (2006), is to add a further level of diversification to the repair process.

Parallel-Set Partitioning Operator. This is the most complex repair method. Starting from the removed requests, the partially destroyed routes, and the airplanes that have not been used, if any, it builds a complete solution by invoking the heuristic of Section B.5.1. It invokes the heuristic β times, storing not only the best solution but also all routes from the generated solutions. These routes are then passed to the SP model of Section B.4, which is executed for a limited time. The best solution obtained is then returned.

B.5.4 Adaptive Weight Adjustment

We follow an approach that mimics the classical one of Ropke and Pisinger (2006). At each iteration, we randomly select first a destroy method and next a repair method according to probabilities that depend on the previous results obtained during the ALNS search. Let M be the set of available destroy methods, $o \in M$ the index of a destroy method, and t the index of the ALNS iteration. At iteration t, each method o is associated with a non-negative weight w^d_{ot}. The weights are used to select a method, according to probabilities $p^d_o = w^d_{ot} / \sum_{q \in M} w^d_{qt}$, for $o \in M$. The same process is applied to select a repair method, for which we produce instead w^r_{ot} weights and p^r_o probabilities.

At the first iteration, all weights are set to the same value, both for destroy and repair, so that they have identical probabilities. Every time a new solution is accepted (as described in Section B.5.6 below) the weights of the selected pair of destroy and repair methods are updated. The rationale behind our weight updating is to reward methods that find new improved solutions, with possibly a low computational effort. If a destroy method o has been selected at iteration t and produced an accepted solution, its weight at the next iteration is updated as

$$w^d_{o,t+1} = w^d_{ot} - \text{time}_o/\text{time}_{max} + \Delta_t/\Delta_{max},$$

where time_o is the computing time spent by o, Δ_t is the difference between the cost of the previous and new solutions, computed according to (B.3), and time_{max} and Δ_{max} are normalization parameters. The same process is used to update the removal weights w^r_{ot}.

B.5.5 Local Search

The local search approach that we implemented is shown in Algorithm 8. It attempts to improve the input solution by means of four different neighborhoods, invoked one after the other.

Algorithm 8 Local Search

```
1: procedure LS(x_input)
2:     x_LS ← Move(x_input)
3:     x_LS ← Swap(x_LS)
4:     if x_LS = x_input then x_LS ← Inter-move(x_LS)
5:     x_LS ← Time-window manipulation(x_LS)
6:     return x_LS
7: end procedure
```

Move. It is an intra-route search that attempts moving a vertex r_i from its current position to another position in route r. For each attempted move, the feasibility of all constraints is

checked. In addition, the relocation of a vertex in another position might lead the route to perform two consecutive visits to the same airstrip. In such a case, the two visits are merged into a unique one. We consider a route r at a time, and a vertex in the route at a time, starting from r_2 and continuing until $r_{|r|}$. We attempt each possible relocation, and the one being feasible leading to the highest value, if any, is implemented. The procedure is performed for all routes.

Swap. This procedure is also an intra-route local search. A swap is an intra-route interchange of the positions of two vertices. The process is equivalent to Move, with the only exception that, instead of moving a vertex r_i after another vertex r_j, it swaps r_i with r_j. The swap is checked with respect to feasibility and cost, possibly considering the merging of two visits to the same airstrip into a single visit.

Inter-move. This is the only inter-route search that we implemented. Because it is quite expensive in terms of computing effort, we invoke it only in case the previous intra-route searches failed in finding an improvement. We consider a route and attempt removing from it a pair of consecutive vertices. This is done only if there are no requests that use just one of the two vertices. In such a case, indeed, the removal would create infeasibilities for such requests. We accept, instead, the case in which some requests are picked up in the first vertex and dropped-off in the second. In this case, the requests are also removed from the route. Once the pair of consecutive vertices, and the associated requests, have been removed, we attempt inserting it in all possible positions in the other routes. The insertion being feasible and leading to the highest value reduction, if any, is implemented. The process is iterated until all routes have been scanned.

Time-window manipulation. It is in an intra-route search that evaluates, for each route and for each visited vertex, if it is convenient to increase the time spent by the airplane on the ground. It is just focused on user inconvenience minimization. The sequence of visits of the route is untouched. We consider the first vertex in the route and try to increase the time on ground from the original minimal ground time of the airstrip, by attempting all increases of two minutes each, up to a total of a one-hour increase. The time giving the minimal overall user inconvenience is selected, and the process is iterated from the next vertex until all vertices have been scanned.

B.5.6 Acceptance and Stopping Criteria

We use a simulated annealing acceptance criterion and a geometrical cooling schedule (Delahaye et al., 2019). Given a current solution x, a new solution x' is accepted with probability $P = e^{-(z(x')-z(x))/T_t}$, where $T_t > 0$ is the temperature at iteration t. The temperature

starts at $T_1 = T_{\text{start}}$ and is decreased every θ iterations using the expression $T_{t+1} = \alpha T_t$, where $0 < \alpha < 1$ is the cooling rate. Good values for parameters T_{start}, θ and α have been decided on the basis of preliminary computational tests. These tests considered the setting for both the first $iter_{phase1}$ calls to the ALNS method, where we aim at a large diversification, and for the successive calls, where we aim at intensifying the search in promising areas. The value of T_{start} is not fixed as a general input parameter, but we calculate it for each instance, considering the solution of our constructive heuristic. Further details are provided in Section B.6.

The ALNS is stopped as soon as one of the following conditions is met: $I_{\max}$ iterations without accepting a new solution have elapsed; a minimum threshold temperature $t_{\min}$ has been reached; or $\Theta_{\max}$ iterations in total, independently from acceptance, have elapsed.

B.6 Computation Results

In this section, we report the outcome of extensive computational tests that we performed to evaluate the iterated ALNS heuristic. The parameters required by the algorithm were set on the basis of preliminary tests, as follows: in Algorithm 6, $iter_{\max}$=10 and $iter_{phase1}$=$iter_{\max}/2$; for the destroy operators, $[d_{min}, d_{max}]$=[0.2, 0.6] and $\delta_{\min}$=2000; for the Parallel-Set Partitioning repair operator, β is one third of the number of passengers removed from the destroy method (rounded up to the next integer if fractional); for the adaptive weight adjustment, $time_{max}$=20 seconds and Δ_{max}=10000. In terms of acceptance and stopping criteria, during phase 1 we set T_{start}=25000, θ=55, α=0.87, $I_{\max}$=30, $t_{\min}$=150, and $\Theta_{\max}$=20000, whereas in the second phase we set T_{start}=2500, θ=30, α=0.95, $I_{\max}$=80, $t_{\min}$=50, and $\Theta_{\max}$=20.

The algorithm was implemented in C++, and CPLEX 12.9 was used as MILP solver. Computations were made on the computer cluster Beluga from CIRRELT, which uses Intel Gold 6148 Skylake processors running at 2.40 GHz. The tests were performed on a set of real-world instances obtained from the industrial partner. Details on the instances are given in Section B.6.1, while in Section B.6.2 we contrast our results with those of the company and present a detailed computational analysis.

B.6.1 Instances

The instance set was created by considering 24 days of activities of the company, as outlined in Table B.1. The days are distributed in different times of the year and are consequently characterized by different tourist requests. We divided the instances into three groups: "small" instances have fewer than 150 requests; "medium" instances between 150 and 280 requests; and "large" instances more than 280 requests. Apart from the number of requests, we also provide the number of airstrips, airplane types, and airplanes available. It can be noticed that the test set is quite varied, involving cases having between 91 and 343

requests, between 14 and 21 airstrips, and between 7 and 15 airplanes.

Table B.1: Instance characteristics

demand	ID	n	$\|V\|$	$\|F'\|$	$\|F\|$	demand	ID	n	$\|V\|$	$\|F'\|$	$\|F\|$	demand	ID	n	$\|V\|$	$\|F'\|$	$\|F\|$
small	1	91	16	1	11	medium	9	220	19	1	12	large	17	288	18	2	14
small	2	96	16	1	7	medium	10	222	18	2	12	large	18	289	18	2	11
small	3	101	14	1	9	medium	11	226	13	2	11	large	19	292	20	2	14
small	4	110	21	1	10	medium	12	252	16	2	13	large	20	300	16	2	14
small	5	112	19	1	10	medium	13	269	16	2	12	large	21	316	18	2	15
small	6	123	20	1	11	medium	14	271	18	2	12	large	22	320	18	2	15
small	7	125	19	1	10	medium	15	274	19	2	12	large	23	332	21	2	15
small	8	138	21	2	10	medium	16	285	17	1	13	large	24	343	14	2	14

As outlined in Section B.2.4, for each instance we tested two cost scenarios. The first one corresponds to a short-term view of the problem, in which the daily cost for using an airplane simply amounts to its mandatory daily fee. The second one corresponds instead to a long term view, in which the airplane cost also includes the daily operating cost of staff salary and amortization.

B.6.2 Results on the short-term scenario and comparison with the company

In Table B.2, we present the results we obtained on the short-term scenario, and compare them with the solution implemented by the company. For each instance and each solution, we provide in order: the number of airplanes used (denoted by $|\widetilde{F}|$ in the table); the fuel consumed in liters (fuel); the total distance traveled in kilometers (km); the number of intermediate stops performed (IS); the total time window violation in minutes (TWV); and the objective function values, namey cost (c, computed as in (B.1)), penalty (u, computed as in (B.2)), and overall objective (z, computed as in (B.3) as the sum of c and u). For the iterated ALNS, we also provide the percentage gap from the company solution, computed as $(z_{ALNS} - z_{company})/z_{company} \times 100$, and the overall execution time, in the format h:mm:ss.

From the table, it can be noticed that the iterated ALNS finds solutions that consistently outperform those produced by the company, with percentage gaps ranging from -13.7% to -57.7%, and being -33.4% on average. The solutions by the company are produced manually: the geographical area is divided into two sub-areas, North and South of Tanzania; two employees construct the partial solutions for each area, with the use of Excel files; finally, the two solutions are merged together with some possible adjustments. The process of creating the solution (which we recall is executed the day before) can also be affected by some partial or late information on requests and airplane status, which might cause further adjustments. In this context, the use of the iterated ALNS is well motivated, also due to the fact that the processing times are not excessive, ranging from about seven minutes to about three and a half hours, and being on average around one hour and a quarter.

Table B.2: Comparison with company solutions on the short-term scenario

	company								iterated ALNS									
ID	$\|\widetilde{F}\|$	fuel	km	IS	TWV	c (1)	u (2)	z (3)	$\|\widetilde{F}\|$	fuel	km	IS	TWV	c (1)	u (2)	z (3)	gap%	time$_{tot}$
1	11	5852	10888	57	4088	17806.5	5058	22864.5	10	4197	7464	49	1847	12539.5	2337	14876.7	-34.9	0:07:36
2	7	3069	5416	71	2859	9218.7	3889	13107.7	7	2934	5189	80	1744	8772.2	2544	11315.8	-13.7	0:09:32
3	9	3855	7053	63	3112	11750.5	3842	15592.5	9	2909	5136	58	847	8825.9	1427	10253.2	-34.2	0:14:05
4	10	4701	8873	58	1888	14461.4	2748	17209.4	10	4001	7094	32	1067	11968.0	1387	13355.3	-22.4	0:14:33
5	10	4926	9138	76	3120	15085.1	4040	19125.1	10	3495	6156	92	1474	10549.1	2394	12943.0	-32.3	0:14:54
6	11	5449	10154	52	2812	16649.3	3492	20141.3	9	1363	2430	148	2860	4813	4340	9153.3	-54.6	0:27:29
7	10	5626	10262	95	4219	16979.3	5349	22328.3	9	4189	7416	107	4880	12473.8	5950	18423.7	-17.5	0:19:37
8	10	5796	10596	43	5636	17480.3	6206	23686.3	10	3882	6838	69	1958	11658.9	2648	14306.9	-39.6	0:20:36
9	12	5907	10689	114	6195	17842.7	7575	25417.7	12	5593	9879	91	2614	16625.1	3524	20148.9	-20.7	0:54:02
10	12	6782	12515	118	3975	20517.4	5355	25872.4	12	5897	10469	98	2644	17457.9	3624	21081.7	-18.5	1:08:06
11	11	6442	11928	154	8835	19507.8	10575	30082.8	10	5043	8934	91	3309	14906.7	4219	19125.6	-36.4	0:49:34
12	13	7548	13978	112	10673	22919.7	12013	34932.7	13	5495	9707	103	3325	16359.9	4355	20714.8	-40.7	0:57:16
13	12	7250	13240	74	8488	21794.8	9348	31142.8	12	4944	8738	125	3160	14858.8	4409	19268.3	-38.1	1:33:17
14	12	8414	15199	188	10068	25015.8	12128	37143.8	12	5664	10037	139	6276	16790.2	7666	24455.7	-34.2	1:26:22
15	12	7807	14081	190	7311	23230.9	10151	33381.9	12	3480	6180	303	7795	10984.9	10825	21810.3	-34.7	1:56:16
16	13	7128	12852	130	9720	21351.4	11020	32371.4	13	6483	11447	97	2826	19192.3	3796	22987.9	-29.0	1:41:01
17	14	10650	19333	211	8851	31740.4	11101	42841.4	14	7325	12971	173	4893	21756.2	6623	28379.0	-33.8	1:41:28
18	11	7555	13851	171	13254	22804.8	15044	37848.8	11	5849	10322	173	4253	17329.6	5983	23312.2	-38.4	2:06:11
19	14	8168	14442	152	9536	24116.5	11336	35452.5	14	5911	10422	120	3247	17597.4	4447	22044.7	-37.8	1:09:34
20	14	8497	15578	162	16687	25576.1	19007	44583.1	14	6319	11214	126	3235	18863.4	4495	23358.0	-47.6	2:06:46
21	15	8973	16278	184	14951	26898.6	17871	44769.6	15	2873	5112	381	5499	9648.8	9309	18957.7	-57.7	3:27:20
22	15	8914	16443	135	11592	27014.3	13462	40476.3	15	7302	12956	149	5133	21742.7	6623	28365.9	-29.9	2:02:45
23	15	8704	15343	158	16612	25682.9	18692	44374.9	15	7696	13593	206	4179	22917.7	6239	29156.2	-34.3	2:39:58
24	14	8569	15661	196	9355	25763.3	11475	37238.3	14	7609	13474	172	5003	22507.1	6723	29229.7	-21.5	2:54:58
AVG	12.0	6941	12658	124	8077	20883.7	9616	30499.4	11.8	5019	8882	133	3503	15047.5	4829	19876.0	-33.4	1:16:48

Strong improvements can be noticed both in the fuel consumption and in the distance travelled, as well as on the two penalizations caused by intermediate stops and time window violations. In terms of airplanes used, the iterated ALNS can reduce this number only for a couple of instances, proving that the fleet is usually well dimensioned for the requests under this service level.

We also attempted to evaluate a long-term scenario, where the daily cost of the airplanes has been increased as previously discussed. The aim is to understand if it is acceptable to reduce the fleet size and how this would affect the service level and the other daily operational costs. The results that we obtained are given in Table B.3. The meaning of the columns is the same as in Table B.2. We report again the details of the short-period test to facilitate comparison. We omit the columns with cost c and overall objective function z, as these are affected by the difference in the input costs and cannot be compared between the two scenarios.

We can notice that the number of used airplanes is reduced for all instances when considering the long-term scenario. On average, this number decreases from 11.8 to 9, with a consequent variation of about 24%. This can be imputed to the higher daily usage cost. The side effect is that the routes performed by the airplanes are longer. This can be noticed by considering that the number of airplanes is reduced but at the same time both fuel consumed and traveled distance increase. Another side effect is the increase in the user inconvenience function u. This can be mostly attributed to the increase in the time window violation, that is almost doubled, whereas the number of intermediate stops is not considerably affected.

Table B.3: Comparison of iterated ALNS results between short- and long-term scenarios

	Short-term							Long-term						
ID	$\lvert\widetilde{F}\rvert$	fuel	km	IS	TWV	u (2)	time$_{tot}$	$\lvert\widetilde{F}\rvert$	fuel	km	IS	TWV	u (2)	time$_{tot}$
1	10	4197	7464	49	1847	2337	0:07:36	6	3881	6906	68	3920	4600	0:05:38
2	7	2934	5189	80	1744	2544	0:09:32	6	3232	5723	77	2087	2857	0:48:31
3	9	2909	5136	58	847	1427	0:14:05	6	2909	5129	83	2082	2912	0:07:47
4	10	4001	7094	32	1067	1387	0:14:33	6	3791	6715	84	4398	5238	0:10:53
5	10	3495	6156	92	1474	2394	0:14:54	6	3405	5997	89	4843	5733	0:09:58
6	9	1363	2430	148	2860	4340	0:27:29	6	1700	3018	152	3974	5494	0:23:39
7	9	4189	7416	107	4880	5950	0:19:37	6	4259	7539	115	4580	5730	0:13:22
8	10	3882	6838	69	1958	2648	0:20:36	7	4293	7556	68	5378	6058	0:13:52
9	12	5593	9879	91	2614	3524	0:54:02	9	5400	9535	159	6717	8307	0:37:09
10	12	5897	10469	98	2644	3624	1:08:06	10	5969	10595	89	5053	5943	0:43:07
11	10	5043	8934	91	3309	4219	0:49:34	8	5070	8981	79	7294	8084	0:38:55
12	13	5495	9707	103	3325	4355	0:57:16	11	5556	9814	116	6296	7456	0:58:02
13	12	4944	8738	125	3160	4409	1:33:17	10	5037	8899	87	4886	5756	0:55:15
14	12	5664	10037	139	6276	7666	1:26:22	10	5700	10102	112	7720	8840	0:53:35
15	12	3480	6180	303	7795	10825	1:56:16	9	4318	7669	263	9709	12339	1:26:07
16	13	6483	11447	97	2825	3796	1:41:01	10	6526	11521	145	7416	8866	0:51:35
17	14	7325	12971	173	4893	6623	1:41:28	12	7586	13456	133	9864	11194	0:52:23
18	11	5849	10322	173	4253	5983	2:06:11	10	5800	10255	141	8073	9483	1:03:48
19	14	5911	10422	120	3247	4447	1:09:34	10	6002	10584	103	9343	10373	1:08:44
20	14	6319	11214	126	3235	4495	2:06:46	11	6319	11210	133	8466	9796	1:18:47
21	15	2873	5111	381	5499	9308	3:27:20	8	2633	4686	353	13376	16906	2:49:10
22	15	7302	12956	149	5133	6623	2:02:45	13	7763	13750	128	9245	10525	1:18:35
23	15	7696	13593	206	4179	6239	2:39:58	13	7514	13273	176	8716	10476	1:27:00
24	14	7609	13474	172	5003	6723	2:54:58	12	7810	13829	153	7665	9195	2:01:13
AVG	11.8	5019	8882	133	3503	4828	1:16:48	9.0	5103	9031	129	6713	8007	0:53:25

With the aim of determining the best balance between the time spent in the heuristic search and in the MILP model solution by the solver, we performed an additional computational analysis in which we attempted different values of the number of calls to Algorithm 7 inside the iterated ALNS. This has been obtained by changing the value of $iter_{\max}$ (see step 5 of Algorithm 6), which also represents the number of calls to the set partitioning model (step 19 of Algorithm 7). The results that we obtained are summarized in Table B.4, where each line provides average values over the entire set of 24 instances. The columns have the same meaning as those in the previous tables, with the exception of a new column, called time$_{incumbent}$, which has been included to present the average time in which the incumbent solution was obtained. We can notice that the attempt with $iter_{\max}$=10 (i.e., the value we adopted for all our previous experiments) gives slightly better results than the other attempts, providing lower cost and user inconvenience values. The improvements are quite small with respect to the values obtained with six and 12 iterations, but much better with respect to those obtained with just one or two attempts. This proves that a good number of calls to the set partitioning model is beneficial for the overall algorithm, and that, when this value is sufficiently large, the algorithm becomes robust.

Table B.4: Analysis for different calls to the set partitioning model (24 instances per line)

$iter_{\max}$	$\lvert\widetilde{F}\rvert$	c (1)	u (2)	z (3)	$\text{time}_{incumbent}$	time_{tot}
1	9.0	32363.4	9655	42018.5	00:43:25	00:53:19
2	8.8	31255.5	9596	40851.9	00:41:08	00:54:09
6	9.1	31441.8	8391	39833.0	00:48:51	00:53:46
10	9.0	31018.0	8007	39025.0	00:47:10	00:53:25
12	9.0	31064.5	8008	39072.7	00:52:23	00:55:12

B.7 Conclusions

In this paper, we studied the Dial-A-Flight operations of one of the major Safari airline companies in Tanzania. The problem they face, denoted DAFPAS, is very challenging because it combines a heterogeneous aircraft fleet, multiple depots, flexible time windows, different operational costs, and the need to provide a good service level. The service level is measured by the number of intermediate stops that passengers undertake during transportation, and the possible violation of the flexible time window constraints. Another complicating issue in the problem originates from the fact that refueling is possible only at a limited number of airstrips.

We solved the DAFPAS heuristically with an iterated ALNS algorithm. Consistently with the literature, the ALNS proved to be effective in dealing with large size instances, finding solutions that were consistently better than those produced manually at the company. Local search and a set partitioning model helped improve the performance of the heuristic. In particular, it was shown that it is better to invoke the set partitioning model many times, with a shorter time limit, instead of just once with a longer limit.

An interesting future research direction is to consider the planning of the itineraries for multiple consecutive days, so as to find the best airstrips where to stop during the night and start at the next morning. That would require modifying the heuristic algorithm, both for what concerns ALNS, local search and set partitioning components, by considering the fact that routes selected for a given day should be connected with the routes in the next day. Such an approach could be employed within a rolling horizon framework.

Another interesting research avenue concerns the opportunity to issue low-cost last minute fares, so as to fill remaining seats at the planned trips, or obtain a better use for trips that are only meant at relocating the aircraft to meet successive requests. Alternative means of transport could also be taken into account. Indeed, for itineraries of limited distance, the tourists can also be offered to move on road, as in traditional ground safaris.

Acknowledgements

Computations were made on the supercomputer Beluga from CIRRELT, managed by Calcul Québec and Compute Canada. The operation of the supercomputer is funded by the Canada Foundation for Innovation, Ministre de l'Économie et de l'Innovation, and Fonds de recherche Nature et technologies - Québec. We acknowledge financial support from University of Modena and Reggio Emilia, under grant FAR 2018. We are grateful to Enrico Tognoni for providing us with data and several interesting insights on air safaris.

B.8 Bibliography

C. Barnhart, P. Belobaba, A. Odoni. Applications of operations research in the air transport industry. *Transportation Science*, 37(4):368–391, 2003.

M. Battarra, J.-F. Cordeau, M. Iori, Pickup and delivery problems for goods transportation, in: P. Toth, D. Vigo (Eds.), Vehicle Routing: Problems, Methods, and Applications, MOS-SIAM Series on Optimization, SIAM, 2014, pp. 161–192.

T. Bektaş, E. Demir, G. Laporte, Green vehicle routing, in: Green Transportation Logistics, Springer, 2016, pp. 243–265.

V. Cacchiani, J.-J. Salazar-González. Heuristic approaches for flight retiming in an integrated airline scheduling problem of a regional carrier. *Omega*, 91:102028, 2020.

J.-F. Cordeau, G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003.

J.-F. Cordeau, G. Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

J.-F. Cordeau, G. Laporte, J. Potvin, M. Savelsbergh. Chapter 7 in Transportation on Demand. *Handbooks in Operations Research and Management Science*, 14:429–466, 2007.

R. de Alvarenga Rosa, A. Machado, G. Ribeiro, G. Mauri. A mathematical model and a clustering search metaheuristic for planning the helicopter transportation of employees to the production platforms of oil and gas. *Computers & Industrial Engineering*, 101:303–312, 2016.

D. Delahaye, S. Chaimatanan, M. Mongeau, Simulated annealing: From basics to applications, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, volume 272 of *International Series in Operations Research & Management Science*, Springer, 2019, pp. 1–35.

M. Dell'Amico, J. Díaz Díaz, G. Hasle, M. Iori. An Adaptive Iterated Local Search for the Mixed Capacitated General Routing Problem. *Transportation Science*, 50(4):1223–1238, 2016.

G. Desaulniers, J. Desrosiers, Y. Dumas, M. Solomon, F. Soumis. Daily aircraft routing and scheduling. *Management Science*, 43(6):841–855, 1997.

K. Doerner, J.-J. Salazar-González, Pickup and delivery routing problems for people transportation, in: P. Toth, D. Vigo (Eds.), Vehicle Routing: Problems, Methods, and Applications, MOS-SIAM Series on Optimization, SIAM, 2014, pp. 193–212.

F. Engineer, G. Nemhauser, M. W. Savelsbergh. Dynamic programming-based column generation on time-expanded networks: Application to the dial-a-flight problem. *INFORMS Journal on Computing*, 23(1):105–119, 2011.

D. Espinoza, R. Garcia, M. Goycoolea, G. Nemhauser, M. W. Savelsbergh. Per-seat, on-demand air transportation Part I: problem description and an integer multicommodity flow model. *Transportation Science*, 42(3):263–278, 2008a.

D. Espinoza, R. Garcia, M. Goycoolea, G. Nemhauser, M. W. Savelsbergh. Per-seat, on-demand air transportation Part II: Parallel local search. *Transportation Science*, 42(3):279–291, 2008b.

K. Fagerholt, B. Foss, O. Horgen. A decision support model for establishing an air taxi service: a case study. *Journal of The Operational Research Society*, 60(9):1173–1182, 2009.

M. Fiala Timlin, W. Pulleyblank. Precedence constrained routing and helicopter scheduling: heuristic design. *Interfaces*, 22(3):100–111, 1992.

A. Fügenschuh, G. Nemhauser, Y. Zeng, Scheduling and routing of fly-in safari planes using a flow-over-flow model, in: M. Jünger, G. Reinelt (Eds.), Facets of Combinatorial Optimization, Springer, Berlin, Heidelberg, 2013, pp. 419–447.

T. Gschwind, M. Drexl. Adaptive Large Neighborhood Search with a Constant-Time Feasibility Test for the Dial-a-Ride Problem. *Transportation Science*, 53(2):480–491, 2019.

N. Hermeto, V. Ferreira Filho, L. Bahiense. Logistics network planning for offshore air transport of oil rig crews. *Computers & Industrial Engineering*, 75:41–54, 2014.

S. C. Ho, W. Szeto, Y.-H. Kuo, J. M. Leung, M. Petering, T. W. Tou. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421, 2018.

T. Kannon, S. Nurre, B. Lunday, R. Hill. The aircraft routing problem with refueling. *Optimization Letters*, 9(8):1609–1624, 2015.

P. Keskinocak, S. Tayur. Scheduling of time-shared jet aircraft. *Transportation Science*, 32(3):277–294, 1998.

D. Klabjan, Large-scale models in the airline industry, in: G. Desaulniers, J. Desrosiers, M. Solomon (Eds.), Column Generation, Springer, Boston, MA, 2005, pp. 163–195.

E. Lacasse-Guay, G. Desaulniers, F. Soumis. Aircraft routing under different business processes. *Journal of Air Transport Management*, 16(5):258–263, 2010.

C. Martin, D. Jones, P. Keskinocak. Optimizing on-demand aircraft schedules for fractional aircraft operators. *Interfaces*, 33(5):22–35, 2003.

M. Masmoudi, K. Braekers, M. Masmoudi, A. Dammak. A hybrid Genetic Algorithm for the Heterogeneous Dial-A-Ride Problem. *Computers & Operations Research*, 81:1–13, 2017.

R. Masson, F. Lehuédé, O. Péton. An Adaptive Large Neighborhood Search for the Pickup and Delivery Problem with Transfers. *Transportation Science*, 47(3):344–355, 2013.

F. Menezes, O. Porto, M. Reis, L. Moreno, M. Poggi de Aragão, E. Uchoa, H. Abeledo, N. Carvalho do Nascimento. Optimizing helicopter transport of oil rig crews at petrobras. *Interfaces*, 40(5):408–416, 2010.

P. Munari, A. Alvarez. Aircraft routing for on-demand air transportation with service upgrade and maintenance events: Compact model and case study. *Journal of Air Transport Management*, 75:75–84, 2019.

National Bureau of Statistics, Tanzania, The 2017 International Visitors' Exit Survey Report. *Technical Report*. available at https://www.bot.go.tz/Publications/TTSS/TTSS-2017.pdf (last accessed March 2020), 2018.

A. Parmentier, F. Meunier. Aircraft routing and crew pairing: Updated algorithms at Air France. *Omega*, 93:102073, 2020.

S. Parragh, K. Doerner, R. Hartl. Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 37(6):1129–1138, 2010.

D. Pisinger, S. Ropke, Large neighborhood search, in: M. Gendreau, J. Potvin (Eds.), Handbook of metaheuristics, volume 146 of *International Series in Operations Research & Management Science*, Springer, Boston, MA, 2010, pp. 399–419.

F. Qian, I. Gribkovskaia, G. Laporte, H. sr Ø. Passenger and pilot risk minimization in offshore helicopter transportation. *Omega*, 40(5):584–593, 2012.

D. Ronen. Scheduling charter aircraft. *Journal of the Operational Research Society*, 51(3):258–262, 2000.

S. Ropke, D. Pisinger. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4):455–472, 2006.

P. Shaw, A new local search algorithm providing high quality solutions to vehicle routing problems. *Technical Report*. Dept. of Computer Science, University of Strathclyde, Glasgow, Scotland, UK, 1997.

A. Subramanian, E. Uchoa, L. Ochi. A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519–2531, 2013.

F. Van der Zwan, K. Wils, S. Ghijs, Development of an aircraft routing system for an air taxi operator, in: Aeronautics and Astronautics, IntechOpen, 2011.

J. Wensveen. *Air transportation: A management perspective*. Routledge, 2016.

A. Yamani, T. Hodgson, L. Martin-Vega. Single aircraft mid-air refueling using spherical distances. *Operations Research*, 38(5):792–800, 1990.

W. Yang, I. Karaesmen, P. Keskinocak, S. Tayur. Aircraft and crew scheduling for fractional ownership programs. *Annals of Operations Research*, 159(1):415–431, 2008.

Y. Yao, Ö. Ergun, E. Johnson, W. Schultz, J. Singleton. Strategic planning in fractional aircraft ownership programs. *European Journal of Operational Research*, 189(2):526–539, 2008.

Y. Yuan, A. Mehrez. Refueling strategies to maximize the operational range of a nonidentical vehicle fleet. *European Journal of Operational Research*, 83(1):167–181, 1995.