

CRISTIANE MARIA SANTOS FERREIRA



Scheduling in Collaborative and Dynamic Environments

Submitted to Faculdade de Engenharia da Universidade do Porto in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Industrial Engineering and Management, supervised by Pedro Amorim and Gonalo Figueira, Assistant Professors of Faculdade de Engenharia da Universidade do Porto

DEPARTMENT OF INDUSTRIAL ENGINEERING AND MANAGEMENT
FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

2022

This research was partially supported by the PhD grant 2020.09619.BD awarded by the Portuguese Foundation for Science and Technology. O trabalho de investigação apresentado nesta dissertação foi parcialmente apoiado pela bolsa de Doutoramento 2020.09619.BD, atribuída pela Fundação para a Ciência e a Tecnologia.

“The scientist is not a person who gives the right answers, he’s one who asks the right questions.”

- Claude Levi-Strauss

Acknowledgments

I would like to express my deep gratitude to Professors Pedro Amorim and Gonalo Figueira, my supervisors. Their patient guidance, encouragement and technical advice were fundamental to the achievements of this work. Furthermore, I am eternally grateful to my husband and daughter, Alexandre Medeiros and Mariana Sousa, for their unconditional daily support. I am also thankful to Professors Maria Antonia and Jos  Fernando Oliveira. Both gave me their kind attention and support in several situations. My grateful thanks are extended to Vale, particularly Alexandre Pigatti, for exchanging his expertise in bulk ports with me and providing real data.

Several colleagues and friends from CEGI (Inesctec) contributed to the extent of disciplines and skills I have learned during the last four years. They also supported me with their friendly assistance and addressed several doubts on a daily basis: Alexandra Oliveira, Beatriz Oliveira, Eduardo Curcio, Eduardo Oliveira, Fabio Neves, Flavia Barbosa, Lu s Dias, Maria Jo o Santos, Maria Jo o Pires, Masoud Golalikhani, Pedro Rocha, Sara Ali, Sara Martins, Sofia Gomes, Thomy Saputro and Xavier Andrade. I cannot forget the support received from CEGI's and DEGI's secretaries: M nica Silva, Isabel Ribeiro, Soledade Medeiros and Helena Silva. Moreover, I will always be thankful to my long-standing friend, Haroldo Gambini, who encouraged me to start my PhD studies and gave me relevant insights on Operational Research.

Finally, I would like to thank our Brazilian friends. We have shared the exciting experience of living in Portugal: Marcelo and Suzana; Alexandre and Carolina and their daughter Julieta; Thiago, Paola and their son Pedro. And I also extend my acknowledgement to the Portuguese funding agency, FCT - Funda  o para a Ci ncia e a Tecnologia, for financing the current work.

Abstract

The emergence of Industry 4.0 is bringing considerable improvements to operations, but also new challenges, as processes tend to be more flexible and dynamic. In manufacturing, one of the technologies that are doing exactly this is the new generation of robots, collaborative robots, which can work alongside humans and open new possibilities as Human-Robot Teams. With this improved performance and flexibility, questions arise as to which tasks should be executed by the different elements of the team and how to keep them synchronised. In logistics, we focus on bulk ports, where the increased demand combined with advanced technologies enforce higher service levels and maximum throughput. In this domain, inbound operations are critical and require solving challenging scheduling problems. Furthermore, both applications deal with uncertainty. Humans' behaviour is naturally stochastic, even working in collaboration with robots. In ports, operations are subject to frequent changes and breakdowns. Effective methods for dynamic scheduling are fundamental in both cases. This thesis aims at answering critical questions in both sectors by solving the underlying scheduling problem, the Multimode Multiprocessor Task Scheduling. In the deterministic case, optimisation approaches - exact and heuristic - are proposed and outperform existing literature and practice. These methods allowed studying the potential performance gains provided by collaborative work in human-robot cells. Dispatching rules are developed for addressing uncertainty. Genetic Programming (GP), an evolutionary hyper-heuristic, is the primary learning mechanism for automated rules generation. Moreover, we show that GP generates effective and interpretable rules by employing a guided-learning approach and incorporating domain expertise. Besides benchmarking with previously existing rules, they were also compared to other strategies, such as periodic re-scheduling. Results show that, in general, the evolved rules prevent performance deterioration in settings with high variation in processing times. And they are particularly effective in working cells where humans and robots must collaborate. In the context of ports, a good relative performance of dispatching rules is also found in terminals under high machines utilisation.

Resumo

A Indústria 4.0 trouxe processos mais dinâmicos e efetivos. Entretanto, também trouxe novos desafios para vários setores como manufatura e logística. Na manufatura, há que se considerar uma nova geração de robôs, os colaborativos, que trabalham ao lado de humanos e possibilitam equipas híbridas no chão de fábrica (os chamados *Human-Robot Teams*). Com esse aumento de performance e flexibilidade, surgem novas questões. Por exemplo, quais tarefas devem ser executadas por cada um dos elementos dessas equipas? E como sincronizá-las? Em logística, particularmente em portos de granel, houve um aumento considerável da procura, forçando melhores níveis de serviço e taxas de entrega. Nesse contexto, as operações de entrada nos portos são críticas e requerem a resolução de difíceis problemas de sequenciamento. Ainda, essas duas aplicações lidam com um alto nível de incerteza. O comportamento humano, por exemplo, é naturalmente estocástico, e portanto introduz incerteza em tarefas feitas por *Human-Robot Teams*. Nos portos, as operações geralmente estão sujeitas a frequentes mudanças e interrupções. Consequentemente, métodos específicos para sequenciamento dinâmico são fundamentais em ambos os casos. Esta tese visa responder questões críticas nos dois setores resolvendo o problema de sequenciamento que surge por trás, o *Multimode Multiprocessor Task Scheduling*. Métodos de otimização, exatos e heurísticos, são propostos e superam a literatura e a prática em performance. Os mesmos permitiram estudar os ganhos potenciais que se pode obter com a colaboração em células de trabalho compostas por humanos e robôs. São desenvolvidas também regras de despacho para endereçar a incerteza. *Genetic Programming* (GP), uma Hiper-heurística evolucionária, é o principal mecanismo de aprendizagem para a geração dessas regras. Comparadas a outras estratégias, as regras são promissoras de uma forma geral em ambientes muito sujeitos a incerteza. Além disso, ao empregar uma abordagem de aprendizagem guiada e incorporar conhecimento de domínio, demonstra-se que GP gera regras não somente efetivas, mas também interpretáveis. O comportamento das regras é também comparado ao de outras estratégias, como re-otimização periódica. Os resultados mostram que, em geral, as regras geradas previnem a deterioração de performance em ambientes com alta variação nos tempos de processamento. E são particularmente efetivas em células de trabalho onde humanos e robôs colaboram. No contexto portuário, tais regras também performam bem em portos que operam um número grande de produtos e sob altos níveis de utilização das máquinas.

Contents

1	Motivation and overview	1
1.1	Problems tackled	3
1.2	Research questions and objectives	3
1.3	Thesis structure	6
	Bibliography	7
2	Literature Review	9
2.1	Background solution approaches	9
2.1.1	Static scheduling	10
2.1.2	Dynamic scheduling	11
2.2	State of the art	15
2.2.1	Dynamic job shops	15
2.2.2	Parallel and multiprocessor tasks	18
2.3	Applications	20
2.3.1	Human-robot collaboration	20
2.3.2	Dry bulk ports	24
	Bibliography	26
3	Effective and interpretable rules via Genetic Programming	41
3.1	Introduction	42
3.2	Literature Review	44
3.2.1	Dispatching rules for dynamic job shop scheduling	44
3.2.2	GP on the design of dispatching rules	46
3.3	Methodology	47
3.3.1	Learning algorithm	49
3.3.2	Simulation model	52
3.3.3	Training, validation and test sets	53
3.3.4	Algorithm parameters	54
3.4	Learning iterations	55
3.4.1	Improving the terminal set	58

3.5	Evolved Rules	60
3.5.1	Rules selection	60
3.5.2	Performance under deterministic processing times	61
3.5.3	Performance under stochastic processing times	63
3.6	Conclusions and future work	63
	Bibliography	64
3.A	Job Shop Simulator	70
3.B	Terminal sets considered in GP experiments	72
3.C	Performance of evolved rules under stochastic processing times	73
4	Static scheduling of Human-Robot Teams	75
4.1	Introduction	76
4.2	Related Work	77
4.2.1	Scheduling in HRTs	77
4.2.2	Multiprocessor Task Scheduling	80
4.3	Problem Definition	81
4.3.1	Mixed Integer Programming Formulation	83
4.3.2	Lower bound	84
4.4	Constraint Programming Model	85
4.5	Genetic Algorithm	87
4.5.1	Encoding and Decoding	87
4.5.2	General Structure and Operators	88
4.6	Performance Evaluation	89
4.6.1	Performance Evaluation of the Proposed Methods	90
4.7	Advantage of Collaborative Work	95
4.7.1	Introducing Robots in Manual Lines	96
4.7.2	Multimode Eligibility and Performance	97
4.7.3	Precedence Levels	97
4.7.4	Humans and Robots Relative Performance	99
4.8	Conclusions and Future Research	99
	Bibliography	100
4.A	Proposed Genetic Algorithm	105
4.A.1	Decoding Algorithm	105

4.A.2	Genetic Algorithm - General Structure	105
4.B	Structure of Benchmark Algorithms	106
4.B.1	Lower bound LB2	106
4.B.2	Upper bound UB2	108
4.B.3	Genetic Algorithm (GA-Chen)	108
5	Dynamic scheduling of Human-Robot Teams	111
5.1	Introduction	112
5.2	Related work	114
5.2.1	Scheduling Human-Robot teams	114
5.2.2	Dispatching rules and GP for production scheduling	115
5.3	Problem definition	117
5.4	Methodology	119
5.4.1	Simulation model	120
5.4.2	Genetic Programming algorithm	121
5.4.3	Instance sets	125
5.4.4	Algorithm parameters	127
5.5	Evolved dispatching rules	128
5.5.1	Results validation	128
5.5.2	Best rules for the MTSP	130
5.5.3	Best rules for non-collaborative operations	131
5.6	Dispatching rules against near-optimal solutions	132
5.6.1	Effect of resource eligibility	133
5.6.2	Effect of the cell composition	134
5.6.3	Different levels of precedence constraints	135
5.6.4	Different levels of collaborative operations	136
5.7	Conclusion	137
	Bibliography	138
5.A	HRTs simulator	142
6	Static and dynamic scheduling of bulk ports operations	145
6.1	Introduction	146
6.2	Background	148
6.3	The Wagon Unloading Problem	150

6.4	Exact solution methods	153
6.4.1	Mixed Integer Programming formulation	153
6.4.2	Constraint Programming model	156
6.5	GP-based solution methods	159
6.5.1	Greedy-randomised heuristic	159
6.5.2	Genetic Programming	160
6.6	Experiments design	163
6.6.1	Instances	163
6.6.2	Algorithms parameters	165
6.7	Experiments results	166
6.7.1	Evolved rules	166
6.7.2	Deterministic settings	168
6.7.3	Managing uncertainty	170
6.8	Conclusions and future work	172
	Bibliography	174
6.A	Genetic Programming algorithm	178
6.B	Fictitious ports	179
6.C	Real instances	179
7	Conclusions and future work	181
7.1	Contributions	181
7.2	Answers to the research questions	183
7.3	Future work	185
A	Dispatching rules for job shops with static arrivals	187
A.1	Introduction	187
A.2	Methodology	189
A.2.1	Breeding	189
A.2.2	Function and Terminal Sets	190
A.2.3	Fitness Evaluation	190
A.3	Computational Experiments	191
A.3.1	Experimental Design	192
A.3.2	Results and Discussion	192
A.4	Conclusions and Future Work	195

Bibliography	195
------------------------	-----

Motivation and overview

Industrial environments experience disruptive progress with the advent of Industry 4.0 (i4.0). Several technologies enable real-time interaction between physical and digital machines, such as additive manufacturing, Internet of Things, distributed ledgers, advanced robotics and artificial intelligence [Olsen and Tomlin, 2020]. When working in concert, they leverage operations' performance, providing cost reduction and increased quality. These advancements contribute to higher customisation and an agile production approach in manufacturing and logistics, capable of responding quickly to new demands, breakdowns, and other unforeseen events.

Technological innovation brings efficiency but, at the same time, adds more complexity to operations management. Ivanov et al. [2020] point out several management streams impacted by i4.0, such as supply chain and scheduling. Processes are naturally more dynamic and subject to disruptions: constant job arrivals, lot size changes, cancellations, delays, production failures or machine breakdowns. These settings require rapid decisions to deal with the operations' dynamics [Parente et al., 2020]. This thesis tackles some of the new challenges that this context brings for scheduling and develop methods to address them.

Collaborative robots (cobots) emerge as a solution to achieve higher levels of performance and flexibility. They are light and sufficiently safe to work alongside humans, embodying Human-Robot Teams (HRTs). Closer cooperation between workers and automated systems opens new possibilities, such as quick changeability and real-time assignment of tasks. HRTs benefit from complementary features - while humans are proficient in several tasks and adaptive to changes, robots are precise and work without fatigue. For these reasons, cobots have been receiving a lot of attention in recent years from academia [Ivanov et al., 2020] and practitioners [Weber, 2018]. In 2019, when robots sales suffered a setback, cobots installations, in contrast, grew by 11% and reached 4.8% of the total market [IFR, 2020]. However, some questions and technical challenges must be addressed to boost HRTs implementation. Manufacturers need to redesign their processes and evaluate the benefits of employing HRTs. There are multiple trade-offs to be considered, between the high performance of traditional robots, the great flexibility of human operators, and the compromise solution of collaborative robots. On the other hand, effective dispatching methods must be developed to synchronise the team and take advantage of cobots.

Disruptive changes in operations management are expected beyond automated manufacturing. For instance, sourcing and transportation are directly influenced by artificial intelligence and communication technologies [Ivanov et al., 2020]. In maritime transportation, practitioners may optimise vessels utilisation and queues at ports by using data analytics

and real-time information (e.g., vessel location and asset utilisation). In particular, ore transportation is a key sector. Iron ore shipments amounted to 1.4 billion tons in 2019, corresponding to 8% of the total maritime volume [UNCTAD, 2020]. Indeed, since 2001, the global demand for iron ore doubled while production and logistics struggled to follow [Song et al., 2019]. Higher service levels for operations are naturally required. Miners and logistics operators must be able to enhance their efficiency to cope with these requirements. However, the research effort on optimising port operations concentrates on terminals, while the land-side is also critical. In iron ore ports, the bottleneck sometimes is the product reception, a complex process involving uncertainty in train arrivals, railroad cars manoeuvre, and the synchronisation of several resources through the stockyard.

Either in collaborative assembly or ports operations, processes must be adapted to achieve the desired results. In the case of HRTs, humans' presence introduces unpredictable behaviour into the execution of tasks. In iron ore transportation, higher competitiveness increases the inherent complexity of ports operation, which is subject to frequent unexpected interruptions. Effective methods are necessary to optimise these processes under all these sources of uncertainty and complexity. As a result, the scheduling of operations is also more complicated, as it must be designed to respond in real-time to possible deviations. These facts motivate the interest in dynamic scheduling methods. They are used to promptly re-optimize the operations' sequence in the presence of unexpected events.

Several studies have been conducted on developing robust schedules or on the adaptation of schedules in real-time. To tackle this problem, dispatching rules have been used with good results, particularly in the semiconductor industry [Varadarajan and Sarin, 2006]. As simple sequencing procedures, they are flexible, easy to implement, and extremely fast to execute. The evident limitation is their generally low effectiveness in comparison to more sophisticated heuristics. These rules have been traditionally derived by empirical or analytical studies, fed by strong domain expertise. To boost their effectiveness, machine learning algorithms appear as a promising alternative. However, most work on these algorithms focuses on generating rules by using extensive computational experiments with little or no business insights. As a result, these algorithms usually produce extremely long rules that cannot be interpreted.

This thesis tackles these questions and seeks new methods to address the corresponding scheduling problems. Mixed Integer Programming, Constraint Programming and Meta-heuristics are applied. Special attention is devoted to dynamic scheduling, which emerges as critical for enhancing operations in i4.0. Thus, research is conducted on the search for effective rules and heuristics for reactive schedules generation. Genetic Programming, an evolutionary approach, is applied to the search of those algorithms. The aim is to derive effective, compact and meaningful rules, one step beyond the studies in this area. To achieve these results, one must combine problem expertise with computational power. Therefore, an iterative methodology is followed, where insights on the problem guide the learning algorithms. The next section describes the specific scheduling problems tackled in this thesis.

1.1. Problems tackled

This thesis focus on two main applications: collaborative working cells and bulk cargo operations. The former is an emerging topic in the context of automated manufacturing. Conversely, the latter refers to wagons unloading, a traditional operation found in dry bulk ports. Expected market changes require better methods to deal with the dynamics of the operations in both contexts. Although they constitute challenging and relevant problems, very few studies about them are found in the scheduling literature.

The two applications studied compose very distinct use cases, but with common characteristics. In both cases, there are multiprocessor tasks, i.e., tasks which require multiple machines at the same time. Also, each task may be executed in different modes, where a mode is characterised by a subset of machines. We formalise both scheduling problems as particular applications of the Multimode Multiprocessor Task Scheduling Problem (MMTSP). In the MMTSP, one must assign tasks to modes without overlap in any machine. The aim is to optimise some performance criteria, such as the average or maximum completion time.

Figure 1.1 gives an overview of the thesis scope, composed of three main streams. For each problem/application, both the static and dynamic versions are presented. In static scheduling problems, all data is known in advance, and a complete schedule is generated *a priori*. The dynamic version incorporates uncertainty in the execution of operations and/or deals with unknown jobs arrivals. Optimisation methods, both exact and heuristic, are usually designed to solve the static version of the MMTSP. In the case of dynamic problems, dispatching rules are used. These rules are generated by a guided empirical learning approach, which uses Genetic Programming algorithms. The methodology is first tested in one of the most well studied problems, the Job Shop Scheduling Problem (JSSP), and then is generalised for the Parallel Machine Scheduling Problem and the MMTSP.

1.2. Research questions and objectives

This thesis concerns the development of scheduling methods in the context of flexible and dynamic environments. Five sub-objectives are derived:

- **O1** Develop a methodology for the generation of effective and interpretable dispatching rules.
- **O2** Generalise and validate the methodology in particular applications.
- **O3** Propose optimisation methods to study HRTs cyclic scheduling.
- **O4** Analyse the impacts of uncertainty in HRTs real-time scheduling.
- **O5** Develop scheduling heuristics for real-world bulk cargo operations.

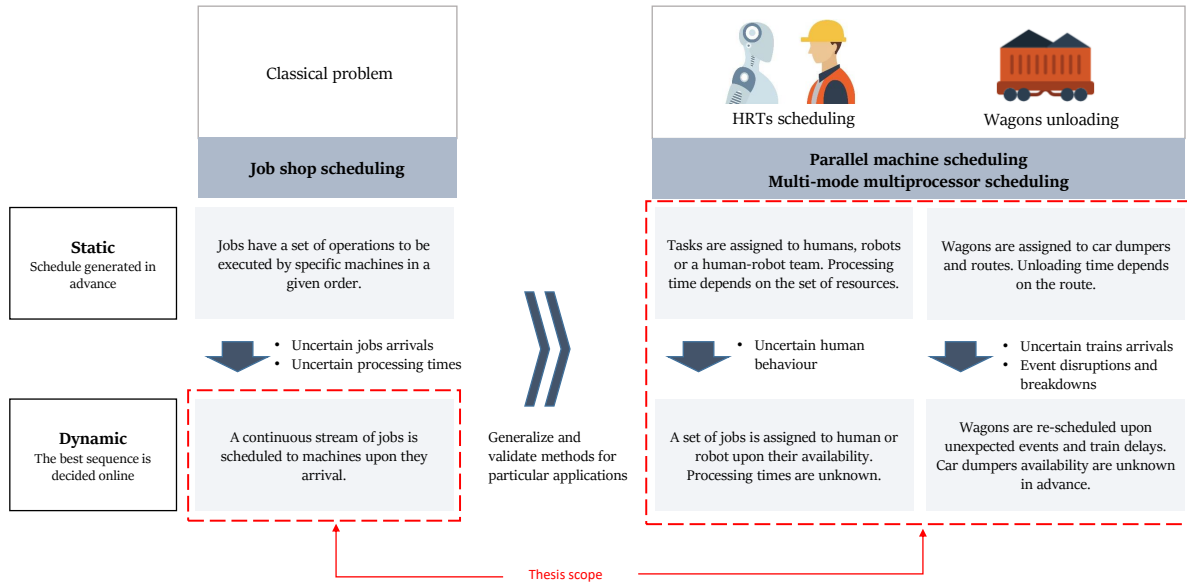


Figure 1.1 – Problems tackled

The methodology and framework to approach O1 pave the way to O2, O4 and O5. The framework is generalised to cope with multi-processor and multi-mode tasks - necessary for achieving O2. It is validated in collaborative cells (O4) and dry bulk ports (O5). The latter application is particularly important in this thesis, as it materialises in a real and challenging use case. Thus, O2 is indirectly achieved when O4 and O5 are concluded. O3 and O4 together address some of the main issues for operations management with the advent of cobots.

To accomplish these five objectives, four research questions are addressed (c.f. Figure 1.2):

- **Q1** How to design effective and interpretable dispatching rules?

Several authors have addressed the development of dispatching rules for scheduling problems, typically the Job Shop Scheduling Problem and its flexible variant. Many works have shown that GP can find rules that outperform the existing ones. However, two main concerns arise with them: (1) the algorithms are trained and validated in very specific settings, hence there is no guarantee that the resulting rules generalise well; (2) the rules generated are too long, possibly with unnecessary parameters. No justification is given for the feature selection, and authors usually cannot explain why the resulting rules work well in those settings.

This research question indirectly asks for the development of a methodology for designing interpretable rules. The goal is to generate rules for dynamic job shops that are at the same time effective, simple and interpretable. One must justify the

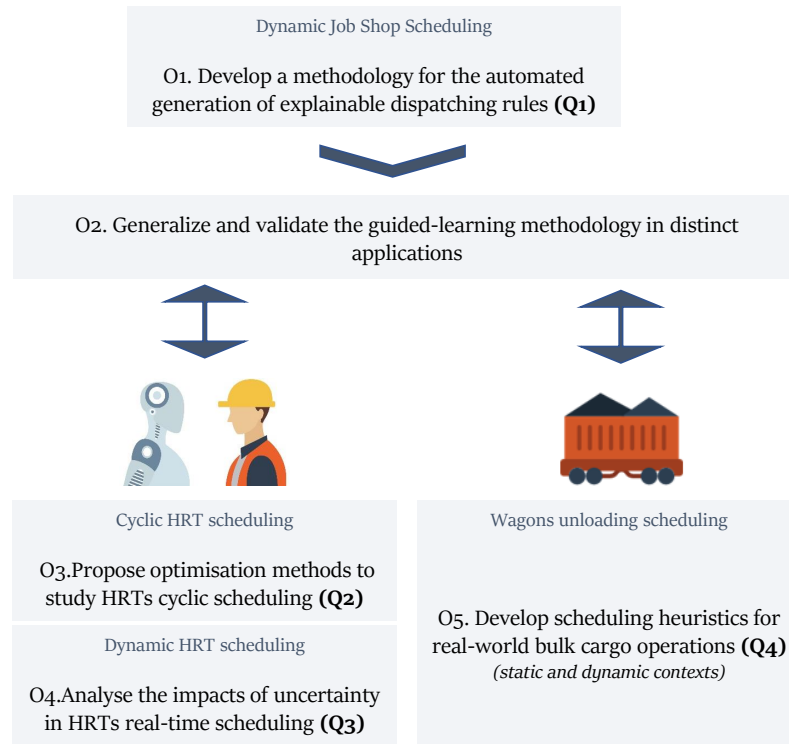


Figure 1.2 – Research objectives and their relationship with the research questions

influence of their parameters, and why that particular mathematical expression works well. Moreover, the rules must be effective in a wide range of settings, with distinct sizes, from highly congested shops to light loaded ones, from tight to loose due dates.

- **Q2** When and how to employ collaborative tasks?

The emergence of Human-Robot Teams (HRTs), where human workers and robots share the same workspace, has brought new challenges to decision-making. Important questions as to when and how to implement these teams are not trivial and require solving complex optimisation problems. Tasks may be executed by only one resource (whether human or robot) or by both in collaboration. Depending on the number of tasks that each resource can execute, their relative productivity (including when in collaboration) and several other aspects (such as task precedence), it might be worth, or not, to employ two resources instead of just a human or a robot alone. Determining how to do it involves solving task allocation and scheduling.

Q2 aims to develop methods to optimise the schedule of HRTs, considering all those parameters. These methods also support evaluating the improvements obtained with the *Cooperation* scenario compared to the more conservative *Coexistence* scenario.

In the former human and robot work alongside in separated tasks, while in the latter a closer interaction occurs with the existence of collaborative tasks.

- **Q3** How to dispatch tasks to HRTs in real-time?

Leveraging flexibility is one of the motivations for HRTs applications. Cobots are sufficiently light to be moved in the shopfloor, and are also easy to reprogram. On the other hand, humans can execute a wide range of tasks and quickly adapt to changes. However, all these features do not guarantee a perfect execution of jobs. Assembly lines are naturally subject to unexpected events, and humans' presence introduces uncertainty in the execution of tasks. It is crucial to overcome such disruptions in collaborative environments, especially if the team must be synchronised.

The third research question refers to the development of dynamic scheduling methods for HRTs, considering such specific characteristics. Robots are usually suitable for a subset of tasks, while humans are more flexible; operations processing times are not precise when executed by humans (uncertain processing times); if collaborative tasks exist, both the human and the robot must be available. Moreover, precedence constraints between tasks must be considered. Q3 asks how to deal with it by applying simple dispatching rules and the gains obtained compared to the deterministic solution.

- **Q4** How to effectively sequence wagons for unloading in dry bulk ports?

Optimising bulk ports is undoubtedly economically relevant due to their high participation in the global commercial trade [UNCTAD, 2020]. To maximise ports throughput, bottlenecks both in the income and outcome operations must be identified and addressed. In particular, iron ore ports get congested if product arrival is not coupled with an effective schedule of wagons unloading. The process is complex because it depends on the availability of several resources (car dumpers, conveyor belts, stackers) and breakdowns are unexpected and frequent. It becomes challenging if there is little flexibility for storage in the stockyard. Moreover, there is stochasticity in trains arrivals.

To answer Q4, the methodology developed for Q1 is adapted to schedule wagons for unloading. The deterministic version of the problem is solved for benchmarking the approaches. The aim is to develop a heuristic that combines dispatching rules for optimising the sequence of wagons. However, the fundamental question refers to the existence of a single strategy to accommodate disruptions in the baseline schedule.

1.3. Thesis structure

This thesis is composed of a collection of papers, and each chapter (or paper) is aligned with each research question previously presented. The current section provides an overview of the chapters that compose the thesis. In Chapter 2, we present the background of solution approaches for the scheduling problems tackled. We also review the related work and state-of-the-art of each application. Chapter 3 explores the proposed methodology by addressing

the Dynamic Job Shop Scheduling Problem. We generate effective and interpretable dispatching rules using Genetic Programming and propose a guided-learning approach. The HRTs application is tackled in Chapters 4 and 5. The former introduces the problem of scheduling HRTs in working cells and presents models and algorithms to the deterministic version. In the latter, we extend the analysis by simulating HRTs in the stochastic context. Dispatching rules are developed to assign and sequence operations to the HRTs dynamically. In Chapter 6, we study income operations in bulk ports. The deterministic version of the problem is formalised and solved using exact and heuristic approaches. We also develop dispatching rules and use data from real ports to simulate several strategies for dynamic scheduling. Finally, Chapter 7 summarises the findings and contributions of the entire work. We also provide answers to the research questions and present some directions for future work.

Bibliography

IFR. World robotics 2020 industrial robots. Technical report, International Federation of Robotics, 2020. URL <https://ifr.org>.

Dmitry Ivanov, Christopher S. Tang, Alexandre Dolgui, Daria Battini, and Ajay Das. Researchers' perspectives on industry 4.0: multi-disciplinary analysis and opportunities for operations management. *International Journal of Production Research*, 0(0):1–24, 2020. doi: 10.1080/00207543.2020.1798035.

Tava Lennon Olsen and Brian Tomlin. Industry 4.0: Opportunities and challenges for operations management. *Manufacturing & Service Operations Management*, 22(1):113–122, 2020. doi: 10.1287/msom.2019.0796.

Manuel Parente, Gonalo Figueira, Pedro Amorim, and Alexandra Marques. Production scheduling in the context of industry 4.0: review and trends. *International Journal of Production Research*, 0(0):1–31, 2020. doi: 10.1080/00207543.2020.1718794.

Yunting Song, Nuo Wang, and Anqi Yu. Temporal and spatial evolution of global iron ore supply-demand and trade structure. *Resources Policy*, 64:101–506, 2019. ISSN 0301-4207. doi: <https://doi.org/10.1016/j.resourpol.2019.101506>.

UNCTAD. Review of maritime transport. Technical report, United Nations Conference on Trade and Development, 2020. URL <https://unctad.org/webflyer/review-maritime-transport-2020>.

Amrisha Varadarajan and Subhash C. Sarin. A survey of dispatching rules for operational control in wafer fabrication. *IFAC Proceedings Volumes*, 39(3):715 – 726, 2006. ISSN

1474-6670. doi: <https://doi.org/10.3182/20060517-3-FR-2903.00363>. 12th IFAC Symposium on Information Control Problems in Manufacturing.

Austin Weber. State of the profession 2018: Smart factories demand smarter engineers. Technical report, Assembly, July 2018. URL <https://www.assemblymag.com/articles/94380-state-of-the-profession-2018-smart-factories-demand-smarter-engineers?>

Literature Review

The relevant literature covers different scheduling problems and methods, and some background is needed to grasp the work that has been done. Therefore, this chapter describes the existing literature on the topics related to the research project. The first section gives a brief overview of the main existing approaches for static and dynamic scheduling. Secondly, previous works on dynamic job shops, parallel and multiprocessor tasks are explored. The last section focuses on applications, namely HRTs and bulk ports.

2.1. Background solution approaches

Scheduling is a decision-making process to manage the execution of tasks to optimise a performance criterion [Drozdowski, 2009]. Tasks may be generally defined as operations that require limited resources to be performed (machines, workers, tools, computer devices, etc.). The relevance of improving scheduling methods is evident, and applications are found in numerous sectors, such as manufacturing, transportation, computer applications, sports events, etc. Scheduling plays a significant role in the industry, in general, and has a crucial impact on optimising the whole production process.

In scheduling problems, the objective function is (directly or indirectly) related to the jobs' processing time. Different criteria are found, and the most common is the makespan (the maximum completion time of jobs), the flowtime and due-date measures as the mean tardiness. Besides, distinct production environments are found. They are classified according to the number of operations of each job, the presence or absence of precedence constraints, the number of eligible machines for each operation, etc. See Pinedo [2008] for a thorough explanation of scheduling settings.

There is no singular solution method for solving all types of scheduling problems. The size and complexity of the environment, constraints and performance criteria influence the best approach. Moreover, depending on the management goals, one may pursue optimality, a good-quality schedule generated in a short time, or a compromise between both. Some standard methods used for static and dynamic scheduling are described next.

2.1.1 Static scheduling

2.1.1.1 Mixed Integer Programming

Mixed Integer Programming (MIP) has been widely applied to scheduling problems. It is generally used to find optimal solutions and as an initial approach to solving a new problem. Advancements in computational resources combined with sophisticated algorithms enabled solving medium-size job shop instances using such models [Ku and Beck, 2016]. MIP formulations for scheduling problems are generally classified into three types: time-indexed [Bowman, 1959], rank-based [Wagner, 1959], and disjunctive formulations [Manne, 1960]. Theoretical and computational comparisons among these formulations suggest that the disjunctive formulation, in general, works better than the others [Demir and Kürşat İşleyen, 2013, Ku and Beck, 2016].

2.1.1.2 Meta-heuristics

Realistic scheduling problems are usually strongly NP-hard, and in practice, solving them to optimality is challenging. In this context, meta-heuristics have been proposed to find near-optimal solutions in a reasonable time. They are high-level heuristics that guide local search methods to escape from local optima exploring a large solution space. In general, their success depends on balancing between diversification (exploration of the search space) and intensification (exploitation of the accumulated search experience).

Tabu search [Shen et al., 2018], GRASP [Bierwirth and Kuhpfahl, 2017] and simulated annealing [van Laarhoven et al., 1992] have been frequently used to solve static deterministic scheduling problems. Additionally, considerable attention has been devoted to population algorithms. They are bio-inspired optimisation methods and operate on a population of solutions (individuals) that evolve at each generation. Crossover and mutation operators are applied to combine solutions and get different individuals. A fitness function is used to evaluate the individuals and select the best ones to proceed for the next generations. Particularly, Genetic Algorithms (GA) have been used in several combinatorial problems [Goldberg, 1989, Herrmann et al., 1995]. A state-of-the-art GA algorithm for the PMSP is found in Edis et al. [2013]. The flexible JSSP has also been successfully solved using a GA algorithm in Pezzella et al. [2008] and Zhang et al. [2011]. For the original JSSP, a randomised-biased variant of GA outperformed existing algorithms for some instances in Gonçalves and Resende [2011].

2.1.1.3 Constraint Programming

Constraint Programming (CP) has been used in the last decade as an efficient method to deal with optimisation problems. It is particularly effective in scheduling problems with a high number of constraints limiting the search space. The recent Parallel Machine literature has promising results of CP in comparison to heuristics and Mixed Integer Programming [Edis and Ozkarahan, 2011, Gökgür et al., 2018]. An overview of Constraint Programming

applied to solve scheduling problems is present in Baptiste et al. [2001].

The purpose of CP algorithms is to solve a Constraint Satisfaction Problem (CSP). The problem is composed of a set of variables, their respective finite set of domain values and a set of constraints over these values [Tsang, 1993]. A solution of a CP model is an assignment of variables to values such that all constraints are satisfied. The algorithms apply constraint propagation techniques for filtering the variables domain (i.e. using inferences obtained from constraints to reduce variables domain) and also explicit search throughout the solution space. For optimisation problems, the CSP has an objective function f , and the algorithm returns the solution S^* that satisfies the CSP and optimises f . If the algorithm completes the search by enumerating all feasible solutions, S^* is the optimal solution of the problem.

2.1.2 Dynamic scheduling

Dynamic scheduling is employed in environments subject to unexpected conditions. In such environments, the sequence of operations should be continuously re-optimised to adapt to the deviations. Several sources of uncertainty are present in production settings. Ouelhadj and Petrovic [2008] classify them into resource-related and job-related. Resource-related uncertainty refers to machines breakdown, unavailability or failures of resources, shortage of materials, etc. Job-related deviations include cancellations, due date changes, early or late arrival, changes in priority, changes in job processing time, and others.

Different strategies may be applied to manage uncertainty in dynamic scheduling. There are proactive approaches that generate a robust schedule based on some prediction of the system behaviour. These methods have gained some research effort, but their performance highly depends on the predictive information obtained. An alternative is performing a hybrid predictive/reactive scheduling, where a baseline of the production system is continuously adapted during the execution of operations. Finally, a completely reactive approach takes no decision in advance. Instead, immediate and local decisions are made in the presence of real-time events. This strategy usually adopts simple and constructive heuristics such as Dispatching Rules (DRs) to prioritise jobs waiting to be processed in a machine's queue [Aytug et al., 2005].

Sabuncuoglu and Bayiz [2000] proposes a classification for dynamic scheduling strategies according to two criteria, *when-to-schedule* and *how-to-schedule*. Three approaches compose the first criteria: *periodic* rescheduling, *event-driven* and *hybrid*. In the *periodic* policy, also known as *rolling time horizon*, a schedule is completely generated at regular intervals, considering all available information. The problem is decomposed into several static problems and usually solved by standard scheduling algorithms. *Event-driven* is the strategy of rescheduling under unexpected events, such as a job arrival or delay. The *hybrid* policy is a composition of the *periodic* and *event-driven* approaches. Regarding *how-to-schedule*, the authors discuss two categories: *full scheduling*, in which all the jobs available are considered; and *partial scheduling*, which includes a percentage of the jobs or a part of

the time horizon. In Sabuncuoglu and Kizilisik [2003], the authors go further and propose a simulation-optimisation algorithm for a dynamic Flexible Manufacturing System. Several strategies are tested considering the *when-to-schedule* and *how-to-schedule* criteria. They conclude that the performance of the systems gets better as the frequency of rescheduling increases. A short response time is also important, and partial scheduling is competitive when there are several machine breakdowns and low flexibility in the system.

In fact, the best way to deal with unexpected interruptions and delays depends on their frequency, nature (whether resource-related or job-related) and environment characteristics. Lawrence and Sewell [1997] compare the performance of simple rules and exact methods in a dynamic scheduling context. They show that the dynamic heuristics outperform optimum schedules even with moderate levels of uncertainty in processing times. For extreme high uncertainty levels, the dispatching rules' performance converges to that of the exact methods. Jain and Foley [2016] explore several strategies, since applying no change in the planned schedule, reschedule using heuristics or simply switch to a dispatching completely reactive mode. The methods are compared using simulation under very distinct scenarios. The authors conclude that the dispatching mode works better than the other strategies at high load levels and longest interruptions. In general, the proposed *Pairwise Exchange Heuristic* outperforms the other approaches at low load level or shortest interruptions.

In real-world applications, the schedule generated in advance usually cannot be accomplished due to unexpected events or deviations on the expected processing times. One may argue that there is no need to generate predictive schedules under those circumstances. Aytug et al. [2005] argue that generating a predictive schedule has some advantages, even under high uncertainty levels. Firstly, these preliminary schedules may be used as capacity-check for tactical-level decisions. They are also useful to guide future actions and smooth the execution of the operations. Finally, if realistic, they set goals that are used in further performance evaluation.

Priority dispatching rules (DRs) are the most common approach for completely reactive scheduling [Aytug et al., 2005]. They are priority functions that assign a score to each job based on the job and system properties. The job with the best score is selected to be executed next. These rules are simple and extremely fast. For this reason, they are extensively adopted in highly dynamic manufacturing systems, where optimal schedules quickly become sub-optimal. Particularly, they have been notably important in the semiconductor industry [Varadarajan and Sarin, 2006]. Besides that, i4.0 processes impose an increased requirement for flexible and decentralised applications [Olsen and Tomlin, 2020, Parente et al., 2020]. For instance, systems with coordinated autonomous agents require local real-time decisions [Rossit et al., 2019]. The use of DRs is a suitable approach for such environments.

Effective DRs have been developed for several scheduling problems and performance criteria. Chen and Matis [2013] design an effective rule for minimising weighted tardiness for job shops. Rules are also derived in Xiong et al. [2017] for a variant of the dynamic JSSP in which jobs arrive in batch. Several due-date-related measures are considered. The DPMS is addressed by Su et al. [2017] using a new composite rule. Priority rules are also pro-

posed by [Chen et al. \[2018\]](#), in the context of the Resource-Constrained Project Scheduling Problem. Differently from the traditional approaches, [Wu et al. \[2020\]](#) integrate dispatching rules with preventive maintenance to prevent machine health deterioration. Finally, [Sweeney et al. \[2019\]](#) investigate the development of rules in a real case study, sequencing vessels in the Upper Mississippi River.

2.1.2.1 Machine learning

In scheduling, the general objective of machine learning methods is to provide the adaption of schedules in real-time. [Pinedo \[2008\]](#) classifies the methods into four categories.

- Rote learning

This category comprises a straightforward mechanism in which the system associates solutions with instances characteristics. However, this procedure is not effective for complex environments, as there is no reasoning on the good performance of those solutions. If the number of jobs and machines is small, and the problem has no much constraints, previous solutions can be directly applied. Otherwise, more sophisticated techniques must be employed.

- Case-based reasoning

This category refers to online mechanisms that operate without supervision. Case-based learning explores the experience from the past to improve solutions under new circumstances. [Huyet and * \[2004\]](#) use simulation to acquire knowledge on system behaviour and propose new scheduling solutions based on the learnt premisses. [Shahzad and Mebarki \[2012\]](#) use data-mining to analyse the behaviour of DRs in instances with distinct features. Based on the analysis, decision trees are generated for DRs selection. Recently, Reinforcement Learning has been extensively used to adapt sequencing policies in real-time [[Stricker et al., 2018](#), [Shi et al., 2020](#)].

- Induction methods

Neural networks and deep learning are the most common induction mechanisms adopted for manufacturing challenges, in general. Although they may be used for schedule generation, in practice, most works use these methods for predictive purposes (maintenance and production quality). [Cadavid et al. \[2020\]](#) provides an overview of recent advances in such methods.

- Classifier systems

This category comprises hyper-heuristics that are used for the automated generation of DRs. The prominent examples are found in the use of evolutionary algorithms, such as in [Ho et al. \[2007\]](#) and [Olafsson and Li \[2010\]](#). With that purpose, there has been increasing attention on Genetic Programming (GP) in the last decade, particularly for production scheduling [[Nguyen et al., 2017a](#)].

This work follows a learning-based approach with GP, which is an evolutionary hyper-heuristic [Burke et al., 2009]. This mechanism evolves computer programs or mathematical expressions that are used to find solutions for a problem. Many works have shown that GP is able to find effective DRs, although the vast majority of them are too large and meaningless. Among the several advantages, GP may produce expressions that are interpretable, in contrast to the black-box models from the other learning approaches. Thus, humans can easily interact with the algorithm, and input domain knowledge in the search process, as human-designed heuristics or good features.

2.1.2.2 Genetic Programming

GP evolves a population of computer programs that are interpreted or executed to solve a problem [Koza, 1994]. The aim is to find the highest fit individual in a search space of computer programs, where each individual's fitness corresponds to how well the respective program solves the problem. Symbolic expression trees are the most common representation of individuals in GP. They are composed of *terminals* and *functions* appropriate to the specific problem. The former are the leaf nodes and include parameters of the problem, operated by the latter. Figure 2.1 illustrates an individual and the corresponding dispatching rule. In this example, the terminals are 2, *PT* (processing time of current operation), *W* (work in the queue) and *NPT* (processing time of the next operation). The mathematical operators + and * compose the function set.

Symbolic expression trees are the most common representation of individuals in GP. They are composed of *terminals* and *functions* appropriate to the specific problem. The former are the leaf nodes and include parameters of the problem, operated by the latter. Figure 2.1 illustrates an individual and the corresponding dispatching rule. In this example, the terminals are 2, *PT* (processing time of current operation), *W* (work in the queue) and *NPT* (processing time of the next operation). The mathematical operators + and * compose the function set. Details on the specific implementation of GP in this work will be given next.

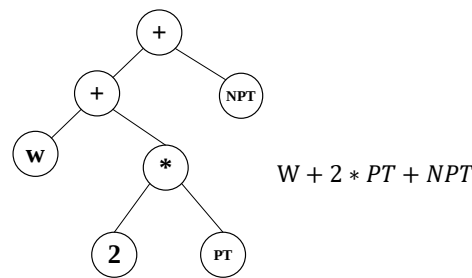


Figure 2.1 – A GP individual tree example and its corresponding DR

Genetic programming has been applied in various studies for DRs in production scheduling, evolving rules that outperform the existing ones in their respective problems. Some examples are briefly described following, and comprehensive surveys on the use of GP for production scheduling is found in Branke et al. [2016], and Nguyen et al. [2017a].

2.2. State of the art

2.2.1 Dynamic job shops

The simulation research on DRs for dynamic job shops has a long tradition. Haupt [1989] and Ramasesh [1990] are relevant works on the topic. They provide thorough explanations of the behaviour and classifications of DRs. Next, we detail the main findings on the performance of the rules for minimising the flowtime and tardiness of the jobs.

In general, the SPT rule, which selects the job with the shortest operation processing time is effective for optimising the flow of jobs. Although, as it is a very myopic strategy, it may prioritise jobs that could find long queues when visiting the next machines. For this reason, in addition to the current operation processing time, most DRs also include some “look-ahead” information. Holthaus and Rajendran [1997] were the first to add the workload in the next machine queue (WINQ) to the SPT, devising the rule PT+WINQ. Later on, Rajendran and Holthaus [1999] included the processing time of the job’s subsequent operation (NPT) and proposed the rule 2PT+WINQ+NPT, which remains the best for minimising the mean flowtime of jobs.

Tardiness and percentage of tardy jobs are the most common due date related measures. The literature has shown that if due dates are loose or the shop utilisation is low, considering the slack of jobs is very important. Otherwise, in highly loaded shops, including this parameter bias and depreciate the behaviour of rules [Haupt, 1989]. In general, processing-time based rules present good behaviour in tight load conditions, and due date based rules are better if shopfloor is overloaded [Ramasesh, 1990]. For this reason, the study of DRs for optimising due date measures is much more complicated.

There is no single rule for all load conditions, although much work has been developed in that direction. Some authors argue that prioritise operations due date is instead of jobs due date results in better sequences. Operations due date usually are calculated by distributing the allowance (remaining time until due date) of jobs per remaining work (the sum of all subsequent operations processing time). Anderson and Nyirenda [1990] proposed the rules CR+SPT and S/RPT+SPT, that prioritise operations according to their due dates. Both rules re-calculate operation due dates at dispatching time. But the former considers the Critical Ratio (CR) and the latter uses the slack divided by the remaining work. The CR of a job is the ratio between its allowance time and its remaining work, and the slack is calculated as job allowance minus its remaining work. The idea behind those rules is that the operation due date should be as short as the portion of job allowance time that remains for that operation. In practice, they apply the SPT rule if the job is delayed and CR or S/RPT in the opposite case.

In the RR rule from Raghu and Rajendran [1993], the slack parameter’s weight is a function of the shop utilisation. The authors use the shop utilisation to balance the current processing time (PT) weight and the operation due date (PT·S/RPT). Moreover, the authors also improve the calculation of the work in the next queue (WINQ), which is usually the sum of the processing time of all operations present in the next machines’ queue. Differently,

RR considers only the operations that would be assigned to a better score than the current operation. The resulting expression is $PT \cdot \exp(u) + PT \cdot S/RPT \cdot \exp(-u) + WINQ$, where u is a parameter that estimates the mean machines' utilisation. Although this DR presents good results in general, under very tight load conditions, the performance of 2PT+WINQ+NPT is the best [Holthaus and Rajendran, 2000].

Instead of pursuing a unique DR, the recent developments focus on selecting rules according to the conditions of the production system. Very different approaches have been proposed in that direction. Romero-Silva et al. [2018] study a flowshop with multi-class jobs. The objective is to analyse if using different dispatching rules in each queue leads to improved mean flowtime and flowtime variance. They concluded that a better performance is obtained when applying a pair of DRs, as the best DRs for the first machine is different from the best DRs for the remaining ones. In Amin and El-Bouri [2018], a preference voting model is employed to select rules based on multiple criteria. The authors designed a Linear Programming Model for classifying rules according to their relative performance in each criterion. Pergher et al. [2020] develop a framework for due date assignment, order release and dispatching rules according to the decision-maker preferences and multiple performance criteria. A decision support system performs an interactive process where the user inputs trade-offs judgements.

A common strategy to achieve better results is to derive more efficient DRs by combining simple rules or learn from their results. Therefore, the research in dynamic scheduling has devoted considerable attention to the development of methods for the automated combination of DRs via machine learning. For example, in Jun et al. [2019] rules for the Flexible JSSP are learnt from solutions obtained with standard optimisation methods. Dispatching decision trees are derived and improved via Random Forests.

Reinforcement Learning (RL) approaches are also found, such as Waschneck et al. [2018], which develop Deep Learning Agents trained in a flexible production environment. The agents optimise DRs for a complex job shop system. The authors list some advantages of the approach, such as flexibility and transparency (by using DRs). Shi et al. [2020] use RL to optimise scheduling policies under uncertain processing times and four different production settings. In both cases, the outcomes are benchmarked with very simple existing rules - Shortest Processing Time (SPT), Earliest Operations Due Date (ODD), First-in-First-Out (FIFO). Furthermore, none of these approaches presents or discuss the generated rules.

The automated design of rules via Genetic Programming has also received a lot of attention in recent years. In the following, we describe the main achievements obtained with that strategy.

2.2.1.1 GP for dynamic job shops

Hildebrandt et al. [2010] and Branke et al. [2015] developed GP algorithms to minimise mean flowtime in job shops. Both compared the resulting rules with the best existing ones and were successful in results obtained. In Nguyen et al. [2015] and Nguyen et al. [2019a] the authors develop DRs that outperform state-of-the-art. They found very efficient rules

for several performance metrics, but they were trained for specific load condition settings. Job shops under stochastic processing times were the focus of [Karunakaran et al. \[2017\]](#) that compared different training approaches to deal with uncertainty in the execution of operations.

Heuristics composed of ensembles of DRs are also a promising alternative. The algorithm from [Hart and Sim \[2016\]](#) applies each rule in turn while [Park et al. \[2018\]](#) compared different combination schemes, like majority voting and a linear combination of the rules. In both works, the resulting ensembles present better performance than single evolved rules. GP may also be combined with other heuristic approaches such as Iterated Local Search [[Nguyen et al., 2015](#)], or more standard evolutionary algorithms [[Pickardt et al., 2012](#)].

Different authors studied surrogate models for improving the algorithm efficiency [[Nguyen et al., 2014](#), [Hildebrandt and Branke, 2015](#), [Nguyen et al., 2017b](#)]. They are used to avoid employing high computational effort in the evaluation of non-promising rules. Running these models is usually much faster than performing a complete simulation for accessing the fitness of individuals.

Previous studies have shown that the tree-based representation is the best approach to evolve powerful scheduling heuristics [[Nguyen et al., 2013](#)]. However, as tree-based evolved rules are suitable to get too large, different representation structures have also been tried [[Nguyen et al., 2013](#), [Branke et al., 2015](#), [Nguyen et al., 2019a](#)]. Besides a good representation, an important decision consists of the terminal set, i.e., the set of the jobs and system attributes that will compose the DRs. [Branke et al. \[2016\]](#) presents a detailed survey on that, and [Mei et al. \[2016\]](#) proposes a mechanism for selecting relevant terminals.

[Nguyen et al. \[2013\]](#) and [Nguyen et al. \[2014\]](#) present multi-objective GP algorithms. The evolved DRs are non-dominated for optimising the percentage of tardy jobs, mean and maximum tardiness, mean and maximum flowtime. Besides that, the resulting rules also present a good performance for each single objective.

[Tay and Ho \[2008\]](#) is the earliest study on the application of GP for the dynamic version of the Flexible Job Shop Scheduling Problem. Their work was followed by [Nie et al. \[2013\]](#) with a Gene Expression Programming (GEP) approach. Differently from GP, GEP-based algorithms make use of a fixed-length representation structure, which resulted in more efficient rules in comparison to the work from [Tay and Ho \[2008\]](#). Lately, [Zhou et al. \[2019\]](#) developed a co-evolutionary GP algorithm to evolve, simultaneously due date assignment and dispatching rules for the flexible JSSP.

Despite the extensive literature on the automated development of rules for job shops, most works do not compare the tardiness from the evolved DRs and state-of-the-art. Others derive different rules for each shop configuration [[Nguyen et al., 2013](#), [Nguyen et al., 2015](#), [Nguyen et al., 2019a](#)] for what there is no single rule for minimising the mean tardiness in dynamic job shops. Moreover, there is a quite limited discussion on the size of the rules. A simplification technique is proposed in [Nguyen et al. \[2017b\]](#), but the resulting DRs are still too complicated and difficult to interpret.

This work has the objective to overcome these gaps and find compact rules that outperform

the existing ones in a wide range of job shops. We aim at involving human intelligence in the process. Some initiatives with a similar purpose are found in previous works, particularly in the study of Hyper-heuristics. In [Burke et al. \[2009\]](#), the authors describe a methodology for the use of GP with the contribution of domain experts. But human participation is limited to the initial implementation decisions. A more iterative process is developed in [Hein et al. \[2018\]](#), where the authors pursue new heuristics for lot sizing. In the study, the outputs from preliminary GP runs are used to understand the search process and improve it. [Nguyen et al. \[2019b\]](#) propose an approach that includes human preferences on the search mechanism, but no input from domain knowledge is incorporated in the search process.

2.2.2 Parallel and multiprocessor tasks

The PMSP literature classifies machine environments into three main classes [[Pinedo, 2008](#)]. (1) *Identical* machines have the same processing time for each job; (2) *Uniform* machines may have different speeds, but the processing time of the jobs is proportional to the speed of each machine; (3) In *Unrelated* machines environments, no particular relationship between the processing time of the jobs exists. HRTs working cells are a suitable application of unrelated machines, as humans and robots usually perform better in distinct types of tasks. Machine eligibility constraints may be viewed as a special case of the unrelated parallel machine environment. Such constraints impose that a given job is allowed to be processed only on a subset of the machines.

Multiprocessor Tasks Scheduling (MTS) is a generalisation of the PMSP where a single task may be executed simultaneously by a set of machines in parallel [[Drozdowski, 2009](#)]. Different variants of MTS problems are described in the literature. They are shown in [Figure 2.2](#). If machines are identical, tasks are classified as *moldable*, *malleable* or *rigid*. In the case of moldable or malleable tasks, the scheduler may choose the number of machines. The processing time of these tasks vary, linearly or not, according to the number of machines given. The distinction between moldable and malleable is that the machines assigned to malleable tasks may change during the execution. Rigid tasks are a particular case of moldable tasks, where the number of machines is fixed. In environments with non-identical machines, we have *multiprocessor* tasks, that may require a specific set of resources (*single-mode*) or accept different combinations of resources (*multiple modes*, as found in the MMTSP). Finally, *uniprocessor* tasks, found in the PMSP, must be assigned for only one machine. They are a specific case of all the other types of tasks.

Several multiprocessor applications are found in parallel computing systems, where tasks may be distributed among several processors [[Błażewicz et al., 2003](#), [Switalski and Sere-dynski, 2015](#), [Drozdowski, 2009](#)]. Multiprocessor single-mode tasks are present in some variants of the Berth Allocation Problem if a ship requires more than one quay crane [[Guan et al., 2002](#), [Türkoğullar et al., 2014](#)]. In workforce allocation, if the tasks' processing time depends on the number of resources assigned, as in [Pappis and Rachaniotis \[2010\]](#) and [Battaia et al. \[2015\]](#), tasks are moldable. In [Artigues et al. \[2013\]](#), the authors describe a case study where malleable jobs have a variable duration, depending on the power given to

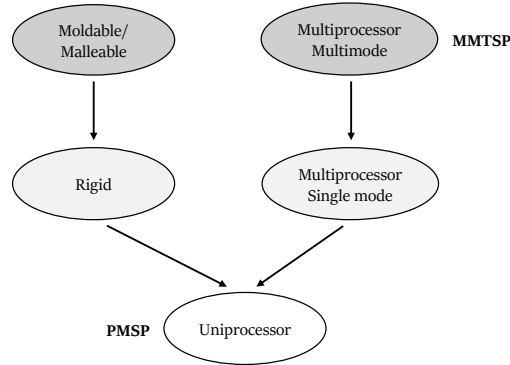


Figure 2.2 – A classification for parallel tasks [Drozdowski, 1996]

the furnace.

The first work to deal with Multiprocessor Tasks Scheduling is Weglarz et al. [1986]. The authors studied the computational complexity of scheduling rigid tasks in the special case of unit processing times. They presented two polynomial-time algorithms with complexity $O(n)$ for non-preemptive scheduling of tasks that require a fixed number of k processors. A comprehensive survey on Multiprocessor Tasks Scheduling can be found in Drozdowski [2009].

In the Multimode (or General) Multiprocessor Tasks Scheduling Problem, each job has a set of possible modes. Each mode has a specific processing time and a subset of machines [Chen and Lee, 1999]. The problem is shown to be NP-hard by demonstrating that the particular case with three machines is strongly NP-hard [Błażewicz et al., 1992]. Bianco et al. [1995] provided a dynamic programming algorithm to solve specific cases of the problem (with $|R| = 2$ and $|R| = 3$). They also proposed a heuristic for the general problem with worst-case ratio of $|R|$. Chen and Lee [1999] improved their work with an approximation ratio of $|R|/2 + \varepsilon$ by using dynamic programming. Later both Chen and Miranda [2001] and Jansen and Porkolab [2005] developed polynomial approximation schemes for the problem. Meta-heuristics based on the concepts of intensification and diversification [Glover, 2000] were developed in the work of Caramia and Giordani [2009] and Caramia and Giordani [2010].

Regarding the variation with precedence constraints, Bianco et al. [1998] are the first authors to tackle this problem. They proposed specific heuristics for the cases with and without budget constraints. In Bianco et al. [1999] three lower bounds were proposed. The best one (LB2) is based on the concept of *constraint graph*, where edges represent jobs that cannot be simultaneously executed. This is the only work to propose lower bounds to this problem. The authors also proposed two constructive heuristics for the problem, namely UB1 and UB2. They showed that these algorithms produce better results in comparison to the former work. To the best of our knowledge, these are the only algorithms developed for

the MMTSP with precedence constraints. Therefore this work is considered for benchmark comparison with the methods here proposed.

The scheduling literature presents several well-studied problems with similar characteristics of the MMTSP. The PMSP with eligibility constraints is also referred as Multi-purpose Machine Scheduling. In Brucker et al. [1997], the authors discuss the complexity of Parallel Machine and shop scheduling problems with such characteristic, and an extensive survey on the topic is found in Leung and Li [2008]. In Brucker and Neyer [1998], the authors deal with a variant of the JSSP where jobs are multimode and multiprocessor. A Tabu Search algorithm with two sophisticated neighbourhoods is proposed and compared with another Tabu Search algorithm, designed for the JSSP with multi-purpose machines.

The Multimode Resource-Constrained Project Scheduling Problem deals with different types of resources. Jobs are associated with a set of modes, each of them with specific processing times and requiring a different amount of resources. The MMTSP is a specific case of that problem where there is exactly one type of each resource. One can find food surveys on this problem and its variations in Brucker et al. [1999] and Hartmann and Briskorn [2010].

Another generalisation of the MMTSP is the Resource Sharing and Scheduling Problem, that also handles a set of multimode jobs. Differently from the MMTSP, during the execution of job j by a set of resources R , the resources in R may be dedicated to j in different periods of time. In Pinto et al. [2009], the authors propose two Branch-and-bound and a Genetic Algorithm for the problem. Although the methods present a promising performance, the proposed instances are quite small, with up to 24 jobs.

The next section summarises the distinct study topics on human-robot collaboration and the research work on HRTs scheduling.

2.3. Applications

2.3.1 Human-robot collaboration

Collaborative environments have received increasing attention during the last two decades. These applications involve very different areas. In robotics, works improve collision detection [Lee et al., 2015] and the optimisation of robots' motion planning [Pellegrinelli et al., 2017]. Machine learning algorithms are being developed to enhance gesture recognition [Liu and Wang, 2018], teaching models for the robot to learn from human demonstrations [Wang et al., 2019] and the prediction of human activities [Zanchettin et al., 2019]. In some contexts, an empirical examination of human-factors is critical, as trust [Hancock et al., 2011], situational awareness and human preferences [Gombolay et al., 2015, 2017]. There are also concerns about the ideal design of tasks in a collaborative assembly cell, given that humans and robots have complementary capabilities [Mateus et al., 2019]. Extensive reviews on the technological advances and issues related to HRTs are found in Krüger et al. [2009] and in Ajoudani et al. [2018]. A recent survey is also found in Math-

eson et al. [2019], where the authors discuss the rapid increase in the number of studies. They show that productivity and safety have always been the main focuses of these works, although human-robot interaction is receiving an increasing number of studies.

The term “Human-Robot Collaboration” is generally used in workplaces where humans and robots are present, without fences. But several interaction levels of human-robot collaboration exist. Some of these settings establish speed and acceleration limits for robots, depending on the proximity to workers. Among all robots developed in this new collaborative perspective, special attention has been devoted to the so-called cobots. They are capable of working at the same station as humans and avoid impact by stopping motion or moving in the opposite direction.

The International Federation of Robotics provides a classification on the different types of HRTs interaction [IFR, 2018]. They describe five scenarios where humans and robots operate in the same working cell, shown in Figure 2.3. The first one, *Cell*, is actually not collaborative, as it refers to the conventional fenced settings. Traditional robots work in protective cages that ensure physical separation from workers under normal operation. In the *Coexistence* scenario, there is no fence, but humans and robots operate in their own workspace. There is no shared area in the workstation, as in Ding et al. [2014]. *Synchronised Collaboration* refers to cells where the operators share the workspace, but they always work in different components, performing sequential and synchronised movements. Examples are found in Chen et al. [2014] and Bänziger et al. [2018].

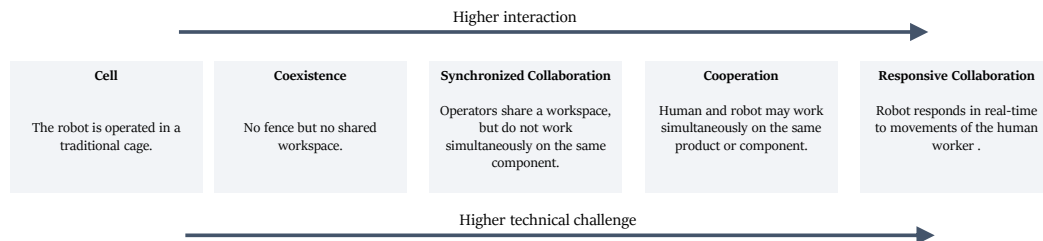


Figure 2.3 – Classification of human-robot collaboration provided by IFR [2018]

A closer interaction occurs in *Cooperation*, where both operators – humans and robots – may work at the same time on the same components, performing collaborative tasks. This is the scope of the HRTs considered in this work, and also considered in Nikolakis et al. [2018], Bogner et al. [2018] and Xu et al. [2020]. Finally, *Responsive Collaboration* is the highest level of interaction, where humans and robots not only share workspace and components, but the robot is able to react in real-time to the movements of a human worker. This is the ultimate goal of collaborative environments and the most difficult to achieve. It involves dynamic optimisation of tasks and gesture recognition and the prediction of human behaviour [Casalino et al., 2019].

Most of the current applications of HRTs are classified as Coexistence and Synchronised Collaboration. This is due to the extensive technological challenges present in the other two scenarios [IFR, 2018]. Among these challenges, the transition to Cooperation involves

some Operations Research (OR) questions related to production planning and scheduling. Although the concept of HRTs date from the '90s, almost no studies are focusing on these questions. [Ivanov et al. \[2020\]](#) and [Olsen and Tomlin \[2020\]](#) point out the relevance of identifying the novel operations questions arising with these technologies.

Table 2.1 presents five groups of relevant problems related to human-robot collaboration that OR may address. The table also presents the number of papers found in each group. Some works have been proposed for scheduling human-robot tasks both in static and dynamic contexts, but some other questions arise. The improvement of working conditions and ergonomic aspects are motivations for the deployment of HRTs. Thus, some authors argue that the assignment of tasks between humans and robots should consider the task complexity, payload, repeatability, and the suitability of each element of the team [[Dianatfar et al., 2019](#)]. Considerable attention has been devoted to the trade-off between safety and efficiency in the workplace design [[Tsarouchi et al., 2017](#)]. Some issues related to the facility layout have also been raised. There should be a physical separation between completely automated and collaborative zones because robots must slow down in humans' presence. This premise raises a new optimisation problem related to facility layout. However, all works related to workplace design and layout only introduce the problem or propose simple procedures [[Michalos et al., 2015](#)]. In total, 36 papers discuss the relevance or propose methods for those problems. Beyond the main issues refereed there, the works also present specific questions and constraints. Ergonomics aspects are discussed in seven works, while nine works also deal with safety issues.

Table 2.1 – Decision-making on collaborative assembly lines: main research topics on operations research and number of papers found

Problem	Main issue	Number of papers	
		Discuss problem	Propose method
Static scheduling	Decide the best assignment and execution sequence of tasks.	1	8
Dynamic scheduling	Dynamically assign, sequence and dispatch tasks, according to environment changes.	2	4
Assignment	Assign tasks as a multi-criteria decision, considering efficiency, safety and work balance.	5	3
Workplace	Safe and efficient design of the workplace and workflow of pieces.	10	3
Facility layout	Define the plant layout. Define which are the collaborative working spaces and which tasks should be executed there.	4	1

Considering the challenges for decision-making, the question of how to effectively schedule HRTs is the most studied topic in these new environments. In the following, the works found on this topic are described.

Scheduling HRTs

Some studies report dynamic scheduling problems, such as [Johannsmeier and Haddadin \[2017\]](#) and [Darvish et al. \[2018\]](#). In both works, the authors consider that the assembly paths may be decided in real-time, and the different assembly routes are described in a AND/OR Graph. In the former, the assignment of tasks is multimode, given that the algorithm may also determine which tasks should be executed by both human and robot. Dynamic scheduling is also present in [Ding et al. \[2014\]](#) and [Nikolakis et al. \[2018\]](#). The former proposes a heuristic for the assignment and sequencing of a PLC I/O Module operations to avoid safety hazards. The latter design a tool that evaluates the assignment alternatives for each operation according to a utility function composed of many criteria. Recently, [Casalino et al. \[2019\]](#) developed a Petri Net model and a scheduler algorithm to simulate and optimise an HRT working cell, given the human uncertainty. The scheduler performs an exhaustive search in the reachability graph to prioritise the next task for the robot. They use a prediction algorithm to identify patterns in human activities and adapt the robot schedule accordingly. Recently, the approach has been improved with the real-time optimisation of sequencing policies under uncertainty in execution times [[Zanchettin, 2021](#)].

All these works focus on providing sufficient empirical evidence by validating the approach in a real verification scenario, where humans and robots simulate a small assembly line. Although, with the exception of [Zanchettin \[2021\]](#), the optimisation algorithms are extremely simple. In many cases, the authors adapt greedy heuristics or simply enumerate of all possible alternatives.

On a deterministic setting, [Gombolay et al. \[2018\]](#) propose an algorithm to multi-robot scheduling, which incorporates the preferences of a human supervisor. The existence of a human worker imposes some spatial constraints to support safety distance regulations. The objective is to minimise the difference to a previous schedule and spatial interfaces between the robots. But the algorithm does not schedule human tasks.

Cyclic scheduling with HRTs is also the focus of some works. [Bänziger et al. \[2018\]](#) describe a genetic algorithm to address the assignment and sequencing of tasks in an HRTs workplace. Solutions obtained are evaluated by a simulation tool designed for that purpose and validated in a real manufacturing environment. The objective is the minimisation of completion time, walking distances and waiting time. In [Bogner et al. \[2018\]](#), a use case of printed circuit boards assembly is presented. The environment consists of workstations with a single robot in collaboration with one or more human operators. There are precedence constraints and minimum safety distances in both problems. Recently, [Xu et al. \[2020\]](#) develop a Bee Algorithm to disassembly tasks in a working cell composed of two operators, a human and a robot. Tasks are classified according to the execution time and difficulty of each operator. The algorithm is multi-objective, considering to minimise the completion time, costs, and difficulty in executing tasks.

[Chen et al. \[2014\]](#) describe a power box module assembly performed by HRTs. The objective consists of both the minimisation of makespan and production costs. Their work

is based on a collaborative manufacturing environment where a human has as co-workers one or more robots to jointly assemble a power module box in a layout configuration similar to that shown by Figure 4.1. The setting is justified because robotic manipulators are nowadays widely used in electronics assembly, and the introduction of humans improve efficiency in the execution of specific manual tasks. The authors propose a Genetic Algorithm that optimises each HRT separately, and a constructive heuristic merges the independent solutions generated.

All these works on static scheduling seek to maximise the productivity of HRTs in manufacturing cells, which are characterised by cyclic production schedules. Even if minor deviations are necessary, operating a cyclic production process is easier and avoids reprogramming the robots every time. In these situations, it is necessary to minimise the cycle's production time, i.e., its makespan. [Chen et al. \[2014\]](#) studies this problem in collaborative working cells but disregard collaborative tasks. [Bogner et al. \[2018\]](#) consider the latter but in a single-mode environment. Therefore, despite optimising makespan in HRTs cells, neither can evaluate the aforementioned managerial questions.

2.3.2 Dry bulk ports

Considerable amount of the global maritime trade operates dry bulk cargo like iron ore, coal and grains such as wheat and soybean and others [[UNCTAD, 2020](#)]. However, bulk ports received little attention from Operations Research literature compared to container ports. A few studies on the optimisation of their operations are found and are briefly described in this section.

Dry bulk ports usually serve as export or import terminals exclusively. They have three subsystems: the product reception, the stockyard and the terminal gate (where the product is loaded to the final destination). In high volume export ports (e.g. dedicated to iron ore or coal), the product arrives by train and must be transported via a conveyor chain to the storage area. Temporally stockpiles are used to store the material; then, the material is reclaimed via conveyor belts to load the shipments. Depending on the cargo type, the stockpiles are reserved for specific shipments or not. In some iron ports, the stockyard also serves to blend the product. Finally, the berths receive the vessels, which are loaded via ship loaders. Several resources are involved in the operation of such ports, requiring efficient plans and management. Figure 2.4 summarises the main optimisation problems found in each of these subsystems.

The problem of assigning vessels to berths is known as the Berth Allocation Problem (BAP), well-studied in several variants with distinct objectives and constraints. Some works on the specific case of bulk cargo are found. [Barros et al. \[2011\]](#) consider the challenging tidal constraints from a real Vale's port *. They also deal with stock level issues. [Umang et al. \[2013\]](#) introduce cargo type consideration in the vessels allocation. This is a relevant feature because distinct resources are necessary for each product type. In the tactical level, [Bilgen and Ozkarahan \[2007\]](#) combine the BAP with a transport problem.

*In Ponta da Madeira, a port in the Northeast of Brazil

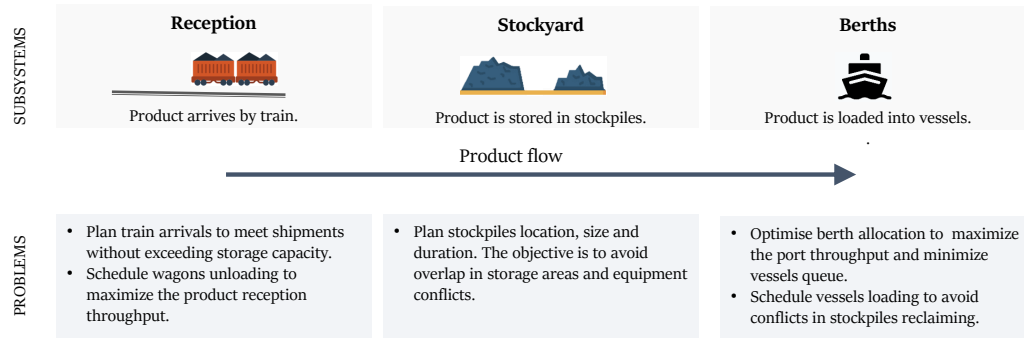


Figure 2.4 – Bulk ports are composed of three integrated subsystems with specific optimisation problems.

They consider the whole process of blending the products, vessels loading and maritime transport to customers. Several constraints are considered, as stock levels and stockyard capacities. The problem is formulated as a MIP model, and the objective is to minimise the total planning cost (i.e., blending, loading, transportation and inventory costs).

Optimising the stockyard allocation is also relevant. The stockpiles location and the equipment used highly influence the throughput of the product reception and the berths. Thus, the income and outcome operations in the storage area must be carefully scheduled to avoid conflicts. In [Sun and Tang \[2013\]](#), the authors deal with an import terminal that operates iron ore to feed a steel plant. A Benders decomposition algorithm is developed to schedule the stockyard operations, but the problem is simplified and does not consider the conveyor belts. Some works integrate the BAP with the scheduling of the stockyard operations. [Robenek et al. \[2014\]](#) developed a branch-and-price algorithm. The approach's effectiveness is shown in instances taken from the SAQR port, the largest bulk port in the middle east, that operates both dry and liquid cargo types. [Unsal and Oguz \[2019\]](#) propose a Benders decomposition approach with the same purpose and solved realistic size instances in a relatively short time (up to one hour).

Some works also focus on the product reception but integrating all port operations. In [Menezes et al. \[2017\]](#), the authors propose a branch-and-price algorithm for planning the stockyard allocation and, at the same time, schedule all port operations. The work also incorporates blending decisions and focuses on a real problem found in a Vale's port. [Rocha de Paula et al. \[2019\]](#) developed exact and heuristic methods to maximise the throughput of a coal port, considering a real use case in Australia. The problem includes several features: the railroad operations and constraints on the port's entrance (the channel). But the authors do not address the scheduling of the wagons unloading, only daily capacities model the product reception. [Burdett et al. \[2020\]](#) advanced on the topic by proposing an effective heuristic for scheduling all operations from the product reception to the vessels loading. But the conveyor belts present in the stockyard are not considered in the optimisation. The problem is formulated as an extension of the Flexible Job Shop Scheduling. The

effectiveness of the approach is demonstrated in real instances from Australia.

The tactical optimisation of import terminals is solved using MIP models in Kim et al. [2009] and Ago et al. [2007]. Both works involve the whole process: the reception (berths), the stockyard allocation and the transport to steel facilities. The later also consider the case of dynamic transport requests.

Table 2.2 classifies the works described above according to the subsystems considered and their goals (Mid-term/Tactical planning or Short-term/Operational). For each combination of subsystems and goal, particular problems are addressed. The congestion of trains in the port arrival is reported as a challenge to be addressed [Menezes et al., 2017, Burdett et al., 2020], but notice there no study on the development of specific methods for scheduling the product reception. In those works, the operations in product reception are modelled in hourly time-buckets or the selection of routes is not included in the problem.

Table 2.2 – Research work on optimisation problems in bulk ports

Problem	Mid-term	Short-term
Berths		Barros et al. [2011] Umang et al. [2013]
Stockyard		Sun and Tang [2013]
Stockyard and berths	Bilgen and Ozkarahan [2007]	Robenek et al. [2014] Unsal and Oguz [2019]
All subsystems in the port	Menezes et al. [2017] Rocha de Paula et al. [2019] Kim et al. [2009] Ago et al. [2007]	Menezes et al. [2017] Burdett et al. [2020]

Bibliography

- Ago, M., Nishi, T., Konoshi, M., 2007. Simultaneous optimization of storage allocation and routing problems for belt-conveyor transportation. *Journal of Advanced Mechanical Design, Systems, and Manufacturing* 1, 250–261. doi:[10.1299/jamdsm.1.250](https://doi.org/10.1299/jamdsm.1.250).
- Ajoudani, A., Zanchettin, A.M., Ivaldi, S., Albu-Schäffer, A., Kosuge, K., Khatib, O., 2018. Progress and prospects of the human–robot collaboration. *Autonomous Robots* 42, 957–975. URL: <https://doi.org/10.1007/s10514-017-9677-2>, doi:[10.1007/s10514-017-9677-2](https://doi.org/10.1007/s10514-017-9677-2).
- Amin, G.R., El-Bouri, A., 2018. A minimax linear programming model for dispatching rule selection. *Computers & Industrial Engineering* 121, 27 – 35. doi:<https://doi.org/10.1016/j.cie.2018.05.021>.
- Anderson, E.J., Nyirenda, J.C., 1990. Two new rules to minimize tardiness in a job shop. *International Journal of Production Research* 28, 2277–2292. doi:[10.1080/00207549008942866](https://doi.org/10.1080/00207549008942866).

- Artigues, C., Lopez, P., Haït, A., 2013. The energy scheduling problem: Industrial case-study and constraint propagation techniques. *International Journal of Production Economics* 143, 13 – 23. doi:[10.1016/j.ijpe.2010.09.030](https://doi.org/10.1016/j.ijpe.2010.09.030).
- Aytug, H., Lawley, M.A., McKay, K., Mohan, S., Uzsoy, R., 2005. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research* 161, 86 – 110. doi:<https://doi.org/10.1016/j.ejor.2003.08.027>. iEPM: Focus on Scheduling.
- Bänziger, T., Kunz, A., Wegener, K., 2018. Optimizing human–robot task allocation using a simulation tool based on standardized work descriptions. *Journal of Intelligent Manufacturing* doi:[10.1007/s10845-018-1411-1](https://doi.org/10.1007/s10845-018-1411-1).
- Baptiste, P., Le Pape, C., Nuijten, W., 2001. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. volume 39. Springer US. doi:[10.1007/978-1-4615-1479-4](https://doi.org/10.1007/978-1-4615-1479-4).
- Barros, V.H., Costa, T.S., Oliveira, A.C., Lorena, L.A., 2011. Model and heuristic for berth allocation in tidal bulk ports with stock level constraints. *Computers & Industrial Engineering* 60, 606–613. doi:<https://doi.org/10.1016/j.cie.2010.12.018>.
- Battaïa, O., Delorme, X., Dolgui, A., Hagemann, J., Horlemann, A., Kovalev, S., Malyutin, S., 2015. Workforce minimization for a mixed-model assembly line in the automotive industry. *International Journal of Production Economics* 170, 489 – 500. doi:[10.1016/j.ijpe.2015.05.038](https://doi.org/10.1016/j.ijpe.2015.05.038). current Research Issues in Production Economics.
- Bianco, L., Błażewicz, J., Dell’Olmo, P., Drozdowski, M., 1995. Scheduling multiprocessor tasks on a dynamic configuration of dedicated processors. *Annals of Operations Research* 58, 493–517. doi:[10.1007/BF02057160](https://doi.org/10.1007/BF02057160).
- Bianco, L., Dell’Olmo, P., Giordani, S., Speranza, M.G., 1999. Minimizing makespan in a multimode multiprocessor shop scheduling problem. *Naval Research Logistics* 46, 893 – 911. doi:[10.1002/\(SICI\)1520-6750\(199912\)46:8<893::AID-NAV2>3.0.CO;2-7](https://doi.org/10.1002/(SICI)1520-6750(199912)46:8<893::AID-NAV2>3.0.CO;2-7).
- Bianco, L., Dell’Olmo, P., Speranza, M.G., 1998. Heuristics for multimode scheduling problems with dedicated resources. *European Journal of Operational Research* 107, 260 – 271. doi:[10.1016/S0377-2217\(97\)00347-0](https://doi.org/10.1016/S0377-2217(97)00347-0).
- Bierwirth, C., Kuhpfahl, J., 2017. Extended grasp for the job shop scheduling problem with total weighted tardiness objective. *European Journal of Operational Research* 261, 835–848. doi:<https://doi.org/10.1016/j.ejor.2017.03.030>.
- Bilgen, B., Ozkarahan, I., 2007. A mixed-integer linear programming model for bulk grain blending and shipping. *International Journal of Production Economics* 107, 555–571. doi:<https://doi.org/10.1016/j.ijpe.2006.11.008>. operations Management in China.

- Błażewicz, J., Dell’Olmo, P., Drozdowski, M., Maczka, P., 2003. Scheduling multiprocessor tasks on parallel processors with limited availability. *European Journal of Operational Research* 149, 377 – 389. doi:[10.1016/S0377-2217\(02\)00760-9](https://doi.org/10.1016/S0377-2217(02)00760-9). sequencing and Scheduling.
- Błażewicz, J., Dell’Olmo, P., Drozdowski, M., Speranza, M., 1992. Scheduling multiprocessor tasks on three dedicated processors. *Information Processing Letters* 41, 275 – 280. doi:[10.1016/0020-0190\(92\)90172-R](https://doi.org/10.1016/0020-0190(92)90172-R).
- Bogner, K., Pferschy, U., Unterberger, R., Zeiner, H., 2018. Optimised scheduling in human–robot collaboration – a use case in the assembly of printed circuit boards. *International Journal of Production Research* , 1–19doi:[10.1080/00207543.2018.1470695](https://doi.org/10.1080/00207543.2018.1470695).
- Bowman, E.H., 1959. The schedule-sequencing problem. *Operations Research* 7, 621–624. doi:[10.1287/opre.7.5.621](https://doi.org/10.1287/opre.7.5.621).
- Branke, J., Hildebrandt, T., Scholz-Reiter, B., 2015. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. *Evolutionary Computation* 23, 249–277. doi:https://doi.org/10.1162/evco_a_00131.
- Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M., 2016. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation* 20, 110–124.
- Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E., 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research* 112, 3 – 41. doi:[10.1016/S0377-2217\(98\)00204-5](https://doi.org/10.1016/S0377-2217(98)00204-5).
- Brucker, P., Jurisch, B., Krämer, A., 1997. Complexity of scheduling problems with multi-purpose machines. *Annals of Operations Research* 70, 57–73. doi:[10.1023/A:1018950911030](https://doi.org/10.1023/A:1018950911030).
- Brucker, P., Neyer, J., 1998. Tabu-search for the multi-mode job-shop problem. *OR Spektrum* 20, 21–28.
- Burdett, R.L., Corry, P., Eustace, C., Smith, S., 2020. A flexible job shop scheduling approach with operators for coal export terminals – a mature approach. *Computers & Operations Research* 115, 104834. doi:<https://doi.org/10.1016/j.cor.2019.104834>.
- Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R., 2009. Exploring Hyper-heuristic Methodologies with Genetic Programming. Springer Berlin Heidelberg, Berlin, Heidelberg. chapter 1. pp. 177–201. doi:[10.1007/978-3-642-01799-5_6](https://doi.org/10.1007/978-3-642-01799-5_6).
- Cadavid, J.P.U., Lamouri, S., Grabot, B., Pellerin, R., Fortin, A., 2020. Machine learning applied in production planning and control: a state-of-the-art in the era of industry 4.0. *Journal of Intelligent Manufacturing* 31, 1173–1181. doi:[10.1007/s10845-019-01531-7](https://doi.org/10.1007/s10845-019-01531-7).

- Caramia, M., Giordani, S., 2009. A new approach for scheduling independent tasks with multiple modes. *Journal of Heuristics* 15, 313–329. doi:[10.1007/s10732-007-9062-y](https://doi.org/10.1007/s10732-007-9062-y).
- Caramia, M., Giordani, S., 2010. A fast metaheuristic for scheduling independent tasks with multiple modes. *Computers & Industrial Engineering* 58, 64–69. doi:[10.1016/j.cie.2009.08.004](https://doi.org/10.1016/j.cie.2009.08.004).
- Casalino, A., Zanchettin, A.M., Piroddi, L., Rocco, P., 2019. Optimal scheduling of human-robot collaborative assembly operations with time petri nets. *IEEE Transactions on Automation Science and Engineering* , 1–15doi:[10.1109/TASE.2019.2932150](https://doi.org/10.1109/TASE.2019.2932150).
- Chen, B., Matis, T.I., 2013. A flexible dispatching rule for minimizing tardiness in job shop scheduling. *International Journal of Production Economics* 141, 360 – 365. doi:<https://doi.org/10.1016/j.ijpe.2012.08.019>. meta-heuristics for manufacturing scheduling and logistics problems.
- Chen, F., Sekiyama, K., Cannella, F., Fukuda, T., 2014. Optimal subtask allocation for human and robot collaboration within hybrid assembly system. *IEEE Transactions on Automation Science and Engineering* 11, 1065–1075. doi:[10.1109/TASE.2013.2274099](https://doi.org/10.1109/TASE.2013.2274099).
- Chen, J., Lee, C.Y., 1999. General multiprocessor task scheduling. *Naval Research Logistics* 46, 72–74. doi:[10.1002/\(SICI\)1520-6750\(199902\)46:1<57::AID-NAV4>3.3.CO;2-8](https://doi.org/10.1002/(SICI)1520-6750(199902)46:1<57::AID-NAV4>3.3.CO;2-8).
- Chen, J., Miranda, A., 2001. A polynomial time approximation scheme for general multiprocessor job scheduling. *SIAM Journal on Computing* 31, 1–17. doi:[10.1137/S0097539798348110](https://doi.org/10.1137/S0097539798348110).
- Chen, Z., Demeulemeester, E., Bai, S., Guo, Y., 2018. Efficient priority rules for the stochastic resource-constrained project scheduling problem. *European Journal of Operational Research* 270, 957 – 967. doi:<https://doi.org/10.1016/j.ejor.2018.04.025>.
- Darvish, K., Wanderlingh, F., Bruno, B., Simetti, E., Mastrogiovanni, F., Casalino, G., 2018. Flexible human–robot cooperation models for assisted shop-floor tasks. *Mechatronics* 51, 97 – 114. URL: <http://www.sciencedirect.com/science/article/pii/S0957415818300485>, doi:<https://doi.org/10.1016/j.mechatronics.2018.03.006>.
- Demir, Y., Kürşat İşleyen, S., 2013. Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling* 37, 977–988. URL: <https://www.sciencedirect.com/science/article/pii/S0307904X12001837>, doi:<https://doi.org/10.1016/j.apm.2012.03.020>.
- Dianatfar, M., Latokartano, J., Lanz, M., 2019. Task balancing between human and robot in mid-heavy assembly tasks. *Procedia CIRP* 81, 157–161. URL: <https://www>.

- [sciencedirect.com/science/article/pii/S2212827119303336](https://www.sciencedirect.com/science/article/pii/S2212827119303336), doi:<https://doi.org/10.1016/j.procir.2019.03.028>. 52nd CIRP Conference on Manufacturing Systems (CMS), Ljubljana, Slovenia, June 12-14, 2019.
- Ding, H., Schipper, M., Matthias, B., 2014. Optimized task distribution for industrial assembly in mixed human-robot environments - case study on IO module assembly, in: 2014 IEEE International Conference on Automation Science and Engineering (CASE), pp. 19–24. doi:[10.1109/CoASE.2014.6899298](https://doi.org/10.1109/CoASE.2014.6899298).
- Drozdowski, M., 1996. Scheduling multiprocessor tasks - an overview. *European Journal of Operational Research* 94, 215–230. doi:[10.1016/0377-2217\(96\)00123-3](https://doi.org/10.1016/0377-2217(96)00123-3).
- Drozdowski, M., 2009. *Parallel Tasks*. Springer London, London. chapter 5. pp. 87–208. doi:[10.1007/978-1-84882-310-5_5](https://doi.org/10.1007/978-1-84882-310-5_5).
- Edis, E.B., Oguz, C., Ozkaran, I., 2013. Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *European Journal of Operational Research* 230, 449 – 463. doi:[10.1016/j.ejor.2013.02.042](https://doi.org/10.1016/j.ejor.2013.02.042).
- Edis, E.B., Ozkaran, I., 2011. A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. *Engineering Optimization* 43, 135–157. doi:[10.1080/03052151003759117](https://doi.org/10.1080/03052151003759117).
- Glover, F., 2000. *Multi-Start and Strategic Oscillation Methods — Principles to Exploit Adaptive Memory*. Springer US, Boston, MA. pp. 1–23. doi:[10.1007/978-1-4615-4567-5_1](https://doi.org/10.1007/978-1-4615-4567-5_1).
- Gökgür, B., Hnich, B., Özpeynirci, S., 2018. Parallel machine scheduling with tool loading: a constraint programming approach. *International Journal of Production Research* 56, 5541–5557. doi:[10.1080/00207543.2017.1421781](https://doi.org/10.1080/00207543.2017.1421781).
- Goldberg, D.E., 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Gombolay, M.C., Bair, A., Huang, C., Shah, J.A., 2017. Computational design of mixed-initiative human–robot teaming that considers human factors: situational awareness, workload, and workflow preferences. *The International Journal of Robotics Research* 36, 597–617. doi:[10.1177/0278364916688255](https://doi.org/10.1177/0278364916688255).
- Gombolay, M.C., Gutierrez, R.A., Clarke, S.G., Sturla, G.F., Shah, J.A., 2015. Decision-making authority, team efficiency and human worker satisfaction in mixed human–robot teams. *Autonomous Robots* 39, 293–312. doi:[10.1007/s10514-015-9457-9](https://doi.org/10.1007/s10514-015-9457-9).
- Gombolay, M.C., Wilcox, R.J., Shah, J.A., 2018. Fast scheduling of robot teams performing tasks with temporospatial constraints. *IEEE Transactions on Robotics* 34, 220–239. doi:[10.1109/TR0.2018.2795034](https://doi.org/10.1109/TR0.2018.2795034).

- Gonçalves, J.F., Resende, M.G.C., 2011. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics* 17, 487–525. URL: <https://doi.org/10.1007/s10732-010-9143-1>, doi:10.1007/s10732-010-9143-1.
- Guan, Y., Xiao, W.Q., Cheung, R.K., Li, C.L., 2002. A multiprocessor task scheduling model for berth allocation: heuristic and worst-case analysis. *Operations Research Letters* 30, 343 – 350. doi:10.1016/S0167-6377(02)00147-5.
- Hancock, P.A., Billings, D.R., Schaefer, K.E., Chen, J.Y.C., de Visser, E.J., Parasuraman, R., 2011. A meta-analysis of factors affecting trust in human-robot interaction. *Human Factors* 53, 517–527. doi:10.1177/0018720811417254.
- Hart, E., Sim, K., 2016. A hyper-heuristic ensemble method for static job-shop scheduling. *Evolutionary Computation* 24, 609–635.
- Hartmann, S., Briskorn, D., 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* 207, 1 – 14. doi:10.1016/j.ejor.2009.11.005.
- Haupt, R., 1989. A survey of priority rule-based scheduling. *Operations Research - Spektrum* 11, 3–16. doi:10.1007/BF01721162.
- Hein, F., Almeder, C., Figueira, G., Almada-Lobo, B., 2018. Designing new heuristics for the capacitated lot sizing problem by genetic programming. *Computers & Operations Research* 96, 1 – 14. doi:<https://doi.org/10.1016/j.cor.2018.03.006>.
- Herrmann, J.W., Lee, C.Y., Hinchman, J., 1995. Global job shop scheduling with a genetic algorithm. *Production and Operations Management* 4, 30–45. doi:10.1111/j.1937-5956.1995.tb00039.x.
- Hildebrandt, T., Branke, J., 2015. On using surrogates with genetic programming. *Evol. Comput.* 23, 343–367. URL: https://doi.org/10.1162/EVCO_a_00133, doi:10.1162/EVCO_a_00133.
- Hildebrandt, T., Heger, J., Scholz-Reiter, B., 2010. Towards improved dispatching rules for complex shop floor scenarios: A genetic programming approach, in: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, NY, USA, pp. 257–264. doi:10.1145/1830483.1830530.
- Ho, N.B., Tay, J.C., Lai, E.M.K., 2007. An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research* 179, 316 – 333. doi:<https://doi.org/10.1016/j.ejor.2006.04.007>.
- Holthaus, O., Rajendran, C., 1997. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics* 48, 87 – 105. doi:[https://doi.org/10.1016/S0925-5273\(96\)00068-0](https://doi.org/10.1016/S0925-5273(96)00068-0).
- Holthaus, O., Rajendran, C., 2000. Efficient jobshop dispatching rules: Further developments. *Production Planning & Control* 11, 171–178. doi:10.1080/095372800232379.

- Huyet, A.L., *, J.L.P., 2004. Synergy between evolutionary optimization and induction graphs learning for simulated manufacturing systems. *International Journal of Production Research* 42, 4295–4313. doi:[10.1080/00207540410001708489](https://doi.org/10.1080/00207540410001708489).
- IFR, 2018. Demystifying Collaborative Industrial Robots. Technical Report. International Federation of Robotics. URL: <https://ifr.org>.
- Ivanov, D., Tang, C.S., Dolgui, A., Battini, D., Das, A., 2020. Researchers' perspectives on industry 4.0: multi-disciplinary analysis and opportunities for operations management. *International Journal of Production Research* 0, 1–24. doi:[10.1080/00207543.2020.1798035](https://doi.org/10.1080/00207543.2020.1798035).
- Jain, S., Foley, W., 2016. Dispatching strategies for managing uncertainties in automated manufacturing systems. *European Journal of Operational Research* 248, 328 – 341. doi:<https://doi.org/10.1016/j.ejor.2015.06.060>.
- Jansen, K., Porkolab, L., 2005. General multiprocessor task scheduling: Approximate solutions in linear time. *SIAM Journal on Computing* 35, 519–530. doi:[10.1137/S0097539799361737](https://doi.org/10.1137/S0097539799361737).
- Johannsmeier, L., Haddadin, S., 2017. A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes. *IEEE Robotics and Automation Letters* 2, 41–48. doi:[10.1109/LRA.2016.2535907](https://doi.org/10.1109/LRA.2016.2535907).
- Jun, S., Lee, S., Chun, H., 2019. Learning dispatching rules using random forest in flexible job shop scheduling problems. *International Journal of Production Research* 57, 3290–3310. doi:[10.1080/00207543.2019.1581954](https://doi.org/10.1080/00207543.2019.1581954).
- Karunakaran, D., Yi Mei, Gang Chen, Mengjie Zhang, 2017. Evolving dispatching rules for dynamic job shop scheduling with uncertain processing times, in: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 364–371.
- Kim, B.I., Koo, J., Park, B.S., 2009. A raw material storage yard allocation problem for a large-scale steelworks. *The International Journal of Advanced Manufacturing Technology* 41, 880–884. doi:[10.1007/s00170-008-1538-x](https://doi.org/10.1007/s00170-008-1538-x).
- Koza, J.R., 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing* 4, 87–112.
- Krüger, J., Lien, T., Verl, A., 2009. Cooperation of human and machines in assembly lines. *CIRP Annals* 58, 628–646. doi:[10.1016/j.cirp.2009.09.009](https://doi.org/10.1016/j.cirp.2009.09.009).
- Ku, W.Y., Beck, J.C., 2016. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research* 73, 165–173. URL: <https://www.sciencedirect.com/science/article/pii/S0305054816300764>, doi:<https://doi.org/10.1016/j.cor.2016.04.006>.
- van Laarhoven, P.J.M., Aarts, E.H.L., Lenstra, J.K., 1992. Job shop scheduling by simulated annealing. *Operations Research* 40, 113–125. doi:[10.1287/opre.40.1.113](https://doi.org/10.1287/opre.40.1.113).

- Lawrence, S.R., Sewell, E.C., 1997. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. *Journal of Operations Management* 15, 71 – 82. doi:[https://doi.org/10.1016/S0272-6963\(96\)00090-3](https://doi.org/10.1016/S0272-6963(96)00090-3).
- Lee, S., Kim, M., Song, J., 2015. Sensorless collision detection for safe human-robot collaboration, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2392–2397. doi:[10.1109/IROS.2015.7353701](https://doi.org/10.1109/IROS.2015.7353701).
- Leung, J.Y.T., Li, C.L., 2008. Scheduling with processing set restrictions: A survey. *International Journal of Production Economics* 116, 251 – 262. URL: <http://www.sciencedirect.com/science/article/pii/S0925527308003071>, doi:<https://doi.org/10.1016/j.ijpe.2008.09.003>.
- Liu, H., Wang, L., 2018. Gesture recognition for human-robot collaboration: A review. *International Journal of Industrial Ergonomics* 68, 355 – 367. doi:[10.1016/j.ergon.2017.02.004](https://doi.org/10.1016/j.ergon.2017.02.004).
- Manne, A.S., 1960. On the job-shop scheduling problem. *Operations Research* 8, 219–223. doi:[10.1287/opre.8.2.219](https://doi.org/10.1287/opre.8.2.219).
- Mateus, J., Claeyss, D., Limère, V., Cottyn, J., Aghezzaf, E.H., 2019. A structured methodology for the design of a human-robot collaborative assembly workplace. *The International Journal of Advanced Manufacturing Technology* 102. doi:[10.1007/s00170-019-03356-3](https://doi.org/10.1007/s00170-019-03356-3).
- Matheson, E., Minto, R., Zampieri, E.G.G., Faccio, M., Rosati, G., 2019. Human-robot collaboration in manufacturing applications: A review. *Robotics* 8. URL: <https://www.mdpi.com/2218-6581/8/4/100>, doi:[10.3390/robotics8040100](https://doi.org/10.3390/robotics8040100).
- Mei, Y., Zhang, M., Nyugen, S., 2016. Feature selection in evolving job shop dispatching rules with genetic programming, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, Association for Computing Machinery, New York, NY, USA. p. 365–372. doi:[10.1145/2908812.2908822](https://doi.org/10.1145/2908812.2908822).
- Menezes, G.C., Mateus, G.R., Ravetti, M.G., 2017. A branch and price algorithm to solve the integrated production planning and scheduling in bulk ports. *European Journal of Operational Research* 258, 926–937. doi:<https://doi.org/10.1016/j.ejor.2016.08.073>.
- Michalos, G., Makris, S., Tsarouchi, P., Guasch, T., Kontovrakis, D., Chryssolouris, G., 2015. Design considerations for safe human-robot collaborative workplaces. *Procedia CIRP* 37, 248 – 253. doi:[10.1016/j.procir.2015.08.014](https://doi.org/10.1016/j.procir.2015.08.014).
- Nguyen, S., Mei, Y., Xue, B., Zhang, M., 2019a. A hybrid genetic programming algorithm for automated design of dispatching rules. *Evolutionary Computation* 27, 467–496. doi:[10.1162/evco_a_00230](https://doi.org/10.1162/evco_a_00230).
- Nguyen, S., Mei, Y., Zhang, M., 2017a. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems* 3, 41–66.

- Nguyen, S., Zhang, M., Alahakoon, D., Tan, K., 2019b. People-centric evolutionary system for dynamic production scheduling. *IEEE TRANSACTIONS ON CYBERNETICS* PP. doi:[10.1109/TCYB.2019.2936001](https://doi.org/10.1109/TCYB.2019.2936001).
- Nguyen, S., Zhang, M., Johnston, M., Tan, K., 2013. Learning iterative dispatching rules for job shop scheduling with genetic programming. *The International Journal of Advanced Manufacturing Technology* 67. doi:[10.1007/s00170-013-4756-9](https://doi.org/10.1007/s00170-013-4756-9).
- Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2013. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation* 17, 621–639.
- Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2013. Dynamic Multi-objective Job Shop Scheduling: A Genetic Programming Approach. Springer Berlin Heidelberg, Berlin, Heidelberg. chapter 10. pp. 251–282. doi:https://doi.org/10.1007/978-3-642-39304-4_10.
- Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2014. Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation* 18, 193–208.
- Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2014. Selection schemes in surrogate-assisted genetic programming for job shop scheduling, in: Dick, G., Browne, W.N., Whigham, P., Zhang, M., Bui, L.T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K.C., Tang, K. (Eds.), *Simulated Evolution and Learning*, Springer International Publishing, Cham. pp. 656–667.
- Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2015. Automatic programming via iterated local search for dynamic job shop scheduling. *IEEE Transactions on Cybernetics* 45, 1–14.
- Nguyen, S., Zhang, M., Tan, K.C., 2017b. Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. *IEEE Transactions on Cybernetics* 47, 2951–2965.
- Nie, L., Gao, L., Li, P., Li, X., 2013. A gep-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. *Journal of Intelligent Manufacturing* 24, 763–774. doi:<https://doi.org/10.1007/s10845-012-0626-9>.
- Nikolakis, N., Kousi, N., Michalos, G., Makris, S., 2018. Dynamic scheduling of shared human-robot manufacturing operations. *Procedia CIRP* 72, 9 – 14. doi:[10.1016/j.procir.2018.04.007](https://doi.org/10.1016/j.procir.2018.04.007). 51st CIRP Conference on Manufacturing Systems.
- Olafsson, S., Li, X., 2010. Learning effective new single machine dispatching rules from optimal scheduling data. *International Journal of Production Economics* 128, 118 – 126. doi:<https://doi.org/10.1016/j.ijpe.2010.06.004>. integrating the Global Supply Chain.

- Olsen, T.L., Tomlin, B., 2020. Industry 4.0: Opportunities and challenges for operations management. *Manufacturing & Service Operations Management* 22, 113–122. doi:[10.1287/msom.2019.0796](https://doi.org/10.1287/msom.2019.0796).
- Ouelhadj, D., Petrovic, S., 2008. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling* 12, 417. doi:[10.1007/s10951-008-0090-8](https://doi.org/10.1007/s10951-008-0090-8).
- Pappis, C.P., Rachaniotis, N.P., 2010. Scheduling in a multi-processor environment with deteriorating job processing times and decreasing values: the case of forest fires. *Journal of Heuristics* 16, 617–632. doi:[10.1007/s10732-009-9110-x](https://doi.org/10.1007/s10732-009-9110-x).
- Parente, M., Figueira, G., Amorim, P., Marques, A., 2020. Production scheduling in the context of industry 4.0: review and trends. *International Journal of Production Research* 0, 1–31. doi:[10.1080/00207543.2020.1718794](https://doi.org/10.1080/00207543.2020.1718794).
- Park, J., Mei, Y., Nguyen, S., Chen, G., Zhang, M., 2018. An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. *Applied Soft Computing* 63, 72 – 86. doi:<https://doi.org/10.1016/j.asoc.2017.11.020>.
- Pellegrinelli, S., Orlandini, A., Pedrocchi, N., Umbrico, A., Tolio, T., 2017. Motion planning and scheduling for human and industrial-robot collaboration. *CIRP Annals* 66, 1 – 4. doi:<https://doi.org/10.1016/j.cirp.2017.04.095>.
- Pergher, I., Frej, E.A., Roselli, L.R.P., [de Almeida], A.T., 2020. Integrating simulation and fitradeoff method for scheduling rules selection in job-shop production systems. *International Journal of Production Economics* 227, 107669. doi:<https://doi.org/10.1016/j.ijpe.2020.107669>.
- Pezzella, F., Morganti, G., Ciaschetti, G., 2008. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research* 35, 3202–3212. URL: <https://www.sciencedirect.com/science/article/pii/S0305054807000524>, doi:<https://doi.org/10.1016/j.cor.2007.02.014>. part Special Issue: Search-based Software Engineering.
- Pickardt, C., Hildebrandt, T., Branke, J., Heger, J., Scholz-Reiter, B., 2012. Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. *International Journal of Production Economics* 145. doi:[10.1016/j.ijpe.2012.10.016](https://doi.org/10.1016/j.ijpe.2012.10.016).
- Pinedo, M.L., 2008. *Scheduling - Theory, Algorithms and Systems*. 3rd ed., Springer.
- Pinto, G., Ainbinder, I., Rabinowitz, G., 2009. A genetic algorithm-based approach for solving the resource-sharing and scheduling problem. *Computers & Industrial Engineering* 57, 1131 – 1143. URL: <http://www.sciencedirect.com/science/article/pii/S0360835209001338>, doi:<https://doi.org/10.1016/j.cie.2009.05.003>.
- Raghu, T., Rajendran, C., 1993. An efficient dynamic dispatching rule for scheduling in a job shop. *International Journal of Production Economics* 32, 301 – 313. doi:[https://doi.org/10.1016/0925-5273\(93\)90044-L](https://doi.org/10.1016/0925-5273(93)90044-L).

- Rajendran, C., Holthaus, O., 1999. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research* 116, 156 – 170. doi:[https://doi.org/10.1016/S0377-2217\(98\)00023-X](https://doi.org/10.1016/S0377-2217(98)00023-X).
- Ramasesh, R., 1990. Dynamic job shop scheduling: A survey of simulation research. *Omega* 18, 43 – 57. doi:[https://doi.org/10.1016/0305-0483\(90\)90017-4](https://doi.org/10.1016/0305-0483(90)90017-4).
- Robenek, T., Umang, N., Bierlaire, M., Ropke, S., 2014. A branch-and-price algorithm to solve the integrated berth allocation and yard assignment problem in bulk ports. *European Journal of Operational Research* 235, 399–411. URL: <https://www.sciencedirect.com/science/article/pii/S0377221713006802>, doi:<https://doi.org/10.1016/j.ejor.2013.08.015>. *maritime Logistics*.
- Rocha de Paula, M., Boland, N., Ernst, A.T., Mendes, A., Savelsbergh, M., 2019. Throughput optimisation in a coal export system with multiple terminals and shared resources. *Computers & Industrial Engineering* 134, 37–51. doi:<https://doi.org/10.1016/j.cie.2019.05.021>.
- Romero-Silva, R., Shaaban, S., Marsillac, E., Hurtado, M., 2018. Exploiting the characteristics of serial queues to reduce the mean and variance of flow time using combined priority rules. *International Journal of Production Economics* 196, 211 – 225. doi:<https://doi.org/10.1016/j.ijpe.2017.11.023>.
- Rossit, D.A., Tohmé, F., Frutos, M., 2019. Production planning and scheduling in cyber-physical production systems: a review. *International Journal of Computer Integrated Manufacturing* 32, 385–395. doi:[10.1080/0951192X.2019.1605199](https://doi.org/10.1080/0951192X.2019.1605199).
- Sabuncuoglu, I., Bayız, M., 2000. Analysis of reactive scheduling problems in a job shop environment. *European Journal of Operational Research* 126, 567 – 586. doi:[https://doi.org/10.1016/S0377-2217\(99\)00311-2](https://doi.org/10.1016/S0377-2217(99)00311-2).
- Sabuncuoglu, I., Kizilisik, O.B., 2003. Reactive scheduling in a dynamic and stochastic fms environment. *International Journal of Production Research* 41, 4211–4231. doi:[10.1080/0020754031000149202](https://doi.org/10.1080/0020754031000149202).
- Shahzad, A., Mebarki, N., 2012. Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem. *Engineering Applications of Artificial Intelligence* 25, 1173–1181. URL: <https://www.sciencedirect.com/science/article/pii/S0952197612000899>, doi:<https://doi.org/10.1016/j.engappai.2012.04.001>.
- Shen, L., Dautère-Pères, S., Neufeld, J.S., 2018. Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research* 265, 503–516. URL: <https://www.sciencedirect.com/science/article/pii/S037722171730752X>, doi:<https://doi.org/10.1016/j.ejor.2017.08.021>.

- Shi, D., Fan, W., Xiao, Y., Lin, T., Xing, C., 2020. Intelligent scheduling of discrete automated production line via deep reinforcement learning. *International Journal of Production Research* 58, 3362–3380. doi:[10.1080/00207543.2020.1717008](https://doi.org/10.1080/00207543.2020.1717008).
- Stricker, N., Kuhnle, A., Sturm, R., Friess, S., 2018. Reinforcement learning for adaptive order dispatching in the semiconductor industry. *CIRP Annals* 67, 511 – 514. URL: <http://www.sciencedirect.com/science/article/pii/S0007850618300659>, doi:<https://doi.org/10.1016/j.cirp.2018.04.041>.
- Su, H., Pinedo, M., Wan, G., 2017. Parallel machine scheduling with eligibility constraints: A composite dispatching rule to minimize total weighted tardiness. *Naval Research Logistics (NRL)* 64, 249–267. doi:[10.1002/nav.21744](https://doi.org/10.1002/nav.21744).
- Sun, D., Tang, L., 2013. Benders approach for the raw material transportation scheduling problem in steel industry, in: 2013 10th IEEE International Conference on Control and Automation (ICCA), pp. 481–484. doi:[10.1109/ICCA.2013.6565191](https://doi.org/10.1109/ICCA.2013.6565191).
- Sweeney, K.D., Sweeney, D.C., Campbell, J.F., 2019. The performance of priority dispatching rules in a complex job shop: A study on the upper mississippi river. *International Journal of Production Economics* 216, 154 – 172. doi:<https://doi.org/10.1016/j.ijpe.2019.04.024>.
- Switalski, P., Seredynski, F., 2015. Scheduling parallel batch jobs in grids with evolutionary metaheuristics. *Journal of Scheduling* 18, 345–357. doi:[10.1007/s10951-014-0382-0](https://doi.org/10.1007/s10951-014-0382-0).
- Tay, J.C., Ho, N.B., 2008. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering* 54, 453 – 473. doi:<https://doi.org/10.1016/j.cie.2007.08.008>.
- Tsang, E., 1993. *Foundations of Constraint Satisfaction*. 2 ed., Academic Press Limited. doi:[10.1016/C2013-0-07627-X](https://doi.org/10.1016/C2013-0-07627-X).
- Tsarouchi, P., Michalos, G., Makris, S., Athanasatos, T., Dimoulas, K., Chrysosolouris, G., 2017. On a human-robot workplace design and task allocation system. *International Journal of Computer Integrated Manufacturing* 30, 1272–1279. doi:[10.1080/0951192X.2017.1307524](https://doi.org/10.1080/0951192X.2017.1307524).
- Türkoğullar, Y.B., Taşkın, Z.C., Aras, N., Altınel, İ.K., 2014. Optimal berth allocation and time-invariant quay crane assignment in container terminals. *European Journal of Operational Research* 235, 88 – 101. doi:[10.1016/j.ejor.2013.10.015](https://doi.org/10.1016/j.ejor.2013.10.015).
- Umang, N., Bierlaire, M., Vacca, I., 2013. Exact and heuristic methods to solve the berth allocation problem in bulk ports. *Transportation Research Part E: Logistics and Transportation Review* 54, 14–31. doi:<https://doi.org/10.1016/j.tre.2013.03.003>.
- UNCTAD, 2020. *Review of Maritime Transport*. Technical Report. United Nations Conference on Trade and Development. URL: <https://unctad.org/webflyer/review-maritime-transport-2020>.

- Unsal, O., Oguz, C., 2019. An exact algorithm for integrated planning of operations in dry bulk terminals. *Transportation Research Part E: Logistics and Transportation Review* 126, 103–121. doi:<https://doi.org/10.1016/j.tre.2019.03.018>.
- Varadarajan, A., Sarin, S.C., 2006. A survey of dispatching rules for operational control in wafer fabrication. *IFAC Proceedings Volumes* 39, 715 – 726. doi:<https://doi.org/10.3182/20060517-3-FR-2903.00363>. 12th IFAC Symposium on Information Control Problems in Manufacturing.
- Wagner, H.M., 1959. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly* 6, 131–140. doi:<https://doi.org/10.1002/nav.3800060205>.
- Wang, W., Li, R., Chen, Y., Diekel, Z.M., Jia, Y., 2019. Facilitating human–robot collaborative tasks by teaching-learning-collaboration from human demonstrations. *IEEE Transactions on Automation Science and Engineering* 16, 640–653. doi:[10.1109/TASE.2018.2840345](https://doi.org/10.1109/TASE.2018.2840345).
- Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., Kyek, A., 2018. Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP* 72, 1264 – 1269. URL: <http://www.sciencedirect.com/science/article/pii/S221282711830372X>, doi:<https://doi.org/10.1016/j.procir.2018.03.212>. 51st CIRP Conference on Manufacturing Systems.
- Weglarz, J., Drabowski, M., Błażewicz, J., 1986. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Transactions on Computers* 35, 389–393. doi:[10.1109/TC.1986.1676781](https://doi.org/10.1109/TC.1986.1676781).
- Wu, C.H., Yao, Y.C., Dauzère-Pérès, S., Yu, C.J., 2020. Dynamic dispatching and preventive maintenance for parallel machines with dispatching-dependent deterioration. *Computers & Operations Research* 113, 104779. doi:<https://doi.org/10.1016/j.cor.2019.104779>.
- Xiong, H., Fan, H., Jiang, G., Li, G., 2017. A simulation-based study of dispatching rules in a dynamic job shop scheduling problem with batch release and extended technical precedence constraints. *European Journal of Operational Research* 257, 13 – 24. doi:<https://doi.org/10.1016/j.ejor.2016.07.030>.
- Xu, W., Tang, Q., Liu, J., Liu, Z., Zhou, Z., Pham, D.T., 2020. Disassembly sequence planning using discrete bees algorithm for human-robot collaboration in remanufacturing. *Robotics and Computer-Integrated Manufacturing* 62, 101860. URL: <http://www.sciencedirect.com/science/article/pii/S0736584519301887>, doi:<https://doi.org/10.1016/j.rcim.2019.101860>.
- Zanchettin, A.M., 2021. Robust scheduling and dispatching rules for high-mix collaborative manufacturing systems. *International Journal of Flexible Manufacturing Systems* 0, to appear.

- Zanchettin, A.M., Casalino, A., Piroddi, L., Rocco, P., 2019. Prediction of human activity patterns for human–robot collaborative assembly tasks. *IEEE Transactions on Industrial Informatics* 15, 3934–3942. doi:[10.1109/TII.2018.2882741](https://doi.org/10.1109/TII.2018.2882741).
- Zhang, G., Gao, L., Shi, Y., 2011. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications* 38, 3563–3573. URL: <https://www.sciencedirect.com/science/article/pii/S095741741000953X>, doi:<https://doi.org/10.1016/j.eswa.2010.08.145>.
- Zhou, Y., jun Yang, J., Huang, Z., 2019. Automatic design of scheduling policies for dynamic flexible job shop scheduling via surrogate-assisted cooperative co-evolution genetic programming. *International Journal of Production Research* 0, 1–20. doi:[10.1080/00207543.2019.1620362](https://doi.org/10.1080/00207543.2019.1620362).

Effective and interpretable rules via Genetic Programming

Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning

Cristiane Ferreira^{*†} · Gonalo Figueira^{*†} · Pedro Amorim^{*†}

Accepted in *Omega*

Abstract The emergence of Industry 4.0 is making production systems more flexible and also more dynamic. In these settings, schedules often need to be adapted in real-time by dispatching rules. Although substantial progress was made until the '90s, the performance of these rules is still rather limited. The machine learning literature is developing a variety of methods to improve them. However, the resulting rules are difficult to interpret and do not generalise well for a wide range of settings. This paper is the first major attempt at combining machine learning with domain problem reasoning for scheduling. The idea consists of using the insights obtained with the latter to guide the empirical search of the former. We hypothesise that this guided empirical learning process should result in effective and interpretable dispatching rules that generalise well to different scenarios. We test our approach in the classical dynamic job shop scheduling problem minimising tardiness, one of the most well-studied scheduling problems. The simulation experiments include a wide spectrum of scenarios for the first time, from highly loose to tight due dates and from low utilisation conditions to severely congested shops. Nonetheless, results show that our approach can find new state-of-the-art rules, which significantly outperform the existing literature in the vast majority of settings. Overall, the average improvement over the best combination of benchmark rules is 19%. Moreover, the rules are compact, interpretable, and generalise well to extreme, unseen scenarios. Therefore, we believe this methodology could be a new paradigm for applying machine learning to dynamic optimisation problems.

Keywords Scheduling · Dynamic Job Shop · Dispatching Rules · Genetic Programming

^{*}Faculdade de Engenharia, Universidade do Porto, Porto, Portugal

[†]INESC TEC - Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Cincia, Porto, Portugal

3.1. Introduction

Industrial environments experience disruptive progress with the advent of Industry 4.0 (i4.0). Several technologies enable real-time interaction between physical and digital machines, such as additive manufacturing, Internet of Things, distributed ledgers, advanced robotics and artificial intelligence [Olsen and Tomlin, 2020]. These advancements provide more flexible production systems capable of responding quickly to new demands, breakdowns, and other unforeseen events. However, this adds also more complexity to production scheduling. Processes are naturally more dynamic and subject to disruptions: new job arrivals, lot size changes, cancellations, production failures or machine breakdowns. The scheduling of operations must therefore be designed to respond in real-time to possible deviations [Parente et al., 2020]. Thus, research is needed to leverage dynamic scheduling to prompt and effectively re-optimize operations' sequence under unexpected events.

Different strategies are applied to manage uncertainty in dynamic scheduling. In an entirely reactive approach, no schedule is generated in advance. Instead, real-time decisions are made upon the occurrence of the events. This strategy usually adopts simple heuristics such as Dispatching Rules (DRs) to prioritise jobs waiting to be processed in a machine's queue [Ouelhadj and Petrovic, 2008]. As simple sequencing procedures, DRs are flexible, easy to implement, and extremely fast to execute. However, their performance depends heavily on the context. Lawrence and Sewell [1997] compare the performance of DRs and exact methods in a dynamic scheduling context. They show that DRs' performance converges to that of a dynamic and sophisticated heuristic for high uncertainty levels. Similarly, Jain and Foley [2016] conclude that under high load levels and long interruptions, the rules work better than adapting a complete schedule.

Effective DRs have been developed for different problems, such as the classical job shops [Chen and Matis, 2013], shops with batch release [Xiong et al., 2017] and parallel machines [Su et al., 2017]. The applications are not limited to manufacturing. For instance, Xu et al. [2016] approach a problem faced by subcontractors in the construction industry. Jung et al. [2019] use scheduling rules for operating rooms in such a way that emergency surgeries are promptly attempted. Sweeney et al. [2019] investigate the development of rules for sequencing vessels in the Upper Mississippi River.

In this work, we aim at designing rules for the Job Shop Scheduling Problem (JSSP), one of the most studied problems in dynamic scheduling. The JSSP is defined on a set of jobs $\mathcal{J}$ that require operations to be processed in a set of machines $\mathcal{M}$. Each job $j \in \mathcal{J}$ has a set of operations O_j that must be executed in a predefined sequence. An operation $i \in O_j$ has a specific processing time p_{ij} and must be executed by a particular machine. Each machine can only process one operation at a time [Pinedo, 2008]. In the dynamic version of the problem, the shop serves a continuous stream of jobs whose arrivals are usually unknown in advance. Thus, the operations' sequence must be continuously re-optimized to adapt to unexpected deviations [Holthaus and Rajendran, 1997]. The metric considered is the mean tardiness, which is challenging because the performance of rules highly depends on the shop load conditions, i.e., there is no rule with superior performance in both tight and light

load conditions [Holthaus and Rajendran, 2000].

Dispatching rules have been traditionally derived by empirical or analytical studies, sustained by scheduling theory. To boost their effectiveness, machine learning algorithms appear as a promising alternative. For example, Shahzad and Mebarki [2012] and Jun et al. [2019] use data-mining and Random Forests, respectively, to learn new dispatching rules from schedules previously obtained. There are also Reinforcement Learning approaches, using Neural Networks either to select dispatching rules or to select the next operation [Waschneck et al., 2018, Luo, 2020] directly. All these approaches use complex, black-box models, such as Neural Networks and tree ensembles, which cannot be easily interpreted. These models are designed and validated with little domain expertise. As a result, despite their good behaviour in some tested cases, they frequently do not generalise well to unseen scenarios.

Our approach aims at designing interpretable DRs and incorporating domain-specific knowledge. This is made possible by the use of Genetic Programming (GP), an evolutionary hyper-heuristic [Burke et al., 2009]. GP evolves computer programs or mathematical expressions, which are executed to solve a problem. The expressions might be used in different ways, such as selecting heuristics [Drake et al., 2020] or computing directly the decisions, like a dispatching rule. One of its advantages compared to other algorithms is the possibility of generating reasonably sized, interpretable-by-design models. By evolving models with GP, researchers might be able to understand their behaviour and even increase knowledge on the problem [Kronberger, 2010].

In the last decade, there has been increasing attention to the use of GP for production scheduling with promising results [Nguyen et al., 2017a]. However, most papers focus on algorithmic efficiency and produce extremely long rules. Moreover, these rules work well in particular instances, but none performs well in a wide range of settings. We propose a guided empirical learning process, whereby insights derived from problem reasoning and existing rules are used to adjust the algorithm search space and thus find better rules. We aim at deriving effective, compact and meaningful rules with good performance in general. To assess the generalisability of the evolved rules, we perform comprehensive computational experiments, varying due date tightness, shop utilisation and number of machines, in a wide spectrum of values. We also test the rules in settings with stochastic processing times. Our hypothesis is that incorporating problem expertise in the algorithm configuration should result in well-performing and more general rules, hopefully also being interpretable.

The remainder of this paper is organised as follows. A literature review is provided in Section 3.2, which first describes the studies on dispatching rules and then reports the main works on GP applied to JSSP. In Section 3.3, we describe our guided empirical learning approach and the algorithm implemented. We detail the learning process and the four GP variants in Section 3.4. In Section 3.5, the evolved rules and their performance comparison are found. We conclude the paper in Section 3.6 with the overall achievements and some final considerations.

3.2. Literature Review

This section describes the related work on dynamic job shop scheduling. At first, we describe the best dispatching rules manually designed, focusing on the mean tardiness metric. Then we review the main achievements on the automated generation of rules using GP.

3.2.1 Dispatching rules for dynamic job shop scheduling

The literature on dynamic scheduling is typically grouped into proactive, reactive and predictive-reactive approaches [Aytug et al., 2005]. The former generate a schedule in advance and tries to minimise the impact of an unexpected event on system performance. Xiong et al. [2013], for instance, propose a multi-objective evolutionary algorithm for generating robust schedules with minimum makespan, and use surrogate models to measure the robustness of the solutions. Predictive-reactive methods provide a baseline schedule and adapt it during the execution. In Jain and Foley [2016], a pairwise exchange heuristic is developed for adjusting the current schedule upon the occurrence of unexpected interruptions. This work advance completely reactive approaches, which are most commonly employed by dispatching rules. Previous studies compared the performance of the rules against heuristics for schedules adaptation. In general, these works conclude that dispatching rules are more suitable than predictive approaches in environments with high uncertainty [Lawrence and Sewell, 1997, Sabuncuoglu and Bayiz, 2000] and high load levels [Jain and Foley, 2016].

Dispatching rules are used to prioritise jobs awaiting to be processed in a queue. They assign scores to each job based on its properties or the system status. Research on the performance of these rules has a long tradition, particularly in dynamic job shops. Haupt [1989] and Ramasesh [1990] are relevant works on the topic, providing thorough explanations on the behaviour and classifications of dispatching rules. The most common performance metrics may be grouped into flowtime-related and due-date-related measures. Next, we discuss the most relevant findings on the use of dispatching rules for job shops.

The flowtime is the time the job spends in the shop from release to its completion time. In general, the SPT rule, which selects the job with the shortest operation processing time, effectively minimises mean flowtime. However, the SPT is myopic; it may choose jobs that could find long queues when visiting the subsequent machines. For this reason, in addition to the current operation processing time, some rules also include *look-ahead information*. Holthaus and Rajendran [1997] were the first to add the workload in the next machine queue (WINQ) to the SPT, devising the rule PT+WINQ. Later on, Rajendran and Holthaus [1999] included the processing time of the job's subsequent operation (NPT) and proposed the rule 2PT+WINQ+NPT, which remains the best for minimising the mean flowtime of jobs.

Tardiness and percentage of tardy jobs are the most common due date related measures. Past research has shown that if due dates are loose or the shop utilisation is low, the dispatching rule should consider some due date information of jobs. Otherwise, in highly

loaded shops, processing-time bases rules work better [Blackstone et al., 1982, Haupt, 1989, Ramasesh, 1990]. Optimising due date measures is challenging as there is no simple rule for all load conditions. However, much work has been developed in that direction. For instance, simulation studies show that prioritising *operations* due date instead of *jobs* due date is more effective. Operations due dates are usually calculated by distributing the allowance (remaining time until due date) of jobs per remaining work (the sum of all subsequent operations processing time).

Anderson and Nyirenda [1990] proposed the rules CR+SPT and S/RPT+SPT, that prioritise operations according to their due dates at dispatching time. To calculate due dates, while the former considers the Critical Ratio (CR), the latter uses the job slack divided by the remaining work. The CR of a job is the ratio between its allowance and its remaining work. Slack is calculated by subtracting the remaining work from the job allowance. The idea behind those rules is that the operation due date should be as short as the proportional job allowance for that operation. In practice, they apply the SPT rule if the due date is close or has passed and the operations due date in the opposite case. In Raghu and Rajendran [1993], the authors use shop utilisation as a parameter to balance between operations processing time (PT) and due date (PT·S/RPT). They also improve the calculation of the WINQ parameter by proposing a better estimation of the next machine's queue. The resulting rule (RR) is $PT \cdot \exp(u) + PT \cdot S/RPT \cdot \exp(-u) + WINQ$, where u is the mean shop utilisation. Although this rule presents good results in general, under very tight load conditions, 2PT+WINQ+NPT performs better [Holthaus and Rajendran, 2000].

Instead of pursuing a unique dispatching rule, recent works focus on selecting rules according to the conditions of the production system. Very different approaches have been proposed in that direction. Romero-Silva et al. [2018] study a flowshop with multi-class jobs. The objective is to analyse if using different dispatching rules in each queue leads to improved mean flowtime and flowtime variance. They concluded that a better performance is obtained when applying distinct rules for each machine. In Amin and El-Bouri [2018], a preference voting model is employed to select rules based on multiple criteria. The authors designed a Linear Programming Model for classifying rules according to their relative performance in each criterion. Pergher et al. [2020] develop a framework for due date assignment, order release and dispatching rules according to the decision-maker preferences and multiple performance criteria. A decision support system performs an interactive process where the user inputs trade-offs judgements.

The recent literature on dispatching rules for the JSSP has focused on specific issues, such as arrivals in batch [Xiong et al., 2017] and precedence between jobs [Fan et al., 2021]. For the classical JSSP minimising the mean tardiness, the rules generated in the '90s remain the state-of-the-art. In the last ten years, the automated design of rules has received much attention, mainly with Genetic Programming. In the following, we describe the main achievements obtained using such method for the generation of rules.

3.2.2 GP on the design of dispatching rules

Machine learning has been increasingly adopted for real-time order dispatching in production systems. [Shahzad and Mebarki \[2012\]](#) apply the C4.5 algorithm on a set of job shop instances and solutions to generate decision trees. These trees outperform existing dispatching rules, but the experiments are performed in static and tiny shops with six jobs. [Jun et al. \[2019\]](#) tackle the Flexible Job Shop Scheduling Problem and use Random Forests to generate decision trees from schedules previously obtained. A Genetic Algorithm is then applied to improve the trees. The experiments are conducted in static and small test instances (with up to 30 jobs) as in the previous work. The authors extend the work in [Jun and Lee \[2021\]](#) and generate rules for Single Machine Scheduling with dynamic arrivals.

A variety of Reinforcement Learning (RL) algorithms has also been applied to dynamic job shops. In general, RL agents choose a dispatching rule or directly select the next operation. [Luo et al. \[2021\]](#), for instance, developed two hierarchical RL agents for optimising the total weighted tardiness and the shop utilisation in flexible job shops. Four distinct reward functions are used to balance both objectives – using Deep Neural Networks as function approximators. Tests are conducted on a large set of scenarios and show that the method outperforms the benchmarking rules. However, the experiments did not include several effective well-known rules for weighted tardiness. The work also provides a detailed review of the use of RL for dynamic job shops.

The ability to interpret decisions is not the focus of any of the works referenced above. They all depend on complex models such as Tree ensembles or Neural Networks, which cannot be easily interpreted. Moreover, in most cases, the experiments do not follow the same scenarios and design of the traditional scheduling literature, preventing their comparison with state-of-the-art methods. Genetic Programming can overcome the interpretability issue. The method can search for new dispatching rules by evolving their structure. It has been applied in the last decade to different scheduling problems with promising results. We briefly describe the main achievements for dynamic job shops in the following. For comprehensive surveys on the use of GP for production scheduling, see [Branke et al. \[2016\]](#) and [Nguyen et al. \[2017a\]](#).

[Hildebrandt et al. \[2010\]](#) and [Branke et al. \[2015\]](#) develop GP algorithms to minimise mean flowtime and show that their results outperform classical dispatching rules. In [Nguyen et al. \[2015\]](#) and [Nguyen et al. \[2019\]](#), the authors find effective rules for several performance metrics – including mean tardiness – but they train specific rules for each load condition. To our knowledge, [Karunakaran et al. \[2017\]](#) and [Ferreira et al. \[2020\]](#) are the only authors focusing on shops with stochastic processing times. However, the former do not benchmark the evolved rules with existing methods, and the latter consider deterministic job arrivals.

Some studies show that GP finds effective ensembles of heuristics for job shops. In [Hart and Sim \[2016\]](#), the authors use GP on the design of a hyper-heuristic for the static version of the JSSP. Their heuristic applies several rules in turn and outperforms single dispatching rules. [Park et al. \[2018\]](#) compares different combination schemes for ensembles. They show that using a linear combination of the rules' scores works better than applying single

dispatching rules or ensembles with majority voting. GP may also be combined with other heuristic approaches such as Iterated Local Search [Nguyen et al., 2015], or more standard evolutionary algorithms [Pickardt et al., 2012].

Different authors studied methods for improving algorithmic efficiency. Surrogate models, for instance, provide faster fitness calculation. They are used to avoid employing high computational effort in the evaluation of non-promising rules. Running these models is usually much efficient than performing a complete simulation for accessing the fitness of individuals [Nguyen et al., 2014, Hildebrandt and Branke, 2015, Nguyen et al., 2017b]. Early studies have shown that the tree-based representation is the best approach to evolve powerful scheduling heuristics Nguyen et al. [2013]. However, as they tend to grow during the evolutionary process, alternative representation structures have also been tried [Nguyen et al., 2013, Branke et al., 2015, Nguyen et al., 2019]. Besides a good representation, an important decision consists of the terminal set, i.e., the set of the jobs and system attributes that compose the rules. Branke et al. [2016] presents a detailed survey on that, and Mei et al. [2016] proposes a mechanism for selecting relevant terminals.

The research on GP for the JSSP extends to variants, as the Flexible JSSP. Tay and Ho [2008] were the first to evolve rules for the problem. Their work was followed by Nie et al. [2013] with a Gene Expression Programming (GEP) approach. Differently from GP, GEP-based algorithms make use of a fixed-length representation structure. We also highlight the work from Zhou et al. [2019], which develops a co-evolutionary GP algorithm for due date assignment and dispatching rules. A multi-objective JSSP was tackled by Nguyen et al. [2013] and Nguyen et al. [2014]. The authors evolve rules that are non-dominated for the percentage of tardy jobs, mean and maximum tardiness, mean and maximum flowtime. Recently, Fan et al. [2021] evolve dispatching rules for the JSSP with extended precedence constraints.

Despite the extensive literature on the automated development of rules for job shops, most works do not benchmark their results with state-of-the-art rules. Others train distinct rules for load condition [Nguyen et al., 2013, Nguyen et al., 2015, Nguyen et al., 2019]. None of the previous works presents a single rule for minimising the mean tardiness that outperforms existing ones. Moreover, there is little discussion on the size of the rules. A simplification technique is proposed in Nguyen et al. [2017b], but their results are still too complex and challenging to interpret. This work aims at overcoming these gaps and finding compact rules that outperform the existing ones in distinct job shops settings and load conditions.

3.3. Methodology

Despite the good performance of machine learning algorithms, such as Neural Networks and ensembles, most generate complex models. They are usually composed of constants and weights that cannot be easily interpreted. On the other hand, the optimal solutions of some optimisation problems consist of simple symbolic expressions (e.g. the Economic Order Quantity, which can be easily found by GP – Lopes et al. [2020]). In scheduling, Xu

et al. [2016] show that under varying and stochastic processing costs, jobs are optimally prioritised by the *Minimum Slack and Longest Remaining Work* policy. We suspect that on more complex problems, if appropriately guided, GP would still find reasonably sized and effective (if not optimal) symbolic expressions.

In order to guide GP, we use problem knowledge in the learning process. Some initiatives with similar purpose are found in previous works, particularly in the study of hyper-heuristics. Burke et al. [2009] describes a methodology for the use of GP with contribution from domain experts. There are six steps, starting from the analysis of existing heuristics for the problem. However, human participation is limited to the initial decisions when the algorithm and function set are defined. There is no feedback from GP solutions to improve the algorithmic decisions. A more iterative process is followed by Lopes et al. [2020]. The authors apply a cooperative co-evolutionary GP algorithm that evolves simultaneously two expressions. Each expression is assigned a specific function set, with some symbol engineering, defined based on the analysis of existing heuristics. The proposed GP finds the EOQ rule for deterministic demand scenarios and meaningful and effective expressions in the stochastic case.

In this paper, we go beyond those studies and formalise the iterative process, as illustrated in Figure 3.1. Analytical reasoning on the problem (left-hand side) provides the necessary insights to specify data, parameters and constraints to boost the computational experiments (on the right). The left-hand side describes the process often found in traditional studies of dispatching rules (Section 3.2.1) while the right-hand side consists of the typical methodology from the GP literature (Section 3.2.2). The aim here is to cross-fertilise these two approaches into a more holistic methodology. This has resulted in the six steps of Figure 3.1. Precedence relations exist between some, but not all, steps – many of them may be conducted in parallel.

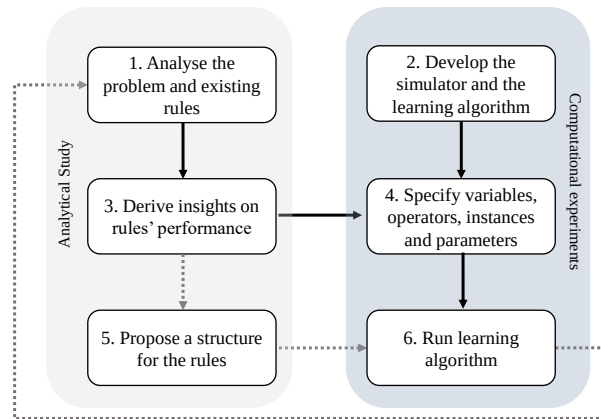


Figure 3.1 – Proposed methodology, an iterative approach where reasoning on the problem and results conduct to inputs for the computational experiments. The algorithm search is combined with human intelligence.

In step 1, the problem is studied. Theoretical foundation and existing rules are analysed.

This study provides conclusions on the behaviour of rules and specific terminals (step 3). There are many possible outcomes from step 3 regarding the best learning configuration, terminals to be considered, GP parameters, or constraints that must be incorporated. They may be used in step 4 to define new terminals and operators (or remove existing ones) for GP, refine the training set, or even tune the algorithm parameters. Depending on the results of step 3, one may fix a specific structure for the evolved rules and guide the learning algorithm through a restricted search space. In step 6, GP runs, generates new rules to be analysed in step 1, and a new iteration begins. Step 2 refers to the developments of the GP and the simulator, which is performed only once.

Next, in Section 3.3.1, we describe the proposed GP algorithm and explain the fitness calculation, which is the average mean tardiness produced by each rule relative to a benchmark. The same metric used for training (in Section 3.4) is applied for the validation of the results, in Section 3.5. The simulation model is detailed in Section 3.3.2, and the design of the experiments are explained in Section 3.3.3. We conclude with the parameters tuning in Section 3.3.4.

3.3.1 Learning algorithm

Genetic Programming (GP) evolves a population of mathematical expressions interpreted or executed to solve a problem [Koza, 1994]. We remark the distinction between GP and the well-known Genetic Algorithms (GA). Despite both are evolutionary and populational approaches, GA evolves solution candidates for a problem. GP goes beyond and can search on the *space of heuristics that can be created* for a problem. The aim is to find the highest fit individual in a search space, where each individual's fitness corresponds to how well the respective expression solves the problem. This approach uses a flexible representation scheme that, combined with a powerful search mechanism, can explore both the structure and parameters of individuals. A generic GP algorithm consists of creating an initial population that is evolved over many generations. At each generation, all individuals from the population are evaluated by the fitness function. The best ones are reproduced and combined for the next generations. The process continues until the algorithm reaches a stopping criterion.

Symbolic expression trees are the most common representation of individuals in GP. They are composed of a function set: *terminals* and *operators* appropriate to the specific problem. The former are the leaf nodes and include parameters of the problem operated by the latter. Figure 3.2 illustrates an individual and the corresponding dispatching rule. In this example, the terminals are 2, *PT* (processing time of current operation), *W* (work in the next queue) and *NPT* (processing time of the next operation). The mathematical operators + and * compose the operators set. Details on the specific implementation of GP in this work are given next.

The pseudo-code of the proposed GP is provided in Algorithm 1. Each individual *A* represents a dispatching rule whose terminals are parameters from the jobs and the shopfloor. The proposed GP runs a generational evolution where a whole population P_g of individuals

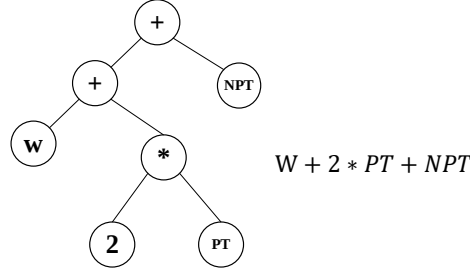


Figure 3.2 – A GP individual tree example and its corresponding dispatching rule

is evaluated and updated at each generation g . The generation of the initial population in line 3 uses the algorithm Ramped Half-and-Half [Koza, 1994], which creates random trees by combining two methods, "full" and "grow", with equal probability. The minimum and maximum depth for trees must be predefined. In trees created by the "full" method, the path length from the root to every terminal is equal to the specified maximum depth. The "grow" method instead generates trees with variable shapes and sizes.

Input: Training set I , benchmark values $\bar{T}_{Bi}$

Result: Dispatching rule A^*

```

1  $A^* \leftarrow \emptyset$ ;
2  $g \leftarrow 1$ ;
3 Initialise population  $P_g$ ;
4 for  $g = 1$  to  $max\_g$  do
5   for  $e = 1$  to 2 do
6     foreach individual  $A \in P_g$  do
7       foreach scenario  $i \in I$  do
8          $\bar{T}_{Ai} \leftarrow \frac{1}{n_e} \cdot \sum_{k=1}^{n_e} simulate(A, i)$ ;
9       end
10       $\tau_A = \prod_{i \in I} \{\bar{T}_{Ai} / \bar{T}_{Bi}\}^{1/|I|}$ ;
11       $fitness(A, g) = \tau_A + |A| \cdot g \cdot \rho$ ;
12    end
13     $P_g \leftarrow \text{Best 50\% individuals of } P_g$ ;
14  end
15   $A^* \leftarrow \arg \min_{A \in P_g \cup A^*} fitness(A, g)$ ;
16   $P_{g+1} \leftarrow \text{Crossover and mutation in } P_g$ ;
17 end

```

Algorithm 1: Genetic Programming algorithm

The fitness computation requires running a job shop simulator, the most expensive procedure in the whole algorithm. Simulation studies of dynamic job shops usually apply several replicas of the same scenario – each with thousands of job completions. In lines 6-

10, the fitness of individual A is calculated. The simulator returns the mean tardiness when applying rule A for scenario i . A reliable estimation of rules performance requires many replications of a job shop simulation. Therefore we adopt a two-step evaluation procedure. The first and lightweight step (when $e = 1$) runs a small number of replications n_1 for the whole population. Then, the 50% worse individuals are discarded (line 11), and P_g keeps only the most promising individuals. In the second step, a larger number of simulation replications n_2 is applied in the remaining population.

An individual evaluation combines the tardiness performance and the tree size. The tardiness of a job is the difference between its completion time c_j and due date d_j if the job is delayed. Given a set of jobs $\mathcal{J}$, we calculate the mean tardiness $\bar{T}$ as follows.

$$\bar{T} = \frac{\sum_{j \in \mathcal{J}} \max(0, c_j - d_j)}{|\mathcal{J}|}$$

The fitness function computes the mean tardiness performance of each individual A on a training data set. The simulation procedure applies rule A in n_e replications of scenario i , and returns the average mean tardiness $\bar{T}_{Ai}$ (line 8 in Algorithm 1). We define set $R = \{\text{RR}, \text{2PTWINQNPT}, \text{CR+SPT}, \text{S/RPT+SPT}, \dots\}$, composed of benchmark rules mentioned in seminal works [Anderson and Nyirenda, 1990, Raghu and Rajendran, 1993, Rajendran and Holthaus, 1999, Holthaus and Rajendran, 1997, 2000]. $\bar{T}_{ri}$ is the mean tardiness found when running the simulator with rule $r \in R$ for a given scenario i . The overall performance of A is estimated by τ_a , the mean relative performance in all training scenarios – the smaller, the better. The relative performance of A for i is calculated as follows.

$$\tau_{Ai} = \max(\bar{T}_{Ai}, \varepsilon) / \max(\min_{r \in R} \bar{T}_{ri}, \varepsilon)$$

Thus, the relative performance of A is the ratio between the mean tardiness found when running this rule and the best mean tardiness found using existing rules. To avoid divisions by zero, we use a small positive number ($\varepsilon = 10^{-4}$) when the value of $\min_{r \in R} \bar{T}_{ri}$ is null – the same applies to $\bar{T}_{Ai}$ for a fair comparison on their values. We calculate τ_A by the geometric mean (see line 9 in Algorithm 1), as it handles ratios more consistently than the arithmetic mean, i.e., given two rules A and B , the geometric mean of $\bar{T}_{Ai}/\bar{T}_{Bi}$ and $\bar{T}_{Bi}/\bar{T}_{Ai}$ is 1, while the arithmetic mean depends on the values of $\bar{T}_{Ai}$ and $\bar{T}_{ri}$. Moreover, as the training set represents very different settings, the ratio $\bar{T}_{Ai}/\bar{T}_{ri}$ may scale to distinct magnitude orders. Thus, the geometric mean smooths large deviations from 1.

In addition to relative performance, the fitness function also considers a strategy to avoid large trees. GP trees tend to grow during the search and result in complex rules – an effect known as bloat. This is a drawback of the symbolic tree representation, as complex trees are computationally inefficient and cannot be easily interpreted. Bloat control strategies are usually applied to avoid this behaviour and pursue compact trees. Our approach penalises the tree size in the fitness function using a bloat control factor ρ . The total penalty depends on the number of nodes $|A|$, the number of generations g and the factor ρ . Considering the twofold objective of getting effective and minor rules, the fitness of individual A in

generation g $f(A, g)$ is calculated as follows (line 10 of Algorithm 1):

$$f(A, g) = \tau_A + |A| \cdot g \cdot \rho$$

In line 11, the population of the next generation is created from the current population P_g . There are two breeding pipelines, crossover and mutation, and each is applied according to a probability. The crossover pipeline selects two individuals as parents to produce two offspring. This operation selects a node at random in each tree and swaps the two sub-trees rooted by those nodes. Mutation replaces a sub-tree with a randomly generated one, generating a new individual. Both pipelines use tournament for parent selection, returning the best of t individuals selected randomly (t is said tournament size). The breeding process picks certain kinds of nodes with different probabilities. For example, one can state to pick non-terminals 90% of the time and terminals 10% of the time. The algorithm runs until it reaches a maximum number of generations max_g and returns the best individual found A^* .

3.3.2 Simulation model

The simulation model follows the literature except for the number of machines $|\mathcal{M}|$. Most researchers consider a fixed number of ten machines [Holthaus and Rajendran, 1997], but we simulate scenarios with distinct values. The number of operations of each job is sampled from a discrete uniform distribution whose limits depend on $|\mathcal{M}|$. Each job has a fixed route, which is randomly generated in advance by assigning operations to machines via a uniform distribution. Re-circulation is allowed but not in consecutive operations. We assume that operations are not preemptable, and setup times are included in operations processing time.

The simulation starts with an empty setting, and a warm-up time is considered for the system to reach a steady state. Following the typical procedure, jobs are numbered in increase sequence upon arrival. Metrics are calculated only for jobs 501 to 2500 (a total of 2000 completed jobs). The arrival of jobs follows a Poisson distribution which rate λ is based on the utilisation level of the shop (u). Given the mean number of operations per job $\bar{n}$ and the mean processing time of operations $\bar{p}$, the arrival rate is calculated as follows:

$$\lambda = \frac{u \cdot |\mathcal{M}|}{\bar{n} \cdot \bar{p}}$$

When they arrive, jobs are released to the shop and are pushed into the first operation machines' queue. Once an operation finishes processing, the job moves to the next machine. Whenever a machine is available, the dispatching rule runs and assigns a score to each job in the queue. The job with the least score is selected for execution. Given the release time r_j of job j , its due date d_j depend on the allowance factor a and is calculated via Total Work Content [Baker, 1984]:

$$d_j = r_j + a \cdot \sum_{i \in O_j} p_{ij}$$

Lawrence and Sewell [1997] studied the performance of dispatching rules in job shops with uncertain processing times. Based on their work, we introduced stochastic operation times in our simulation model. Given the processing time of an operation p_{ij} and a coefficient of variation (cv), the actual processing times are found by a Gamma distribution with mean $E[p_{ij}]$ and standard deviation $\sigma_{ij} = E[p_{ij}] \times cv$. The expected processing times $E[p_{ij}]$ are the same as in the deterministic version. A pseudo-code of the simulator algorithm is found in Section 3.A from Appendix. The next section describes the data sets.

3.3.3 Training, validation and test sets

A scenario is defined by a tuple $\langle |\mathcal{M}|, \bar{p}, a, u, cv \rangle$ whose parameters are described in Table 3.1. As mentioned in Section 3.2, the rules' performance depends on the load conditions, which are defined by the due date allowance (a) and machines utilisation (u). To consider both tight and light conditions, job shop simulation studies typically use two or three values for a and u , in most cases varying from 3 to 8 and 85% to 95%, respectively [Holthaus and Rajendran, 1997, Rajendran and Holthaus, 1999, Dominic et al., 2004, Nguyen et al., 2019]. In this work, a large spectrum of 49 combinations of a and u is considered for the first time. The coefficient of variation cv determines the uncertainty level in operations processing times (usually deterministic, $cv = 0$). The simulation scenarios are grouped into three sets: training, validation and test. The training set is used during GP runs, while the validation set is used to analyse the outputs of GP and compare different versions. We use the test set to evaluate the final selected rules. To ensure simulating distinct jobs at each stage, we use different random seeds for each set.

Table 3.1 – Scenarios used in simulations for training and evaluation.

Parameter	Description	Training	Validation	Test
$ \mathcal{M} $	Number of machines	10	10	10, 20, 50, 100
$\bar{p}$	Mean proc. time	25, 50	25, 50, 100	25, 50, 100, 250, 500
a	Due dates allowance factor	3, 4, 6	2, 3, 4, 6, 8	1.3, 2, 3, 4, 6, 8, 10
u	Shop utilisation level	.85, .90, .97	.80, .85, .90, .95, .97	.70, .80, .85, .90, .95, .97, .99
cv	Uncertainty in proc. times	0	0	0.0, 0.2, 0.4, 0.6, 0.8, 1.0
# Scenarios $\times$ replications		18 $\times$ 12	75 $\times$ 100	897 $\times$ 100

The training set is sufficiently diversified to assess rules in distinct load conditions but does not include extreme cases or stochastic times. This allows us to assess whether the rules can generalise and extrapolate to scenarios never seen before. We simulate ten machines, and frequently used values for $\bar{p}$, a , and u . Therefore, the size of the training set is $1 \cdot 2 \cdot 3 \cdot 3 \cdot 1 = 18$. As described in Algorithm 1, there are two evaluation phases. The simulator runs two replications for each training scenario in the first evaluation phase, and ten in the second ($n_1 = 2, n_2 = 10$).

To represent a more extensive range of scenarios, the validation set is composed of five values for a and u , resulting in 25 settings in total. A full-factorial design of all values is used to compose the validation set (as in the training set). These values enable simulating loose due dates, where the best rule (RR, from [Raghu and Rajendran \[1993\]](#)) produces no tardiness and high congested shops. The shop utilisation also includes the extreme values found in previous works. For validating the rules, we ran 100 simulation replications for each scenario. The number of replications is adequate to ensure that the validation of the results is statistically significant for a confidence level of 95%. The same number of simulation replications is applied to the test set, which is explained next.

In addition to all the values of the validation set, the test set includes extreme values of allowance factor (1.3 and 10) and shop utilisation (0.70 and 0.99). We aim to test the evolved rules in all load conditions found in the literature. Moreover, processing times are sampled with cv at different levels, similarly to [Lawrence and Sewell \[1997\]](#). Previous studies show that the shop size does not affect the relative performance of the rules [[Blackstone et al., 1982](#)]. For that reason, almost all studies on dispatching rules for the JSSP consider ten machines or less in the simulation model. However, that may not be true for some recent and more sophisticated parameters used in certain rules. Some of them, such as WOR, the *workload in the next machines' queue*, from [Nguyen et al. \[2019\]](#), may be influenced by the number of machines in the shop. Therefore, and to attest the generalisability of our rules, we consider larger values of $|\mathcal{M}|$ in our test set. The number of operations is selected from $U(2f, 14f)$, where $f = |\mathcal{M}|/10$. For $|\mathcal{M}| = 10$, the number of operations follows [Holthaus and Rajendran \[2000\]](#) and is selected from $U(2,14)$. Given the large set of parameters, we do not apply a full-factorial design on all their values on the test set. The extreme load conditions (where $a = 1.3$ or 10, $u = 0.7$ or 0.99) are tested only for $|\mathcal{M}| = 10$, $\bar{p} = \{25, 50, 100\}$ and $cv = 0$.

3.3.4 Algorithm parameters

Preliminary experiments were conducted to tune the algorithm parameters. We observed that GP is not very sensitive to small changes in the crossover and mutation probabilities. Thus, we selected two extreme configurations, as shown in Table 3.2, and performed 30 independent runs for each combination of crossover/mutation probability and ρ (the bloat control factor). The table shows the average performance (τ) of the best evolved rules, considering the validation set. We also report the average and maximum rules size, i.e., the number of nodes in the corresponding trees. Notice that, in general, the highest mutation probability leads to better performance, regardless of the bloat control factor. Moreover, for $\rho = 0$, the smaller crossover probability produces a smaller average size. The crossover operation seems to promote unnecessarily long trees.

The GP parameters used in the final experiments are found in Table 3.3. Limiting the tree depth to small values prematurely prunes good branches, so we set the maximum depth of trees to a large number commonly used (17). As the bloat control factor ρ impacts both the size of the rules and their performance, it received special attention. Section 3.4 details the results obtained for different values of ρ . We observed that the larger the population size,

Table 3.2 – Performance and size of rules for distinct values of crossover and mutation rate.

ρ	Crossover = 0.8/ Mutation = 0.2			Crossover = 0.2/Mutation = 0.8		
	Avg. τ	Avg. size	Max. size	Avg. τ	Avg. size	Max. size
10^{-4}	2.46	12.7	27	2.43	13.6	27
10^{-5}	2.31	39.1	101	2.06	40.2	73
0	2.39	197.8	389	2.05	98.6	207

the better the average results obtained, although, for values above 200, the improvement in solution quality is negligible. The number of generations (*max_g*) is limited to 50, as in preliminary runs, no improvement in the fitness function was observed beyond generation 35. As the population is usually too homogeneous after some generations, the tournament size was set to the smaller possible value (2) to promote diversity. We set the values for the probability of selecting a terminal and a non-terminal (0.1 and 0.9, respectively) and the parameters of the Ramped Half-and-Half algorithm (minimum tree depth of 2 and maximum 6). All algorithms were coded in Java, and the GP procedure uses the evolutionary computation library ECJ [Luke, 2017].

Table 3.3 – GP Parameters Setting

Population size	200
Number of generations (<i>max_g</i>)	50
Tournament size (<i>t</i>)	2
Maximum trees depth	17
Crossover probability	0.2
Mutation probability	0.8

3.4. Learning iterations

We performed four iterations on the procedure described in Figure 3.1, which have resulted in four GP variants: *GPLit*, *GPBasic*, *GPImp* and *GPStruct*. The first two variants are based on the set of terminals used by the GP and scheduling literature, respectively. In the last two iterations, we improve the function set and propose a specific structure for the rules based on the analysis of the problem and the results obtained. The following sections describe each variant and the results obtained with them.

We present beforehand the list of terminals of *GPBasic*, *GPImp* and *GPStruct* in Table 3.4. The second column of the table shows the size associated with each terminal, i.e., the number of atomic parameters and operators. CR, for instance, has size three, corresponding to AJ/RPT (“AJ”, “/”, “RPT”) and PT·CR has size five, corresponding to “PT”, “.”, “AJ”, “/” and “RPT”. To calculate WINQ (the work in the next machines’ queue), we use the procedure described in Raghu and Rajendran [1993]. As the terminal set of *GPLit* is too extensive, it is presented in Section 3.B of Appendix.

In the first two iterations, the function sets follow the literature in two different ways. While the first (*GPLit*) reproduces previous GP experiments, the second (*GPBasic*) is based on

Table 3.4 – Terminal sets considered in GPBasic, GPImp and GPStruct

Terminal	Size	Description	GPBasic	GPImp	GPStruct	
					$Slack > 0$	$Slack \leq 0$
PT	1	Processing time of current operation	x	x	x	x
WINQ	1	Work in the next queue	x	x	x	x
NPT	1	Processing time of next operation	x			
U	1	Shop utilisation	x			
PT·S/RPT	5	Operation due date (using slack)	x			
PT·CR	5	Operation due date (using allowance)	x			
S ⁺	1	Slack if positive		x	x	
AJ ⁺	1	Allowance if positive		x	x	
S ⁺ /RPT	3	Relative slack if positive		x	x	
CR ⁺	3	Critical ratio if positive		x	x	
PT·S ⁺ /RPT	5	Operation due date (using slack) if positive		x	x	
PT·CR ⁺	5	Operation due date (using allowance) if positive		x	x	
RPT	1	Remaining processing time		x	x	

the classical dispatching rules. Next, we detail both sets and the average results obtained with them.

We provide two terminal sets for GPLit, namely GPLit_a from [Nguyen et al. \[2013\]](#) and GPLit_b based on [Nguyen et al. \[2019\]](#). In both works, GP finds well-performing rules for specific job shop settings. Ephemeral random constants are not included in our experiments to promote interpretability and generalisability. In the operators set, we use the basic mathematical operations (+, -, ·, /), *max* and *min*, as in the works mentioned. GPLit_b also contains an *If* operator, as in [Nguyen et al. \[2019\]](#). In all experiments the protected division applies, i.e., $a/b = 1$ if $b = 0$.

In the second iteration, we aim at providing GP with a smaller set of selected terminals and operators. The earlier studies on dynamic job shop scheduling show that look-ahead parameters should be incorporated into the SPT rule. Moreover, due date information (slack or allowance) must be considered to prioritise the most urgent operations. These parameters are found in the best existing rules, such as RR, 2PT+WINQ+NPT, CR+SPT and S/RPT+SPT. Thus, the terminal set of GPBasic contains only the parameters present in those rules, including the processing time of current operation (PT), due date information (S/RPT and CR), next operation processing time (NPT) and work in the next queue (WINQ). The parameters $exp(u)$ and $exp(-u)$ from RR are omitted since there is no justification for using the exponential function. GPBasic and the following two variants use only the basic four mathematical operations (+, -, · and /) in the operators set.

We describe the results of these two variants in the following. The rules' performance is calculated using the metric for training described in Section 3.3.1. We set four distinct values for the bloat control factor ($\rho = \{10^{-3}, 10^{-4}, 10^{-5}, 0\}$) to explore the trade-off between rules size and performance. Following [Nguyen et al. \[2019\]](#), 30 independent executions were performed for each GP variant and value of ρ . When $\rho = 10^{-3}$, both versions of GPLit get stuck in tiny rules with poor performance ($\tau \sim 3$). Similarly, GPBasic produces

PT+WINQ, which is ineffective for mean tardiness ($\tau = 2.36$) as it does not consider due dates. The results of the other values of ρ are presented in Figure 3.3. We also present in red dashed lines the average performance of the best existing rules, which are described in Anderson and Nyirenda [1990], Raghu and Rajendran [1993], Holthaus and Rajendran [2000] and Xiong et al. [2017].

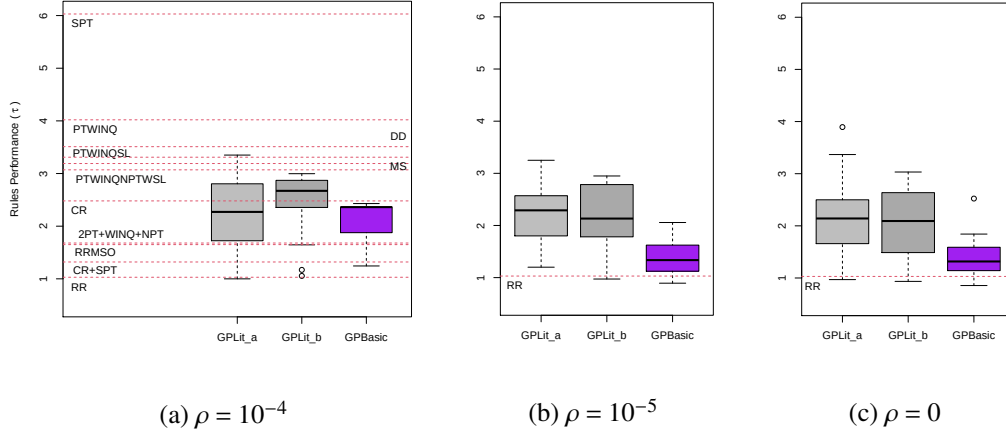


Figure 3.3 – Performance of rules evolved with GPLit_a, GPLit_b and GPBasic

The overall performances of GPLit_a and GPLit_b are similar. In 90 runs, GPLit_a resulted in four good rules ($\tau < 1$) and GPLit_b in only two. Between both versions, GPLit_b found the best rule ($\tau = 0.93$) and its average performance is slightly better, 2.24 against 2.34 from GPLit_a. The best rules are found when no bloat control is applied or using light bloat control. Regarding GPBasic, when $\rho = 10^{-4}$, the algorithm generates PT+WINQ in 17 out of 30 runs, resulting in poor average performance. However, for $\rho > 10^{-4}$, this version performs better than GPLit_a and GPLit_b. Nine rules are better than RR, and the best performance is 0.86. From these results, we conclude that GP could find a better combination of the parameters used by manually devised rules. Moreover, we suspect that the function sets of GPLit_a and GPLit_b are probably unnecessarily large. Consequently, there is no dimensional consistency in the rules devised with them – some terminals refer to the number of operations, others to the remaining work or even the number of machines yet to be visited. Combining these terminals is thus an additional challenge for GP. It is also interesting to observe that terminal U, used to adapt rules to the shop utilisation in the rule RR, is not present in the best evolved rules with GPBasic.

Figure 3.4 presents the size of all rules evolved in these two first iterations and the size of RR for reference. We observe that our bloat control approach is effective to provide smaller rules, although the average performance is similar for $\rho = 10^{-5}$ and $\rho = 0$ (see Figure 3.3). The good performance of GPBasic was obtained at the cost of considerably longer rules. Indeed, there is no significant difference in the size obtained with the different GP variants. These first experiments demonstrate that GP can produce rules that perform

well on average, but there is room for improvement. The well-performing rules are too long and cannot be easily interpreted. In the next section, we detail how the terminal set of GPBasic is improved in the last two iterations.

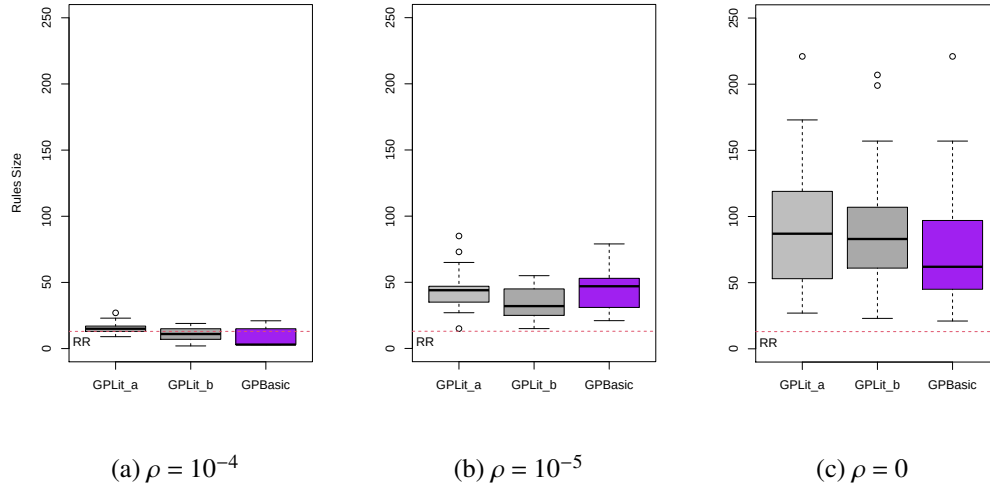


Figure 3.4 – Size of rules evolved with GPLit_a, GPLit_b and GPBasic

3.4.1 Improving the terminal set

Highly loaded settings and close due dates lead to negative values for allowance and slack, which introduces bias in the rules' behaviour [Haupt, 1989, Anderson and Nyirenda, 1990]. The term CR·PT prioritise operations with long instead of short processing times if CR is negative. Moreover, these parameters are not relevant in tight load conditions, even for optimising due date measures [Ramasesh, 1990]. Thus, in GPImp, the slack and allowance of jobs are considered only when positive; we use 0 otherwise. S^+ substitutes terminal S (slack) (slack if positive), and CR becomes CR^+ (critical ratio if positive). Hence, GPImp uses $PT \cdot S^+ / RPT$ instead of $PT \cdot S / RPT$ and $PT \cdot CR^+$ instead of $PT \cdot CR$ as well. We also decompose some terminals into their atomic parameters to verify if GP finds better combinations. For instance, in addition to $PT \cdot S^+ / RPT$, we also include PT, S^+ , RPT and S^+ / RPT . Terminals NPT (next operation processing time) and U (shop utilisation) are discarded, as they do not appear in the best rules evolved with GPBasic.

The last learning iteration designs a preliminary structure for the rules. If job slack is negative, it is not possible to meet the due date. From the good performance of the rules generated with GPImp, we conclude that it is better not to consider any due date information in such cases. Thus, in GPStruct, two expressions are used to calculate the score of each operation, depending on job slack. The idea is similar to CR+SPT and S/RPT+SPT, which apply operations processing time if the due date is too close or has passed. GPStruct evolves two separated sub-trees, A and B, fed with distinct terminal sets, as specified in

Figure 3.5. While A uses the same set of GPImp, B does not consider any due date-related terminal (slack or allowance). Although breeding is performed independently on each sub-tree, the fitness calculation uses the whole tree. The last two columns of Table 3.4 indicate the terminals of GPStruct in both cases.

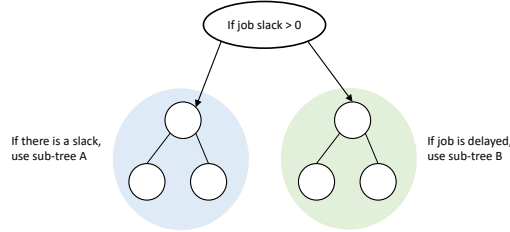


Figure 3.5 – Fixed structure proposed for rules evolved

Figure 3.6 presents the rules' performance by GP variant – GPLit_a and GPLit_b are not included given their poor performance. GPImp provides much better rules than GPBasic, and is in turn outperformed by GPStruct. GPStruct provides the best performing rule ($\tau = 0.80$) and the best average performance. Moreover, GPStruct consistently generates good rules, regardless the value of $\rho - \tau \leq 0.86$ for 79% of the rules evolved. For instance, the rule *If (slack > 0) PT·CR⁺ + 2WINQ Else 2PT + WINQ* ($\tau = 0.84$) was generated in 56% of the runs with $\rho = 10^{-4}$. It is interesting that when PT·CR⁺ is not used, GP finds the same expression from the combination of the nodes {PT, “.”, CR⁺}.

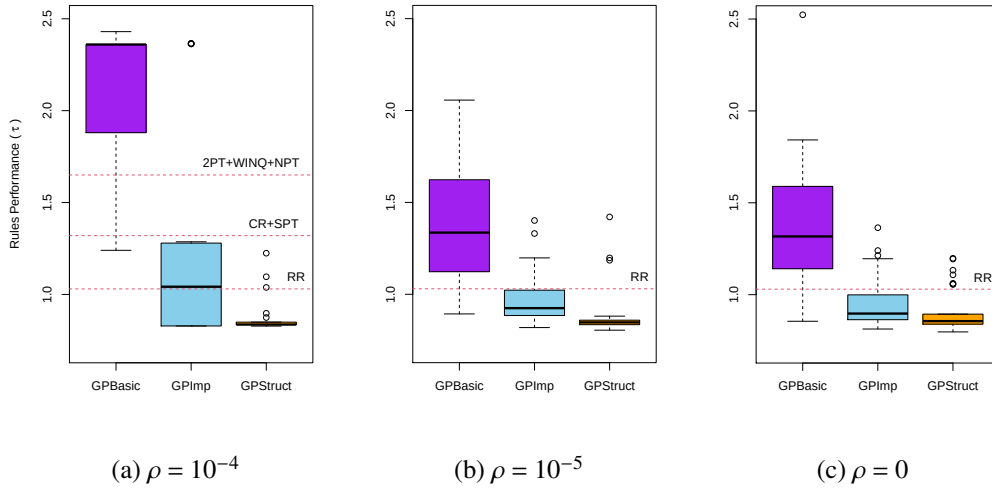


Figure 3.6 – Performance of rules evolved with different GP configurations

Table 3.5 summarises the average performance and the best rule of GPLit (joint GPLit_a and GPLit_b), GPBasic, GPImp and GPStruct. The improvement in performance provided by each iteration is clear, regardless of the bloat control factor (ρ). On average, the rules

evolved with GPBasic present a performance of 1.61 while GPImp and GPStruct find rules with 1.05 and 0.89, respectively. We also report the average size of the rules. As the rules evolved with GPStruct combine two expressions, they are slightly greater than the others.

Table 3.5 – Performance and size of rules evolved using preliminary versions of GP.

ρ	GPLit			GPBasic			GPImp			GPStruct		
	τ	τ_{min}	Size	τ	τ_{min}	Size	τ	τ_{min}	Size	τ	τ_{min}	Size
10^{-4}	2.40	1.00	13.6	2.07	1.24	8.3	1.23	0.83	12.0	0.87	0.83	15.7
10^{-5}	2.20	0.97	39.0	1.39	0.82	45.7	0.97	0.81	47.0	0.89	0.80	34.8
0	2.11	0.93	91.7	1.36	0.86	76.3	0.95	0.81	76.3	0.91	0.80	114.3
Avg.	2.24	0.97	48.1	1.61	1.00	43.4	1.05	0.81	45.1	0.89	0.81	54.9

3.5. Evolved Rules

In this section, we exhibit and analyse the best evolved rules considering a two-criteria objective, finding *effective* and *interpretable* rules. These objectives are measured by the rules' relative performance in the test set and their size, respectively. For presenting the detailed results in the test set, we select non-dominated rules in both criteria. Next, we explain this selection.

3.5.1 Rules selection

The chart in Figure 3.7 presents the performance on the validation set (*y-axis*) and size (*x-axis*) of all evolved rules with $\tau \leq 1$. Both axes are on a logarithmic scale. Observe that the rules generated with GPLit (GPLit_a and GPLit_b) are all dominated. The chart highlights three non-dominated rules (R1, R2, R3), evolved with GPImp and GPStruct. Among them, R1 is the shortest and least performing (0.83). Rule R3 is the largest and best performing (0.80), although R2 presents a very similar performance (of around 0.80).

Table 3.6 presents the mathematical expression of each non-dominated rule. R2 was simplified by applying mathematical operations, e.g., $a + a + a$ is converted to $3a$, and a/a is 1. Column "Size" shows the size of the expression after these modifications and the original size in brackets. R3 is too large and cannot be easily interpreted. Thus we do not present it. We detail R1 and R2 next.

R1 has a structure similar to RR from Raghu and Rajendran [1993]. Both rules are composed of three terms, PT (operations processing time), PT·S/RPT (operations due date) and WINQ (work in the next machines' queue). The difference is in the balance between PT and PT·S/RPT. The rule RR use weights $exp(u)$ and $exp(-u)$ for PT and PT·S/RPT, respectively, adapting the influence of those terms to the shop utilisation u . R1, instead, omits the term PT·S/RPT for delayed jobs and applies weight 2 for PT (in RR, the weight of PT varies from 2.0 to 2.7). For negative or null slacks, R1 behaves just like $2PT + WINQ$, which is an expression frequently found by GPStruct for the branch of non-positive slacks.

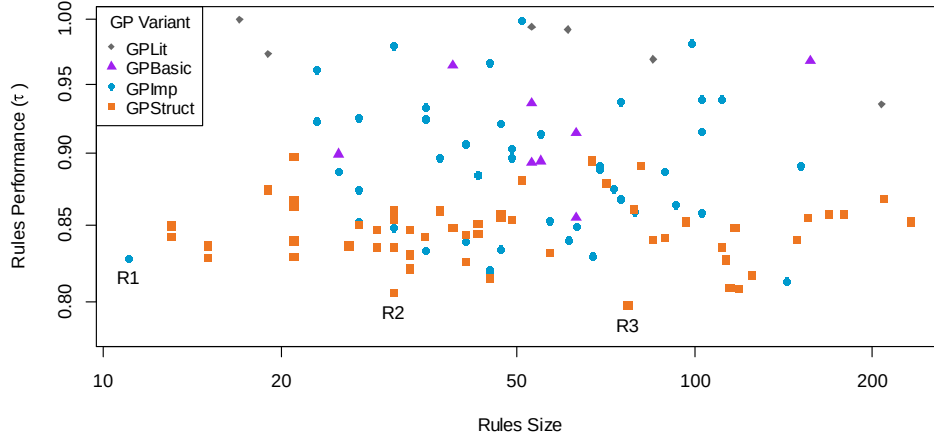


Figure 3.7 – Size and relative performance of the evolved rules.

Table 3.6 – Evolved rules

Rule	Mathematical expression	Size	τ	Algorithm
R1	$2PT + PT \cdot S^+ / RPT + WINQ$	11	0.828	GPImp
R2	<i>If (Slack > 0) Then</i>	19 (31)	0.805	GPStruct
	$PT + PT \cdot CR^+ + WINQ$			
	<i>Else</i>			
	$3PT + WINQ - (WINQ/PT)$			
R3	(large expression)	77	0.797	GPStruct

The rule R2 is equivalent to R1 if the job slack is positive. The terminal CR corresponds to AJ/RPT . As slack (S) is $AJ - RPT$, $CR = (S + RPT)/RPT = 1 + S/RPT$. Thus, $PT \cdot (2 + S/RPT)$, from R1, corresponds to $PT \cdot (1 + CR)$, from R2. The behaviour of both rules is different only when the slack is negative or zero. In that case, R2 assigns weight 3 for PT and subtract $WINQ/PT$, favouring jobs with short processing times. The balance between operations processing time (PT) and work in the next queue (WINQ) in R2 is 3:1, similar to $2PT + WINQ + NPT$ – the best existing rule for extreme tight load settings. Except when PT is considerably small, the term $WINQ/PT$ is much smaller than the other terms in the expression, with almost no influence on the resulting score. Thus, R2 emphasise the use of the shortest processing time for jobs in congested shops.

3.5.2 Performance under deterministic processing times

This section details the results obtained with R1 and R2 in our test set for deterministic processing times. We recall that the performance τ_A is the mean relative performance of rule A against the best existing rules (described in Section 3.2.1). For each scenario i , we calculate $\bar{T}_{Ai}/\bar{T}_{ri}$, where r is the best rule for i among all of the benchmarked rules. Thus, if $\tau_A < 1$, rule A outperforms, on average, all existing rules combined.

Table 3.7 presents the value of τ for each combination of shop utilisation and allowance factor. In these experiments we use the most common values for processing times ($\bar{p} = \{25, 50, 100\}$) and number of machines ($|\mathcal{M}| = 10$). We performed statistical tests to compare the result of each evolved rule to the best existing rules. We obtained significant results at the 0.05 level for almost all settings – the exceptions are marked with †. In these cells, we consider a tie between the evolved rule and the benchmark as in cells with $\tau = 1.00$.

The cells in the top right corner correspond to light load settings – large allowance factor and low shop utilisation. In these settings, the (evolved and benchmark) rules usually produce no tardiness, resulting in $\tau = 1.00$ (tie). Considering all settings, R1 and R2 perform better than the benchmark. R1 outperforms existing rules in 33 out of 49 settings and is worse in 7. R2 has better average performance than R1 and works better in more settings, 38 out of 49. The effectiveness of both rules is due to a better balance between operations processing time and due date, especially when job slack is negative. However, the evolved rules deteriorate under higher utilisation levels. Observe that $\tau > 1$ more frequently in cells where $u \geq 0.97$. As the average queue size increases with utilisation level, the value of WINQ becomes larger, but the weight of PT does not follow. We conclude that it could be desirable to train specific rules for these tight load settings.

Table 3.7 – Performance under deterministic processing times and distinct values of shop utilisation and allowance factor.

$u \backslash a$	R1								R2							
	1.3	2	3	4	6	8	10	Avg	1.3	2	3	4	6	8	10	Avg
0.70	0.98	0.94	0.78	0.53	1.00	1.00	1.00	0.87	0.98	0.94	0.78	0.56	1.00	0.79	1.00	0.85
0.80	0.98	0.94	0.87	0.73	0.42	1.00	1.00	0.82	0.99	0.94	0.88	0.73	0.41	1.00	0.46	0.73
0.85	0.99	0.97	0.85	0.82	0.56	0.30	1.00	0.73	0.99	0.96	0.83	0.81	0.57	0.26	1.00	0.72
0.90	1.00	0.99	0.90	0.86	0.90	0.69	0.07	0.62	0.98	0.98	0.89	0.85	0.86	0.62	0.25	0.72
0.95	0.98	0.98	0.96	0.86	0.91	0.95	0.81	0.92	0.98	0.97	0.93	0.85	0.90	0.89	0.81	0.90
0.97	0.99 †	0.98	0.98	0.96	1.08	0.96	1.07	1.00	0.98	0.99 †	0.97	0.93	0.97	0.89	1.06	0.97
0.99	1.02	0.98	1.01 †	1.10	1.21	1.23	1.20	1.10	0.99 †	0.98	1.00	1.00	1.22	1.18	1.10	1.06
Avg	0.99	0.97	0.91	0.82	0.82	0.81	0.69	0.85	0.99	0.96	0.89	0.81	0.80	0.74	0.73	0.84

Table 3.8 extends the analysis to a larger number of machines and different processing time distributions. Each cell shows the average result for $u = \{0.8, 0.85, 0.9, 0.95, 0.97\}$, and all results are significant at the 0.05 level. As the number of machines increases, R1 and R2 tend to perform better than the benchmark rules (the value of τ decreases). The exception is the settings with very close due dates ($a = 2$), where both rules deteriorate performance. We observed that as $|\mathcal{M}|$ increases, the average queue size also increases, from 4.5 (with ten machines) to 7 (with 100 machines). This system behaviour affects the balance between terminals WINQ and PT. This is likely the reason for the worse performance of the rules in such settings. Another exception is $a = 8$ and $|\mathcal{M}| = 100$, where RR (the benchmark rule), R1, and R2 produce zero tardiness, resulting in $\tau = 1.00$. The processing time distribution has little effect on the performance of the rules.

Table 3.8 – Performance for different number of machines and processing time distribution.

$\bar{p}$	$ M \backslash a$	R1						R2					
		2	3	4	6	8	Avg	2	3	4	6	8	Avg
25 - 100	10	0.97	0.91	0.84	0.73	0.72	0.83	0.97	0.90	0.83	0.71	0.67	0.81
	20	1.00	0.95	0.79	0.50	0.38	0.68	1.00	0.93	0.77	0.49	0.36	0.66
	50	1.06	0.88	0.33	0.41	0.40	0.55	1.04	0.88	0.36	0.45	0.40	0.57
	100	1.08	0.59	0.17	0.18	1.00	0.45	1.06	0.64	0.33	0.33	1.00	0.59
250	10	0.96	0.91	0.84	0.73	0.76	0.84	0.96	0.90	0.82	0.73	0.70	0.82
500	10	0.96	0.92	0.85	0.72	0.72	0.83	0.96	0.89	0.83	0.71	0.67	0.80

3.5.3 Performance under stochastic processing times

To assess the effectiveness of the evolved rules under uncertain processing times, we compare the mean tardiness generated by each rule for different cv values. As explained in Section 3.3.3, cv is the coefficient of variation for sampling the actual processing times, (larger values for cv correspond to higher uncertainty levels; when $cv = 0$, processing times are deterministic). In Table 3.9, we report the average value of τ under stochastic processing times. Although the relative performance of both rules deteriorates, they are still better than the benchmark – notice that the training set is composed of scenarios with deterministic processing times. The average performance of R1 and R2 for each setting is presented in Section 3.C of Appendix.

Table 3.9 – Overall performance under stochastic processing times

Rule	cv						Avg
	0.0	0.2	0.4	0.6	0.8	1.0	
R1	0.83	0.91	0.92	0.90	0.92	0.91	0.90
R2	0.81	0.85	0.86	0.87	0.88	0.89	0.86

Our analysis finishes with the effect of uncertainty in the mean tardiness for distinct load conditions. The charts in Figure 3.8 present the mean tardiness $\bar{T}$ obtained for rules RR, 2PT+WINQ+NPT, R1 and R2, considering all uncertainty levels. Notice that now we report absolute values (unlike the previous sections). These results refer to scenarios where $\bar{p} = 50$ and $|M| = 10$. In general, all dispatching rules produce higher tardiness as cv increases. Comparing the existing rules, RR and 2PT+WINQ+NPT are good in light and tight load conditions, respectively, but none of them works well in both cases. In medium conditions (Figure 3.8b), RR performs well if the cv is low; as uncertainty increases its performance degrades fast. R2 consistently produces lower tardiness than both, regardless of the setting, and R1 is good in medium conditions.

3.6. Conclusions and future work

This work focuses on the use of problem expertise to leverage the computational power of machine learning. Genetic Programming is applied as the learning approach to design

dispatching rules for the Dynamic Job Shop Scheduling Problem. The objective is to find rules that minimise the mean tardiness, one of the most challenging performance criteria in dynamic job shops. We propose a guided empirical learning procedure, where reasoning on the problem derives insights to guide the algorithmic search iteratively. Modifications are applied to the terminal set or rules structure at each iteration, and new results are generated and analysed. Unlike traditional approaches, GP is also provided with problem expertise from the existing literature.

We found that GP can combine parameters from existing rules and generate better ones if fed with the appropriate terminal set. There is a vast number of combinations of terminals and operators. Thus, GP cannot successfully explore the search space in feasible computational time. We found that working with a small terminal set of selected parameters enables evolving more effective dispatching rules than large terminal sets. Moreover, we conclude that the overall effectiveness of the evolved rules is even better when providing a fixed good structure.

All rules generated were evaluated in two criteria, performance and size, and the non-dominated were selected for testing. Besides having an interpretable structure, these rules present an overall improvement of 19% compared to the best possible combination of the state-of-the-art rules. This superiority is achieved in almost all settings, from extreme light load conditions (where all jobs meet the due date) to congested ones. Among all tested settings, only for very tight conditions, where shop utilisation is around 99%, the average performance of the new rules does not outperform the literature. Furthermore, the good behaviour of the rules is confirmed in shops with a different number of machines and under stochastic processing times.

As future work, our guided empirical learning approach should be tested in different problems, such as the Flexible Job Shop Scheduling or Parallel Machine Scheduling. Also, multi-objective optimisation becomes a relevant research topic because there are several performance criteria in all these scheduling problems.

Acknowledgments

This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme, and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project SAICTPAC/0034/2015- POCI-01-0145-FEDER-016418.

Bibliography

Amin, G.R., El-Bouri, A., 2018. A minimax linear programming model for dispatching rule selection. *Computers & Industrial Engineering* 121, 27 – 35. doi:<https://doi.org/10.1016/j.cie.2018.05.021>.

- Anderson, E.J., Nyirenda, J.C., 1990. Two new rules to minimize tardiness in a job shop. *International Journal of Production Research* 28, 2277–2292. doi:[10.1080/00207549008942866](https://doi.org/10.1080/00207549008942866).
- Aytug, H., Lawley, M.A., McKay, K., Mohan, S., Uzsoy, R., 2005. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research* 161, 86 – 110. doi:<https://doi.org/10.1016/j.ejor.2003.08.027>. iEPM: Focus on Scheduling.
- Baker, K.R., 1984. Sequencing rules and due-date assignments in a job shop. *Management Science* 30, 1093–1104. doi:[10.1287/mnsc.30.9.1093](https://doi.org/10.1287/mnsc.30.9.1093).
- Blackstone, J.H., Phillips, D.T., Hogg, G.L., 1982. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research* 20, 27–45. doi:[10.1080/00207548208947745](https://doi.org/10.1080/00207548208947745).
- Branke, J., Hildebrandt, T., Scholz-Reiter, B., 2015. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. *Evolutionary Computation* 23, 249–277. doi:https://doi.org/10.1162/evco_a_00131.
- Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M., 2016. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation* 20, 110–124.
- Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R., 2009. Exploring Hyper-heuristic Methodologies with Genetic Programming. Springer Berlin Heidelberg, Berlin, Heidelberg. chapter 1. pp. 177–201. doi:[10.1007/978-3-642-01799-5_6](https://doi.org/10.1007/978-3-642-01799-5_6).
- Chen, B., Matis, T.I., 2013. A flexible dispatching rule for minimizing tardiness in job shop scheduling. *International Journal of Production Economics* 141, 360 – 365. doi:<https://doi.org/10.1016/j.ijpe.2012.08.019>. meta-heuristics for manufacturing scheduling and logistics problems.
- Dominic, P.D.D., Kaliyamoorthy, S., Kumar, M.S., 2004. Efficient dispatching rules for dynamic job shop scheduling. *The International Journal of Advanced Manufacturing Technology* 24, 70–75. doi:[10.1007/s00170-002-1534-5](https://doi.org/10.1007/s00170-002-1534-5).
- Drake, J.H., Kheiri, A., Özcan, E., Burke, E.K., 2020. Recent advances in selection hyper-heuristics. *European Journal of Operational Research* 285, 405–428. URL: <https://www.sciencedirect.com/science/article/pii/S0377221719306526>, doi:<https://doi.org/10.1016/j.ejor.2019.07.073>.
- Fan, H., Xiong, H., Goh, M., 2021. Genetic programming-based hyper-heuristic approach for solving dynamic job shop scheduling problem with extended technical precedence constraints. *Computers & Operations Research* 134, 105401. URL: <https://www.sciencedirect.com/science/article/pii/S0305054821001672>, doi:<https://doi.org/10.1016/j.cor.2021.105401>.

- Ferreira, C., Figueira, G., Amorim, P., 2020. Optimizing dispatching rules for stochastic job shop scheduling, in: Madureira, A.M., Abraham, A., Gandhi, N., Varela, M.L. (Eds.), *Hybrid Intelligent Systems*, Springer International Publishing, Cham. pp. 321–330.
- Hart, E., Sim, K., 2016. A hyper-heuristic ensemble method for static job-shop scheduling. *Evolutionary Computation* 24, 609–635.
- Haupt, R., 1989. A survey of priority rule-based scheduling. *Operations Research - Spektrum* 11, 3–16. doi:[10.1007/BF01721162](https://doi.org/10.1007/BF01721162).
- Hildebrandt, T., Branke, J., 2015. On using surrogates with genetic programming. *Evol. Comput.* 23, 343–367. URL: https://doi.org/10.1162/EVCO_a_00133, doi:[10.1162/EVCO_a_00133](https://doi.org/10.1162/EVCO_a_00133).
- Hildebrandt, T., Heger, J., Scholz-Reiter, B., 2010. Towards improved dispatching rules for complex shop floor scenarios: A genetic programming approach, in: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, NY, USA. pp. 257–264. doi:[10.1145/1830483.1830530](https://doi.org/10.1145/1830483.1830530).
- Holthaus, O., Rajendran, C., 1997. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics* 48, 87 – 105. doi:[https://doi.org/10.1016/S0925-5273\(96\)00068-0](https://doi.org/10.1016/S0925-5273(96)00068-0).
- Holthaus, O., Rajendran, C., 2000. Efficient jobshop dispatching rules: Further developments. *Production Planning & Control* 11, 171–178. doi:[10.1080/095372800232379](https://doi.org/10.1080/095372800232379).
- Jain, S., Foley, W., 2016. Dispatching strategies for managing uncertainties in automated manufacturing systems. *European Journal of Operational Research* 248, 328 – 341. doi:<https://doi.org/10.1016/j.ejor.2015.06.060>.
- Jun, S., Lee, S., 2021. Learning dispatching rules for single machine scheduling with dynamic arrivals based on decision trees and feature construction. *International Journal of Production Research* 59, 2838–2856. doi:[10.1080/00207543.2020.1741716](https://doi.org/10.1080/00207543.2020.1741716).
- Jun, S., Lee, S., Chun, H., 2019. Learning dispatching rules using random forest in flexible job shop scheduling problems. *International Journal of Production Research* 57, 3290–3310. doi:[10.1080/00207543.2019.1581954](https://doi.org/10.1080/00207543.2019.1581954).
- Jung, K.S., Pinedo, M., Sriskandarajah, C., Tiwari, V., 2019. Scheduling elective surgeries with emergency patients at shared operating rooms. *Production and Operations Management* 28, 1407–1430. doi:<https://doi.org/10.1111/poms.12993>.
- Karunakaran, D., Yi Mei, Gang Chen, Mengjie Zhang, 2017. Evolving dispatching rules for dynamic job shop scheduling with uncertain processing times, in: *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 364–371.
- Koza, J.R., 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing* 4, 87–112.

- Kronberger, G.K., 2010. Symbolic Regression for Knowledge Discovery - Bloat, Overfitting, and Variable Interaction Networks. Ph.D. thesis. Johannes Kepler University. Linz, Austria.
- Lawrence, S.R., Sewell, E.C., 1997. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. *Journal of Operations Management* 15, 71 – 82. doi:[https://doi.org/10.1016/S0272-6963\(96\)00090-3](https://doi.org/10.1016/S0272-6963(96)00090-3).
- Lopes, R.L., Figueira, G., Amorim, P., Almada-Lobo, B., 2020. Cooperative coevolution of expressions for (r,q) inventory management policies using genetic programming. *International Journal of Production Research* 58, 509–525. doi:[10.1080/00207543.2019.1597293](https://doi.org/10.1080/00207543.2019.1597293).
- Luke, S., 2017. Ecj then and now, in: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Association for Computing Machinery, New York, NY, USA. p. 1223–1230. doi:[10.1145/3067695.3082467](https://doi.org/10.1145/3067695.3082467).
- Luo, S., 2020. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing* 91, 106208. doi:<https://doi.org/10.1016/j.asoc.2020.106208>.
- Luo, S., Zhang, L., Fan, Y., 2021. Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning. *Computers & Industrial Engineering* 159, 107489. URL: <https://www.sciencedirect.com/science/article/pii/S0360835221003934>, doi:<https://doi.org/10.1016/j.cie.2021.107489>.
- Mei, Y., Zhang, M., Nyugen, S., 2016. Feature selection in evolving job shop dispatching rules with genetic programming, in: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, Association for Computing Machinery, New York, NY, USA. p. 365–372. doi:[10.1145/2908812.2908822](https://doi.org/10.1145/2908812.2908822).
- Nguyen, S., Mei, Y., Xue, B., Zhang, M., 2019. A hybrid genetic programming algorithm for automated design of dispatching rules. *Evolutionary Computation* 27, 467–496. doi:[10.1162/evco_a_00230](https://doi.org/10.1162/evco_a_00230).
- Nguyen, S., Mei, Y., Zhang, M., 2017a. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems* 3, 41–66.
- Nguyen, S., Zhang, M., Johnston, M., Tan, K., 2013. Learning iterative dispatching rules for job shop scheduling with genetic programming. *The International Journal of Advanced Manufacturing Technology* 67. doi:[10.1007/s00170-013-4756-9](https://doi.org/10.1007/s00170-013-4756-9).
- Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2013. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation* 17, 621–639.
- Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2013. *Dynamic Multi-objective Job Shop Scheduling: A Genetic Programming Approach*. Springer Berlin Heidelberg,

- Berlin, Heidelberg. chapter 10. pp. 251–282. doi:https://doi.org/10.1007/978-3-642-39304-4_10.
- Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2014. Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation* 18, 193–208.
- Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2014. Selection schemes in surrogate-assisted genetic programming for job shop scheduling, in: Dick, G., Browne, W.N., Whigham, P., Zhang, M., Bui, L.T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K.C., Tang, K. (Eds.), *Simulated Evolution and Learning*, Springer International Publishing, Cham. pp. 656–667.
- Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2015. Automatic programming via iterated local search for dynamic job shop scheduling. *IEEE Transactions on Cybernetics* 45, 1–14.
- Nguyen, S., Zhang, M., Tan, K.C., 2017b. Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. *IEEE Transactions on Cybernetics* 47, 2951–2965.
- Nie, L., Gao, L., Li, P., Li, X., 2013. A gep-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. *Journal of Intelligent Manufacturing* 24, 763–774. doi:<https://doi.org/10.1007/s10845-012-0626-9>.
- Olsen, T.L., Tomlin, B., 2020. Industry 4.0: Opportunities and challenges for operations management. *Manufacturing & Service Operations Management* 22, 113–122. doi:[10.1287/msom.2019.0796](https://doi.org/10.1287/msom.2019.0796).
- Ouelhadj, D., Petrovic, S., 2008. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling* 12, 417. doi:[10.1007/s10951-008-0090-8](https://doi.org/10.1007/s10951-008-0090-8).
- Parente, M., Figueira, G., Amorim, P., Marques, A., 2020. Production scheduling in the context of industry 4.0: review and trends. *International Journal of Production Research* 0, 1–31. doi:[10.1080/00207543.2020.1718794](https://doi.org/10.1080/00207543.2020.1718794).
- Park, J., Mei, Y., Nguyen, S., Chen, G., Zhang, M., 2018. An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. *Applied Soft Computing* 63, 72 – 86. doi:<https://doi.org/10.1016/j.asoc.2017.11.020>.
- Pergher, I., Frej, E.A., Roselli, L.R.P., [de Almeida], A.T., 2020. Integrating simulation and fitradeoff method for scheduling rules selection in job-shop production systems. *International Journal of Production Economics* 227, 107669. doi:<https://doi.org/10.1016/j.ijpe.2020.107669>.

- Pickardt, C., Hildebrandt, T., Branke, J., Heger, J., Scholz-Reiter, B., 2012. Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. *International Journal of Production Economics* 145. doi:[10.1016/j.ijpe.2012.10.016](https://doi.org/10.1016/j.ijpe.2012.10.016).
- Pinedo, M.L., 2008. *Scheduling - Theory, Algorithms and Systems*. 3rd ed., Springer.
- Raghu, T., Rajendran, C., 1993. An efficient dynamic dispatching rule for scheduling in a job shop. *International Journal of Production Economics* 32, 301 – 313. doi:[https://doi.org/10.1016/0925-5273\(93\)90044-L](https://doi.org/10.1016/0925-5273(93)90044-L).
- Rajendran, C., Holthaus, O., 1999. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research* 116, 156 – 170. doi:[https://doi.org/10.1016/S0377-2217\(98\)00023-X](https://doi.org/10.1016/S0377-2217(98)00023-X).
- Ramasesh, R., 1990. Dynamic job shop scheduling: A survey of simulation research. *Omega* 18, 43 – 57. doi:[https://doi.org/10.1016/0305-0483\(90\)90017-4](https://doi.org/10.1016/0305-0483(90)90017-4).
- Romero-Silva, R., Shaaban, S., Marsillac, E., Hurtado, M., 2018. Exploiting the characteristics of serial queues to reduce the mean and variance of flow time using combined priority rules. *International Journal of Production Economics* 196, 211 – 225. doi:<https://doi.org/10.1016/j.ijpe.2017.11.023>.
- Sabuncuoglu, I., Bayız, M., 2000. Analysis of reactive scheduling problems in a job shop environment. *European Journal of Operational Research* 126, 567 – 586. doi:[https://doi.org/10.1016/S0377-2217\(99\)00311-2](https://doi.org/10.1016/S0377-2217(99)00311-2).
- Shahzad, A., Mebarki, N., 2012. Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem. *Engineering Applications of Artificial Intelligence* 25, 1173–1181. URL: <https://www.sciencedirect.com/science/article/pii/S0952197612000899>, doi:<https://doi.org/10.1016/j.engappai.2012.04.001>.
- Su, H., Pinedo, M., Wan, G., 2017. Parallel machine scheduling with eligibility constraints: A composite dispatching rule to minimize total weighted tardiness. *Naval Research Logistics (NRL)* 64, 249–267. doi:[10.1002/nav.21744](https://doi.org/10.1002/nav.21744).
- Sweeney, K.D., Sweeney, D.C., Campbell, J.F., 2019. The performance of priority dispatching rules in a complex job shop: A study on the upper mississippi river. *International Journal of Production Economics* 216, 154 – 172. doi:<https://doi.org/10.1016/j.ijpe.2019.04.024>.
- Tay, J.C., Ho, N.B., 2008. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering* 54, 453 – 473. doi:<https://doi.org/10.1016/j.cie.2007.08.008>.
- Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., Kyek, A., 2018. Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP* 72, 1264 – 1269. URL: <http://www.sciencedirect.com/>

- science/article/pii/S221282711830372X, doi:<https://doi.org/10.1016/j.procir.2018.03.212>. 51st CIRP Conference on Manufacturing Systems.
- Xiong, H., Fan, H., Jiang, G., Li, G., 2017. A simulation-based study of dispatching rules in a dynamic job shop scheduling problem with batch release and extended technical precedence constraints. *European Journal of Operational Research* 257, 13 – 24. doi:<https://doi.org/10.1016/j.ejor.2016.07.030>.
- Xiong, J., ning Xing, L., wu Chen, Y., 2013. Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns. *International Journal of Production Economics* 141, 112–126. URL: <https://www.sciencedirect.com/science/article/pii/S0925527312001739>, doi:<https://doi.org/10.1016/j.ijpe.2012.04.015>. meta-heuristics for manufacturing scheduling and logistics problems.
- Xu, Y., Shi, C., Duenyas, I., 2016. Priority rules for multi-task due-date scheduling under varying processing costs. *Production and Operations Management* 25, 2086–2102. doi:[10.1111/poms.12606](https://doi.org/10.1111/poms.12606).
- Zhou, Y., jun Yang, J., Huang, Z., 2019. Automatic design of scheduling policies for dynamic flexible job shop scheduling via surrogate-assisted cooperative co-evolution genetic programming. *International Journal of Production Research* 0, 1–20. doi:[10.1080/00207543.2019.1620362](https://doi.org/10.1080/00207543.2019.1620362).

Appendix 3.A Job Shop Simulator

The pseudo-code of the job shop simulator is described in Algorithm 2. The inputs are scenario parameters $\langle |M|, \bar{p}, a, u, cv \rangle$ and a dispatching rule. As described in Section 3.3.2, each job arrival r_j is sampled from a Poisson distribution which mean rate depends on the value of u . The allowance factor a is considered to calculate jobs' due dates. The parameter $\bar{p}$ is the mean processing time, and cv is the coefficient of variation for sampling the actual processing times. Table 3.10 describes some of the variables defined in the algorithm and the respective initialisation. The next operation of each job enters machines' queue Q_m at time $start_j$, when they are available for execution.

Table 3.10 – Main variables considered by Algorithm 2

Variable	Description	Initialisation
num_jobs	Number of finished jobs	0
$\mathcal{J}'$	Jobs not completed	$\mathcal{J}$
$start_j$	Minimum start time of job j next operation	r_j
m_next_j	Next machine of job $j \in \mathcal{J}$	First machine of j
Q_m	Queue in machine m	$\emptyset$
A_m	Next available time of machine m	0

The simulation starts with an empty shopfloor and continues until a given number of jobs have been computed (max_jobs). The next operation of each job enters the machines' queue (Q_m) at time start_j , when they are available for execution. In line 2, the algorithm verifies which is the earliest event between the availability of a new operation of a job (start_j) and the availability of machine (A_m). An operation becomes available upon job arrival or when the predecessor operation finishes. In both cases, the operation may enter the next machines' queue (line 4), and the value of start_j is set to ∞ . Once a machine whose queue is not empty becomes available (line 7), the next operation to be executed is selected. In lines 8 to 10, the dispatching rule runs and calculates a score for each job in the queue. The job j^* with the least score is executed (in case of ties, the algorithm selects the job that arrived earlier in the queue). If j^* has no operations left, it is completed, removed from $\mathcal{J}'$, and the tardiness is calculated (lines 15-18). Otherwise, the algorithm updates the start time of the jobs' next operation (line 20). We recall that jobs are numbered on arrival in a crescent sequence order, and the tardiness T_j is computed only for jobs numbered from min to max . Following the values adopted in literature, we set $\text{min} = 501$ and $\text{max} = 2500$,

$\text{max_jobs} = \text{max} - \text{min} = 2000$.

Input: Scenario $\langle |\mathcal{M}|, \bar{p}, a, u, \text{cv} \rangle$ and DR

Result: Mean Tardiness $\bar{T}$

```

1 while num_jobs < max_jobs do
2   if  $\min_{j \in \mathcal{J}'} \{start_j\} < \min_{m \in \mathcal{M}: Q_m \neq \emptyset} \{A_m\}$  then
3      $j^* \leftarrow \arg \min_{j \in \mathcal{J}'} \{start_j\}$ ;
4      $Q_{m_{next_{j^*}}} \leftarrow Q_{m_{next_{j^*}}} + j^*$ ;
5      $start_{j^*} \leftarrow \infty$ ;
6   else
7      $m \leftarrow \arg \min_{m \in \mathcal{M}: Q_m \neq \emptyset} \{A_m\}$ ;
8     foreach job  $j \in Q_m$  do
9        $score_j \leftarrow \text{run\_DR on } j$ ;
10    end
11     $j^* \leftarrow \arg \min_{j \in Q_m} \{score_j\}$ ;
12    execute  $j^*$  in  $m$ ;
13     $A_m \leftarrow finish\_time_{j^*}$ ;
14    if  $m = \text{last\_machine of } j^*$  then
15       $\mathcal{J}' \leftarrow \mathcal{J}' \setminus j^*$ ;
16      if  $j^* \in \{\text{min} \dots \text{max}\}$  then
17         $\text{num\_jobs} \leftarrow \text{num\_jobs} + 1$ ;
18         $T_{j^*} \leftarrow \max(0, d_{j^*} - finish\_time_{j^*})$ ;
19         $\bar{T} \leftarrow \bar{T} + T_{j^*}$ ;
20      end
21    else
22       $start_{j^*} \leftarrow finish\_time_{j^*}$ ;
23    end
24  end
25 end
26  $\bar{T} \leftarrow \bar{T} \div \text{max\_jobs}$ ;

```

Algorithm 2: Job Shop Simulator

Appendix 3.B Terminal sets considered in GP experiments

Table 3.11 lists all terminals used in the first learning iteration, where two distinct sets are proposed. For calculating the evolved rules' size, we assume that all terminals in these sets have size one.

Table 3.11 – Terminal sets for GPLit

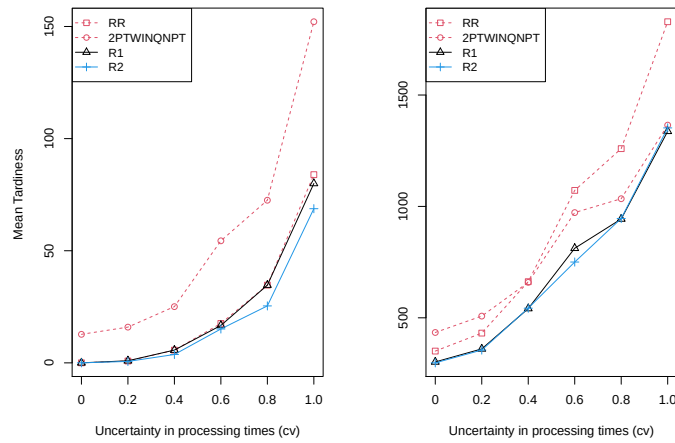
Description	GPLit _a	GPLit _b
Average processing time of operations in current queue	x	x
Expected processing time of current operation	x	x
Job allowance		x
Job arrival time	x	
Job due date	x	
Job queuing time		x
Machine ready time	x	x
Maximum due date time among jobs in queue		x
Maximum processing time among operations in queue		x
Minimum due date time among jobs in queue		x
Minimum processing time among operations in queue		x
Number of jobs in the system		x
Number of operations left	x	x
Operation ready time	x	
Processing time of the next operation of the job	x	x
Queue size in current machine		x
Queue size in the next machine		x
Remaining processing time of job	x	x
Remaining processing time of all jobs in the queue		x
Remaining time of the successive operations		
Slack of job	x	x
Time in the system	x	x
Work in current machines' queue		
Work in the next machines' queue	x	x
Work in all successive machines' queues		x

Appendix 3.C Performance of evolved rules under stochastic processing times

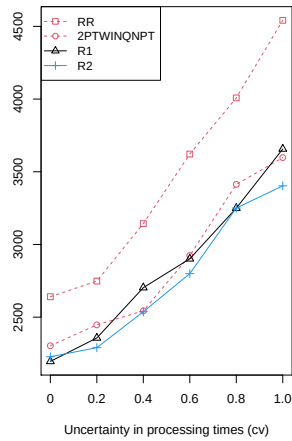
Table 3.12 presents the performance τ of R1 and R2 for each combination of allowance factor and shop utilisation. Despite some variance in the results, the relative performance of both rules tends to be more homogeneous compared to settings with deterministic processing times. Moreover, both rules behaviour is better under light load conditions.

Table 3.12 – Performance under stochastic processing times for each setting.

$u \backslash a$	R1						R2					
	2	3	4	6	8	Avg	2	3	4	6	8	Avg
0.80	0.95	0.89	0.83	0.78	0.97	0.88	0.95	0.88	0.82	0.68	0.76	0.81
0.85	0.97	0.89	0.85	0.77	0.77	0.84	0.96	0.88	0.83	0.74	0.62	0.80
0.90	0.96	0.93	0.88	0.82	0.78	0.87	0.95	0.91	0.86	0.79	0.73	0.85
0.95	0.97	0.97	0.93	0.89	0.88	0.93	0.96	0.96	0.92	0.86	0.82	0.90
0.97	0.98	0.97	0.97	0.96	0.94	0.97	0.97	0.97	0.93	0.93	0.91	0.94
Avg	0.97	0.93	0.89	0.84	0.86	0.90	0.96	0.92	0.87	0.80	0.76	0.86



(a) Light load conditions, $a = 8$ and $u = 0.89$ (b) Medium load conditions, $a = 4$ and $u = 0.90$



(c) Tight load conditions, $a = 2$ and $u = 0.97$

Figure 3.8 – Mean tardiness generated by evolved and existing rules for stochastic processing times.

Static scheduling of Human-Robot Teams

Scheduling Human-Robot Teams in collaborative working cells

Cristiane Ferreira^{*†} · Gonalo Figueira^{*†} · Pedro Amorim^{*†}

Published in *IJPE*, 2021

Abstract Soon, a new generation of Collaborative Robots embodying Human-Robot Teams (HRTs) is expected to be more widely adopted in manufacturing. The adoption of this technology requires evaluating the overall performance achieved by an HRT for a given production workflow. We study this performance by solving the underlying scheduling problem under different production settings. We formulate the problem as a Multimode Multiprocessor Task Scheduling Problem, where tasks may be executed by two different types of resources (humans and robots), or by both simultaneously. Two algorithms are proposed to solve the problem - a Constraint Programming model and a Genetic Algorithm. We also devise a new lower bound for benchmarking the methods. Computational experiments are conducted on a large set of instances generated to represent a variety of HRT production settings. General instances for the problem are also considered. The proposed methods outperform algorithms found in the literature for similar problems. For the HRT instances, we find optimal solutions for a considerable number of instances, and tight gaps to lower bounds when optimal solutions are unknown. Moreover, we derive some insights on the improvement obtained if tasks can be executed simultaneously by the HRT. The experiments suggest that collaborative tasks reduce the total work time, especially in settings with numerous precedence constraints and low robot eligibility. These results indicate that the possibility of collaborative work can shorten cycle time, which may motivate future investment in this new technology.

Keywords Scheduling · Human Robot Collaboration · Multiprocessor Tasks · Constraint Programming

^{*}Faculdade de Engenharia, Universidade do Porto, Porto, Portugal

[†]INESC TEC - Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Cincia, Porto, Portugal

4.1. Introduction

Since 2013, demand for industrial robots has been rising by 19% on average per year due to the global tendency towards automation in industry. In absolute terms, this number translates to annual global sales of 16.5 billion USD in 2018 [IRF, 2019]. The growth is expected to continue, particularly with the emergence of collaborative robots (cobots) which are expected to be responsible for 24% of robots sales in 2021 [Statista, 2019]. Cobots belong to a new generation of robots, capable of working alongside humans, embodying Human-Robot Teams (HRTs).

HRTs benefit from the distinctive features of these two resources. Humans are efficient in a wide range of tasks and adaptive to changes, while robots are precise and not subject to fatigue. Therefore, these teams combine productivity and flexibility while improving overall working conditions. Different interaction levels of human-robot collaboration exist. One may employ a mere coexistence of humans and robots in the same workstation or have both interacting with synchronised movements on sequential or parallel tasks. A higher interaction level occurs when tasks may be executed by human and robot in collaboration (the so-called “collaborative tasks”). The close cooperation between both opens up new possibilities for the manufacturing process. Hence, they have been receiving a lot of interest from practitioners in recent years [Weber, 2018].

However, technical challenges must be addressed before HRTs can be broadly implemented, including safety and synchronisation issues. Indeed, the higher the interaction between resources, the more challenging the deployment of the HRT. At the same time, managerial questions arise, as the investment in these robots should depend on the gains obtained. Thus, manufacturers need to know when is it worth to employ HRTs and how to optimise their processes accordingly. Depending on the level of collaboration, the number of tasks that each resource executes, their relative performance and several other aspects (e.g., precedence constraints), it might be worth, or not, to employ two resources instead of just a human or a robot alone. Optimising their processes consists of allocating and sequencing tasks on these resources.

The scheduling literature has studied this problem, known as Multimode Multiprocessor Task Scheduling Problem (MMTSP), but devoted limited attention. Most of the work has been conducted on the Single-mode MTSP. Multiprocessor tasks are those that require more than one resource at the same moment of time [Drozdowski, 2009]. These tasks appear in different applications, such as parallel computing systems [Switalski and Sere-dynski, 2015] and berth allocation [Türkoğullar et al., 2014]. In our case, multiprocessor tasks represent the collaborative tasks of the HRT. However, jobs can also be executed in other modes – the human operator or the robot, separately. The MMTSP, therefore, consists of assigning modes to jobs, in addition to their sequencing, being a generalisation of the Parallel Machine Scheduling Problem [Bianco et al., 1999].

Collaborative robots have recently started to be analyzed, mainly in the robotics literature [Xu et al., 2020, Bogner et al., 2018, Bänziger et al., 2018, Chen et al., 2014]. All these works seek to maximize the productivity of HRTs in manufacturing cells, which are characterized by cyclic production schedules. Even if minor deviations are necessary, operating a cyclic production process is easier and avoids reprogramming the robots every time. In these situations, it is necessary to minimise the production time of the cycle, i.e., its makespan. Chen et al. [2014] studies this problem in collaborative working cells but disregards collaborative tasks. Bogner et al. [2018] consider the latter but in a single-mode environment. Therefore, despite optimising makespan in HRTs cells, neither of them is able to evaluate the aforementioned managerial questions.

In this work, we analyse and extend the problem described by Chen et al. [2014], considering collaborative tasks, multiple modes and other features commonly found in HRTs, such as resource eligibility. We propose a Mixed Integer Programming formulation for the MMTSP with the objective of minimising the makespan and two different methods to solve the problem. Namely, we outperform the literature benchmarks with a Constraint Programming model and a Genetic Algorithm. A lower bound based on a Linear Programming model allows us to compute relatively tight optimality gaps and, hence, attest the quality of the proposed methods. Furthermore, these computational experiments, we derive a variety of managerial insights, which are important for the evaluation and deployment of collaborative robots.

The remainder of this paper is organised as follows. Section 4.2 reviews the literature on Decision-Making in Human-Robot Environments. It also reports previous work on Multiprocessor Task Scheduling and related problems. Section 4.3 describes the problem and presents a Mixed Integer Programming Model. A Linear Programming Model used for obtaining lower bounds is also presented. The proposed Constraint Programming Model and Genetic Algorithm are described in Sections 4.4 and 4.5, respectively. Computational results are presented in Section 4.6, where we compare the performance of the proposed methods and in Section 4.7, where we discuss insights on the improvements provided by the collaborative work. Finally, the paper is concluded with Section 4.8.

4.2. Related Work

This section describes the related work with a two-fold objective. At first, we summarise the works that describe scheduling problems in industrial environments of HRTs. Then we review the studies on Multiprocessor Task Scheduling and the state of the art for the MMTSP.

4.2.1 Scheduling in HRTs

Human-Robot Collaboration has received increasing attention during the last few years, in very different research areas. In robotics, works refer to the need for improving collision detection [Lee et al., 2015] and the optimisation of robots' motion planning [Pellegrinelli

et al., 2017]. In some contexts, human- factors are critical elements. Thus, several studies report an empirical examination of human-robot trust [Hancock et al., 2011]. Machine learning algorithms are being developed to enhance gesture recognition [Liu and Wang, 2018] and human activity prediction [Zanchettin et al., 2019] so that the robot can identify and adapt to the operators' behaviour. There are also concerns about plant layout design [Tsarouchi et al., 2016] and about designing tasks in a collaborative assembly cell, considering the different capabilities of humans and robots [Mateus et al., 2019]. An extensive review of the technological advances and issues related to HRTs is found in Krüger et al. [2009] and more recently in Ajoudani et al. [2018].

Some studies report dynamic scheduling problems, such as Johannsmeier and Haddadin [2017] and Darvish et al. [2018]. In both works, the authors consider that the assembly paths may be decided in real-time and the different assembly routes are described in a AND/OR Graph. In the former, the assignment of tasks is multimode, given that the algorithm may also determine which tasks should be executed by both human and robot. Dynamic scheduling is also present in Ding et al. [2014] and Nikolakis et al. [2018]. The former proposes a heuristic for the assignment and sequencing of assembly operations of a PLC I/O Module to avoid safety hazards. The latter design a tool that evaluates the assignment alternatives for each operation according to a utility function composed of many criteria. Recently, Casalino et al. [2019] developed a Petri Net model and a scheduler algorithm to simulate and optimise an HRT working cell, given the human uncertainty. The scheduler performs an exhaustive search in the reachability graph to prioritise the next task for the robot. They use a prediction algorithm to identify patterns in human activities and adapt the robot schedule accordingly.

On a deterministic setting, Gombolay et al. [2018] propose an algorithm to multi-robot scheduling, which incorporates the preferences of a human supervisor. The existence of a human worker imposes some spatial constraints to support safety distance regulations. The objective is to minimise the difference to a previous schedule and spatial interfaces between the robots. But the algorithm does not schedule human tasks.

Cyclic scheduling with HRTs is also the focus of some works. Bänziger et al. [2018] describe a genetic algorithm to address the assignment and sequencing of tasks in an HRTs workplace. Solutions obtained are evaluated by a simulation tool designed for that purpose and validated in a real manufacturing environment. The objective is the minimisation of completion time, walking distances and waiting time. In Bogner et al. [2018], a use case of printed circuit boards assembly is presented. The environment consists of workstations with a single robot in collaboration with one or more human operators. There are precedence constraints and minimum safety distances in both problems. Recently, Xu et al. [2020] develop a Bee Algorithm to disassembly tasks in a working cell composed of two operators, a human and a robot. Tasks are classified according to the execution time and difficulty of each operator. The algorithm is multi-objective, considering to minimise the completion time, costs and the difficulty in the execution of tasks. Chen et al. [2014] describe a power box module assembly performed by HRTs. The objective consists of both the minimisation of makespan and production costs. Their work is based on a collaborative manufacturing environment where a human has as co-workers one or more robots to jointly

assemble a power module box, in a layout configuration similar to that shown by Figure 4.1. The setting is justified by the fact that robotic manipulators are nowadays widely used in electronics assembly and the introduction of humans improve efficiency in the execution of specific manual tasks. The authors propose a Genetic Algorithm that optimises each HRT separately, and the independent solutions generated are merged by a constructive heuristic. The algorithm that merges the independent solutions considers the Symmetric Assignment Constraints (cf. Section 4.3).

The International Federation of Robotics provides a classification on the different types of HRTs interaction [IFR, 2018]. They describe five scenarios where humans and robots that operate in the same working cell. The first one is actually not collaborative, as it refers to fenced robots. The other four are summarised in Table 4.1, which also classifies the HRTs found in the works referred here.

In the *Coexistence* scenario, there is no fence, but humans and robots operate in their own workspace (there is no shared area in the workstation). *Synchronised Collaboration* refers to cells where the operators share the workspace, but they always work in different components, performing sequential and synchronised movements. A closer interaction occurs in *Cooperation*, where both operators – humans and robots - may work at the same time on the same components, performing collaborative tasks. This is the scope of the HRTs considered in this work. Finally, *Responsive Collaboration* is the highest level of interaction, where humans and robots not only share workspace and components, but the robot is able to react in real-time to the movements of a human worker.

Table 4.1 – Categorisation of works in the literature, according to the classification of human-robot collaboration provided by IFR [2018]

HRTs Type	Coexistence	Synchronised Collaboration	Cooperation	Responsive Collaboration
Description	No fence but no shared workspace.	Operators share a workspace, but do not work simultaneously on the same component.	Human worker and robot work simultaneously on the same product or component.	Robot responds in real-time to movement of the worker
Examples	Ding et al. [2014]	Chen et al. [2014], Bänziger et al. [2018]	Nikolakis et al. [2018], Bogner et al. [2018], Xu et al. [2020]	Johannsmeier and Haddadin [2017], Darvish et al. [2018], Casalino et al. [2019]

Although the last two scenarios are the most advanced and should grow in the long term, most of the current applications of HRTs are Coexistence and Synchronised Collaboration. This is due to the extensive technological challenges present in the other two scenarios [IFR, 2018]. Moreover, the transition to Cooperation involves analysing some performance trade-offs, primarily related to the impact in productivity provided by the collaborative tasks. This is the purpose of this paper.

Our paper primarily differs from the literature in considering multitask tasks, that are tasks suitable to be executed in different modes. None of the works reported considers optimis-

ing the decision between multiprocessor (collaborative) or single-processor task executing. As described above, this is a fundamental question when studying the improvement opportunities of collaborative environments. Moreover, except for [Chen et al. \[2014\]](#) and [Bogner et al. \[2018\]](#), the literature focuses on teams composed by only a human and a robot, solving the problems using relatively simple or exhaustive methods. Next, we review research on multiprocessor tasks scheduling, where tasks are executed by more than one resource.

4.2.2 Multiprocessor Task Scheduling

In classical Parallel Machine Scheduling, the environment is composed of a set of resources that work in parallel, and jobs require a single operation from one of the resources [[Pinedo, 2008](#)]. Multiprocessor Task Scheduling (MTS) is a generalisation of Parallel Machine Scheduling where a single job may be executed simultaneously by a set of resources in parallel [[Drozdowski, 2009](#)]. Different variants of MTS problems are described in the literature. If all resources are identical, each job may require a specific *number of resources* (rigid tasks) or a given *amount of processing time* from resources (moldable or malleable tasks). Otherwise, in non-identical resources, jobs may require *a specific set of resources* (single mode) or accept *different combinations* of resources (multiple modes).

Multiprocessor applications are found beyond HRTs. In parallel computing systems, some works deal with tasks that identical processors may simultaneously execute [[Błażewicz et al., 2003](#), [Switalski and Seredynski, 2015](#), [Drozdowski, 2009](#)]. In the Berth Allocation Problem, multiprocessor tasks are present if a ship is served by more than one quay crane, as in [Guan et al. \[2002\]](#), with rigid tasks, and [Türkoğullar et al. \[2014\]](#), moldable tasks. Moldable tasks are also found in workforce allocation if the tasks' processing time depends on the number of resources assigned, as in [Pappis and Rachaniotis \[2010\]](#), and [Battaia et al. \[2015\]](#).

The first work to deal with Multiprocessor Task Scheduling is [Weglarz et al. \[1986\]](#). The authors studied the computational complexity of scheduling rigid tasks in the special case of unit processing times. They presented two polynomial-time algorithms with complexity $O(n)$ for non-preemptive scheduling of tasks that require a fixed number of k processors. A comprehensive survey on Multiprocessor Task Scheduling can be found in [Drozdowski \[2009\]](#).

In the Multimode (or General) Multiprocessor Task Scheduling Problem, each job has a set of possible modes. Each mode has a specific processing time and a subset of machines [[Chen and Lee, 1999](#)]. The problem is shown to be NP-hard by demonstrating that the particular case with three machines is strongly NP-hard [[Błażewicz et al., 1992](#)]. [Bianco et al. \[1995\]](#) provided a dynamic programming algorithm to solve specific cases of the problem (with $|R| = 2$ and $|R| = 3$). They also proposed a heuristic for the general problem with worst-case ratio of $|R|$. [Chen and Lee \[1999\]](#) improved their work with an approximation ratio of $|R|/2 + \varepsilon$ by using dynamic programming. Later both [Chen and Miranda \[2001\]](#) and [Jansen and Porkolab \[2005\]](#) developed polynomial approximation schemes for the problem. Meta-heuristics based on the concepts of intensification and diversification

[Glover, 2000] were developed in the work of Caramia and Giordani [2009] and Caramia and Giordani [2010].

Bianco et al. [1998] are the first authors to tackle the variant of the MMTSP with precedence constraints. They proposed specific heuristics for the cases with and without budget constraints. Bianco et al. [1999] is the only work proposing specific lower bounds for the MMTSP. The authors present three distinct methods, LB1, LB1r and LB2 - the later dominates the other two. LB2 generates a weighted *conflict graph* $\mathcal{G}$, in which each vertex represents a job, and each arc (j_1, j_2) indicates that jobs j_1 and j_2 cannot overlap in time. The weight of arc (j_1, j_2) is the shortest processing time of j_1 among all its modes. Arc (j_1, j_2) is created if either of the following conditions is met: (a) there is a path between j_1 and j_2 in the precedence graph; (b) all eligible modes from j_1 and j_2 have some machine in common. LB2 is given by the maximum weighted clique in $\mathcal{G}$. The authors also proposed two constructive heuristics for the problem, namely UB1 and UB2. They showed that these algorithms produce better results in comparison to the former work. To the best of our knowledge, these are the only algorithms developed for the MMTSP with precedence constraints. Therefore this work is considered for benchmark comparison with the methods here proposed. A more detailed description of them is given in Appendix.

The scheduling literature presents several well-studied problems with similar characteristics of the MMTSP. The existence of machine eligibility constraints is usually referred to as Multi-purpose Machine Scheduling. In Brucker et al. [1997], the authors discuss the complexity of Parallel Machine and shop scheduling problems with such characteristic, and an extensive survey on the topic is found in Leung and Li [2008]. The Multimode Resource-Constrained Project Scheduling Problem deals with different types of resources. Jobs are associated with a set of modes, each of them with specific processing times and requiring a different amount of resources. The MMTSP is a specific case of that problem where there is exactly one type of each resource. One can find food surveys on this problem and its variations in Brucker et al. [1999] and Hartmann and Briskorn [2010].

Another generalisation of the MMTSP is the Resource Sharing and Scheduling Problem, that also handles a set of multimode jobs. Differently from the MMTSP, during the execution of job j by a set of resources R , the resources in R may be dedicated to j in different periods of time. In Pinto et al. [2009], the authors propose two Branch-and-bound and a Genetic Algorithm for the problem. Although the methods present a promising performance, the proposed instances are quite small, with up to 24 jobs.

In the following we formalise the MMTSP and a variant of it suitable to be found in collaborative environments. A Mixed Integer Programming formulation and a lower bound are also described.

4.3. Problem Definition

The Multimode Multiprocessor Task Scheduling Problem [Drozdowski, 1996] is described over a set of non-identical resources R and a set of jobs J . A mode m is composed of a

subset of resources R_m such that $R_m \subseteq R$. A job j must be executed in one of the possible modes M_j , each requiring a specific processing time p_{jm} . One has to define a non-preemptive schedule for the jobs in such a way that each resource only processes one job at a time. The problem is NP-hard since a particular case is strongly NP-hard [Błażewicz et al., 1992].

The classic objective function of such problem is to minimise the makespan, i.e., the maximum completion time of jobs [Drozdowski, 2009]. This is also one of the common performance criteria considered in the scheduling of HRTs, mainly because of its use in cyclic scheduling (Chen et al. [2014], Bogner et al. [2018]). Additionally, these settings often involve precedence constraints, which are common in a variety of production environments, such as assembly systems.

Collaborative working cells are composed of two types of resources, humans and robots, that share the same physical environment and work on the same jobs. A job may be executed by a human or a robot (working alone), or an HRT (both working in collaboration). Figure 4.1 illustrates an example of a working cell composed of two robots, one human, and ten jobs to be executed in each cycle. A feasible solution is presented in the Gantt Chart: the human executes alone jobs A_1, A_2, E_1 and E_2 ; the first robot executes B_1 and D_1 while the second executes B_2 and D_2 ; both C_1 and C_2 are executed by an HRT. Note that there are two HRTs: HRT_1 , the first robot and the human; and HRT_2 , the second robot and the human.

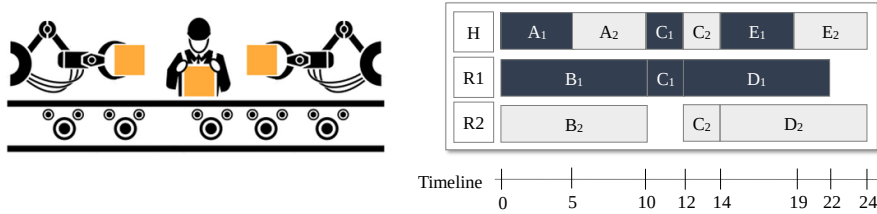


Figure 4.1 – An hybrid working cell and a solution to the MMTSP

One may also observe specific constraints that go beyond the traditional MMTSP. In the HRT application presented by Chen et al. [2014], the working cell assembles multiple components into a single piece. Since the cell comprises two similar robots (just like in the example above), it produces two equal pieces in each cycle. In that case, we say that the working cell has two different working spaces, one per HRT.

We define a variant of the MMTSP where the set of jobs J is divided into working spaces. Each working space $w = \{1, \dots, n\}$ is responsible for a single piece per cycle, which requires a set of jobs $J_w \subset J$. Each job $j \in J_w$ has n replications (one in each working space). In the example of Figure 4.1, there are two working spaces ($w \in \{1, 2\}$), thus $J = J_1 \cup J_2$, and $J_1 = \{A_1, B_1, C_1, D_1, E_1\}$, $J_2 = \{A_2, B_2, C_2, D_2, E_2\}$. Jobs in J_1 can only be executed by resources from $HRT_1 = \{H, R_1\}$, while jobs in J_2 can only be executed by $HRT_2 = \{H, R_2\}$. Observe that the job A_2 is the replication of A_1 in working space 2. The same

applies for $B1$ and $B2$, and so on.

The MMTSP-SAC includes additional constraints to MMTSP, the *Symmetric Assignment Constraints* (SAC). They impose that: i) jobs in J_w must be assigned to resources from HRT_w , and ii) the assignment of jobs to modes must be the same for all their replications. The solution in Figure 4.1 satisfies these constraints. The importance of considering these constraints is justified by [Chen et al. \[2014\]](#) as to avoid the human's mental burden of executing different types of tasks in each team he participates in. The easiness of coordinating and reprogramming the robots is another important reason.

4.3.1 Mixed Integer Programming Formulation

In this section, we describe a mixed integer programming formulation (MIP) for the MMTSP and its variant with Symmetric Assignment Constraints (MMTSP-SAC). Let Y_{jm} be a binary variable that indicates if job j is assigned to mode m . Variables S_{jm} and F_{jm} are, respectively, the starting and ending times of job j in mode m (if this mode is not selected, those variables take the value 0). The binary variable X_{rij} indicates that job i precedes job j in resource r (this assumes that both jobs i and j are assigned to modes that contain resource r). To assist in formulating the precedence constraints, we define $A = \{(i, j) : i, j \in J, i < j\}$, i.e., the set of all pairs of jobs where job i must be executed before job j . To assist in the no-overlap constraints, we define $M_{jr} = \{m \in M_j : r \in R_m\}$, i.e., the subset of modes of job j that contain resource r . The objective (4.1) is to minimise the makespan given by C . The parameter H is an upper bound for C , calculated as $\sum_{j \in J} \hat{p}_j$, where $\hat{p}_j = \max_{m \in M_j} p_{jm} \forall j \in J$. The model is given below.

$$\text{Min } C \tag{4.1}$$

Subject to:

$$\sum_{m \in M_j} Y_{jm} = 1 \quad \forall j \in J \tag{4.2}$$

$$F_{jm} \geq S_{jm} + p_{jm} + H \cdot (Y_{jm} - 1) \quad \forall j \in J, \forall m \in M_j \tag{4.3}$$

$$S_{jm} \geq F_{im'} + H \cdot (Y_{jm} - 1) \quad \forall (i, j) \in A, \forall m \in M_j, \forall m' \in M_i \tag{4.4}$$

$$S_{jm} \geq F_{im'} + H \cdot (Y_{jm} + Y_{im'} + X_{rij} - 3) \quad \forall i, j \in J, \forall r \in R, \forall m \in M_{jr}, \forall m' \in M_{ir} \tag{4.5}$$

$$S_{im'} \geq F_{jm} + H \cdot (Y_{jm} + Y_{im'} - X_{rij} - 2) \quad \forall i, j \in J, \forall r \in R, \forall m \in M_{jr}, \forall m' \in M_{ir} \tag{4.6}$$

$$C \geq F_{jm} \quad \forall j \in J, \forall m \in M_j \tag{4.7}$$

$$Y_{jm} \in \{0, 1\} \quad \forall j \in J, \forall m \in M_j \tag{4.8}$$

$$X_{rij} \in \{0, 1\} \quad \forall i, j \in J, \forall r \in R \tag{4.9}$$

Constraints (4.2) ensure that each job is assigned to one and only one mode. Constraints (4.3) calculate the ending time of each job, according to its assignment. Constraints (4.4) establish the pre-defined precedence relations. Constraints (4.5) and (4.6) ensure that each resource will not process more than one job at a time ('no-overlap'). Constraints (4.7) calculate the makespan. Constraints (4.8) and (4.9) define the domain of the variables. The continuous decision variables are non-negative: $S_{jm}, F_{jm} \geq 0, \forall j \in J, \forall m \in M_j$.

In the variant MMTSP-SAC, jobs are replicated in n working spaces, each one composed of a different robot. Given two working spaces (1 and 2), there will be two sets of jobs (J_1 and J_2), and each job $j_1 \in J_1$ will have a replication j_2 in J_2 . A solution of this problem must ensure that if job j is assigned to mode m , all replications of j are also assigned to m in their respective working spaces. Given that j_1 belongs to J_1 , and j_w is the replication of j_1 in working space J_w , the MMTSP-SAC includes an additional constraint set:

$$X_{j_1m} = X_{j_wm} \quad \forall j_1 \in J_1, \forall m \in M_j, \forall w \in \{2, \dots, n\} \quad (4.10)$$

4.3.2 Lower bound

As the MMTSP is an NP-hard problem, a lower bound was developed to assess the quality of solutions. We use a Linear Programming Model (LP) based on the formulation described in Section 4.3.1.

Binary assignment variables Y_{jm} are relaxed. Therefore, jobs may be partially assigned to different modes. Variables X_{rij} , S_{jm} and F_{jm} are discarded. Variables Z_r designate the maximum completion time of each resource. The objective function is the same as (4.1), to minimize the makespan C . All constraints of the LP model are described as follows.

$$\sum_{m \in M_j} Y_{jm} = 1 \quad \forall j \in J \quad (4.11)$$

$$Z_r = \sum_{j \in J} \sum_{m \in M_j: r \in R_m} Y_{jm} \cdot p_{jm} \quad \forall r \in R \quad (4.12)$$

$$C \geq Z_r \quad \forall r \in R \quad (4.13)$$

$$Y_{jm} \geq 0 \quad \forall j \in J, \forall m \in M_j \quad (4.14)$$

Constraints (4.11) ensure that all jobs will be entirely assigned to modes, given that they may be partially assigned to more than one mode. Constraints (4.12) calculate the maximum completion time of each resource, according to the jobs assigned to them. Constraints (4.13) calculate the makespan and Constraints (4.14) define the relaxed domain of variables Y_{jm} .

The following sections describe the Constraint Programming Model and the Genetic Algorithm, which were implemented to find near-optimal solutions for the problem. Both

are present in literature as efficient methodologies to deal with such complex scheduling problems (Gökgür et al. [2018], Afzalirad and Rezaeian [2016]).

4.4. Constraint Programming Model

Constraint Programming (CP) has been used in the last decade as an efficient method to deal with optimisation problems. It is particularly effective in scheduling problems with a high number of constraints limiting the search space. The recent Parallel Machine literature has promising results of CP in comparison to heuristics and Mixed Integer Programming (Edis and Ozkarahan [2011], Gökgür et al. [2018]). An overview of Constraint Programming applied to solve scheduling problems is present in Baptiste et al. [2001].

The purpose of CP algorithms is to solve a Constraint Satisfaction Problem (CSP). The problem is composed of a set of variables, their respective finite set of domain values and a set of constraints over these values [Tsang, 1993]. A solution of a CP model is an assignment of variables to values such that all constraints are satisfied. The algorithms apply constraint propagation techniques for filtering the variables domain (i.e. using inferences obtained from constraints to reduce variables domain) and also explicit search throughout the solution space. For optimisation problems, the CSP has an objective function f , and the algorithm returns the solution S^* that satisfies the CSP and optimises f . If the algorithm completes the search by enumerating all feasible solutions, S^* is the optimal solution of the problem.

Our CP model aims at finding optimal or at least near-optimal solutions for the MMTSP. The list of variables is found in Table 4.2, composed of two different sets of interval variables and a set of sequence variables. Interval variables are used to represent a period during which a non-preemptive operation is performed, most commonly found in scheduling models. The interval variable I_j is used to model the start and finish time of job $j \in J$, while variable L_{jm} links the job execution time to the assignment decision. If job j is assigned to mode $m \in M_j$, the value of L_{jm} is equal to the value of I_j , otherwise L_{jm} is empty. Sequence variables define an order for a list of operations. For each resource $r \in R$, a variable Seq_r is defined with the purpose of sequencing its operations. Thus, Seq_r is described over a set of interval variables $L_{jm} : r \in R_m$ and specify in which sequence their values must be set.

Table 4.2 – Variables of the Constraint Programming model

Variable	Type	Description
I_j	Interval	Execution time (start and end) of job $j \in J$
L_{jm}	Interval	Execution time of job $j \in J$ if assigned to mode $m \in M_j$
Seq_r	Sequence	Sequence of operations variables $L_{jm} r \in R_m$

Figure 4.2 presents the value of variables L_{jm} and Seq_r for the example illustrated in Section 4.3. The problem instance is composed of the human H , robots $R1$ and $R2$, and ten jobs (A1, B1, C1, D1, E1, A2, B2, C2, D2, E2). There are five possible execution modes in

total: the human alone (H), the robot alone (R1 or R2) and an HRT ($\{H, R1\}$ or $\{H, R2\}$). The robots R1 or R2 are eligible for different jobs, depending on the working space. So job C1 may be assigned to H, R1 or $\{H, R1\}$, while job C2 may be assigned to H, R2 or $\{H, R2\}$. In the solution of Figure 2, job C1 is assigned to $\{H, R1\}$ and job C2 is assigned to $\{H, R2\}$.

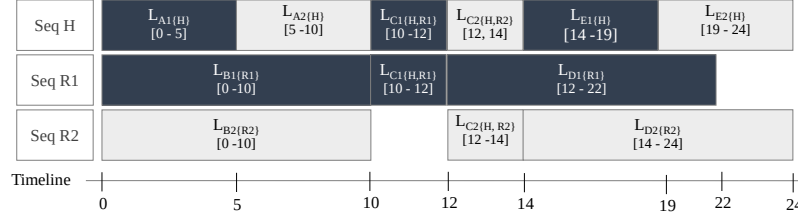


Figure 4.2 – Representation of sequence variables Seq_H , Seq_{R1} and Seq_{R2} . Values in sequence variables represent the value of the respective interval variable L_{jm}

In the example, the value of $L_{A1,\{H\}}$ is the interval [0-5], and $L_{A1,\{R1\}} = L_{A1,\{H,R1\}} = \emptyset$. It means that job A1 is scheduled to be executed from time 0 to 5, by the human alone. The value of Seq_H is the sequence $\{L_{A1,\{H\}}, L_{A2,\{H\}}, L_{C1,\{H\}}, L_{C2,\{H\}}, L_{E1,\{H\}}, L_{E2,\{H\}}\}$. Observe that $L_{C1,\{H,R1\}} = [10, 12]$, which means that job C1 starts execution in time 10 and finishes at 12. Because mode $\{H, R1\}$ is composed of two resources, that variable is associated both to Seq_H and Seq_{R1} .

The objective function and constraints of the model are described below, given that $start_of(I_j)$ and $end_of(I_j)$ are the start and finish time of variable I_j , respectively.

$$\text{Min} \quad \max_{j \in J} end_of(I_j) \quad (4.15)$$

$$start_of(I_j) \geq end_of(I_i) \quad \forall (i, j) \in A \quad (4.16)$$

$$alternative(I_j, \{[L_{jm}, p_{jm}] | m \in M_j\}) \quad \forall j \in J \quad (4.17)$$

$$no_overlap(Seq_r) \quad \forall r \in R \quad (4.18)$$

$$presence_of(L_{jm}) \Rightarrow presence_of(L_{j_wm}) \quad \forall j \in J_1, \forall m \in M_j, \forall w \in \{2, \dots, n\} \quad (4.19)$$

The objective function (4.15) is to minimise the maximum completion time $end_of(I_j)$

among all jobs $j \in J$. The pre-specified precedence constraints between jobs are described by constraints (4.16).

The function $alternative(I_j, \{[L_{jm}, p_{jm}] | m \in M_j\})$ in Constraints (4.17) sets the value of I_j equals to the value of one of the variables L_{jm} , i.e., one mode $m \in M_j$ must be selected for j . As the size of I_j corresponds to the processing time of job j , $alternative()$ also ensures that the size of I_j will be p_{jm} , where m is the selected mode. In a feasible solution, each resource r can process at most one job at any time. Thus, the value of all $L_{jm}, r \in R_m$ cannot overlap. That is guaranteed in our model by in Constraints (4.18), that enables only feasible and no-overlapping sequences to Seq_r .

Finally, the *Symmetric Assignment Constraints* are defined by (4.19). As described in Section 4.3, these constraints are considered in the MMTSP-SAC, a variant of the problem where there are n replications of the same job. In that version, the set of jobs J is divided into working spaces $w = \{1, \dots, n\}$, each one composed of a subset $J_w \subset J$. We say that job j_w is the replication of job j in working space w . If $presence_of(L_{jm}) = true$, job j is assigned to mode m . The model assures that for each working space w and $j \in J_1$, if $presence_of(L_{jm}) = true$, mode m is also assigned to job j_w .

As the constraint programming solvers might not scale well for larger problem instances, a Genetic Algorithm was developed and benchmarked with both the CP and previous methods proposed in the literature. The algorithm is detailed in the following section.

4.5. Genetic Algorithm

Genetic Algorithms are bio-inspired optimisation methods used for a vast number of combinatorial problems [Goldberg, 1989, Herrmann et al., 1995]. They operate on a population of solutions (individuals) that evolve at each generation. Crossover and mutation operators are applied to combine solutions and get different individuals. A fitness function is used to evaluate the individuals and select the best ones to proceed for the next generations. Those algorithms are most common approaches used for several variants of Parallel Machine Scheduling problems [Edis et al., 2013]. In this work, we design a specific Genetic Algorithm for the MMTSP, GA-MM, which details are given next.

4.5.1 Encoding and Decoding

Two strings are used to represent individuals: *v-mode* and *v-order*. The first one is a vector containing the assigned execution mode for each job. The list *v-order* represents the sequence order of jobs, where each position refers to one job.

Figure 4.3 shows an example of the individual representation for an instance composed of one human, one robot and ten jobs. In *v-mode*, H, R and T respectively indicate an assignment to human, robot and both simultaneously. In this example, the sequence of jobs as established by *v-order* is 0, 2, 3, 6, 8, 1, 4, 5, 7 and 9.

The decoding procedure generates a schedule from both vectors. The algorithm iteratively

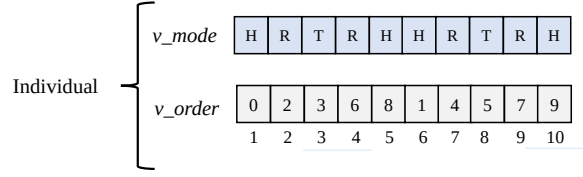


Figure 4.3 – Individual representation for a solution with 10 jobs.

picks a job j from v_order and inserts it into the schedule. The starting time is the minimum feasible time given by R_m , where m is the mode assigned to j in v_mode . When considering job j , if some of its predecessors are not already in the partial schedule, the algorithm includes it in a pending list. Jobs in the pending list are considered again after the inclusion of all their predecessors. The pseudo-code and an explanation of the procedure are provided in Algorithm 3 of Appendix 4.A.1.

4.5.2 General Structure and Operators

The proposed GA-MM is a multi-start procedure that runs until a maximum number of generations without improvement $maxG$. The first step of the algorithm is a random generation of p individuals for the initial population. For each individual, the values in vector v_mode are randomly defined, considering the possible modes for each job. The vector v_order is iteratively populated. At each iteration, a job is randomly chosen and inserted in the last position of the list. This iterative procedure only sorts jobs from the subset of jobs whose predecessors are already in v_order . Thus in the initial population, the order of jobs in v_order satisfies the precedence constraints.

At each generation, the algorithm applies the crossover and mutation operations and includes the new individuals in the population of the following generation. With a frequency f , a local search procedure is applied to ls individuals randomly chosen from the best $b\%$ in the population. At the end of each generation, the current population is completely updated, and the algorithm proceeds to a new generation. The stopping criterion is to find a target solution or reach a time limit. Algorithm 4 in Section 4.A.2 (Appendix) presents a pseudo-code on the general structure of the algorithm.

There are two crossover operators, both of them applied with a rate of c_r with a two-point procedure. The parent selection uses tournament groups: a number of individuals are randomly selected, and the best is used as a parent in crossover. In each crossover operation, two tournament selections pick up parents S_1 and S_2 to generate children S_a and S_b . The first operator randomly defines two crossover points j and i in vector v_mode . Based on that, two children S_a and S_b are created crossing values of vector v_modes between points j and i . The values in list v_order are the same as the parents. In case of the MMTSP-SAC, jobs are grouped by working space, $J = \cup_{w=1}^n J_w$. One must ensure that job $j_1 \in J_1$ and its replication $j_w \in J_w$ have the same assignment. So the first crossover operator is applied only to jobs from J_1 (the first working space), and the changes are copied to the

other jobs.

The second operator randomly selects two crossover points j and i in the vector v -order, such that $j < i$. Children vectors S_a and S_b are created as copies of the parents S_1 and S_2 . A sub-vector from S_1 between positions $j+1$ and i is inserted into child S_b at position $j+1$. The same occurs between parent S_2 and child S_a . At this point, the v -order vector of both children has size $|J| + i - j$, and have duplicated values. A procedure is applied to remove the second occurrence of each value for both children. This operator may be illustrated with a small example where v -order is $\{1, 2, 3, 4, 5\}$ for S_1 and $\{5, 4, 3, 2, 1\}$ for S_2 . If $j = 1$ and $i = 4$, the sub-vectors $\{2, 3, 4\}$ and $\{4, 3, 2\}$ will be inserted in children S_b and S_a , respectively, at position 2. Thus, v -order for S_a and S_b would be, respectively, $\{1, 4, 3, 2, 2, 3, 4, 5\}$ and $\{5, 2, 3, 4, 4, 3, 2, 1\}$. After removing the duplicated values, v -order for S_a and S_b would be, respectively, $\{1, 4, 3, 2, 5\}$ and $\{5, 2, 3, 4, 1\}$.

There are also two mutation operators. Each of them is applied with a rate of m_r . Both operators randomly select a parent S from the best $b\%$ of the population. The first operator randomly selects a job j and an execution mode in M_j different from the current one in vector v -mode, generating a child S_a . The second operator randomly selects a pair of jobs and swaps their values in vector v -order, generating a child S_b .

As aforementioned, with a frequency of f , a local search is conducted from the best l_s individuals. The local search procedure improves solution S by applying modifications in jobs allocation or sequencing. It sequentially considers three different neighbourhoods in the following order:

- *Change Allocation* - Changes the assignment to a different mode.
- *Swap Allocations* - Pairwise exchanges the assignment of jobs.
- *Change Order* - Pairwise exchanges the order of jobs.

Each neighbourhood is applied until the algorithm visits all jobs with no improvement in S . Before comparing each new solution with the incumbent one, the algorithm runs the decoding procedure described in Section 4.5.1. An alternative version of the algorithm calculated the bottleneck path of the schedule and applied each neighbourhood only to jobs in that path. But the time taken to calculate the path did not compensate for the improvement in solution quality.

4.6. Performance Evaluation

This section describes the results of a performance comparison of the proposed methods for different problem sizes and instance sets. All methods were coded in C++ and ran on a personal computer including an Intel(R) Core(TM) i7 CPU with 2.8 GHz of speed and 12 GB of RAM. For the Constraint Programming Model, a commercial language and solver is used (IBM ILOG CP Optimizer 12.6).

We evaluate the proposed methods and compare them to UB2, the algorithm proposed by Bianco et al. [1999], and the Genetic Algorithm proposed by Chen et al. [2014], GA-Chen. Both algorithms are described in appendix 4.B.2. We remark that the algorithm proposed by Chen et al. [2014] is different from our proposal, GA-MM, as it generates a separate solution for each working cell. These benchmark methods were implemented and adapted to run as multi-start algorithms to improve their competitiveness. A randomised version of the constructive heuristic UB1 from Bianco et al. [1999] was also implemented, but its results were slightly worse than UB2's. Thus, for a matter of concision, we report only the results of UB2.

4.6.1 Performance Evaluation of the Proposed Methods

Two instance sets were generated for the performance comparison between methods. The first set simulates HRTs working cells and the second set is composed of general instances. In the following, we present the results for both instance sets. Table 4.3 presents the calibration of the parameters for the Genetic Algorithm, whose values were defined in preliminary tests and depend on the version of the problem considered, whether MMTSP or MMTSP-SAC.

Table 4.3 – Parameter calibration for proposed algorithms

Parameter	MMTSP	MMTSP-SAC
Max No-Improvement Generations ($MaxG$)	100	30
Population Size (p)	10000 $/ J $	5000 $/ J $
LS Frequency (f)	10	5
Num Individuals to LS (ls)	10	10
% for Parent Selection (b)	20%	10%
Crossover rate (c_r)	0.9	0.9
Mutation rate (m_r)	0.1	0.1

4.6.1.1 HRTs Instance Set

Table 4.4 summarises the parameters that are used in the HRTs Instance Set. These instances are composed of a set of humans and a set of robots. A working space w is composed of a robot and human operators that may collaborate with it. Each job j belonging to w may be assigned only to resources in HRT_w , depending on the modes present in M_j . We generated 120 instances from the combination of these values. The methods were run in two problem variants: the original MMTSP and MMTSP-SAC, the version with the *Symmetric Assignment Constraints*.

The number of humans and robots was defined according to what was found in robotics literature [Ajoudani et al., 2018]. The processing times were sampled from a uniform distribution as commonly found in scheduling literature. The resources of the same type (whether human or robot) have the same processing time for each job and the relation

between humans and robots speed vary from 0.5 to 2, also in accordance to use cases and prototypes from literature (Chen et al. [2014], Xu et al. [2020]).

As in Bianco et al. [1999], the precedence graph is randomly generated, and each job may have up to 3 predecessors. In preliminary experiments, we have considered the maximum of 1 and 2 predecessors, but changing this parameter does not influence the relative comparison between the methods. The remaining parameter values were defined in such a way to produce a diverse instance set.

Table 4.4 – Parameters setting for instance generation

Parameter	Description	Adopted Values
<i>NH</i>	Number of humans.	1, 2
<i>NR</i>	Number of robots.	2, 3
<i>Jobs/Working Space</i>	Number of jobs for each working space (a robot and human operators).	20, 50
<i>OpTime</i>	Processing times for humans.	$U(1, 100)$
<i>Human/Robot Speed</i>	Average relationship between robots and humans processing time.	0.5, 1, 1.5, 2
<i>RE</i>	Robot eligibility, i.e. percentage of jobs that can be processed by robots.	25%, 50%, 100%
<i>MM Elig</i>	Multimode eligibility. Defined as percentage of jobs with the option to be executed simultaneously by two resources.	0%, 25%, 50%, 75%, 100%
<i>Collaboration Time</i>	Processing time of a job if executed in the collaborative mode (simultaneously by more than one resource) in comparison to the processing time of the faster resource.	$U(0.5, 1)$

At first, we evaluate the proposed lower bound (LB3) against the one proposed by Bianco et al. [1999] (LB2, briefly described in the appendix 4.B.1). In the HRT instance set, the lower bounds provided by LB3 are always tighter than the values of LB2. The Relative Percentage Deviation (RPD) of LB3 results (lb) in comparison to the best-known solution values (ub) are, in average, 2.3% while the average RPD of LB2 results is 42.6%. RPD is calculated as $RPD = (ub - lb)/ub$. The results of LB2 are close to LB3 for the instances with one human and less than 100% of robot eligibility (RPD around 1%). For the other instances, LB2 provided a quite weak lower bound, with an average of 64.5%. These results are explained by the structure of the Conflict Graph generated by LB2. The graph is too sparse in case of existing redundant resources for a job.

For the next experiments, all methods ran with a time limit of 1800s. We defined the time limit based on preliminary experiments that show that the CP model stabilised the improvements after 10 minutes of execution. In order to compare the results of the different methods, we use the relative gap of their results (ub) in comparison to lower bounds (lb), where lb is the optimal value if it is known or the result of LB3 otherwise ($Gap = (ub - lb)/lb$). The optimal solutions are obtained when CP is able to finish the search or when the gap to the lower bound is zero.

Table 4.5 summarises the results for the MMTSP, where the performance of the MIP model

is compared to the CP model, GA-MM and UB2 [Bianco et al., 1999].* The results are grouped by the number of humans (NH), the number of robots (NR) and robot eligibility (RE). There are ten instances for each combination $\langle \text{NH}, \text{NR}, \text{RE} \rangle$, two different values for the number of jobs per working cell (20, 50) and five values for multimode eligibility (0%, 25%, 50%, 75%, 100%). The results reported for CP, GA-MM and UB2 are the average of 10 executions. Each row in the table corresponds to the average results of 10 instances. Column “# Opt.” refers to the number of instances where each method found the optimal solution. Values in column “Gap (%)” which are significantly smaller than the value presented by the other methods are marked with a star (for a significance level of 0.05).

The MIP model was able to find and prove optimal solutions for only five instances. For the other 115, it has reached the time limit without even finding integer solutions. The CP model solved 26 instances to optimality, found the optimal solution of other 12, and provided an average gap of 3.1%. It reaches the time limit for 33 out of 40 instances with 100% of robot eligibility. The algorithm is highly influenced by the size of the solution space, as it applies an exploratory search on that space. Thus, the number of resources, robot eligibility and multimode eligibility are all relevant parameters to define the difficulty of the instances for that method.

Regarding the heuristics, the algorithms GA-MM and UB2 could find the optimal solutions for 25 and 5 problem instances, respectively. Thus, we observe a much better performance of GA-MM in comparison to UB2. Still, the number of resources and robot eligibility indeed influence the difficulty of the instances to these methods. On average, the results of both are far from the CP model, with average gaps of 12% for GA-MM and 22% for UB2. Moreover, they both could not find better solutions than CP for any of the instances generated.

*The MIP formulation proposed in Bianco et al. [1999] was implemented but did not find any integer solution.

Table 4.5 – Comparison of results between the proposed approaches and the algorithm proposed by Bianco et al. [1999] (UB2), on MMTSP instances that simulate HRTs.

NH	NR	RE	MIP Model			CP Model			GA-MM			UB2		
			# Opt.	Time (s)	Gap (%)	# Opt.	Time (s)	Gap (%)	# Opt.	Time (s)	Gap (%)	# Opt.	Time (s)	Gap (%)
1	2	25%	2	1621	0.0	9	765	0.3	7	637	0.4	2	1440	5.2
		50%	0	1800	-	7	922	0.4	6	1034	1.3	0	1800	12.0
		100%	1	1800	0.0	3	1275	2.2*	1	1687	6.2	0	1800	21.8
	3	25%	1	1621	0.0	7	1260	0.6	5	917	0.6	3	1264	2.6
		50%	1	1626	0.0	7	1068	0.4	4	1094	0.9	0	1800	12.0
		100%	0	1800	-	2	1648	3.6*	0	1800	14.6	0	1800	23.9
2	2	25%	0	1800	-	5	1800	0.6*	1	1634	2.8	0	1800	8.7
		50%	0	1800	-	1	1627	2.3*	0	1800	6.9	0	1800	17.0
		100%	0	1800	-	1	1625	3.6*	0	1800	14.8	0	1800	26.0
	3	25%	0	1800	-	5	1800	1.7*	1	1649	4.4	0	1800	9.9
		50%	0	1800	-	0	1800	2.8*	0	1800	10.1	0	1800	23.4
		100%	0	1800	-	1	1658	3.8*	0	1800	26.7	0	1800	35.5
			5	1756	0.0	47	1437	3.1	25	1471	12.0	5	1725	22.5

The next experiments consider the same instance set, but they include the additional SAC constraints. Table 4.6 reports a comparison between the results of MIP, CP and GA-MM to the MMTSP-SAC. As in Table 4.5, each row corresponds to the average results of 10 instances and values in column “Gap(%)” that are significantly smaller are marked with a star.

The three methods present better results for the variant of the MMTSP. The MIP model was able to prove optimality of 17 solutions instead of the five on the original problem, although it could not find integer solutions for the remaining instances. CP proves optimality for 62 solutions and finds the optimal solution for other six instances. The overall gap is quite low, around 2%. The existence of additional constraints favours the performance of the algorithm, as it can reduce the domain of the variables by filter propagation. As a result, the performance of the CP model is particularly good in the small instances (one human and $RE \leq 50\%$), being able to prove the optimality for 28 out of 40 solutions.

GA-MM is able to find the optimal solution for 50 instances. Considering only the instances with known optimal values, GA-MM presented an average gap of 0.2% - only in 4/68 cases, this gap was above 1%. However, for the whole HRT instance set, the gap is average 2.9%, as shown in the table. The results of GA-MM and CP are not significantly different for instances with only one human. Moreover, in this version of the problem, some solutions found by GA-MM were better than CP’s, especially for higher values of robot eligibility.

An extra set of instances was generated to compare results with the algorithm proposed by Chen et al. [2014] (GA-Chen). In this case, the precedence graph is composed of assembly phases to represent the power box assembly process. The results of GA-Chen are far from the other methods, with an average gap of 22.1% against 0.5% from CP and 0.7% from the proposed GA-MM. Moreover, GA-Chen could find only one optimal solution out of 36 instances, where CP found 35 and GA-MM 28.

Table 4.6 – Comparison of results between the proposed approaches on MMTSP-SAC instances.

<i>NH</i>	<i>NR</i>	<i>RE</i>	MIP Model			CP Model			GA-MM		
			# Opt.	Time (s)	Gap (%)	# Opt.	Time (s)	Gap (%)	# Opt.	Time (s)	Gap (%)
1	2	25%	2	1631	0.0	10	548	0.0	8	403	0.0
		50%	1	1800	0.0	8	461	0.6	7	694	0.8
		100%	3	1800	0.0	5	924	2.1	5	1312	2.7
	3	25%	2	1449	0.0	8	773	0.3	8	387	0.3
		50%	3	1442	0.0	8	552	0.4	8	469	0.4
		100%	1	1800	0.0	5	913	2.9	4	1315	2.5
2	2	25%	1	1800	0.0	3	1278	2.1*	3	1465	2.6
		50%	2	1800	0.0	5	973	2.6*	2	1596	4.0
		100%	1	1800	0.0	5	988	2.2*	2	1482	4.8
	3	25%	0	1800	-	2	1455	2.9	2	1445	3.1
		50%	0	1800	-	4	1216	3.9*	0	1802	5.3
		100%	1	1705	0.0	5	973	5.3*	1	1634	7.7
			17	1719	0.0	68	921	2.1	50	1167	2.9

4.6.1.2 General Instance Set

The second set of experiments aims to compare the performance of the methods in general and larger instances. The instance generation followed the process described by Bianco et al. [1999], where the number of resources varied from 5 to 15 and the number of jobs is up to 400. Each job may be executed in up to three modes, and the number of resources in each mode is sampled from $U(1,5)$. Given a mode m and a job j , p_{jm} is sampled from $U(1, k)$, where k is the maximum between 1 and $10/|R_m|$. Thus, there is a balance between processing times and the number of resources of each mode. As this procedure may generate redundant modes, before running each method we discharge mode m if exists a mode m' in which $R_{m'} \subset R_m$ and $p_{jm'} \leq p_{jm}$ for all jobs $j \in J$.

The results from the two lower bound methods were compared to the best-known solutions, resulting in an average RPD of 33% for LB2 and 20% for LB3. LB2 presented better average results than LB3 for the instances with a smaller relation between the number of jobs and the number of resources (20 jobs/5 resources and 50 jobs/10 resources). For the remaining instances, LB3 performed better. When comparing only to the known optimal solutions, the average RPD of LB2 is 18% while the average of LB3 is 22%.

Regarding the upper bound comparison, Table 4.7 presents the results of the proposed approaches and the heuristic from literature (UB2). The first and second columns contain the number of resources and jobs, and each row summarises average values for ten instances. The gap refers to the best lower bound for each instance, or the optimal value when it is known. Here, again, the MIP model presented a poor performance in comparison to the other methods, as it found only integer solutions for ten instances - the same for which the model prove optimality. In contrast, the CP model found the optimal solutions for 22 instances, with a relatively small gap when the number of resources is five.

The performance of the heuristics was again worse than the CP model, although GA-MM found better solutions than UB2 for almost all instances. GA-MM, as CP, found the optimal solutions for all instances with five resources and 20 jobs while UB2 found only one. In general, all methods depreciated the quality of the solutions found when the number of resources increases.

Table 4.7 – Comparison of results between the proposed approaches and the algorithm proposed by Bianco et al. [1999] (UB2), on general MMTSP instances.

$ R $	$ J $	MIP Model			CP Model			GA-MM			UB2		
		# Opt.	Time (s)	Gap (%)	# Opt.	Time (s)	Gap (%)	# Opt.	Time (s)	Gap (%)	# Opt.	Time (s)	Gap (%)
5	20	10	435	0.0	10	4	0.0	10	120	0.0	1	1620	8.3
	50	0	1800	-	5	1161	5.1*	0	1800	7.5	0	1800	28.6
	100	0	1800	-	0	1800	8.5*	0	1800	15.5	0	1800	42.1
	200	0	1800	-	0	1800	8.0*	0	1800	21.1	0	1800	40.4
10	50	0	1800	-	7	726	12.2*	0	1800	16.6	0	1800	44.9
	100	0	1800	-	0	1800	25.5*	0	1800	40.9	0	1800	78.4
	200	0	1800	-	0	1800	22.6*	0	1800	48.8	0	1800	75.3
	300	0	1800*	-	0	1800	24.8	0	1800	56.9	0	1800	74.2
15	100	0	1800	-	0	1800	32.4*	0	1800	52.4	0	1800	98.2
	200	0	1800	-	0	1800	28.2*	0	1800	63.9	0	1800	87.9
	300	0	1800	-	0	1800	28.8*	0	1800	71.3	0	1800	86.2
	400	0	1800	-	0	1800	30.6*	0	1800	75.9	0	1800	83.3
		10	1686	0.0	22	1508	18.9	10	1660	42.2	1	1785	62.3

As the CP model is able to find the optimal solution for instances with reasonable size, this method is now used to run an extensive set of experiments, in order to derive managerial insights on the advantage of collaborative work. The next section describes the results obtained in these experiments.

4.7. Advantage of Collaborative Work

In this section, we describe computational experiments performed to analyse how collaboration may improve the production workflow. Particularly, most of the experiments aim to estimate the makespan reduction when comparing settings with and without collaborative tasks. We conduct the analysis on HRTs working cells, the focus of this work, but it generally applies to settings where distinct types of resources execute multiprocessor tasks. We ran the CP model on a variety of parameter settings in order to analyse the effect of different problem characteristics. Moreover, by using the MMTSP-SAC, we aim to identify the characteristics that provide the largest improvement opportunities.

The experiments are based on small instances for the CP model to be able to prove optimality in seconds (preliminary tests showed that the charts are similar for larger instances). Thus, for all instances we considered 20 jobs per working space and two working spaces, i.e., 40 jobs in total. For each combination of parameters, we calculated the number of replications necessary for a 0.05 significance level. We concluded that 800 replications is

sufficient to guarantee an error of at most 0.005 for all experiments.

All charts in this section report the average makespan reduction (MR) calculated by:

$$MR = (m_1 - m_2)/m_1,$$

where m_1 and m_2 are the optimal makespan from two different problem instances.

4.7.1 Introducing Robots in Manual Lines

The objective of the first tests is to compare the makespan that results from environments composed only by humans to the makespan obtained with HRT. The purpose is to analyse the potential impacts of including cobots in manual working cells, considering different relative performances between humans and robots.

Results are shown in Figure 4.4, where m_1 refers to problem instances composed only by 1 human and m_2 refer to problem instances with 1 human and 2 robots. Therefore MR indicates the improvement obtained with the addition of robots, given that their participation is limited by robot eligibility. The value of parameter $MM\ Elig$ is equal to the value of RE .

As expected, the chart shows that including robots improves makespan ($MR > 0$). Moreover, the values of robot eligibility approximately correspond to the improvements obtained, i.e. 20% of gain for 20% of robot eligibility. This result motivates the use of cobots, even if they are only suitable for a small subset of jobs.

With increased robot eligibility, the influence of the robot's performance becomes noticeable. The curves approximate to the theoretical upper bound for the makespan reduction of introducing two resources in a setting composed of only one. This result reveals that the existence of precedence levels and different jobs processing times do not significantly reduce the expected improvement gains of introducing robots in the working cells.

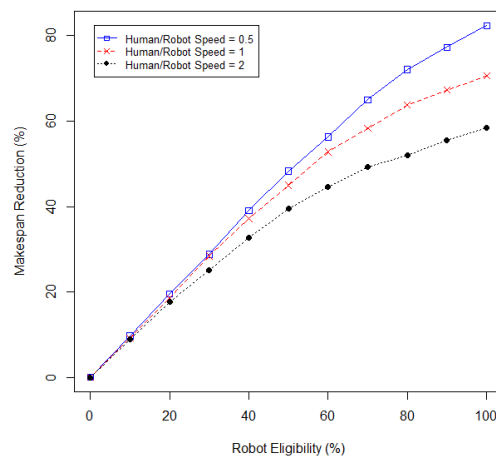


Figure 4.4 – Makespan reduction by introducing two robots in manual working cells

From this experiment onwards, all instances are composed of one human and two robots. The following charts report the average MR where m_1 and m_2 only differ on parameter $MM\ Elig$: in m_1 there is no collaborative task ($MM\ Elig = 0$). Hence it is reasonable to say that for these charts, the makespan reduction estimates the productivity gain directly provided by collaborative work. When not explicitly described, consider 100% of robot eligibility for both m_1 and m_2 and 100% of multimode eligibility for m_2 .

4.7.2 Multimode Eligibility and Performance

Figure 4.5 presents the average makespan reduction obtained with respect to multimode eligibility (4.5a) and collaborative operation time (4.5b).

The charts report results for three levels of robot eligibility: 20%, 50% and 100%. We observe that there is a trend toward more significant gains in scenarios with lower values of robot eligibility. This is explained by the fact that settings with lower robot eligibility tend to have higher idleness of robots, favouring the simultaneous use of two resources.

Leveraging the possibility of collaboration (multimode eligibility in Figure 4.5a) has a direct and positive impact on makespan reduction. Figure 4.5b shows that reducing collaborative time has a positive and linear impact on the scheduling performance measure. As more synergies are obtained by the simultaneous execution of a task, the better the final result in terms of overall completion time. The (close to) linear behaviour in both charts is observed for all values of robot eligibility.

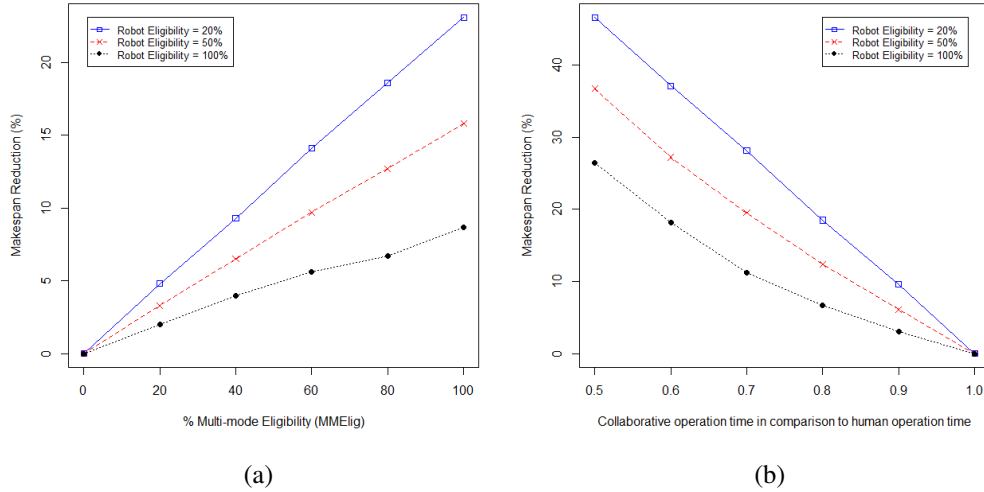


Figure 4.5 – Makespan improvement by introducing collaborative tasks

4.7.3 Precedence Levels

The precedence level is the number of precedence constraints in comparison to the number of jobs, where 0% means that no job has predecessors and 100% means that there is only

one possible sequence of execution of the jobs. This is one of the most structural characteristics of scheduling problems. Therefore, we study the impact of this parameter on the reduction of makespan when considering multimode eligibility.

This set of experiments has the objective to analyse the improvements obtained in different precedence levels and different relative performances between resources. Results are shown in Figure 4.6.

In Figure 4.6a, we observe that the precedence level is an essential factor to influence gains with collaboration for higher values of robot eligibility. Otherwise, if robots are suitable to operate on a limited set of jobs ($RE = 20\%$), the potential gains are noticeable higher (as in Figure 4.5), but they seem to be independent on the precedence level.

The chart in Figure 4.6b presents the makespan reduction by varying the precedence level on settings with different relationship between the performance of the resources. If humans are faster than or equal to robots ($Human/Robot\ Speed \geq 1$), improvements obtained with collaborative work increase as the precedence level increases. A possible reason for this behaviour is that the higher the number of precedence constraints, the longer the idleness of resources. Hence, there is an opportunity to reduce jobs completion time by assigning the collaborative modes.

In case robots are faster than human ($Human/Robot\ Speed < 1$), potential gains are not so expressive. This can be explained by the fact that when robots are faster, it is more advantageous to keep them working alone instead of making use of collaborative tasks.

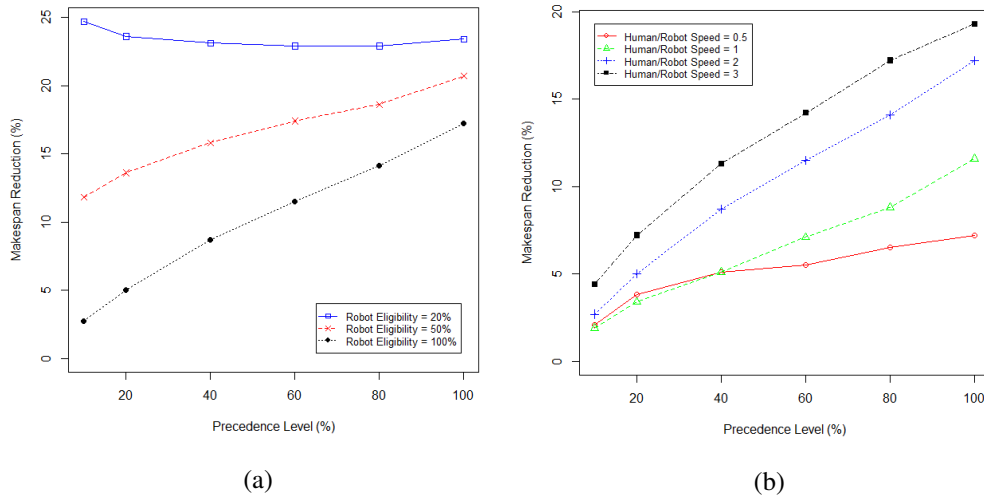


Figure 4.6 – Makespan reduction over precedence level. (a) Three levels of robot eligibility and Human/Robot Speed fixed to two (b) Different values for Human/Robot Speed and robot eligibility fixed to 100%.

4.7.4 Humans and Robots Relative Performance

The last set of experiments has as objective to further analyse the influence of the relative performance of resources (*Human/Robot Speed*) on improvement opportunities. We generated problem instances varying this parameter from 0.5 (the robots are faster than the human) to 15 (the human is in average 15 times faster than the robots). The collaborative operation time is 0.7 times the fastest resource.

The chart of Figure 4.7 presents three levels of robot eligibility and confirms that lower values of robot eligibility favour the use of collaborative work. We conclude that if the robots are faster than the human (*Human/Robot Speed* = 0.5), the makespan reduction is substantially dependent on the value of *RE*. However, this behaviour tends to be nullified as the human turns to be faster than robots. This result is similar to what we observed in the chart of Figure 4.6b. When robots are faster, it is not worth to make use of collaborative tasks, unless they cannot operate alone in jobs (i.e., low values of *RE*).

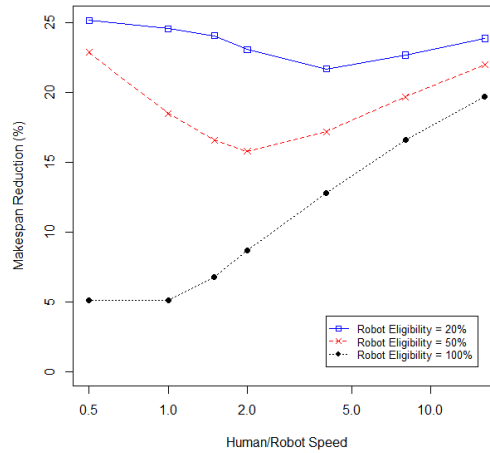


Figure 4.7 – Makespan improvement over the average relationship between robot and human performance

4.8. Conclusions and Future Research

This work presents one of the first steps in the direction of analysing improvement opportunities with Human-Robot Collaboration environments. We examined and extended problems described in robotics literature and explored the traditional scheduling literature with a problem that can be used in this context, the Multimode Multiprocessor Task Scheduling Problem, which has received little attention.

We propose a different Mixed Integer Programming formulation from the one existing in literature for the MMTSP and a variant with Symmetric Assignment constraints. We also implemented a Constraint Programming model that was able to find the optimal solution for 29% of the problem instances generated for the MMTSP and 52% in the case of the variant. In the same time limit, the CP model presented better results than the heuristics

found in the literature.

Moreover, a Linear Programming model provided tight lower bounds for the unsolved instances and also better results than literature. We proposed a Genetic Algorithm in order to generate good quality solutions for the set of larger instances of the MMTSP and the variant with Symmetric Assignment constraints. The approach resulted in an average gap of 1% to the known optimal solutions. We observed that the proposed Genetic Algorithm, GA-MM, performed better for problems with higher levels of robot eligibility in contrast with the better performance that the CP model presented for problems with low levels of robot eligibility. We believe that the heuristic may be enhanced in future work by incorporating some approaches found in the scheduling literature, such as a different scheduling generation scheme or alternative individual representations. Adaptations of state-of-the-art Tabu Search algorithms for scheduling are also promising ideas.

We also derive a variety of managerial insights, critical to companies that want to invest in HRTs solutions or are evaluating that possibility, which is expected to grow strongly in the near future. The computational experiments show that the possibility of collaborative work can provide significant improvement opportunities, particularly in settings with low robot eligibility or a large number of precedence constraints.

Because collaborative environments are subject to strict security regulations, it is meaningful to design a variant of the MMTSP for HRTs that considers different aspects of the standards regulations. Pursuing this path should lead to insights on some trade-offs between the use of traditional robots in separated environments and the use of cobots in safe environments for human workers. Further studies may also extend this work to consider ergonomic aspects in task allocation, contributing to the well-being of their workers. The improvement of working conditions and proper introduction of robots in the operators' environment will make manufacturers more attractive and socially responsible, as well as improve their productivity.

Bibliography

- Afzalirad, M., Rezaeian, J., 2016. Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. *Computers & Industrial Engineering* 98, 40 – 52. doi:<https://doi.org/10.1016/j.cie.2016.05.020>.
- Ajoudani, A., Zanchettin, A.M., Ivaldi, S., Albu-Schäffer, A., Kosuge, K., Khatib, O., 2018. Progress and prospects of the human–robot collaboration. *Autonomous Robots* 42, 957–975. URL: <https://doi.org/10.1007/s10514-017-9677-2>, doi:[10.1007/s10514-017-9677-2](https://doi.org/10.1007/s10514-017-9677-2).
- Bänziger, T., Kunz, A., Wegener, K., 2018. Optimizing human–robot task allocation using a simulation tool based on standardized work descriptions. *Journal of Intelligent Manufacturing* doi:[10.1007/s10845-018-1411-1](https://doi.org/10.1007/s10845-018-1411-1).

- Baptiste, P., Le Pape, C., Nuijten, W., 2001. Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. volume 39. Springer US. doi:[10.1007/978-1-4615-1479-4](#).
- Battaïa, O., Delorme, X., Dolgui, A., Hagemann, J., Horlemann, A., Kovalev, S., Malyutin, S., 2015. Workforce minimization for a mixed-model assembly line in the automotive industry. *International Journal of Production Economics* 170, 489 – 500. doi:[10.1016/j.ijpe.2015.05.038](#). current Research Issues in Production Economics.
- Bianco, L., Błażewicz, J., Dell’Olmo, P., Drozdowski, M., 1995. Scheduling multiprocessor tasks on a dynamic configuration of dedicated processors. *Annals of Operations Research* 58, 493–517. doi:[10.1007/BF02057160](#).
- Bianco, L., Dell’Olmo, P., Giordani, S., Speranza, M.G., 1999. Minimizing makespan in a multimode multiprocessor shop scheduling problem. *Naval Research Logistics* 46, 893 – 911. doi:[10.1002/\(SICI\)1520-6750\(199912\)46:8<893::AID-NAV2>3.0.CO;2-7](#).
- Bianco, L., Dell’Olmo, P., Speranza, M.G., 1998. Heuristics for multimode scheduling problems with dedicated resources. *European Journal of Operational Research* 107, 260 – 271. doi:[10.1016/S0377-2217\(97\)00347-0](#).
- Błażewicz, J., Dell’Olmo, P., Drozdowski, M., Maczka, P., 2003. Scheduling multiprocessor tasks on parallel processors with limited availability. *European Journal of Operational Research* 149, 377 – 389. doi:[10.1016/S0377-2217\(02\)00760-9](#). sequencing and Scheduling.
- Błażewicz, J., Dell’Olmo, P., Drozdowski, M., Speranza, M., 1992. Scheduling multiprocessor tasks on three dedicated processors. *Information Processing Letters* 41, 275 – 280. doi:[10.1016/0020-0190\(92\)90172-R](#).
- Bogner, K., Pferschy, U., Unterberger, R., Zeiner, H., 2018. Optimised scheduling in human–robot collaboration – a use case in the assembly of printed circuit boards. *International Journal of Production Research* , 1–19doi:[10.1080/00207543.2018.1470695](#).
- Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E., 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research* 112, 3 – 41. doi:[10.1016/S0377-2217\(98\)00204-5](#).
- Brucker, P., Jurisch, B., Krämer, A., 1997. Complexity of scheduling problems with multi-purpose machines. *Annals of Operations Research* 70, 57–73. doi:[10.1023/A:1018950911030](#).
- Caramia, M., Giordani, S., 2009. A new approach for scheduling independent tasks with multiple modes. *Journal of Heuristics* 15, 313–329. doi:[10.1007/s10732-007-9062-y](#).

- Caramia, M., Giordani, S., 2010. A fast metaheuristic for scheduling independent tasks with multiple modes. *Computers & Industrial Engineering* 58, 64–69. doi:[10.1016/j.cie.2009.08.004](https://doi.org/10.1016/j.cie.2009.08.004).
- Casalino, A., Zanchettin, A.M., Piroddi, L., Rocco, P., 2019. Optimal scheduling of human-robot collaborative assembly operations with time petri nets. *IEEE Transactions on Automation Science and Engineering* , 1–15doi:[10.1109/TASE.2019.2932150](https://doi.org/10.1109/TASE.2019.2932150).
- Chen, F., Sekiyama, K., Cannella, F., Fukuda, T., 2014. Optimal subtask allocation for human and robot collaboration within hybrid assembly system. *IEEE Transactions on Automation Science and Engineering* 11, 1065–1075. doi:[10.1109/TASE.2013.2274099](https://doi.org/10.1109/TASE.2013.2274099).
- Chen, J., Lee, C.Y., 1999. General multiprocessor task scheduling. *Naval Research Logistics* 46, 72–74. doi:[10.1002/\(SICI\)1520-6750\(199902\)46:1<57::AID-NAV4>3.3.CO;2-8](https://doi.org/10.1002/(SICI)1520-6750(199902)46:1<57::AID-NAV4>3.3.CO;2-8).
- Chen, J., Miranda, A., 2001. A polynomial time approximation scheme for general multiprocessor job scheduling. *SIAM Journal on Computing* 31, 1–17. doi:[10.1137/S0097539798348110](https://doi.org/10.1137/S0097539798348110).
- Darvish, K., Wanderlingh, F., Bruno, B., Simetti, E., Mastrogiovanni, F., Casalino, G., 2018. Flexible human-robot cooperation models for assisted shop-floor tasks. *Mechatronics* 51, 97 – 114. URL: <http://www.sciencedirect.com/science/article/pii/S0957415818300485>, doi:<https://doi.org/10.1016/j.mechatronics.2018.03.006>.
- Ding, H., Schipper, M., Matthias, B., 2014. Optimized task distribution for industrial assembly in mixed human-robot environments - case study on IO module assembly, in: 2014 IEEE International Conference on Automation Science and Engineering (CASE), pp. 19–24. doi:[10.1109/CoASE.2014.6899298](https://doi.org/10.1109/CoASE.2014.6899298).
- Drozdowski, M., 1996. Scheduling multiprocessor tasks - an overview. *European Journal of Operational Research* 94, 215–230. doi:[10.1016/0377-2217\(96\)00123-3](https://doi.org/10.1016/0377-2217(96)00123-3).
- Drozdowski, M., 2009. Parallel Tasks. Springer London, London. chapter 5. pp. 87–208. doi:[10.1007/978-1-84882-310-5_5](https://doi.org/10.1007/978-1-84882-310-5_5).
- Edis, E.B., Oguz, C., Ozkarahan, I., 2013. Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *European Journal of Operational Research* 230, 449 – 463. doi:[10.1016/j.ejor.2013.02.042](https://doi.org/10.1016/j.ejor.2013.02.042).
- Edis, E.B., Ozkarahan, I., 2011. A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. *Engineering Optimization* 43, 135–157. doi:[10.1080/03052151003759117](https://doi.org/10.1080/03052151003759117).
- Glover, F., 2000. Multi-Start and Strategic Oscillation Methods — Principles to Exploit Adaptive Memory. Springer US, Boston, MA. pp. 1–23. doi:[10.1007/978-1-4615-4567-5_1](https://doi.org/10.1007/978-1-4615-4567-5_1).

- Gökgür, B., Hnich, B., Özpeynirci, S., 2018. Parallel machine scheduling with tool loading: a constraint programming approach. *International Journal of Production Research* 56, 5541–5557. doi:[10.1080/00207543.2017.1421781](https://doi.org/10.1080/00207543.2017.1421781).
- Goldberg, D.E., 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Gombolay, M.C., Wilcox, R.J., Shah, J.A., 2018. Fast scheduling of robot teams performing tasks with temporospatial constraints. *IEEE Transactions on Robotics* 34, 220–239. doi:[10.1109/TRO.2018.2795034](https://doi.org/10.1109/TRO.2018.2795034).
- Guan, Y., Xiao, W.Q., Cheung, R.K., Li, C.L., 2002. A multiprocessor task scheduling model for berth allocation: heuristic and worst-case analysis. *Operations Research Letters* 30, 343 – 350. doi:[10.1016/S0167-6377\(02\)00147-5](https://doi.org/10.1016/S0167-6377(02)00147-5).
- Hancock, P.A., Billings, D.R., Schaefer, K.E., Chen, J.Y.C., de Visser, E.J., Parasuraman, R., 2011. A meta-analysis of factors affecting trust in human-robot interaction. *Human Factors* 53, 517–527. doi:[10.1177/0018720811417254](https://doi.org/10.1177/0018720811417254).
- Hartmann, S., Briskorn, D., 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* 207, 1 – 14. doi:[10.1016/j.ejor.2009.11.005](https://doi.org/10.1016/j.ejor.2009.11.005).
- Herrmann, J.W., Lee, C.Y., Hinchman, J., 1995. Global job shop scheduling with a genetic algorithm. *Production and Operations Management* 4, 30–45. doi:[10.1111/j.1937-5956.1995.tb00039.x](https://doi.org/10.1111/j.1937-5956.1995.tb00039.x).
- IFR, 2018. *Demystifying Collaborative Industrial Robots*. Technical Report. International Federation of Robotics. URL: <https://ifr.org>.
- IRF, 2019. *World Robotics*. Technical Report. International Federation of Robotics. URL: <https://ifr.org>.
- Jansen, K., Porkolab, L., 2005. General multiprocessor task scheduling: Approximate solutions in linear time. *SIAM Journal on Computing* 35, 519–530. doi:[10.1137/S0097539799361737](https://doi.org/10.1137/S0097539799361737).
- Johannsmeier, L., Haddadin, S., 2017. A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes. *IEEE Robotics and Automation Letters* 2, 41–48. doi:[10.1109/LRA.2016.2535907](https://doi.org/10.1109/LRA.2016.2535907).
- Krüger, J., Lien, T., Verl, A., 2009. Cooperation of human and machines in assembly lines. *CIRP Annals* 58, 628–646. doi:[10.1016/j.cirp.2009.09.009](https://doi.org/10.1016/j.cirp.2009.09.009).
- Lee, S., Kim, M., Song, J., 2015. Sensorless collision detection for safe human-robot collaboration, in: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2392–2397. doi:[10.1109/IRoS.2015.7353701](https://doi.org/10.1109/IRoS.2015.7353701).

- Leung, J.Y.T., Li, C.L., 2008. Scheduling with processing set restrictions: A survey. *International Journal of Production Economics* 116, 251 – 262. URL: <http://www.sciencedirect.com/science/article/pii/S0925527308003071>, doi:<https://doi.org/10.1016/j.ijpe.2008.09.003>.
- Liu, H., Wang, L., 2018. Gesture recognition for human-robot collaboration: A review. *International Journal of Industrial Ergonomics* 68, 355 – 367. doi:[10.1016/j.ergon.2017.02.004](https://doi.org/10.1016/j.ergon.2017.02.004).
- Mateus, J., Claeys, D., Limère, V., Cottyn, J., Aghezzaf, E.H., 2019. A structured methodology for the design of a human-robot collaborative assembly workplace. *The International Journal of Advanced Manufacturing Technology* 102. doi:[10.1007/s00170-019-03356-3](https://doi.org/10.1007/s00170-019-03356-3).
- Nikolakis, N., Kousi, N., Michalos, G., Makris, S., 2018. Dynamic scheduling of shared human-robot manufacturing operations. *Procedia CIRP* 72, 9 – 14. doi:[10.1016/j.procir.2018.04.007](https://doi.org/10.1016/j.procir.2018.04.007). 51st CIRP Conference on Manufacturing Systems.
- Pappis, C.P., Rachaniotis, N.P., 2010. Scheduling in a multi-processor environment with deteriorating job processing times and decreasing values: the case of forest fires. *Journal of Heuristics* 16, 617–632. doi:[10.1007/s10732-009-9110-x](https://doi.org/10.1007/s10732-009-9110-x).
- Pellegrinelli, S., Orlandini, A., Pedrocchi, N., Umbrico, A., Tolio, T., 2017. Motion planning and scheduling for human and industrial-robot collaboration. *CIRP Annals* 66, 1 – 4. doi:<https://doi.org/10.1016/j.cirp.2017.04.095>.
- Pinedo, M.L., 2008. *Scheduling - Theory, Algorithms and Systems*. 3rd ed., Springer.
- Pinto, G., Ainbinder, I., Rabinowitz, G., 2009. A genetic algorithm-based approach for solving the resource-sharing and scheduling problem. *Computers & Industrial Engineering* 57, 1131 – 1143. URL: <http://www.sciencedirect.com/science/article/pii/S0360835209001338>, doi:<https://doi.org/10.1016/j.cie.2009.05.003>.
- Statista, 2019. Share of traditional and collaborative robot unit sales worldwide from 2017 to 2021. Technical Report. Statista. URL: <https://www.statista.com/statistics/1018935/traditional-and-collaborative-robotics-share-worldwide/>.
- Switalski, P., Seredynski, F., 2015. Scheduling parallel batch jobs in grids with evolutionary metaheuristics. *Journal of Scheduling* 18, 345–357. doi:[10.1007/s10951-014-0382-0](https://doi.org/10.1007/s10951-014-0382-0).
- Tsang, E., 1993. *Foundations of Constraint Satisfaction*. 2 ed., Academic Press Limited. doi:[10.1016/C2013-0-07627-X](https://doi.org/10.1016/C2013-0-07627-X).
- Tsarouchi, P., Spiliotopoulos, J., Michalos, G., Koukas, S., Athanasatos, A., Makris, S., Chrysosolouris, G., 2016. A decision making framework for human robot collaborative workplace generation. *Procedia CIRP* 44, 228 – 232. doi:[10.1016/j.procir.2016.02.103](https://doi.org/10.1016/j.procir.2016.02.103). 6th CIRP Conference on Assembly Technologies and Systems (CATS).

- Türkoğullar, Y.B., Taşkın, Z.C., Aras, N., Altınel, İ.K., 2014. Optimal berth allocation and time-invariant quay crane assignment in container terminals. *European Journal of Operational Research* 235, 88 – 101. doi:[10.1016/j.ejor.2013.10.015](https://doi.org/10.1016/j.ejor.2013.10.015).
- Weber, A., 2018. State of the Profession 2018: Smart Factories Demand Smarter Engineers. Technical Report. Assembly. URL: <https://www.assemblymag.com/articles/94380-state-of-the-profession-2018-smart-factories-demand-smarter-engineers?>
- Weglarz, J., Drabowski, M., Błażewicz, J., 1986. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Transactions on Computers* 35, 389–393. doi:[10.1109/TC.1986.1676781](https://doi.org/10.1109/TC.1986.1676781).
- Xu, W., Tang, Q., Liu, J., Liu, Z., Zhou, Z., Pham, D.T., 2020. Disassembly sequence planning using discrete bees algorithm for human-robot collaboration in remanufacturing. *Robotics and Computer-Integrated Manufacturing* 62, 101860. URL: <http://www.sciencedirect.com/science/article/pii/S0736584519301887>, doi:<https://doi.org/10.1016/j.rcim.2019.101860>.
- Zanchettin, A.M., Casalino, A., Piroddi, L., Rocco, P., 2019. Prediction of human activity patterns for human–robot collaborative assembly tasks. *IEEE Transactions on Industrial Informatics* 15, 3934–3942. doi:[10.1109/TII.2018.2882741](https://doi.org/10.1109/TII.2018.2882741).

Appendix 4.A Proposed Genetic Algorithm

4.A.1 Decoding Algorithm

This section details the decoding procedure of the Genetic Algorithm proposed in this work. The pseudo-code is described in Algorithm 3.

The algorithm starts with an ordered list of jobs $J^{schedule}$ to be scheduled and an empty set of pending jobs J^{wait} . As a job j is removed from $J^{schedule}$, the algorithm includes it in the solution. This allocation only occurs if all predecessor jobs have previously been included in the solution. Otherwise, j is temporarily included in J^{wait} .

To include a job j assigned to mode m in a partial solution, the algorithm defines as starting time the first time when all resources in R_m are available, and all precedent jobs have finished. At each iteration, the algorithm verifies if any of the pending jobs may return to the ordered list $J^{schedule}$. If so, those jobs are placed again at the beginning of $J^{schedule}$.

4.A.2 Genetic Algorithm - General Structure

This section presents an overall structure of the Genetic Algorithm. The pseudo-code is found in Algorithm 4. Each run starts by initialising the population P with p individuals randomly generated.

```

1  $J^{schedule} \leftarrow$  Jobs sorted according to  $v$ -order;
2  $J^{wait} \leftarrow \emptyset$ ;
3 while  $|J^{schedule}| > 0$  do
4    $j \leftarrow$  Pop front  $J^{schedule}$ ;
5   if Predecessors of  $j$  are scheduled then
6     Include  $j$  in solution according to the assignment in  $v$ -modes;
7   else
8      $J^{wait} \leftarrow J^{wait} \cup \{j\}$ ;
9   end
10  for each job  $j$  in  $J^{wait}$  do
11    if Predecessors of  $j$  are scheduled then
12      Pop  $j$  from  $J^{wait}$ ;
13      Push front  $j$  in  $J^{schedule}$ ;
14    end
15  end
16 end

```

Algorithm 3: Decoding Algorithm

The best solution for each run is S^{run} . At each generation, the algorithm updates S^{run} and initialises the generation counter if it has found a better solution. After that, it applies the crossover and mutation operations to P and includes the new individuals in set P^{new} , the population of the following generation.

With a frequency f , a local search procedure is applied to ls individuals randomly chosen from the best $b\%$ in P . This mechanism generates ls new individuals that are incorporated to P^{new} . In this case, the worst individuals in P^{new} are removed, so the algorithm ensures that the next population size will not be larger than p .

At the end of each generation, the current population P is completely updated according to the individuals in the set P^{new} and the algorithm proceeds to a new generation. The stopping criterion is to find a target solution or reach a time limit. The algorithm returns the best solution found S^* .

Appendix 4.B Structure of Benchmark Algorithms

4.B.1 Lower bound LB2

The lower bound LB2 proposed by Bianco et al. [1999] is based on $\mathcal{G} = (J, E, W)$, the *constraint graph*, where the vertex set J is composed of the jobs, the edge set E represents the constraints between jobs and W assigns a weight π_j to each vertex $j \in J$:

$$\pi_j = \min_{m \in M_j} \{p_{jm}\}$$

An edge $(i, j) \in E$ indicates that jobs j and i can not be processed simultaneously. In that

```

1  $S^* \leftarrow \emptyset$ ;
2 while not (Time Limit) do
3    $S^{run} \leftarrow \emptyset$ ;
4   Initialize population P;
5    $gen \leftarrow 1$ ;
6   while  $gen < MaxG$  do
7      $S \leftarrow \text{Best individual in } P$ ;
8     if  $S$  better than  $S^{run}$  then
9        $gen \leftarrow 1$ ;
10       $S^{run} \leftarrow S$ ;
11    end
12     $P^{new} \leftarrow \text{Crossover in } P$ ;
13     $P^{new} \leftarrow P^{new} + \text{Mutation in } P$ ;
14    if  $gen \bmod f = 0$  then
15       $P^{new} \leftarrow P^{new} + \text{Local search in } ls \text{ individuals in } P$ ;
16      Remove worst individuals in } P^{new};
17    end
18     $P \leftarrow P^{new}$ ;
19     $gen \leftarrow gen + 1$ ;
20  end
21  if  $S^{run}$  better than  $S^*$  then
22     $S^* \leftarrow S^{run}$ ;
23  end
24 end

```

Algorithm 4: Genetic Algorithm

case, one of the following two conditions is met.

- i Precedence: There is a path in the precedence graph between jobs j and i
- ii Common resources: $R_m \cap R_{m'} \neq \emptyset$ for each pair of modes $m \in M_j, m' \in M_i$

Given that $\mathcal{W}$ is the maximum weighted clique in $\mathcal{G}$, LB2 is calculated as follows:

$$LB2 = \sum_{j \in \mathcal{W}} \pi_j$$

This lower bound is proved to dominate LB1 and LB1r, the other methods proposed by Bianco et al. [1999].

4.B.2 Upper bound UB2

As UB2 is a constructive heuristic, we implemented it as a randomised algorithm with a multi-start procedure that returns the best solution found. In this section, we briefly describe both implementations.

The algorithm is a stochastic constructive heuristic that inserts jobs into solution according to an increasing simulation time t . At each iteration, the candidate list is composed of all jobs available at the current time t , given that the availability of a job depends on its predecessors and the availability of the resources. When no candidate is available at the current simulation time, t is updated to the next feasible time.

The fitness function $b_{j,m}$ of UB2 prioritises jobs and modes with higher suitability to be assigned. The suitability of a pair $\langle j, m \rangle$ depends on the processing time of job j if executed in mode m in comparison to the other pairs. UB2 uses a roulette selection to sample the pair $\langle j, m \rangle$ to be inserted into solution, where the probability of selection of a pair depends on $b_{j,m}^\alpha$. The value of α defines the randomness of the algorithm, the larger, the more greedy.

Algorithm 5 describes the general structure of UB2. There is a list of unscheduled jobs $\mathcal{T}0$ initialised with the entire jobs set J . The algorithm executes until $\mathcal{T}0$ is empty. At each iteration, a candidate list $\mathcal{T}C$ is built based on the fitness function $b_{j,m}$. A job j and a corresponding execution mode m is sampled from $\mathcal{T}C$ and included in the current solution S^{run} . The stopping criteria is to find a target solution or to reach a time limit. The algorithms return the best solution found S^* . The parameter α was set to 1 after preliminary tests.

4.B.3 Genetic Algorithm (GA-Chen)

The benchmark Genetic Algorithm is very high-speed, even with a large number of generations. Thus we also used a multi-start procedure attempting to compare it with the proposed

```

1  $S^* \leftarrow \emptyset$ ;
2 while not (Time Limit) do
3    $\mathcal{T}0 \leftarrow J$ ;
4    $S^{run} \leftarrow \emptyset$ ;
5   while  $\mathcal{T}0$  not  $\emptyset$  do
6      $\mathcal{TC} \leftarrow \text{Select candidates } \langle j, m \rangle \text{ from jobs in } \mathcal{T}0$ ;
7     Sort  $\mathcal{TC}$  according to  $b_{j,m}$  and  $\alpha$ ;
8     Sample  $\langle j, m \rangle$  from  $\mathcal{TC}$ ;
9     Insert job  $j$  assigned to mode  $m$  into  $S^{run}$ ;
10    Pop  $j$  from  $\mathcal{T}0$ ;
11  end
12  if  $S^{run}$  better than  $S^*$  then
13     $S^* \leftarrow S^{run}$ ;
14  end
15 end

```

Algorithm 5: Adaptation of UB2

methods in the same time limit. A slight adaptation was also performed to incorporate robot eligibility.

The method presented in [Chen et al. \[2014\]](#) runs an independent genetic algorithm for each working space, generating n different solutions. The individual representation is an assignment of jobs to resources, and the sequence of jobs is greedily calculated based on the assignment decision. The population size is fixed to two individuals. In the first generation, one individual represents an assignment of all jobs to the human, and the other is an assignment of all jobs to the robot. The final solution of each run is obtained by merging the partial solutions generated for each working space, ensuring that the *Symmetric Assignment Constraints* are respected. The process is repeated until the stopping criterion is met.

Dynamic scheduling of Human-Robot Teams

Managing Human-Robot Teams in real-time with dispatching rules

Cristiane Ferreira^{*†} · Gonalo Figueira^{*†} · Pedro Amorim^{*†}

Submitted to *IJPE*

Abstract

Collaborative robots are capable of working next to humans, forming Human-Robot Teams (HRTs). They benefit from the complementarity of these resources but need to be synchronised to ensure a smooth-paced production. This goal is challenging due to the variable execution time of human workers. In addition, the settings in which these teams operate are often dynamic, having order arrivals continuously along time. Effective methods for dynamic scheduling of HRTs are therefore necessary to react in real-time. This work develops dispatching rules for scheduling HRTs, which are fast procedures for sequencing jobs. Genetic Programming, an evolutionary approach, is used as a learning mechanism of the rules. This method is able to balance efficacy and interpretability. The evolved rules are evaluated in a diversified instance set representing a wide range of HRTs applications. We consider both dynamic and static job arrivals, and four performance metrics are analysed. Computational experiments show that the generated rules are considerably more effective than the best existing ones. Moreover, they are easy to understand and generalise well to very distinct settings, even in the more well-studied parallel machine case. Finally, a comparison to exact methods is also included. Results show that dispatching rules outperform near-optimal (static/deterministic) solutions under medium-high uncertainty levels.

Keywords Dynamic Scheduling · Human-Robot collaboration · Dispatching Rules · Genetic Programming · Explainable Artificial Intelligence

^{*}Faculdade de Engenharia, Universidade do Porto, Porto, Portugal

[†]INESC TEC - Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ci4ncia, Porto, Portugal

5.1. Introduction

Manufacturing systems have been subject to a disruptive transformation with the advent of Industry 4.0. New technologies leverage production efficiency towards higher customisation and competitiveness requirements [Olsen and Tomlin, 2020]. Particularly, advances in robotics and artificial intelligence resulted in a new generation of robots capable of sharing the working spaces with humans – the so-called collaborative robots (cobots). These Human-Robot Teams (HRTs) increase flexibility in the shop-floor with new possibilities such as quick changeability of tasks. While humans are more adaptive to changes and suitable to complex tasks, robots are precise and work without fatigue. For these reasons, cobots have been receiving a lot of attention in recent years from academia [Ivanov et al., 2020] as well as practitioners [Weber, 2018]. In 2019, when robots sales suffered a setback, cobots installations, in contrast, grew by 11% and reached 4.8% of the total market [IFR, 2020].

The term “Human-Robot Collaboration” is generally used when humans and robots work alongside, although they may interact at distinct levels. In some cases, there is no shared area in the working cell. In other cases, collaborative operations are present, requiring both operators to work simultaneously on the same task. In the highest level of interaction, humans and robots share the workspace and components, reacting to each other in real-time. This is the ultimate goal of collaborative environments and the most difficult to achieve [IFR, 2018].

In collaborative applications, some tasks should be executed separately by each operator, but they must also be synchronised for occasional collaborative operations [Casalino et al., 2019]. Moreover, there are dependencies between tasks. Thus, if an operator takes longer to finish a given task, the others may have to wait. Other than that, one can take alternative tasks to avoid delays in the overall production setting. This re-scheduling has to be made frequently, as humans’ processing times are generally uncertain. Thus, managing schedules is challenging in these settings. Indeed, generally, production scheduling tends to be more complex with the new technologies from Industry 4.0 [Parente et al., 2020].

Effective methods are necessary to adapt schedules in real-time, particularly in these HRT applications. Some initiatives are found in the literature, as in Zanchettin [2021], where the authors simulate an assembly system composed of humans and robots and propose an algorithm for generating dispatching policies that are used for real-time scheduling. Pupa et al. [2021] propose a framework for human-robot interaction that incorporate algorithms for dynamic scheduling. However, the focus of these works is on the validation of the approach in real scenarios. The algorithms for adapting schedules under deviations usually enumerate all alternatives and thus are able to solve only extremely small instances. Moreover, the authors do not benchmark their methods with existing algorithms for dynamic scheduling.

Scheduling under uncertainty is indeed a long-studied topic, well beyond HRTs [Aytug et al., 2005]. Studies can be classified as proactive, by developing robust schedules, or reactive, working on the adaptation of schedules in real-time. To tackle this problem, dis-

patching rules have been used with good results in job shops and other settings [Lawrence and Sewell, 1997, Romero-Silva et al., 2018, Sweeney et al., 2019]. As simple sequencing procedures, they are flexible, easy to implement, and extremely fast to execute. The evident limitation is their generally low effectiveness in comparison to more sophisticated heuristics. These rules have been traditionally derived by empirical or analytical studies, fed by strong domain expertise. Recently, research has been conducted on the automated design of rules using machine learning and evolutionary algorithms. Most of the works on the topic create dispatching rules via Genetic Programming, a hyper-heuristic that evolves mathematical expressions [Nguyen et al., 2019]. In Ferreira et al. [2021a], interpretable rules for dynamic job shops are evolved by an iterative process. Authors use insights from scheduling literature to guide the algorithm towards effective and meaningful rules.

This work is the first on the generation of specific dispatching rules for HRTs. Previous works on real-time HRTs scheduling are considered to cover the problems and settings therein. Our objective is to develop rules that are, at the same time, effective in a wide spectrum of settings and easy to interpret. In the underlying scheduling problem, humans and robots are resources that must execute a set of operations, some of them in collaboration. The problem is modelled as a Multi-processor Task Scheduling Problem (MTSP), a generalisation of the Parallel Machine Scheduling Problem (PMP) in which some operations must be executed by two resources. We consider two variants of the problem, with and without dynamic arrivals of jobs. The most common performance criteria in the scheduling literature are taken for each case. Under static arrivals, the aim is to minimise the makespan and mean completion time. Contrary, if arrivals are uncertain and dynamic, we consider the mean tardiness and mean flowtime.

Genetic Programming is applied to the design of the rules, extending the work from Ferreira et al. [2021a] to the context of HRTs. The evolved rules are compared to the best existing ones for each criterion. We consider working cells with a distinct number of humans and robots, distinct distribution of tasks for each resource type, congested and light loaded settings. Since uncertainty in the processing time of tasks executed by humans is a key aspect, we analyse the influence of the uncertainty level in the performance of the evolved rules. We also compare the proposed approach to existing effective methods for the deterministic version of the MTSP and to a proactive-reactive approach. Finally, we derive insights on the most suitable settings for applying these reactive methods and gains obtained in comparison to near-optimal solutions for the deterministic case.

The paper is organised as follows. Section 5.2 describes related work on scheduling HRTs. We also present existing literature on dispatching rules for production scheduling and the use of Genetic Programming for the automated generation of rules. In Section 5.3, the two variants of the problem and the performance criteria are presented. The methodology adopted is detailed in Section 5.4, where we explain the HRTs simulator, the Genetic Programming algorithm and the design of the experiments. The presentation of the results is divided into two sections. In Section 5.5, we compare the performance of the generated rules to the best existing ones. An analysis of the effectiveness of dispatching rules against near-optimal solutions generated for the deterministic case is provided in Section 5.6. The motivation for such analysis is to provide insights on which settings are suitable for the use

of reactive methods instead of proactive ones. Conclusions and future work are given in Section 5.7.

5.2. Related work

Our work is at the intersection of two related research areas: HRTs scheduling and dispatching rules. Therefore, we review the literature on both of them. At first, we summarise the studies on scheduling HRTs, focusing on real-time decision making. Then, we review the general literature on dispatching rules for a variety of problems, and using different methods.

5.2.1 Scheduling Human-Robot teams

Although collaborative settings have received considerable attention from robotics in the past ten years, few works propose optimisation methods for those environments. Some studies deal with a deterministic context, such as [Bogner et al. \[2018\]](#), who propose two heuristics and a matheuristic for a use case of printed circuit boards assembly. The environment consists of working cells composed of a single robot and one or more humans. A distinct setting is presented in [Chen et al. \[2014\]](#), which describe an assembly cell composed of a human and one to three robots and develop a Genetic Algorithm to schedule them. [Bänziger et al. \[2018\]](#) and [Xu et al. \[2020\]](#) also propose heuristics for scheduling HRTs working cells. The former developed a Genetic Algorithm for optimising the cycle time in cells composed of one human and one robot. The latter design a Bee algorithm for disassembly tasks and minimise three metrics: labour costs, difficulty in executing tasks, and cycle time. Recently, [Ferreira et al. \[2021b\]](#) propose a Constraint Programming model and a Genetic Algorithm for scheduling HRTs that outperform previous algorithms. Precedence constraints are present in all works, and the optimisation criterion is generally minimising the makespan. Additional objectives are also studied, such as minimising the walking distance [[Bänziger et al., 2018](#)].

Real-time scheduling of HRTs is also tackled. [Johannsmeier and Haddadin \[2017\]](#) and [Darvish et al. \[2018\]](#) propose an algorithm for assigning and scheduling tasks in real-time—they may be executed by humans, robots or by both simultaneously. In both works, the operation time of tasks executed by humans is uncertain, although there is no further detail on how uncertainty is modelled. Both algorithms perform exhaustive searches on the solution space. Some works include prediction and monitoring of human behaviour to evaluate alternatives. In [Casalino et al. \[2019\]](#), a dispatcher uses a Petri Net model to simulate HRTs and optimise the makespan. Given the prediction of human activities, the algorithm assigns tasks to the robot to synchronise both resources and avoid human idleness. Whenever the robot is available, a dispatching rule is applied to select the best task. This approach has been expanded by [Zanchettin \[2021\]](#), incorporating a digital twin of a high-mix multi-product setting. A throughput controller is attached and, using a hierarchical clustering algorithm, generates dispatching policies in real-time. In the framework

of Pupa et al. [2021], an initial schedule is generated using a Mixed Integer Programming model that optimises two criteria, the mean completion time and the suitability of tasks to humans. During the execution, the tool monitors human activities and adapts the schedule accordingly to avoid idleness.

Recently, Reinforcement Learning (RL) has also been considered. In Liu et al. [2021], a dual-agent system is proposed, containing one RL agent for the human and the other for the robot. The robot's agent is trained using Deep Q networks, and the human is represented by simulation. Results show the feasibility of the approach, but there is no comparison with other methods. Zhang et al. [2022] design an RL algorithm for assigning tasks to a human and a robot (that work in the same cell). Two agents generate actions for each of the resources. Noise is incorporated into the human agent behaviour to represent human fatigue and other uncertain factors. The objective is to minimise the overall tasks difficulty and the makespan. The authors benchmark their algorithm with a multi-agent deep deterministic policy and show that the proposed method's learning process is more effective. However, the approach is validated in a unique instance of 18 operations, and no comparison on the performance metrics is shown.

The works on real-time scheduling referred above focus on providing sufficient empirical evidence. They validate their approaches in real verification scenarios. However, the optimisation algorithms are extremely simple, greedy heuristics or exhaustive enumeration of all possible alternatives except for the RL approaches. The effectiveness of the methods is not compared to existing optimisation methods, and the design of the experiments does not follow the scheduling literature. As most of these algorithms execute an exhaustive search, the number of jobs considered is limited to ten.

We aim at overcoming these limitations by proposing effective dispatching rules to assign and sequence tasks in real-time. These are computationally cheap methods extensively used in dynamic scheduling as a reactive approach. Next, we summarise the best existing dispatching rules and the main achievements on the use of GP.

5.2.2 Dispatching rules and GP for production scheduling

Dispatching rules are used in production scheduling for prioritising jobs in real-time. Once a resource is available, the rule assigns a score for each job waiting to be processed in a queue, usually based on the jobs' properties or system status. There is a long tradition on the study of the rules behaviour, particularly in job shops [Ramasesh, 1990]. However, most of the proposed rules are designed for job shops. Several works have studied the rules performance and show that one of the most critical variables is the next operations' queue [Raghu and Rajendran, 1993, Xiong et al., 2017, Sweeney et al., 2019]. This does not apply to collaborative working cells, which have a single queue. Therefore, we summarise here only existing rules that may be used for the settings studied in this work.

Lawrence and Sewell [1997] is a seminal study on dispatching rules for optimising the makespan. In that work, the results of seven rules are compared to the optimum and a sophisticated heuristic, the Shifting Bottleneck. Among the studied rules are the Short-

est Processing Time (SPT), the Most Operations Remaining (MOR), and the Most Work Following (MWF) that prioritises jobs according to the remaining processing time on successor machines. The authors simulate job shops with uncertain processing times and static arrivals and conclude that dispatching rules are competitive to other methods under high uncertainty levels. Later on, [Ferreira et al. \[2020\]](#) generated rules with GP that outperform those classical ones. The authors also extended the work of [Lawrence and Sewell \[1997\]](#) by incorporating sequence-dependent setup times.

In environments with dynamic arrivals, the mean flowtime and mean tardiness measures have been extensively studied. Past research has shown that rules' performance varies depending on how congested the shop-floor is and how tight due dates are [[Raghu and Rajendran, 1993](#)]. For this reason, designing a single rule for minimising the mean tardiness in all load conditions is challenging. The SPT, EDD (earliest due date) and MS (minimum job slack) are examples of rules that work well in some particular settings but not in general. Some authors have developed adaptive approaches, as the MDD (modified due date), CR+SPT and S/RPT+SPT (these two are adaptive rules designed by [Anderson and Nyirenda \[1990\]](#)). These rules adapt the score calculation as due dates get closer. For minimising the mean flowtime in classical settings, SPT is the best [[Romero-Silva et al., 2018](#)].

In the pursuit of more effective dispatching rules, machine learning has been recently applied with promising results. An example is [Jun et al. \[2019\]](#), where the authors generate Random Forests of decision trees to select rules. [Luo \[2020\]](#) proposes a Reinforcement Learning approach that use Neural Networks to select dispatching rules. Despite the good performance of such algorithms for some cases, most of them generate complex models that frequently do not generalise well to unseen scenarios. In this work, we use GP as the learning method, focusing on the generation of interpretable and effective rules.

Genetic Programming has been applied to production scheduling with promising results [[Nguyen et al., 2017](#)]. [Hildebrandt et al. \[2010\]](#) and [Branke et al. \[2015\]](#) developed GP algorithms to minimise mean flowtime and show that their results outperform classical dispatching rules. In [Nguyen et al. \[2015\]](#) and [Nguyen et al. \[2019\]](#), the authors found effective rules for several performance metrics –including mean tardiness – but they trained specific rules for each load condition. To our knowledge, [Karunakaran et al. \[2017\]](#) and [Ferreira et al. \[2020\]](#) are the only authors focusing on shops with stochastic processing times. However, the former does not benchmark the evolved rules with existing methods, and the latter considers static job arrivals. [Tay and Ho \[2008\]](#) and [Nie et al. \[2013\]](#) evolve rules for the Flexible Job Shop Scheduling. Despite minimising tardiness and flowtime, typically measured in dynamic arrivals of thousands of jobs, the former validates their rules in static settings and the latter in settings with only 50 jobs. Most of these works have a relevant drawback; they involve extensive computational experiments and little input from domain expertise, resulting in complex and meaningless rules. In [Ferreira et al. \[2021a\]](#), a methodology is proposed for the development of interpretable dispatching rules using GP. The authors follow an iterative process, where lessons from scheduling literature and previous experiments guide the search for more effective rules. Moreover, by using bloat control, selecting meaningful terminals and fixing the rules structure, the evolved rules are

compact and easily interpreted.

Very few studies are found in parallel machine environments. [Su et al. \[2017\]](#) adapt the Apparent Tardiness Cost rule to minimise the weighted tardiness and generate static schedules. However, the designed rule involves the preliminary estimation of several parameters. [Đurasević et al. \[2016\]](#) evolve priority rules for scheduling unrelated parallel machines in deterministic settings. Four criteria are considered: the makespan, weighted tardiness, weighted number of tardy jobs and mean flowtime. In [Đurasević and Jakobović \[2019\]](#) and [Jaklinović et al. \[2021\]](#), the authors extend their previous work by applying GP on the generation of the rules. The former evolve ensembles of dispatching rules and the latter incorporates additional constraints as eligibility, setup times and unavailability of machines. Both works present extensive experiments and a thorough analysis on the performance of the results obtained. However, the authors do not present examples of the evolved rules. All these studies validate their methods in simulations of static arrivals.

With this work, we aim at covering many issues not addressed by the literature described above. First, most works focus on the classical Job Shop Scheduling Problem; only a few design rules for the classical static PMP (without relevant constraints as machine eligibility) and HRTs are overlooked. Thus, specific rules must be designed for HRTs settings and for prioritising collaborative operations, where two resources must be synchronised to execute tasks. Moreover, the extensive literature on GP has focused on the algorithm efficiency and the generation of well-performing dispatching rules for particular settings, neglecting their interpretability and generalisability. Finally, we found no studies on dynamic job arrivals on parallel machines environments, which is necessary for measuring tardiness and flowtime with statistical significance [[Raghu and Rajendran, 1993](#)].

5.3. Problem definition

We consider a working cell operated by human workers and robots. Some tasks may be executed by each resource separately or by an HRT (one human and one robot working in collaboration). The problem may be modelled as a particular case of the Multi-processor Task Scheduling Problem (MTSP), where some operations must be executed simultaneously by two resources. The MTSP is NP-hard since even a particular case, without the assignment decision, is NP-hard [[Drozdowski, 2009](#)].

We denote by $\mathcal{M}$ a set of resources partitioned into two subsets, humans $\mathcal{H}$ and robots $\mathcal{R}$. A set of jobs $\mathcal{J}$ must be executed, and each job $j \in \mathcal{J}$ has a set of operations O_j . There are two types of operations: single-processor (executed by a single resource) and collaborative (or duo-processor, which always involves one human and one robot). Each single-processor operation o has a set of eligible resources $M_o \subseteq \mathcal{M}$. Each collaborative operation may be executed by any HRT, i.e., any pair $\langle h, r \rangle$, where $h \in \mathcal{H}$ and $r \in \mathcal{R}$. The processing time of each single-processor operation o if executed by resource $m \in M_o$ is given by p_{om} , while the processing time of a collaborative operation is p_o . For each job j , there are precedence constraints between its operations O_j . One must assign resources to operations and sequence them without overlap in any resource. The goal is to optimise one

of four metrics: makespan, mean completion time, mean flowtime and mean tardiness.

Figure 5.1 illustrates an example of a working cell composed of two robots (r_1 and r_2) and one human (h). Note that there are two HRTs: h working along with r_1 and h working with r_2 . In the example, three jobs, each with three operations, must be executed (j_o is operation o of job j). Depending on the precedence constraints, operations of the same jobs may be executed in parallel, as o_{22} and o_{23} . A feasible solution is presented where the human executes alone operations o_{11} and o_{33} ; the first robot executes o_{21} and o_{32} , whereas the second executes o_{23} , o_{31} and o_{13} . Operations o_{22} and o_{12} are executed by each of the HRTs.

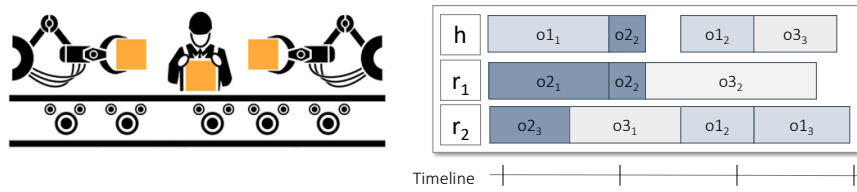


Figure 5.1 – An hybrid working cell and its schedule

The presence of humans adds flexibility to automated environments, as they can adapt to deviations in real-time. However, humans are not as precise as robots in the execution of tasks and introduce unpredictable behaviour. This is particularly relevant in collaborative environments where HRTs need to be synchronised. Thus, processing times are stochastic if tasks are executed by humans or HRTs.

Some collaborative settings consider a set of tasks known in advance [Chen et al., 2014, Xu et al., 2020, Pupa et al., 2021]. Other works, such as Zanchettin [2021], deal with dynamic job arrivals. This is a relevant distinction in scheduling research. If arrivals are static, new jobs are admitted to the shop-floor only after the existing scheduling cycle. A continuous stream of arriving jobs is released to the shop-floor and dispatched for execution in a dynamic model. In order to tackle both static and dynamic cases, this work considers two variants of the MTSP, a static set of jobs (MTSP-S) and dynamic job arrivals (MTSP-D). Because of their different nature, different performance metrics are considered in each case.

In the MTSP-S, a set of jobs is released to the shop-floor at the beginning of the planning horizon. This is the case of cyclic production, where the most common goal is to optimise the completion of the entire set of jobs, i.e., the makespan. Some works also refer to the need to minimise the mean completion of the jobs. Both metrics, the makespan C_{max} and the mean completion time $\bar{C}$, are given by Equation (5.1) and Equation (5.2), respectively (where c_j is the completion time of each job j).

$$C_{max} = \max_{j \in \mathcal{J}} \{c_j\} \quad (5.1)$$

$$\bar{C} = \frac{\sum_{j \in \mathcal{J}} c_j}{|\mathcal{J}|} \quad (5.2)$$

In the MTSP-D, each job has a specific and unknown release time r_j , and one must optimise a time or a due-date related criterion. In such settings, the most common performance metrics are the mean flowtime $\bar{F}$ and the mean tardiness $\bar{T}$. The flowtime refers to the time from job release to completion. The tardiness of a job is the difference between its completion time and due date d_j . Both metrics are calculated as in Equations (5.4) and (5.3).

$$\bar{F} = \frac{\sum_{j \in \mathcal{J}} (c_j - r_j)}{|\mathcal{J}|} \quad (5.3)$$

$$\bar{T} = \frac{\sum_{j \in \mathcal{J}} \max(0, c_j - d_j)}{|\mathcal{J}|} \quad (5.4)$$

Table 5.1 summarises the four problem variants. In all cases, the processing times of operations executed by humans or HRTs are stochastic, and the coefficient of variation is an instance parameter. All variants include instances with and without collaborative operations, which depends on the parameter α (percentage of collaborative operations of the instance). If the setting has only single processor operations, it is an instance of the PMP with machine eligibility and precedence constraints [Pinedo, 2008].

Table 5.1 – Problem variants and performance criteria.

Problem	Job arrivals	Criterion
MTSP-S	Static	Makespan (C_{max})
		Mean completion time ($\bar{C}$)
MTSP-D	Dynamic	Mean flowtime ($\bar{F}$)
		Mean tardiness ($\bar{T}$)

5.4. Methodology

Most of the existing machine learning algorithms provide well-performing but too complex models. Neural Networks and tree ensembles, for instance, usually involve several constants and weights that humans cannot interpret. Our approach uses GP in the search for effective and interpretable dispatching rules. The aim is to evolve compact and meaningful expressions that perform better than the existing classical rules. With that purpose, we follow the methodology proposed in Ferreira et al. [2021a] and use insights from dy-

dynamic scheduling literature. Thus, a small set of selected terminals are used to train the GP algorithm. Moreover, we employ bloat control and enforce dimensional consistency.

For evaluating each rule, an HRT simulator runs several replications over a set of instances. This section explains the two components of our approach, the simulation model and the GP algorithm. The section concludes with the design of the experiments, where we describe the instances and the parameters setting.

5.4.1 Simulation model

In our problem, some operations have multiple alternative resources for execution. Thus, one must decide not only the sequence of operations but also the assigned resource(s). There is a single queue for all resources in the working cell. Once at least one resource is available, a dispatching rule is applied to decide the next operation to be executed and which resources (there could be more than one idle) should be used. Thus, both sequencing and assignment decisions are taken with the same rule. This approach is chosen because of the interdependencies between those two decisions.

When jobs arrive, all operations without predecessors are immediately released to the queue. In the MTSP-S, all jobs arrive at the beginning of the simulation, while in MTSP-D, the shop-floor starts empty. In this latter case, we follow the literature using a warm-up period for the system to reach a steady-state before starting to collect metrics [Raghu and Rajendran, 1993]. Jobs are numbered according to their arrival, and metrics are calculated for jobs that arrive after a warm-up time.

The dynamic arrivals follow a Poisson distribution. As not all resources can perform all operations, the frequency of job arrivals must be adapted to resource eligibility. Otherwise, resources with higher eligibility would frequently incur 100% utilisation and difficult jobs completion. Thus, we calculate the arrivals rate λ based on the following parameters: utilisation level of the shop (u), the mean processing time of operations ($\mathbb{E}[p]$), the mean number of operations per job ($\mathbb{E}[|O|]$), human eligibility (he), robot eligibility (re) and percentage of collaborative operations (α). The expected number of eligible resources at any given time in the shopfloor is $ER = |\mathcal{H}| \cdot he + |\mathcal{R}| \cdot re$. By multiplying the number of resources to their eligibility, shop utilisation will not correspond to u . Therefore, we consider that u is the shop utilisation *relative to the resources eligibility*. The expected service time ST of jobs assumes that collaborative operations consume twice the processing time (since they consume two resources). The parameters p_{om} (processing time of single-processor operations) and p_o (processing time of collaborative operations) are sampled from the same distribution and have the same expected value. Thus, $ST = \mathbb{E}[|O|] \cdot \mathbb{E}[p] \cdot (1 + \alpha)$. Finally, the arrival rate is calculated as follows:

$$\lambda = \frac{u \cdot ER}{ST}$$

Operations are moved to the queue once all their predecessors are completed. Every time

at least one resource is available, the dispatching rule runs and assigns a score to a tuple $\langle \text{resource}, \text{single-processor operation} \rangle$ or $\langle \text{human}, \text{robot}, \text{collaborative operation} \rangle$. In scores calculation, only resources available for execution and operations in the queue are considered. The tuple with the lowest score is selected for execution.

The due date d_j of each job j depends on its arrival time at_j and the average processing time of each operation $\bar{p}_{jo}$. Due dates are calculated via Total Work Content [Baker, 1984], given that the allowance factor a is a parameter of the simulation:

$$d_j = at_j + a \cdot \sum_{o \in O_j} \bar{p}_{jo}$$

The generation of the stochastic processing times follows the procedure of Lawrence and Sewell [1997]. Based on their work, we introduced stochastic operation times in our simulation model. Processing times p_{om} are sampled from a Gamma distribution with given expected value $\mathbb{E}[p_{om}]$ and coefficient of variation cv . The expected processing times $\mathbb{E}[p_{om}]$ are the same as in the deterministic version. The process is analogous for p_o .

Since there is no guarantee that the dispatching rules will assign operations to resources in a balanced way, utilisation may reach almost 100% for some resource during simulation. This is a relevant issue in the presence of collaborative operations, as they require the availability of a human and a robot. In the MTSP-D, where there is no limit for the number of jobs that arrive at the shop-floor, balanced use of resources is necessary to stabilise the queue size and allow all jobs to finish. We observed that the simulation of some rules never achieves a steady-state and keeps increasing the queue size indefinitely. To avoid such behaviour, we defined the maximum queue size max_queue . If simulation, at any time, reaches this value, it promptly finishes without success. A pseudo-code of the simulator algorithm is found in Section 5.A from Appendix.

5.4.2 Genetic Programming algorithm

GP is a populational-based algorithm that evolves mathematical or logical expressions [Koza, 1994]. Thus, instead of directly generating solutions to a problem (like a schedule), the evolved individuals can be used as methods to solve the problem (in this case, dispatching rules that allocate and sequence operations). The approach combines a flexible representation scheme and a powerful search mechanism to explore both the structure and parameters of expressions. The aim is to find the individual with the highest fitness, i.e., the best expression to solve the problem. The GP developed in this work aims at finding expressions that are not only effective for the problem but that also compact and meaningful. Thus, we use a parsimony strategy to prioritise both well-performing and small rules. Moreover, in the pursuit of meaningful rules, we must ensure that they are dimensionally consistent. We employ a Strongly Typed GP, which is aware of the distinct types of the parameters that compose the expressions. Details on the fitness calculation are given in Section 5.4.2.1 and the types and function set are described in Section 5.4.2.2.

Symbolic expression trees are the most common representation of individuals in GP. They are composed of a function set: *terminals* and *operators* appropriate to the specific problem. The former are the leaf nodes and include parameters of the problem operated by the latter. Figure 5.2 illustrates an individual and the corresponding dispatching rule. In this example, the terminals are 2, *PT* (processing time of current operation), *S* (job slack) and *RE* (resource eligibility). The mathematical operators + and * compose the operators set.

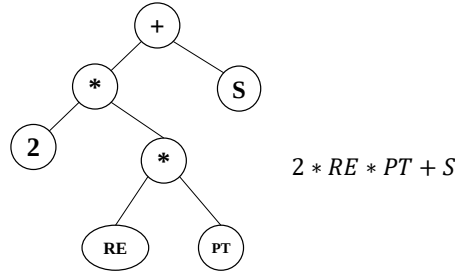


Figure 5.2 – A GP individual tree example and its corresponding dispatching rule

In general, GP algorithms create an initial population and evolve it over many generations. At each generation, individuals are evaluated and assigned a fitness score. The ones with the best fitness are reproduced and for the next generations. The pseudo-code of the proposed GP is provided in Algorithm 6. The algorithm receives as input a performance metric (i.e., makespan, mean completion time, mean flowtime, or mean tardiness), the training set, and the benchmark value pm_{Bi} for each instance i . Each tree A represents a dispatching rule whose terminals are parameters from the jobs and the shop-floor. The generation of the initial population in line 2 uses the algorithm Ramped Half-and-Half [Koza, 1994], which creates random trees by combining two methods, *full* and *grow*, with equal probability. The minimum and maximum depth for the trees must be predefined. In trees created by the *full* method, the length of the path from the root to every terminal is equal to the specified maximum depth. The *grow* method instead generates trees with variable shapes and sizes.

The proposed GP runs a generational evolution where a whole population P_g is evaluated and updated at each generation g . The fitness computation requires running the simulator described in Section 5.4.1, which is the most expensive procedure in the whole algorithm. Studies considering dynamic job arrivals usually simulate several replications of the same instance –each with thousands of job completions. In lines 4-8, the fitness of individual A is calculated. The simulator returns the performance metric pm_{Ai} when applying the rule A for instance i . The fitness of each individual combines this performance metric and the tree size and will be detailed in the next section. In line 10, the population of the next generation is created from the current population P_g . There are two breeding pipelines, crossover and mutation; each one is applied according to a likelihood probability. The crossover pipeline selects two individuals as parents to produce two offspring. This operation selects a node at random in each tree and swaps the two sub-trees rooted by those nodes. Mutation replaces a sub-tree with a randomly generated one, creating a new individual. Both pipelines use *tournament selection* to select the parents, returning the best of ts individuals selected at

Input: Performance Metric, Training set I , benchmark $\bar{p}_{Bi}$

Result: Dispatching rule A^*

```

1  $A^* \leftarrow \emptyset$ ;
2 Initialise population  $P_g$ ;
3 for  $g = 1$  to  $max\_g$  do
4   foreach  $A \in P_g$  do
5     foreach instance  $i \in I$  do
6        $pm_{Ai} \leftarrow 1/n \cdot \sum_{k=1}^n simulate(A, i)$ ;
7     end
8      $\tau_A = 1/|I| \cdot \sum_{i \in I} \{pm_{Ai}/pm_{Bi}\}$ ;
9      $fitness(A, g) = \tau_A + |A| \cdot g \cdot \rho$ ;
10  end
11   $A^* \leftarrow \underset{A \in \{P_g \cup A^*\}}{\operatorname{argmin}} fitness(A, g)$ ;
12   $P_{g+1} \leftarrow \text{Crossover and mutation in } P_g$ ;
13 end

```

Algorithm 6: Genetic Programming algorithm

random (ts is the tournament size). The breeding process picks certain kinds of nodes with different probabilities. For example, one can state to pick non-terminals 90% of the time and terminals 10% of the time. The algorithm runs until it reaches a maximum number of generations max_g and returns the best individual found A^* .

5.4.2.1 Fitness evaluation

The fitness function computes the performance of each individual A on a training set of instances. The simulation procedure applies rule A in several replications of instance i , and returns the average metric pm_{Ai} (line 6 in Algorithm 6). We define a set $DR = \{d_1, d_2, \dots\}$ of benchmark rules. The benchmark value for instance i is the minimum pm_{di} for all rules $d \in DR$. We detail the composition of DR in Section 5.5, where we present the performance comparison of the evolved rules. The overall performance of A is estimated by τ_a , the mean relative performance in all instances, considering the ratio between pm_{Ai} and pm_{di} . As the mean tardiness is null in some cases, we apply a protected division with $\max(pm_{di}, \varepsilon)$, with $\varepsilon = 10^{-3}$. Therefore, the relative performance of A for each instance i is calculated as follows.

$$\frac{\max(pm_{Ai}, \varepsilon)}{\max(\min_{d \in DR} pm_{di}, \varepsilon)}$$

In addition to relative performance, the fitness function also considers a strategy to avoid large trees. GP trees tend to grow during the search and result in complex rules –an effect known as bloat. This is a drawback of the symbolic tree representation, as complex trees are computationally inefficient and cannot be easily interpreted. Bloat control strategies

are usually applied to avoid this behaviour and pursue compact trees. Our approach is to penalise the tree size in the fitness function using a bloat control factor ρ . The penalty depends on the number of nodes $|A|$, the number of generations completed g and ρ . Considering the twofold objective of getting effective and small rules, the fitness of individual A in generation g is calculated as follows:

$$fitness(A, g) = \tau_A + |A| \cdot g \cdot \rho$$

If simulation reaches the maximum queue size, the individual fitness is set to a large constant.

5.4.2.2 Function set and allowed operations

The operator set is composed of the four basic mathematical operations: sum (+), subtraction (/), multiplication (*) and protected division (/) - the latter returns 1 if the denominator is 0. Table 5.2 presents the terminal set considered in this work, which was selected after preliminary experiments. We describe them in the following.

The remaining work of a job (RW) is the sum of all remaining operations' processing time. Time in the system (T) refers to the time since job arrival until the current time and is applied only for minimising the mean flowtime. The allowance of a job is the time until its due date, a relevant parameter for optimising tardiness metrics. We selected three terminals with allowance information from the best existing rules: the job slack (S), which is the allowance minus remaining work; the operation due date (PT*CR), which is based on the critical ratio CR -the allowance divided by the remaining work; and the processing time multiplied by the slack relative to remaining work (PT*S/RW). If the due date has passed, the allowance is negative. Similarly, the job slack is negative if the time until the due date is lower than the remaining work. Previous experiments demonstrated that negative due date information (slack or allowance) introduces bias and deteriorates the performance of dispatching rules. Thus, the evolved rules use CR^+ and S^+ , which assumes zero if the value of CR and S, respectively, is negative.

The terminal NS is the number of immediate successors of the current operation, given the precedence constraints of the job. The number of operations left (NL) counts the number of remaining operations of the current job. Resource eligibility (RE) is the percentage of operations eligible for each resource (human or robot) and assume values from 0 to 1. Likewise, operation flexibility (F) is the percentage of resources eligible for a given operation. Finally, the collaborative indicator (D) is a binary terminal that indicates whether an HRT must execute the operation.

Table 5.2 – Terminals adopted for each performance criterion

Type	Terminal	Code	Makespan	Completion	Flowtime	Tardiness
Time	Operation processing time	PT	x	x	x	x
	Remaining work	RW	x	x	x	x
	Time in system	T			x	
	Job slack	S ⁺				x
	Operation due date	PT*CR ⁺				x
	Processing time and relative slack	PT*S ⁺ /RW				x
#Ops	Number of operations left	NL	x	x	x	x
	Number of successors	NS	x	x	x	x
Flex	Resource eligibility	RE	x	x	x	x
	Operation Flexibility	F	x	x	x	x
Binary	Collaborative indicator	D	x	x	x	x

As all parameters described above have very distinct meanings and dimensions, they are grouped into four types: Time, #Ops (number of operations), Flexibility and Binary. Observe that with this terminal set, some rules may not be consistent in dimensions. For instance, in PT + RW + F, the terminal F is unlikely to determine the rule's behaviour, as it assumes values much smaller than PT and RW. To avoid such expressions and evolve meaningful rules, we apply Strongly-Typed GP, where there are constraints on the allowed operations. For instance, sum and subtraction operations cannot be performed if the terms involved are of different types (e.g., Time and Flexibility cannot be added).

In practice, the algorithm constraints which nodes may serve as children of other nodes. The return type of each operator node depends on the mathematical operation and the type of the children nodes. Terminals types are indicated in Table 5.2. Table 5.3 reports the allowed operations and their respective return types. The population initialisation, crossover and mutation operations generate only individuals that comply with the operations reported there.

Table 5.3 – Allowed operations for each pair of types, and the corresponding return type

	Time			#Ops			Flex			Binary
	+-	*	/	+-	*	/	+-	*	/	*
Time	Time		#Ops		Time		Time			Time
#Ops		Time	Time	#Ops		Flex	#Ops	Flex		#Ops
Flex		Time	Time		#Ops	#Ops	Flex	Flex	Flex	Flex
Binary		Time			#Ops			Flex		

5.4.3 Instance sets

The instance set was carefully designed to accommodate the distinct HRT settings found in the studies discussed in Section 5.2.1. Where no specific information was found, we use the standards adopted by the scheduling literature. Seminal papers are taken as reference in dynamic scheduling [Raghu and Rajendran, 1993, Holthaus and Rajendran, 1997, Rajendran and Holthaus, 1999, Dominic et al., 2004], and the most relevant works on the MTSP

[Bianco et al., 1999, Ferreira et al., 2021b]. An instance is defined by nine parameters, which are listed in Table 5.4.

Processing times, p_{om} and p_o , are sampled from $U(0, 100)$, and the number of operations of each job is sampled from a discrete uniform between 2 and 14 [Holthaus and Rajendran, 2000]. Thus, $\mathbb{E}[p] = 50$ and $\mathbb{E}[|O|] = 8$. The eligibility of single-processor operations to resources are randomly defined using a uniform distribution – the probability of o be eligible for resources $h \in \mathcal{H}$ and $r \in \mathcal{R}$ are he and re , respectively. The duo-processor operations are eligible for any pair $\langle h, r \rangle$. We select the precedence constraints at random, given that each operation may have up to 3 predecessors, as in Bianco et al. [1999] and Ferreira et al. [2021b]. Systems with dynamic job arrivals have no limit to the total number of jobs that may arrive at the shop-floor. Metrics are collected for jobs 501 to 2500 (a total of 2000 completed jobs, see Section 5.4.1). If arrivals are static, the size of $\mathcal{J}$ is limited. In this work, we adopt three values for $|\mathcal{J}|$ (10, 20 and 50), just like other works on the MTSP. As mentioned in Section 5.2, the rules' performance for mean tardiness depends on the load conditions, which are defined by the due date allowance (a) and resources utilisation (u). We vary the values of a and u to simulate both tight and light conditions for validating the tardiness metric. We also vary the utilisation for evaluating the mean flowtime, which is influenced by the shop-floor congestion.

Each working cell is composed of 4 operators distributed into humans and robots in different ways –there is no reference in the literature of larger collaborative cells. The values of he , re and α were defined in such a way to produce diverse instances. The instances sets are composed of a full-factory design of all parameters. The coefficient of variation cv determines the uncertainty level in operations processing times (if deterministic, $cv = 0$).

Table 5.4 – Training, validation sets

Parameter	Description	Training set	Validation and test sets
$ \mathcal{J} $	Number of jobs	10, 20 and 50 for static arrivals 2000 for dynamic arrivals	
a	Due dates allowance factor	3, 6	3, 4, 6
u	Shop utilisation level	0.85, 0.95	0.85, 0.90, 0.95
$ \mathcal{H} \times \mathcal{R} $	Number of humans x robots	(1 x 3), (3 x 1)	(1 x 3), (2 x 2), (3 x 1)
$he \times re$	Human and robot eligibility	(0.25 x 1), (1 x 0.25)	(0 x 1), (0.25 x 1), (0.5 x 1), (1 x 0), (1 x 0.25), (1 x 0.5)
α	Percentage of collaborative operations	0, 0.5	0, 0.25 and 0.5
cv	Uncertainty in processing times	0.2, 0.8	0, 0.2, 0.4, 0.6, 0.8 and 1

We consider three instance sets, as shown in Table 5.4. The training set is used by GP in the fitness calculation. It includes two cell compositions, the extreme values of a , u and α , and a subset of he, re and cv . In the training phase, the number of simulations per instance is the minimum necessary for assessing the effectiveness of each rule: 20 for the

MTSP-D. We apply 100 simulations per instance for the MTSP-S, where the number of jobs is much smaller, and more replications are necessary. The evaluation and selection of the evolved rules are performed using the validation set, which is much larger. This set includes all adopted values for a, u, he and re and all values of cv . To ensure that the results are statistically significant for a confidence level of 95% (for an error of at most 0.1), 100 replications are applied for the MTSP-D and 1000 for the MTSP-S – the minimum number of replications are 92 and 988, respectively. After selecting the best rule, we perform a more detailed analysis using the test set, which has the same size of the validation set but includes distinct instances. The number of jobs is the same in the three sets.

5.4.4 Algorithm parameters

Preliminary experiments were conducted to tune the algorithm parameters. Special attention was given to the bloat control factor (ρ) since it directly influences the fitness function and the size of the rules. As the penalty applied to individuals' size increases with the number of generations (see Section 5.4.2.1), the value of ρ must be studied together with the maximum number of generations max_gen . Thus, we ran 10 executions of the GP for each combination of $\rho = \{10^{-3}, 10^{-4}, 10^{-5}\}$ and $max_gen = \{50, 100, 200\}$. The overall performance obtained with the evolved rules for the makespan is shown in Figure 5.3. Observe that if the bloat control factor is 10^{-3} , there is a huge impact on the final performance of the rules. The average size of the rules is 6, 22 and 67 for ρ of 10^{-3} , 10^{-4} and 10^{-5} , respectively. Although the best performance is achieved for $\rho = 10^{-5}$, there seems to be a good compromise with $\rho = 10^{-4}$. Performing 200 generations in each run provides the best average results, but the use of this value is impractical as each generation is computationally expensive. Moreover, no significance difference on the overall performance of rules is found between $max_g = 50$ and $max_g = 100$ in Figure 5.3b. Thus, we decided to set $\rho = 10^{-4}$ and $max_g = 50$.

We observed that GP is not very sensitive to small changes in the crossover and mutation probabilities. The GP parameters used in the final experiments are exhibited in Table 5.5. The larger the population size, the better the average results obtained, although, for values above 200, the improvement in solution quality is negligible. Limiting the tree depth to small values prematurely prunes good branches, so we decided to set the maximum depth of trees to a large number commonly used (17). As the population is usually too homogeneous after some generations, the tournament size was set to the smallest value possible (2) to promote diversity. We set the values for the probability of selecting a terminal and a non-terminal (0.1 and 0.9, respectively) and the parameters of the Ramped Half-and-Half algorithm (minimum tree depth of 2 and maximum 6). The value of max_queue is set to the maximum queue size observed when running the best existing rules in the validation set. All algorithms were coded in Java, and the GP procedure uses the evolutionary computation library ECJ [Luke, 2017].

The next section describes the evolved rules for each criterion and their performance relative to the best existing dispatching rules for similar problems. Considering the four performance criteria and two types of instances, we executed eight independent experiments

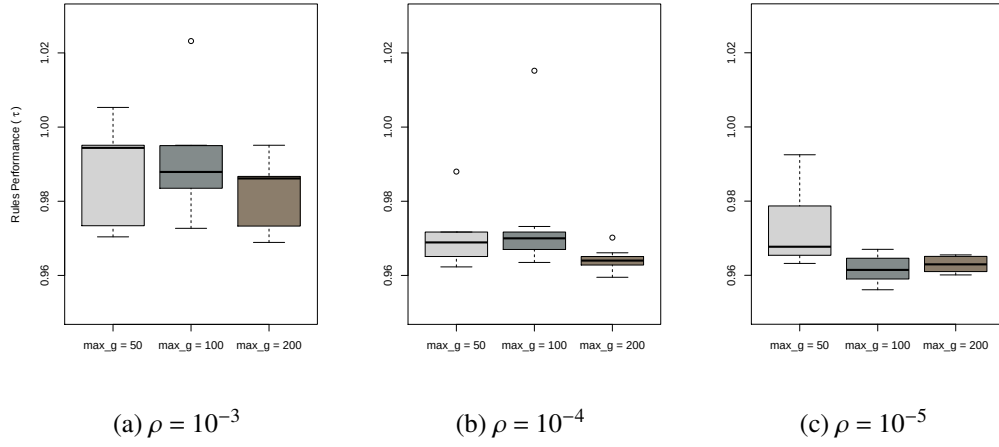


Figure 5.3 – Performance of rules evolved for the makespan criteria under distinct parameters settings

Table 5.5 – GP Parameters Setting

Parameter	Adopted values
Bloat control factor (ρ)	10^{-4}
Population size	200
Number of generations	50
Tournament size	2
Maximum trees depth	17
Crossover probability	0.2
Mutation probability	0.8
Maximum queue size <i>max_queue</i>	9000

and 30 runs of GP for each.

5.5. Evolved dispatching rules

This section describes the computational results for all performance criteria. First, we present an overall comparison of the generated rules with the best existing ones. Then we present the mathematical expression of the best rules and their detailed results. The section concludes with rules evolved for the particular case where no collaborative operations exist, corresponding to the PMP.

5.5.1 Results validation

We aim at evaluating the rules generated with GP against existing dispatching rules for the problem tackled here, considering instances with collaborative operations ($\alpha > 0$). As

a benchmark for the makespan and mean completion time, the seven rules presented by Lawrence and Sewell [1997] are considered. For evaluating the mean tardiness, we selected six existing rules that apply for prioritising jobs in a single queue: SPT, EDD, MDD, MS, CR+SPT and S/RPT+SPT, most of them designed for tardiness related metrics. We use the SPT for the mean flowtime, which performs well for this metric in job shops [Holthaus and Rajendran, 2000]. In the few instances where the SPT cannot reach the steady-state of the system, the FCFS is applied. We recall that the performance τ_A of rule A refers to the ratio between its results and the benchmark values. For each instance i , we calculate pm_{Ai}/pm_{di} , where $d \in DR$ is the best rule for i in the literature. Thus, if $\tau_A < 1$, rule A outperforms, on average, the best combination of the benchmark rules.

The boxplots in Figure 5.4 present the value of τ found using the evolved rules in the validation set –the lower, the better. We also report the performance of the benchmark rules, whose overall performance is below 1.4. Under dynamic arrivals, MS and FCFS are the only benchmark rules that reach the steady-state in all instances. Thus, we exclude from this analysis the cases where simulation reached the maximum queue size. Observe that, in general, the evolved rules outperform the existing ones in all criteria. We also note that there is lower variance in the results for C_{max} and $\bar{C}$, possibly because the MTSP-S deals with a small set of deterministic jobs. Although makespan is the most difficult metric to improve, all rules generated by GP present better average performance than the benchmark, with τ around 0.97 (in the worst case $\tau = 0.98$). In the other three criteria, the evolved rules provide considerable improvements. The average relative performance is 0.74, 0.80 and 0.83 for the mean completion time, mean flowtime and mean tardiness, respectively. In addition to the effectiveness, all evolved rules are able to stabilise the queue in all instances.

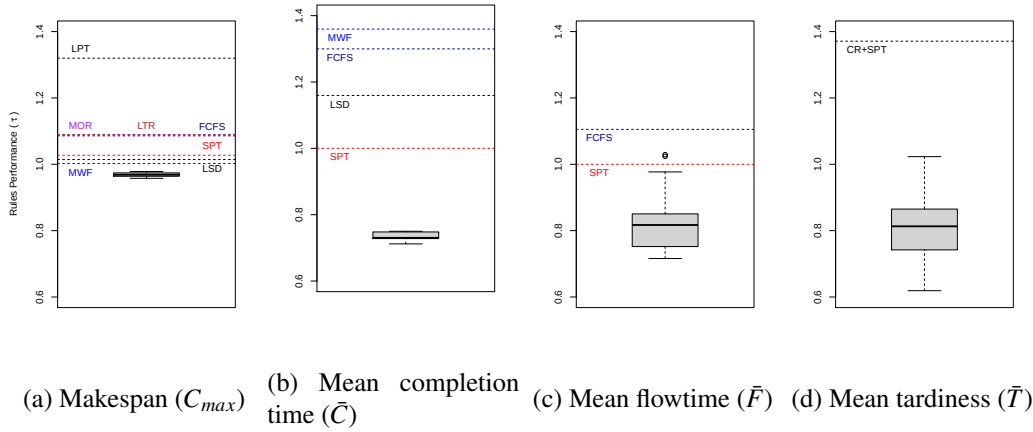


Figure 5.4 – Each chart presents the overall performance of 30 rules generated in independent GP runs, and results from the best existing dispatching rules.

5.5.2 Best rules for the MTSP

Table 5.6 shows the best rules generated for instances with collaborative operations, i.e., the rules with the lowest τ in the validation set. All rules are compact, easy to interpret, and generally prioritise the shortest processing times (PT). Moreover, in the four rules, collaborative operations tend to receive lower scores than single-processor operations. In fact, indicator D has a negative coefficient in R1, R2 and R3. In R4, if $D = 1$, the score is multiplied by RE/NL , which is usually lower than 1; otherwise, if $D = 0$, the score is multiplied by 1 (the result of the protected division $RE/(D*NL)$). Therefore, all rules prioritise collaborative operations, which require two resources available. We also observe that R1 is the only rule in which the coefficient of RW is negative. Thus, for minimising the makespan, one must select jobs with greater remaining work. This is consistent with the behaviour of rules like MOR, LSD and MWF, which try to anticipate the execution of the largest jobs. Schedules with a long tail of large jobs are undesirable for the makespan criterion. To optimise the other criteria, most of the generated rules select jobs with lower remaining work, those that would finish earlier. Notably, due date related terminals have a large influence on minimising the mean tardiness. Resource eligibility is also relevant for mean tardiness (R4) and makespan (R1). The mean flowtime prioritises jobs with fewer operations remaining (NL) and the longest time in the system ($-T$).

Table 5.6 – Evolved rules for the MTSP

	Rule	Metric	Mathematical expression
MTSP-S	R1	Makespan	$(PT - D * RW) / (RW * RE)$
	R2	Mean completion time	$(1 - D) * 2PT + RW$
MTSP-D	R3	Mean flowtime	$PT * NL * NL + RW - T * D$
	R4	Mean tardiness	$S^+ + (PT + PT * CR^+ + RW) * (RE / D * NL)$

In Table 5.7 we present the relative performance of R1-R4, grouped by resources eligibility (he and re). We also group the results of R1 and R2 by the number of jobs ($|\mathcal{J}|$). As the number of jobs is the same for all instances in the MTSP-D, the results of R3 are grouped by shop utilisation (u), as this is the most determining factor. The results of R4 are grouped by allowance factor (a), as its performance is the same regardless of utilisation. Observe that the four rules consistently outperform the benchmark not only on average but in very distinct settings. The value of τ is below 1 in all cells. The effectiveness of the rules is highly influenced by he and re , in different ways for each criterion. The relative makespan, mean flowtime and mean tardiness are better in settings with higher resource eligibility, where humans or robots are suitable for a greater number of tasks. It seems that the classical rules for these criteria are not suitable for assigning operations to resources, particularly in the presence of collaborative operations. For the mean completion time, it is the opposite; better values are found with lower eligibility. Regarding $|\mathcal{J}|$, while it does not influence the performance in makespan, the improvement in the mean completion time

is better with a larger number of jobs. The relative performance for the mean flowtime is better in congested settings (higher value of u), possibly because there is not much room for improvement in light loaded settings. Despite the high variance in the results of R4, the rule is effective in all settings, and particularly when due dates are not loose (medium or low values of a).

Table 5.7 – Performance evaluation of the best evolved rules for the two variants of the MTSP

Problem		MTSP-S								MTSP-D							
Metric		Makespan				Mean completion time				Mean flowtime				Mean tardiness			
		$ \mathcal{J} $				$ \mathcal{J} $				u				a			
<i>he</i>	<i>re</i>	10	20	50	Avg.	10	20	50	Avg.	0.85	0.90	0.95	Avg.	3	4	6	Avg.
1	0	0.99	0.99	0.99	0.99	0.70	0.66	0.62	0.66	0.76	0.73	0.69	0.73	0.77	0.74	0.91	0.81
1	0.25	0.96	0.96	0.96	0.96	0.75	0.72	0.70	0.73	0.71	0.73	0.69	0.71	0.50	0.42	0.58	0.50
1	0.5	0.92	0.92	0.93	0.93	0.80	0.79	0.78	0.79	0.71	0.67	0.60	0.66	0.45	0.40	0.35	0.40
0	1	0.98	0.98	0.99	0.99	0.70	0.66	0.62	0.66	0.76	0.73	0.69	0.73	0.74	0.69	0.95	0.79
0.25	1	0.96	0.96	0.97	0.96	0.75	0.72	0.70	0.72	0.71	0.72	0.69	0.71	0.71	0.66	0.84	0.74
0.5	1	0.93	0.92	0.93	0.92	0.80	0.79	0.78	0.79	0.71	0.66	0.60	0.66	0.50	0.44	0.51	0.48
Average		0.96	0.96	0.96	0.96	0.75	0.72	0.70	0.73	0.73	0.71	0.66	0.70	0.61	0.56	0.69	0.62

5.5.3 Best rules for non-collaborative operations

In preliminary experiments, we observed that it is challenging to generate a single rule that works well for instances with and without collaborative operations. Thus, specific rules were evolved for the case where $\alpha = 0$, as shown in Table 5.8. All rules have a similar structure, prioritising operations with shorter processing time and flexibility. However, there are some particularities. For instance, R5 is consistent with R1 (for makespan with collaborative operations) as it also prioritises operations with longer remaining work. In R8, generated for the mean tardiness, due date information (CR and slack) has a relevant impact on the score calculation, as in R4.

Table 5.8 – Evolved rules for the MTSP without collaborative operations

	Rule	Metric	Mathematical expression
MTSP-S	R5	Makespan	$RW - RW^2/(PT*F)$
	R6	Mean completion time	$PT*F*(NL-NS)$
MTSP-D	R7	Mean flowtime	$PT*F*F$
	R8	Mean tardiness	$PT*CR^+ + (PT*S^+/RW + RW)*F$

We detail the results of R5 - R8 on the test set in Table 5.9. As the instances used here

have no collaborative operation, the values $he = 0$ and $re = 0$ do not apply (otherwise, some resource would be 100% idle). Higher values of he and re require better assignment of resources to operations. As in the MTSP, higher eligibility (either in the value of he and re) provides better relative results for the makespan, suggesting that the benchmark rules are not suitable for assignment decisions. However, the same does not occur for the mean flowtime and mean tardiness. The relative performance of both R7 and R8 is better with lower eligibility (τ around 0.76 against 0.96). As in Table 5.7, the improvement in mean completion time is better for a larger number of jobs and lower eligibility. The allowance factor (a) also influences the performance of R8, which presents better improvements in settings with tight due dates, similarly to R4. We also observe that, except for the makespan, the relative effectiveness of rules is not as good as for the case with collaborative operations. R8 presents an average performance worse than the benchmark in some cases, as in settings with $a = 6$ and high resources eligibility.

Table 5.9 – Performance evaluation of the best evolved rules in environments without collaborative operations

Problem		MTSP-S								MTSP-D							
Metric		Makespan				Mean completion time				Mean flowtime				Mean tardiness			
		$ \mathcal{J} $				$ \mathcal{J} $				u				a			
he	re	10	20	50	Avg.	10	20	50	Avg.	0.85	0.90	0.95	Avg.	3	4	6	Avg.
1	0.25	0.96	0.97	0.98	0.97	0.80	0.73	0.69	0.74	0.84	0.82	0.79	0.82	0.61	0.80	0.77	0.72
1	0.5	0.92	0.92	0.93	0.92	0.85	0.77	0.71	0.77	0.96	0.95	0.95	0.95	0.83	0.92	1.27	1.01
0.25	1	0.96	0.98	0.99	0.98	0.79	0.73	0.68	0.73	0.84	0.82	0.80	0.82	0.75	0.84	0.80	0.80
0.5	1	0.92	0.93	0.94	0.93	0.85	0.76	0.70	0.77	0.96	0.96	0.95	0.96	0.81	1.05	1.06	0.97
Average		0.94	0.95	0.96	0.95	0.82	0.75	0.70	0.75	0.90	0.89	0.87	0.89	0.75	0.90	0.97	0.87

5.6. Dispatching rules against near-optimal solutions

This section focus on the makespan criterion, one of the most common used in HRTs studies [Xu et al., 2020, Bogner et al., 2018, Bänziger et al., 2018, Chen et al., 2014]. We compare the output obtained with the evolved rules to high-quality schedules found for the deterministic version of the problem. The aim is to investigate how uncertainty in processing times influence the good performance of existing methods for the MTSP-S and compare them to the rules generated in this work. We also analyse the influence of other characteristics by performing experiments on distinct HRTs settings.

In Chapter 4, we devised an effective Constraint Programming Model (CP) for the Multimode Multiprocessor Task Scheduling Problem, a generalisation of the MTSP. In that work, CP was applied on HRTs working cells with deterministic processing times. The algorithm found the optimal solution for one-third of the proposed instances and provided an average gap of only 3.1% to a lower bound. The CP model is described in Section 4.4 and used in the experiments of this Section.

The design of the experiments is based on Lawrence and Sewell [1997], which evaluate the makespan performance of dispatching rules under different uncertainty levels for job shops. For each instance, we simulate distinct values of cv , from the deterministic case ($cv = 0$) to the exponential distribution ($cv = 1$). Four strategies are compared:

- **CP** –The solution found by CP for the deterministic case is applied, and no change is allowed during execution.
- **Dynamic Sequence** –The solution from CP is applied, but the sequence may change in the face of delays. For instance, if the next operation should be 2_3 but some of its predecessors have not finished yet, the subsequent operation is taken, and 2_3 will be executed as soon as it is ready.
- **MWF** –We use the best dispatching rule from Lawrence and Sewell [1997].
- **Evolved rule** – $R1$ and $R5$, the best rules found by GP for optimising the makespan are used.

All experiments use instances with the largest number of jobs, 50. For each instance, we calculated the number of replications necessary for a 0.05 significance level. We concluded that 50 replications are sufficient to guarantee an error of at most 0.005 for all experiments. A Linear Programming model proposed in Section 4.3.2 is used to analyse the distance of the makespan generated to a lower bound for the optimal solution. Thus, all charts in this section report the makespan C_{max} relative to the lower bound L found for the same instance (C_{max}/L).

In the next three sections, we analyse the influence of distinct characteristics in the behaviour of the rules: resource eligibility, number of humans, number of robots and number of precedence constraints. In these experiments, collaborative tasks are not considered (all operations are single-processor), and the evolved rule considered for comparison is $R5$. In Section 5.6.4 we consider collaborative operations and use $R1$ in this analysis.

5.6.1 Effect of resource eligibility

In the first set of experiments, we compare the methods in settings with distinct resource eligibility constraints. All instances are composed of two humans and two robots. In the chart from Figure 5.5a, humans are eligible for all operations while robots present 50% of eligibility. The chart from Figure 5.5c presents the opposite case: eligibility constraints apply only to humans. In the experiments of Figure 5.5b, no eligibility constraints are present.

As expected, the original solution provided by CP deteriorates in face of uncertainty in all circumstances. For $cv = 0$, the makespan from CP is, on average, 1% greater than the lower bound and 17% in the higher levels of uncertainty. Using dynamic sequence provides better solutions, at most 9% greater than the lower bound. In the chart from Figure 5.5b, we observe that both dispatching rules have similar performance ($R5$ is slightly better), and

they generate poor solutions in general. This result may be explained by the difficulty in selecting the best resource to execute operations when all resources are eligible. It seems a complex decision for such simple dispatching rules. Even in the deterministic case, the solution of both rules is around 5% greater than the lower bound. However, they become competitive for higher values of cv (0.6 or greater) as they do not deteriorate too much.

From Figures 5.5a and 5.5c, we conclude that the evolved rule is promising in settings where not all resources are eligible for all operations. Independently on the type of resource (humans or robots), if there are eligibility constraints, the solution from R5 is around 3% greater than the lower bound in the deterministic case and is competitive to the solution provided by CP with dynamic sequence when $cv = 0.4$ or greater. However, MWF presents poor behaviour in these settings, probably because it does not include a resource eligibility parameter.

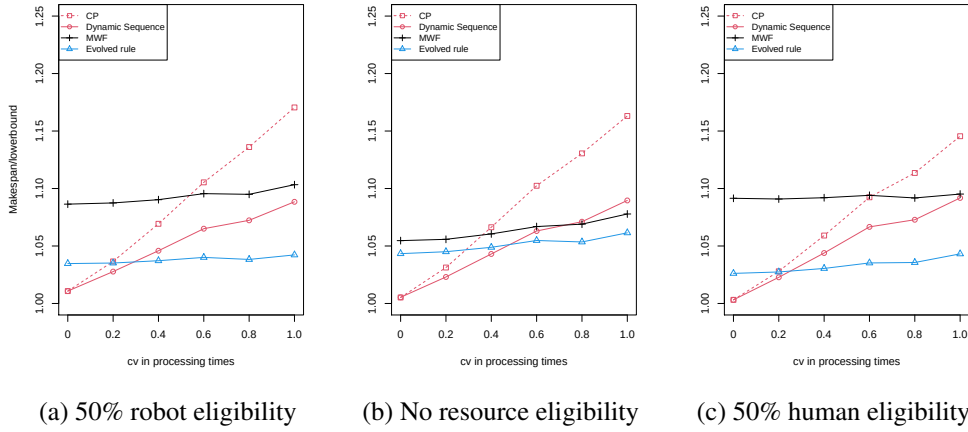


Figure 5.5 – Performance of methods for different types of resource eligibility constraints

As the evolved rule seems suitable for environments with eligibility constraints, we aim to analyse which other characteristics of HRT settings favour its use. In all subsequent experiments, we consider instances where human is eligible for %50 of the operations and no constraints apply to robots.

5.6.2 Effect of the cell composition

In the charts from Figure 5.6, we present results for different cell compositions, intending to evaluate the methods in distinct combinations of humans and robots. We remind that only human behaviour is stochastic; the processing time of operations executed by robots is always deterministic. Thus, by comparing the three charts, it is possible to analyse the influence of the number of humans on the results of the methods.

We observe that the larger the number of humans, all methods deteriorate in performance under uncertainty. The solution from CP, for instance, is up to 14% greater than the lower

bound in cells with only one human and up to 21% in cells with three humans. The behaviour of all the other methods is similar, but note that the evolved rule is the most robust, with worst-case performance around 5% in all scenarios. The other dispatching rule, MWF, produce results that are extremely dependent on the cell composition. In the deterministic case with three humans, the obtained makespan is 20% greater than the lower bound. This poor behaviour is probably due to all parameters from MWF are time-dependent (the processing time of the current operation processing and the remaining work). In R5, contrary, other parameters also drive the rule behaviour (resource eligibility and operation flexibility).

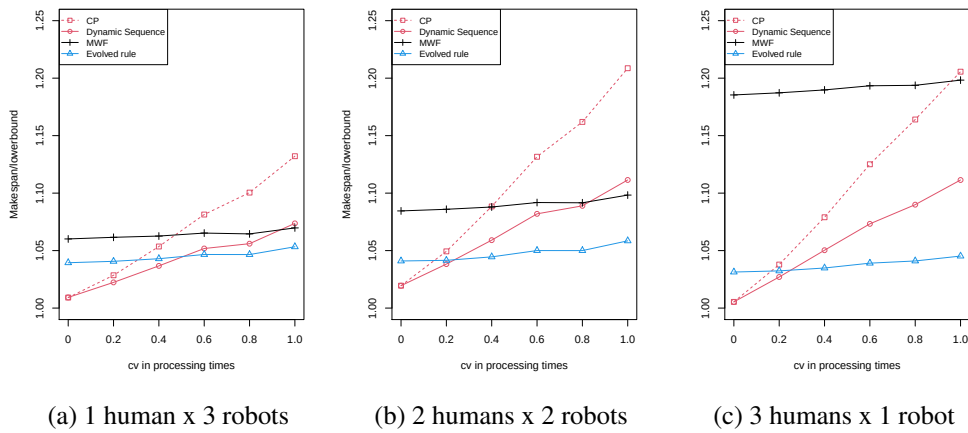


Figure 5.6

5.6.3 Different levels of precedence constraints

The existence of a high number of precedence constraints has a relevant impact on any schedule, particularly when the criterion is makespan. Any delay of an operation in the bottleneck path postpone the metric. In the third set of experiments, we aim at analysing how the methods behave for different precedence levels. In Figure 5.7 we compare three very distinct settings: where each operation has zero to six predecessors (5.7a); zero or one (5.7b); or no predecessor (5.7c). We consider here working cells composed of two humans and two robots.

For all methods but MWF, as the expected predecessors increase, the impact on the makespan is more significant. This effect is more intense for the fixed CP solution that produces a makespan up to 10% greater than the lower bound if no precedence exists. R5 works much better than MWF in general and presents better results in settings with fewer predecessors. It is no trivial to explain why the number of precedence constraints does not influence MWF. A possible reason may be that this rule prioritises jobs with most work remaining, which are more suitable to cascading effects on successive operations.

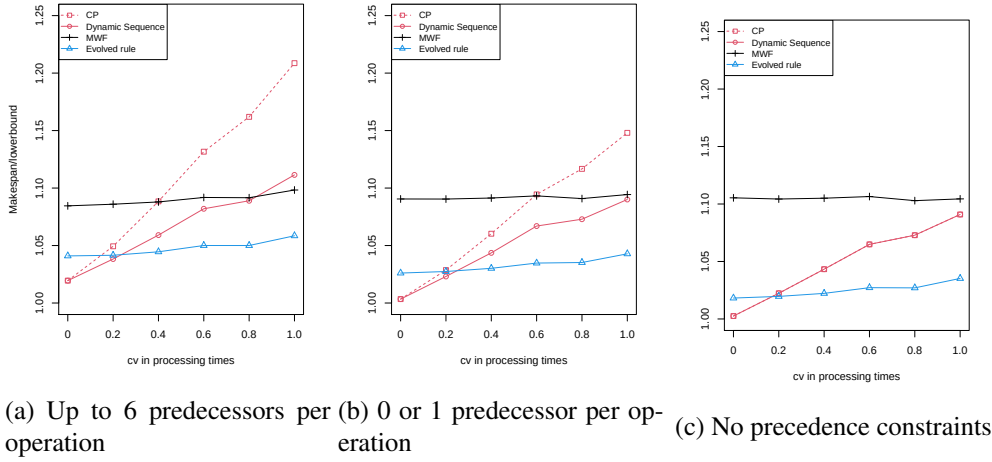


Figure 5.7

5.6.4 Different levels of collaborative operations

Finally, we aim at analysing if all conclusions from previous experiments apply in the presence of collaborative operations, where an HRT must be synchronised to execute tasks. We recall that the evolved rule here is R1, and we simulate two humans and two robots, considering two values for α : 25% and 50%. Here we include an additional strategy: using R1 to sequence operations and assume as mandatory the assignment provided by CP –we call it Evolved rule (CP Assignment). Thus, when using this strategy, there is only one assignment alternative and the dispatching rule is used only to prioritise operations for each resource.

We observe similar results for both charts. The original solution from CP is the better strategy only in the deterministic case and deteriorates quickly under uncertainty. MWF does not work well in general since it does not include any parameter for prioritising collaborative operations. The new strategy (Evolved rule with CP assignment) is the best under low uncertainty levels, when $cv = 0.2$ in Figure 5.8a and for $0.2 \leq cv \leq 0.6$ in Figure 5.8b. A possible reason for the good performance of this strategy is that the assignment decision is critical in the presence of collaborative operations. Thus, it is better to use the CP solution than employing a dispatching rule for this decision. However, as uncertainty increases, the optimal assignment may be completely different from the one found by CP. In such circumstances, a reactive and dynamic approach (with the evolved rule) for the assignment decision works better than using a pre-specified solution.

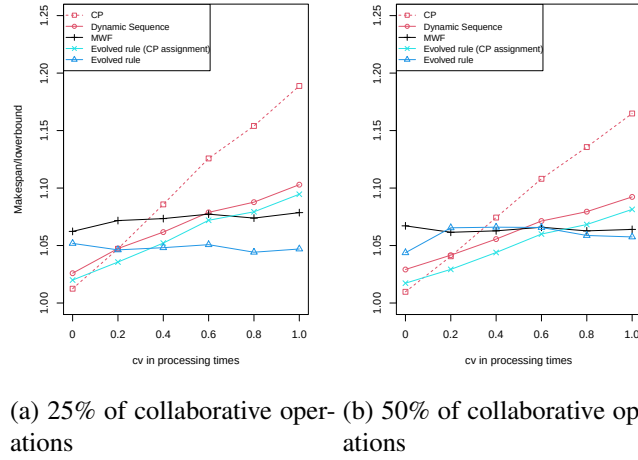


Figure 5.8

5.7. Conclusion

This work aims at providing reactive scheduling methods for an emerging application in manufacturing, collaborative working cells. In such environments, human and robot must be synchronised, and some operations require the simultaneous execution of both operators. We model the problem of scheduling HRTs as the Multiprocessor Task Scheduling Problem with uncertain processing times. Dispatching rules are designed via Genetic Programming for two variants of the problem, with static and dynamic arrivals. Four performance criteria are considered, makespan, mean completion time, mean tardiness and mean flowtime.

We simulate the evolved rules in very distinct collaborative settings, varying the number of jobs, resource eligibility, load conditions, working cell composition and uncertainty level in processing times. In the presence of collaborative operations, the evolved rules outperform the best existing ones in all criteria, providing gains around 3-4% for the makespan, 25-30% for the mean completion time and mean flowtime and 30-40% for the mean flowtime. In addition to the overall good performance, the rules perform well in a wide range of settings. Moreover, they are compact, meaningful and easy to interpret. We also compare the makespan performance of the dispatching rules against a Constraint Programming Model. Results show that solutions generated for the deterministic version of the problem using CP deteriorate under uncertain processing times, favouring the use of dispatching rules in this circumstance. The rules reach the system steady-state in all instances and generalise well in a wide range of settings. We also evolved rules for the particular case without collaborative operations.

Future research should be employed on the validation of the proposed approach in a real use case, either in collaborative working cells or in other manufacturing settings. Moreover, studies should focus on experimental comparisons between dispatching rules and other

dynamic scheduling methods. For instance, one should investigate which environments favour the use of proactive/reactive approaches instead of completely reactive rules. The use of rolling horizons may also be interesting. In practice, scheduling usually deals with several performance criteria. Thus, another relevant study topic is the generation of rules for multi-objective optimisation.

Acknowledgments

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within a PhD scholarship.

Bibliography

- Anderson, E.J., Nyirenda, J.C., 1990. Two new rules to minimize tardiness in a job shop. *International Journal of Production Research* 28, 2277–2292. doi:[10.1080/00207549008942866](https://doi.org/10.1080/00207549008942866).
- Aytug, H., Lawley, M.A., McKay, K., Mohan, S., Uzsoy, R., 2005. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research* 161, 86 – 110. doi:<https://doi.org/10.1016/j.ejor.2003.08.027>. iEPM: Focus on Scheduling.
- Baker, K.R., 1984. Sequencing rules and due-date assignments in a job shop. *Management Science* 30, 1093–1104. doi:[10.1287/mnsc.30.9.1093](https://doi.org/10.1287/mnsc.30.9.1093).
- Bänziger, T., Kunz, A., Wegener, K., 2018. Optimizing human–robot task allocation using a simulation tool based on standardized work descriptions. *Journal of Intelligent Manufacturing* doi:[10.1007/s10845-018-1411-1](https://doi.org/10.1007/s10845-018-1411-1).
- Bianco, L., Dell’Olmo, P., Giordani, S., Speranza, M.G., 1999. Minimizing makespan in a multimode multiprocessor shop scheduling problem. *Naval Research Logistics* 46, 893 – 911. doi:[10.1002/\(SICI\)1520-6750\(199912\)46:8<893::AID-NAV2>3.0.CO;2-7](https://doi.org/10.1002/(SICI)1520-6750(199912)46:8<893::AID-NAV2>3.0.CO;2-7).
- Bogner, K., Pfersch, U., Unterberger, R., Zeiner, H., 2018. Optimised scheduling in human–robot collaboration – a use case in the assembly of printed circuit boards. *International Journal of Production Research*, 1–19doi:[10.1080/00207543.2018.1470695](https://doi.org/10.1080/00207543.2018.1470695).
- Branke, J., Hildebrandt, T., Scholz-Reiter, B., 2015. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. *Evolutionary Computation* 23, 249–277. doi:https://doi.org/10.1162/evco_a_00131.
- Casalino, A., Zanchettin, A.M., Piroddi, L., Rocco, P., 2019. Optimal scheduling of human-robot collaborative assembly operations with time petri nets. *IEEE Transactions on Automation Science and Engineering*, 1–15doi:[10.1109/TASE.2019.2932150](https://doi.org/10.1109/TASE.2019.2932150).

- Chen, F., Sekiyama, K., Cannella, F., Fukuda, T., 2014. Optimal subtask allocation for human and robot collaboration within hybrid assembly system. *IEEE Transactions on Automation Science and Engineering* 11, 1065–1075. doi:[10.1109/TASE.2013.2274099](https://doi.org/10.1109/TASE.2013.2274099).
- Darvish, K., Wanderlingh, F., Bruno, B., Simetti, E., Mastrogiovanni, F., Casalino, G., 2018. Flexible human–robot cooperation models for assisted shop-floor tasks. *Mechatronics* 51, 97 – 114. URL: <http://www.sciencedirect.com/science/article/pii/S0957415818300485>, doi:<https://doi.org/10.1016/j.mechatronics.2018.03.006>.
- Dominic, P.D.D., Kaliyamoorthy, S., Kumar, M.S., 2004. Efficient dispatching rules for dynamic job shop scheduling. *The International Journal of Advanced Manufacturing Technology* 24, 70–75. doi:[10.1007/s00170-002-1534-5](https://doi.org/10.1007/s00170-002-1534-5).
- Drozdzowski, M., 2009. Parallel Tasks. Springer London, London. chapter 5. pp. 87–208. doi:[10.1007/978-1-84882-310-5_5](https://doi.org/10.1007/978-1-84882-310-5_5).
- Ferreira, C., Figueira, G., Amorim, P., 2020. Optimizing dispatching rules for stochastic job shop scheduling, in: Madureira, A.M., Abraham, A., Gandhi, N., Varela, M.L. (Eds.), *Hybrid Intelligent Systems*, Springer International Publishing, Cham. pp. 321–330.
- Ferreira, C., Figueira, G., Amorim, P., 2021a. Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning. URL: <https://arxiv.org/abs/2109.03323>, arXiv:2109.03323.
- Ferreira, C., Figueira, G., Amorim, P., 2021b. Scheduling human-robot teams in collaborative working cells. *International Journal of Production Economics* 235, 108094. doi:<https://doi.org/10.1016/j.ijpe.2021.108094>.
- Hildebrandt, T., Heger, J., Scholz-Reiter, B., 2010. Towards improved dispatching rules for complex shop floor scenarios: A genetic programming approach, in: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, NY, USA. pp. 257–264. doi:[10.1145/1830483.1830530](https://doi.org/10.1145/1830483.1830530).
- Holthaus, O., Rajendran, C., 1997. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics* 48, 87 – 105. doi:[https://doi.org/10.1016/S0925-5273\(96\)00068-0](https://doi.org/10.1016/S0925-5273(96)00068-0).
- Holthaus, O., Rajendran, C., 2000. Efficient jobshop dispatching rules: Further developments. *Production Planning & Control* 11, 171–178. doi:[10.1080/095372800232379](https://doi.org/10.1080/095372800232379).
- IFR, 2018. Demystifying Collaborative Industrial Robots. Technical Report. International Federation of Robotics. URL: <https://ifr.org>.
- IFR, 2020. World Robotics 2020 Industrial Robots. Technical Report. International Federation of Robotics. URL: <https://ifr.org>.

- Ivanov, D., Tang, C.S., Dolgui, A., Battini, D., Das, A., 2020. Researchers' perspectives on industry 4.0: multi-disciplinary analysis and opportunities for operations management. *International Journal of Production Research* 0, 1–24. doi:[10.1080/00207543.2020.1798035](https://doi.org/10.1080/00207543.2020.1798035).
- Jaklinović, K., Đurasević, M., Jakobović, D., 2021. Designing dispatching rules with genetic programming for the unrelated machines environment with constraints. *Expert Systems with Applications* 172, 114548. URL: <https://www.sciencedirect.com/science/article/pii/S0957417420311921>, doi:<https://doi.org/10.1016/j.eswa.2020.114548>.
- Johannsmeier, L., Haddadin, S., 2017. A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes. *IEEE Robotics and Automation Letters* 2, 41–48. doi:[10.1109/LRA.2016.2535907](https://doi.org/10.1109/LRA.2016.2535907).
- Jun, S., Lee, S., Chun, H., 2019. Learning dispatching rules using random forest in flexible job shop scheduling problems. *International Journal of Production Research* 57, 3290–3310. doi:[10.1080/00207543.2019.1581954](https://doi.org/10.1080/00207543.2019.1581954).
- Karunakaran, D., Yi Mei, Gang Chen, Mengjie Zhang, 2017. Evolving dispatching rules for dynamic job shop scheduling with uncertain processing times, in: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 364–371.
- Koza, J.R., 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing* 4, 87–112.
- Lawrence, S.R., Sewell, E.C., 1997. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. *Journal of Operations Management* 15, 71 – 82. doi:[https://doi.org/10.1016/S0272-6963\(96\)00090-3](https://doi.org/10.1016/S0272-6963(96)00090-3).
- Liu, Z., Liu, Q., Wang, L., Xu, W., Zhou, Z., 2021. Task-level decision-making for dynamic and stochastic human-robot collaboration based on dual agents deep reinforcement learning. *The International Journal of Advanced Manufacturing Technology* 115, 3533–3552. doi:<https://doi.org/10.1007/s00170-021-07265-2>.
- Luke, S., 2017. Ecj then and now, in: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Association for Computing Machinery, New York, NY, USA. p. 1223–1230. doi:[10.1145/3067695.3082467](https://doi.org/10.1145/3067695.3082467).
- Luo, S., 2020. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing* 91, 106208. doi:<https://doi.org/10.1016/j.asoc.2020.106208>.
- Nguyen, S., Mei, Y., Xue, B., Zhang, M., 2019. A hybrid genetic programming algorithm for automated design of dispatching rules. *Evolutionary Computation* 27, 467–496. doi:[10.1162/evco_a_00230](https://doi.org/10.1162/evco_a_00230).
- Nguyen, S., Mei, Y., Zhang, M., 2017. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems* 3, 41–66.

- Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2015. Automatic programming via iterated local search for dynamic job shop scheduling. *IEEE Transactions on Cybernetics* 45, 1–14.
- Nie, L., Gao, L., Li, P., Li, X., 2013. A gep-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. *Journal of Intelligent Manufacturing* 24, 763–774. doi:<https://doi.org/10.1007/s10845-012-0626-9>.
- Olsen, T.L., Tomlin, B., 2020. Industry 4.0: Opportunities and challenges for operations management. *Manufacturing & Service Operations Management* 22, 113–122. doi:[10.1287/msom.2019.0796](https://doi.org/10.1287/msom.2019.0796).
- Parente, M., Figueira, G., Amorim, P., Marques, A., 2020. Production scheduling in the context of industry 4.0: review and trends. *International Journal of Production Research* 0, 1–31. doi:[10.1080/00207543.2020.1718794](https://doi.org/10.1080/00207543.2020.1718794).
- Pinedo, M.L., 2008. *Scheduling - Theory, Algorithms and Systems*. 3rd ed., Springer.
- Pupa, A., Van Dijk, W., Secchi, C., 2021. A human-centered dynamic scheduling architecture for collaborative application. *IEEE Robotics and Automation Letters* 6, 4736–4743. doi:[10.1109/LRA.2021.3068888](https://doi.org/10.1109/LRA.2021.3068888).
- Raghu, T., Rajendran, C., 1993. An efficient dynamic dispatching rule for scheduling in a job shop. *International Journal of Production Economics* 32, 301 – 313. doi:[https://doi.org/10.1016/0925-5273\(93\)90044-L](https://doi.org/10.1016/0925-5273(93)90044-L).
- Rajendran, C., Holthaus, O., 1999. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research* 116, 156 – 170. doi:[https://doi.org/10.1016/S0377-2217\(98\)00023-X](https://doi.org/10.1016/S0377-2217(98)00023-X).
- Ramasesh, R., 1990. Dynamic job shop scheduling: A survey of simulation research. *Omega* 18, 43 – 57. doi:[https://doi.org/10.1016/0305-0483\(90\)90017-4](https://doi.org/10.1016/0305-0483(90)90017-4).
- Romero-Silva, R., Shaaban, S., Marsillac, E., Hurtado, M., 2018. Exploiting the characteristics of serial queues to reduce the mean and variance of flow time using combined priority rules. *International Journal of Production Economics* 196, 211 – 225. doi:<https://doi.org/10.1016/j.ijpe.2017.11.023>.
- Su, H., Pinedo, M., Wan, G., 2017. Parallel machine scheduling with eligibility constraints: A composite dispatching rule to minimize total weighted tardiness. *Naval Research Logistics (NRL)* 64, 249–267. doi:<https://doi.org/10.1002/nav.21744>.
- Sweeney, K.D., Sweeney, D.C., Campbell, J.F., 2019. The performance of priority dispatching rules in a complex job shop: A study on the upper mississippi river. *International Journal of Production Economics* 216, 154 – 172. doi:<https://doi.org/10.1016/j.ijpe.2019.04.024>.

- Tay, J.C., Ho, N.B., 2008. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering* 54, 453 – 473. doi:<https://doi.org/10.1016/j.cie.2007.08.008>.
- Đurasević, M., Jakobović, D., 2019. Creating dispatching rules by simple ensemble combination. *Journal of Heuristics* doi:[10.1007/s10732-019-09416-x](https://doi.org/10.1007/s10732-019-09416-x).
- Đurasević, M., Jakobović, D., Knežević, K., 2016. Adaptive scheduling on unrelated machines with genetic programming. *Applied Soft Computing* 48, 419 – 430. doi:<https://doi.org/10.1016/j.asoc.2016.07.025>.
- Weber, A., 2018. State of the Profession 2018: Smart Factories Demand Smarter Engineers. Technical Report. Assembly. URL: <https://www.assemblymag.com/articles/94380-state-of-the-profession-2018-smart-factories-demand-smarter-engineers?>
- Xiong, H., Fan, H., Jiang, G., Li, G., 2017. A simulation-based study of dispatching rules in a dynamic job shop scheduling problem with batch release and extended technical precedence constraints. *European Journal of Operational Research* 257, 13 – 24. doi:<https://doi.org/10.1016/j.ejor.2016.07.030>.
- Xu, W., Tang, Q., Liu, J., Liu, Z., Zhou, Z., Pham, D.T., 2020. Disassembly sequence planning using discrete bees algorithm for human-robot collaboration in remanufacturing. *Robotics and Computer-Integrated Manufacturing* 62, 101860. URL: <http://www.sciencedirect.com/science/article/pii/S0736584519301887>, doi:<https://doi.org/10.1016/j.rcim.2019.101860>.
- Zanchettin, A.M., 2021. Robust scheduling and dispatching rules for high-mix collaborative manufacturing systems. *International Journal of Flexible Manufacturing Systems* 0, to appear.
- Zhang, R., Lv, Q., Li, J., Bao, J., Liu, T., Liu, S., 2022. A reinforcement learning method for human-robot collaboration in assembly tasks. *Robotics and Computer-Integrated Manufacturing* 73, 102227. URL: <https://www.sciencedirect.com/science/article/pii/S0736584521001095>, doi:<https://doi.org/10.1016/j.rcim.2021.102227>.

Appendix 5.A HRTs simulator

The pseudo-code of the HRTs simulator is described in Algorithm 7. The instance parameters $\langle \bar{p}, a, u, cv \rangle$ and a dispatching rule are inputs for the algorithm, which simulates settings with and without dynamic arrivals. As described in Section 5.4.1, in the MTSP-D, jobs arrivals are sampled from a Poisson distribution which mean rate depends on the value of u . The allowance factor a is considered to calculate jobs due dates. In the MTSP-S, all jobs arrive before the beginning of the simulation. The parameter $\bar{p}$ is the mean processing

time, and cv is the coefficient of variation for sampling the actual processing times. The simulation starts with an empty shop-floor and goes on until a number of jobs have been computed (max_jobs). The algorithm maintains a set of non-completed operations $\mathcal{O}'$, initialised with the operations of all jobs. Each operation $o \in \mathcal{O}'$ enters the queue Q at time $start_o$, when their predecessors have finished. Operations with no predecessors enter the queue at their job's arrival time in the shop-floor at_j . The next time resource m is available is kept in A_m .

The simulation goes on until a number of jobs have been computed (max_jobs). In line 2, the algorithm verifies which is the earliest event, between the availability of an operation ($start_o$) and the availability of a resource (A_m). An operation becomes available upon job arrival or when the predecessor operation finishes. In both cases, the operation enters the queue (line 3). Once at least one machine is available (line 5), the next pair of operation and resource(s) is selected. In lines 6 to 10, the dispatching rule runs and calculates a score for each eligible pair <operation, resource>. The process is analogous for collaborative operations: if o is collaborative, the score is calculated for a tuple <operation, human, robot>. The pair or tuple with the least score is executed. If the respective job job_o^* has no operations left, it is completed, and the metric is calculated (lines 14-17). Otherwise, the algorithm updates the start time of the operations' successors (line 20). We recall that jobs are numbered on arrival in a crescent sequence order, and the performance metric P is computed only for jobs numbered from min to max . Following the values adopted in

literature, we set $min = 501$ and $max = 2500$, $max_jobs = max - min = 2000$.

Input: Instance $\langle \bar{p}, a, u, cv \rangle$ and dispatching rule DR

Result: Performance Metric pm

```

1 while num_jobs < max_jobs do
2   if min{starto} < min{Am} then
3     |  $Q \leftarrow Q + \arg \min_{o \in O'} \{start_o\}$  ;
4   else
5     |  $min\_start \leftarrow \arg \min_{m \in M} \{A_m\}$  ;
6     | foreach resource  $m \in M$  do
7       | if  $A_m = min\_start$  then
8         | | foreach operation  $o \in Q$  do
9           | | | if  $m$  eligible for  $o$  then
10            | | | |  $score_{om} \leftarrow run\_DR$ ;
11            | | | end
12            | | end
13            | end
14          end
15           $(o^*, m^*) \leftarrow \arg \min_{(o,m) \in (Q \times M)} \{score_{om}\}$ ;
16           $A_{m^*} \leftarrow finish\_time_{o^*}$ ;
17           $O' \leftarrow O' \setminus o^*$ ;
18          if jobo* is_complete then
19            | calculate_metric  $pm$  ;
20            | if jobo*  $\in \{min...max\}$  then
21              | |  $num\_jobs \leftarrow num\_jobs + 1$ ;
22              | end
23            else
24              | | foreach operation  $o \in Successors_{o^*}$  do
25                | | |  $start_o \leftarrow \max(start_{o^*}, finish\_time_{o^*})$  ;
26                | | end
27              | end
28            end
29          end
30  pm  $\leftarrow pm \div max\_jobs$ ;

```

Algorithm 7: HRTs Simulator

Static and dynamic scheduling of bulk ports operations

Scheduling wagons unloading in bulk ports with uncertain processing times

Cristiane Ferreira^{*†} · Gonalo Figueira^{*†} · Pedro Amorim^{*†} · Alexandre Pigatti[‡]

Submitted to *COR*

Abstract Optimising operations in bulk cargo ports is of great relevance due to their major participation in international trade. However, little attention has been devoted to the inbound operations, critical to meet shipments deadlines. The product typically arrives by train and must be transferred to the stockyard. This process requires several machines and is subject to frequent disruptions leading to uncertain processing times. This work focuses on the scheduling problem of unloading the wagons to the stockyard, approaching both the deterministic and the stochastic versions. For the deterministic problem, we compare three solution approaches: a Mixed Integer Programming model, a Constraint Programming model and a Greedy Randomised algorithm. The selection rule of the latter is evolved via Genetic Programming. The stochastic version is approached by dispatching rules, also evolved via Genetic Programming. The proposed approaches are validated using real data from a leading company in the mining sector. Results show that the new heuristic presents similar results to the company's algorithm in a considerably shorter computational time. Moreover, we perform extensive computational experiments to validate the methods on a wide spectrum of randomly generated instances. Finally, as managing uncertainty is fundamental for the effectiveness of these operations, distinct strategies are compared, ranging from purely predictive to completely reactive scheduling. We conclude that re-scheduling with high frequency is the best approach to avoid performance deterioration under schedule disruptions, and using the evolved dispatching rules incur fewer deviations from the original schedule.

^{*}Faculdade de Engenharia, Universidade do Porto, Porto, Portugal

[†]INESC TEC - Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Cincia, Porto, Portugal

[‡]Vale S.A., Brazil

Keywords Dynamic Scheduling · Bulk Cargo Ports · Dispatching Rules · Genetic Programming

6.1. Introduction

One-third of the global maritime trade operates dry bulk cargo like iron ore, coal and grains (e.g., wheat and soybean). In particular, iron ore shipments amounted to 1.4 billion tons in 2019, corresponding to 8% of the total maritime volume [UNCTAD, 2020]. Ports are critical for iron ore producers as demand is mainly fulfilled overseas to steel companies. Given their inherent complexity involving several assets (e.g., railway) and processes (e.g., product blending), ports are commonly the bottleneck of iron ore supply chains [Comtois and Slack, 2016]. Despite that, little attention has been directed to bulk operations, especially compared to that devoted to container ports.

Dry bulk ports usually serve as export or import terminals exclusively. Export terminals have three subsystems: the product income, the stockyard and the berths. All of them manage complex operations and are subject to uncertainty. Most research efforts have concentrated on the berths and neglected the inbound operations [Neagoe et al., 2018]. The problem of assigning vessels to berths is well-studied and known as the Berth Allocation Problem (BAP). Some works on the specific case of bulk cargo are found, including particular features as the stock levels and blending [Barros et al., 2011, Umang et al., 2013, Bilgen and Ozkarahan, 2007]. The BAP has also been integrated with other operations, especially in real-world problems. For instance, Menezes et al. [2017] and Robenek et al. [2014] approach case studies from Brazil and middle-east, respectively. In both cases, they solve the integrated planning of the berths, stockyard allocation and product income. Rocha de Paula et al. [2019] and Burdett et al. [2020] study and solve the operations of the three subsystems in terminals from Australia. However, in these works, the inbound resources are considered only for capacity checking – in hourly time buckets. The scheduling of machines and tools for wagons unloading has not been addressed so far.

Ports management must ensure the effectiveness of all operations to meet shipments due dates. In large export terminals, the product is dumped from trains to stockpiles, which requires a route of machines (e.g., a car dumper, conveyor belts and stackers). The schedule includes several routes that share the same machines and must work in parallel. It is challenging to assign routes to wagons and sequence them without conflicts. Moreover, one must satisfy trains' arrivals and railway maneuvers. Research is needed to find effective scheduling methods to optimise the throughput of these inbound operations.

Besides synchronising several machines and avoiding conflicts in the routes, managers must also deal with unexpected disruptions. The process is frequently interrupted by machine breakdowns, arrivals of urgent operations or interdiction of some location in the stockyard. Therefore, in addition to generating effective schedules, providing adequate responses to interruptions is also relevant. As the process involves maneuvers, managers must also prevent the overhead caused by several modifications to the base schedule. This study

aims at developing high-quality initial solutions (deterministic case) and approaches for responding to unexpected disruptions and minimising deviations from the original schedule.

Extensive research has been conducted on strategies to manage uncertainty in scheduling. [Aytug et al. \[2005\]](#) group them into three categories: predictive, completely reactive and predictive-reactive. Predictive strategies generate schedules by predicting all the information (either as deterministic parameters or with their distributions) and using it accordingly. Completely reactive methods, on the opposite, do not generate schedules in advance and select the next operation in real-time. In this case, dispatching rules are usually adopted. Predictive-reactive approaches generate a preliminary schedule and adapt it during execution. [Jain and Foley \[2016\]](#) compare all these approaches and conclude that none of them performs better in all settings. For instance, under high load levels and long interruptions, dispatching rules work better than adapting the original schedule. They also analyse the trade-off between minimising deviations from the baseline and pursuing the best performance.

Our paper provides multiple contributions to the study of bulk cargo operations. First, we introduce and formalise a new problem found in export terminals - the Wagon Unloading Problem. Solution methods are proposed for both the deterministic and the stochastic versions of the problem. The effectiveness of these methods is demonstrated in extensive computational experiments. We use real data from two of the world largest iron ore ports, solving huge scheduling problems of hundreds of machines. Moreover, we analyse the performance of dispatching rules and predictive-reactive strategies using a stochastic simulation model. This analysis extends previous works by tackling the trade-off between maximising performance and minimising deviations from a baseline schedule.

The problem is modelled as a variant of the Multimode Multiprocessor Task Scheduling Problem (MMTSP), where each task has distinct execution modes [[Drozdowski, 2009](#)]. In the application tackled here, tasks are unloading operations, and each possible route is an execution mode. Additional constraints are incorporated to address the specificities of the problem, such as trains' arrivals and railway maneuvers. We present a Mixed Integer Programming formulation and a Constraint Programming model for the deterministic version. Moreover, we design a greedy-randomised heuristic using Genetic Programming (GP), an evolutionary hyper-heuristic that has been successfully applied on production scheduling [[Nguyen et al., 2017](#)]. GP is employed on the generation of priority rules that are used to select operations. In the stochastic context, we use GP to evolve dispatching rules. After all, we evaluate three approaches: following the baseline schedule, re-scheduling, and dispatching rules. The aim is to analyse the uncertainty level in which well-performing deterministic methods deteriorate relative to dynamic scheduling strategies and which approach performs best in each circumstance. All these methods are tested in instances from real ports, and we also generate a set of fictitious instances to represent an extensive range of scenarios.

The remainder of the paper is organised as follows. Section 6.2 briefly describes the literature on the MMTSP and similar problems. As we also deal with the stochastic version of the problem, we present the main approaches adopted to cope with uncertainty in schedul-

ing. In Section 6.3, the wagons scheduling problem is described in detail. The exact models are presented in Section 6.4.1. First, we present the Mixed Integer Programming formulation and then the Constraint Programming model. Section 6.5 concerns the Genetic Programming algorithm. We describe the greedy-randomised heuristic and how we use the priority and dispatching rules. We also detail the specific GP developed for this work. Section 6.6 describes the experiments design, the instance sets and the algorithm parameters. The computational results are presented in Section 6.7, where we describe the evolved rules and compare the performance of the proposed approaches. We also present the performance of dispatching rules and other approaches under uncertainty. Conclusions and future work are given in Section 6.8.

6.2. Background

Multiprocessor Task Scheduling (MTS) is a class of scheduling problems in which some tasks are executed simultaneously by more than one resource. Applications are found in parallel computing systems, where jobs are distributed among several processors [Switalski and Seredynski, 2015], in some variants of the BAP when a ship requires more than one quay crane [Türkoğullar et al., 2014], and in workforce allocation when operation times depend on the number of workers assigned [Battaia et al., 2015]. Drozdowski [2009] presents an extensive review and classification of the distinct variants of MTS problems.

The Multimode Multiprocessor Task Scheduling Problem (MMTSP) is a variant of MTS in which tasks have multiple execution modes. Each mode comprises a subset of machines and a specific processing time. Although several applications may be found in the industry, the problem received little attention from scheduling literature. Błażewicz et al. [1992] prove that the problem is NP-hard by showing that the particular case with three machines is strongly NP-hard. Meta-heuristics based on intensification and diversification were developed in the work of Caramia and Giordani [2009] and Caramia and Giordani [2010]. In Bianco et al. [1999], the authors design two constructive heuristics that outperform former work in the variant with precedence constraints. In Ferreira et al. [2021a], the authors study the scheduling of Human-Robot teams as a specific case of the MMTSP. A GP algorithm and a Constraint Programming model are proposed. Both methods present promising results for several instances of the problem.

None of these previous works took into account the stochastic aspect of real-world applications. In practice, unexpected events often interrupt the execution of planned schedules. Therefore, besides providing optimal and near-optimal solutions for deterministic settings, one must study methods for repairing the schedules under disruptions. This is particularly relevant in the current application as port management involves several machines and is subject to frequent disruptions [Pant et al., 2014]. In such environments, the sequence of operations is dynamically adapted during the execution.

There are several strategies for dynamic scheduling. This work follows the classification from Aytug et al. [2005] that groups them into proactive, reactive and predictive-reactive approaches. Proactive approaches generate a robust schedule based on some prediction

of the system behaviour. Xiong et al. [2013], for instance, propose a multi-objective evolutionary algorithm for generating robust schedules with minimum makespan. Surrogate models use the system information to measure the robustness of the solutions. Although these proactive methods have gained some research effort, their implementation is complex because the performance depends on the predictive information obtained.

Reactive approaches take no decision in advance. These strategies usually adopt fast and simple dispatching rules, which prioritise jobs for execution in real-time. Research on the performance of these rules has a long tradition, particularly in dynamic job shops. We refer to Haupt [1989] and Ramasesh [1990] for a comprehensive classification of them. They have been extensively adopted in dynamic manufacturing systems, where optimal schedules quickly become sub-optimal. Effective dispatching rules have been developed for several scheduling problems, particularly in the semiconductor industry [Varadarajan and Sarin, 2006]. Chen and Matis [2013] design a well-performing rule for minimising weighted tardiness for job shops. Rules are also derived in Xiong et al. [2017] for a variant of the dynamic Job Shop Scheduling Problem in which jobs arrive in batch. In Su et al. [2017], authors address a dynamic parallel machine scheduling problem using a new composite rule. Recently, GP has been employed on the automated generation of rules with promising results [Nguyen et al., 2017, Ferreira et al., 2021a].

This paper discusses the performance of predictive-reactive scheduling, an intermediate between the other two approaches, where a baseline schedule of the production system is adapted during the execution. Particularly, we examine the use of re-scheduling and dispatching rules combined with a predictive schedule. Ouelhadj and Petrovic [2008] classify re-scheduling strategies according to two criteria, *when-to-schedule* and *how-to-schedule*. Three approaches compose the first criteria: *periodic* re-scheduling, *event-driven* and *hybrid*. In the *periodic* policy, also known as *rolling horizon*, a schedule is completely generated at regular intervals, considering all available information. The problem is decomposed into several static problems and is usually solved by standard scheduling algorithms. *Event-driven* is the strategy of re-scheduling under unexpected events, such as a job arrival or delay. The *hybrid* policy is a composition of the *periodic* and *event-driven* approaches. In the *how-to-schedule* criterion, authors present two categories: *full scheduling*, in which all the jobs available are considered; and *partial scheduling*, which includes a percentage of the jobs or a part of the time horizon.

Sabuncuoglu and Bayız [2000] developed a relevant study on re-scheduling strategies. They compare the performance of dispatching rules against the use of a beam search algorithm for partial scheduling through extensive computational experiments. The authors conclude that dispatching rules perform well in face of unexpected disruptions and that the performance of the partial scheduling approach improves as the frequency of re-scheduling increases. A classification of previous studies is also presented. In Sabuncuoglu and Kizilisik [2003], the authors propose a simulation-optimisation algorithm for a dynamic Flexible Manufacturing System. Several strategies are tested considering the *when-to-schedule* and *how-to-schedule* criteria. The conclusion is the same as the previous study – the system’s performance gets better as the frequency of re-scheduling increases. Partial scheduling is particularly competitive when there are several machine breakdowns and low flexibility in

the system.

Lou et al. [2012] combine robust scheduling with a reactive approach. The authors use a stochastic model to generate a predictive schedule and repair algorithms to adapt it when unexpected events occur. They show that the combination of both approaches generates more robust solutions than pure reactive methods. Liu and Zhou [2013] address parallel machines re-scheduling under job rework disruptions. Distinct conflicting criteria are considered in a bi-objective problem to minimise the completion time and the number of disrupted jobs. A polynomial-time algorithm is proposed to generate all efficient solutions in both criteria. In Yin et al. [2016], the authors face a bi-objective re-scheduling problem in deterministic parallel machines. A set of jobs must be re-scheduled to minimise both the mean completion time and the difference from the original schedule. The aim is to determine the set of Pareto-optimal solutions. They work with two variants of the problem (with fixed and variable number of machines) and show that both are NP-hard in the strong sense.

In a nutshell, the best way to deal with unexpected disruptions in scheduling depends on their frequency, nature (whether resource-related or job-related) and environment characteristics. Jain and Foley [2016] explore three approaches: no change in the original schedule, re-scheduling using heuristics, and switching to a completely reactive mode. Several methods are proposed and compared using simulation in manufacturing scenarios. The authors conclude that the proposed *Pairwise Exchange Heuristic* outperforms the other methods at low load level or shortest interruptions. The dispatching rules work better at high load levels and prolonged interruptions. The authors also highlight the relevance of avoiding modifications in the original schedule. They argue that the response to unexpected events needs to be surgical to avoid significant changes in promised delivery times.

Our work distinguishes from the above studies in three main aspects. Firstly, this is the only work addressing the optimisation of inbound operations in bulk ports. Despite some methods for scheduling multi-mode tasks have been proposed, none of them applies to the particularities of this application. Secondly, Jain and Foley [2016] employ a single interruption and measure the system behaviour for some hours after the interruption. We extend their work by exploring those strategies under uncertain processing times and using a stochastic simulation model. Third, none of the studies above evaluated their methods in large-sized practical instances. Several works are limited to 10 machines, while real application problems involve hundreds of machines as in bulk ports. Real scenarios from large ports are considered in our experiments, demonstrating the scalability and applicability of the proposed approaches.

6.3. The Wagon Unloading Problem

The problem concerns unloading trains to stockpiles. In this operation, the product is dumped from wagons and moved through the stockyard. We denote the set of all machines that transport the product from a wagon to a stockpile by route. A route starts at a car dumper, comprises several conveyor belts and ends at a stacker that piles up the material.

Stackers are assigned to a location of the stockyard and reach only the stockpiles located there. A relevant aspect of the problem is that all machines in the route are used during all the unloading operation. To minimise the number of railway maneuvers, wagons are grouped into lots. Each lot has a set of eligible stockpiles – each route a small set of piles. Since each stockpile can be reached by a subset of routes, we can say that each lot has also a set of eligible routes. One must assign lots to routes and sequence them without overlap in every machine. The objective is to maximise the number of wagons unloaded in a given planning horizon.

Formally, let $\mathcal{L}$ be the set of lots to be unloaded, $\mathcal{M}$ be the set of machines and $\mathcal{R}$ be the set of routes in the port. The machines of route $r \in \mathcal{R}$ are given by $\mathcal{M}_r \subset \mathcal{M}$. Each lot $l \in \mathcal{L}$ has a set of eligible routes $R_l \subset \mathcal{R}$, composed of all routes whose stacker reaches some pile eligible for l . The operation of lot l must start after its arrival time a_l . The processing time of lot l using route r is p_{lr} . We model this problem as an MMTSP, which is strongly NP-hard [Drozdowski, 2009]. In the MMTSP, a set of tasks must be performed by machines. Each task must be assigned to an execution mode, and each mode is a subset of machines. In this work, the set of tasks is given by $\mathcal{L}$ and each lot $l \in \mathcal{L}$ has a set of execution modes given by R_l .

Figure 6.1 illustrates a stockyard composed of four piles. At the bottom of the figure, a graph represents the routes – each node is a machine and arcs indicate the flow direction. In the example, $\mathcal{L}$ is composed of two lots of five wagons each. Suppose that, given their product composition, only stockpiles 1 and 4 are eligible for these lots. They must be assigned to routes that end at stackers that reach those piles. Two possible routes are $CD1 - CB1 - CB4 - CB6 - S1$ and $CD3 - CB3 - CB4 - CB7 - S4$. As these routes share a machine (CB4), they cannot operate simultaneously.

To accommodate the specificities of the wagons unloading application, additional features are incorporated into the classical MMTSP:

- **Railway maneuver** – At the train arrival, an *initial maneuver* is performed to transport the lots to a buffer yard. The arrival time of lot l , a_l , includes the train arrival time plus the time for this maneuver. Just before starting any unloading operation, the *standard maneuver* takes the lot to the car dumper. During this maneuver, the car dumper must remain idle.
- **Sequence-dependent setup times** – Each machine is subject to a setup between two operations of distinct routes. If two subsequent operations use distinct routes that have at least one machine in common, a setup time must be inserted between those operations.
- **Machines maintenance** – Ports conduct frequently planned maintenance tasks, which take significant time from the machines, and hence must be considered when scheduling them. During the maintenance time of some machine, no operation using that machine can be performed.
- **Simultaneous use of machine** – Some ports, as the ones from our case study, have

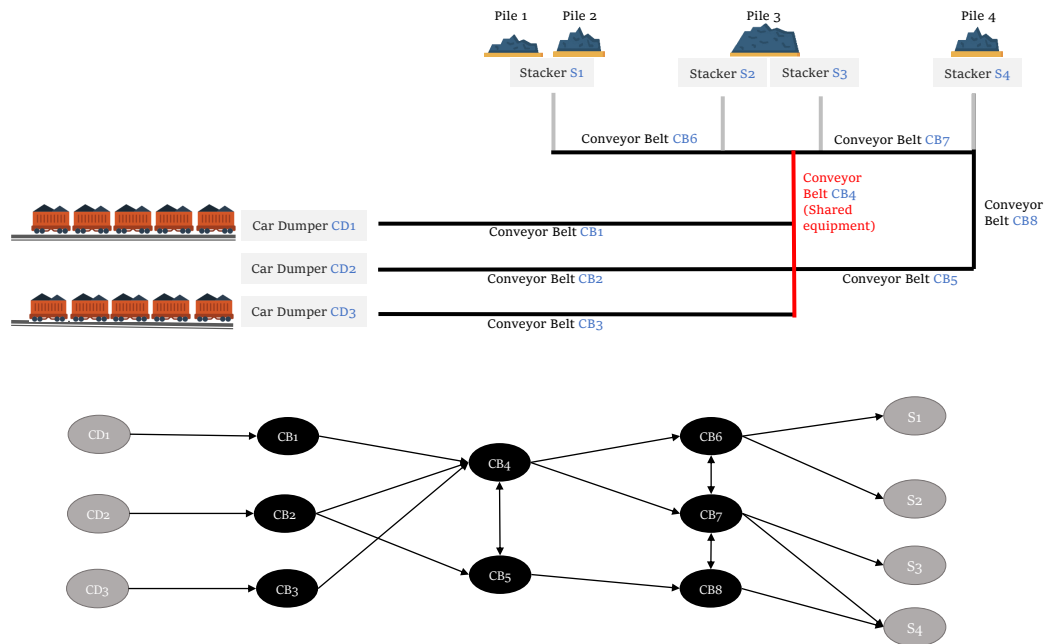


Figure 6.1 – The wagons unloading operation and the unloading routes.

conveyors and stackers with double the capacity of the car dumpers. These machines may be used simultaneously by two distinct lots if the two operations are directed to the same stockpile.

Figure 6.2 presents a Gantt chart of the car dumpers, where a schedule of eight lots and one maintenance task is given. Before each operation, a maneuver is executed. In general, setups are performed in parallel to the maneuvers. Exceptionally, the operation of lot 8 must wait for the finish of lot 5 and the setup of machine S3. In the example, machines CB4 and S2 have double capacity, which allows two lots simultaneously. That is the case of the lots in blue boxes (1 and 3, as well as 6 and 7). The lots in grey boxes are unloaded by dedicated routes.

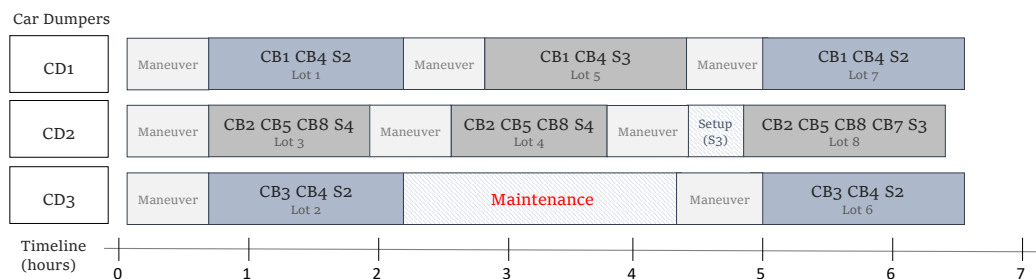


Figure 6.2 – Feasible solution of the problem.

In addition to the aforementioned extensions to the classical MMTSP, uncertainty in the processing times can be considered. We assume that actual processing times are unknown until execution. Thus, the original schedule must be adapted during execution to accommodate deviations and maintain an effective throughput. However, before starting the unloading operation, lots must be transported to car dumpers via railway maneuvers. Changes on the baseline schedule lead to additional maneuvers and possibly overhead in the number of resources required. To avoid this overhead, managers must find a good compromise between modifying the baseline and optimising the throughput. Therefore, the stochastic version of the problem aims at maximising the throughput, while minimising the number of deviations from the baseline schedule. Yin et al. [2016] measure schedule deviations as the difference from original jobs completion times. As our objective is to avoid railway maneuvers, we count the number of changes in car dumpers and in the sequence of lots. The number of deviations of lot l is given by the difference from its original to the new sequence. Moreover, if l is assigned to distinct car dumpers in each schedule, the number of deviations is incremented by one. If l changes the route but maintains the original car dumper and the sequence, no additional maneuver is necessary and no deviation is incurred.

6.4. Exact solution methods

This section describes the exact methods proposed to solve the deterministic version of the problem. At first, we introduce a Mixed Integer Programming (MIP) model, where the formulation proposed by Ferreira et al. [2021a] to the MMTSP is adapted to the current application. Then we present the Constraint Programming model.

6.4.1 Mixed Integer Programming formulation

The problem is defined on a set of lots $\mathcal{L}$, machines $\mathcal{M}$ and routes $\mathcal{R}$. We also define MT , the set of planned maintenance tasks, and each task $j \in MT$ is associated to a machine $m_j \in \mathcal{M}$. The parameters of the problem are summarised in Table 6.1. If m is a car dumper, the setup time $st_{r_1 r_2}^m$ corresponds to the standard maneuver time. Otherwise, $st_{r_1 r_2}^m$ applies only if $r_1 \neq r_2$.

Table 6.1 – Problem Parameters

Parameter	Description
at_l	Arrival time of lot l
p_{lr}	Processing time of lot l in route r
w_l	Number of wagons in lot l
$st_{r_1 r_2}^m$	Setup time of machine m from route r_1 to route r_2
sm_j	Start time of maintenance task j
fm_j	Finish time of maintenance task j
m_j	Machine associated to maintenance task j

Table 6.2 shows the variables of the model. To assist in the formulation, we define set

$A_{l_1 r_1}$ composed of all pairs (l_2, r_2) eligible for simultaneous execution with (l_1, r_1) . In such case, l_1 and l_2 are eligible for at least one pile in common and this pile is reached by routes r_1 and r_2 . Moreover, $(l_2, r_2) \in A_{l_1 r_1}$ only if all machines in $\mathcal{M}_{r_1} \cap \mathcal{M}_{r_2}$ have double capacity. Variable $YP_{l_1 r_1 l_2 r_2}^m$ indicates that l_1 and l_2 share machine m and use routes r_1 and r_2 , respectively. By definition, if $(l_2, r_2) \notin A_{l_1 r_1}$ or if m is a single-capacity machine, $YP_{l_1 r_1 l_2 r_2}^m = 0$.

Table 6.2 – MIP Variables

Name	Type	Description
Y_{lr}	Binary	1 if lot l is unloaded via route r ; 0 otherwise.
$X_{ml_1 l_2}$	Binary	1 if l_1 and l_2 use machine m and l_1 unloads before l_2 ; 0 otherwise.
$YP_{l_1 r_1 l_2 r_2}^m$	Binary	1 if l_1 and l_2 unload using routes r_1 and r_2 , respectively, and sharing machine m ; 0 otherwise.
XMB_{lj}	Binary	1 if lot l use machine m_j before maintenance task j ; 0 otherwise.
XMA_{lj}	Binary	1 if lot l use machine m_j after maintenance task j ; 0 otherwise.
N_{lr}	Integer	Number of wagons of lot l unloaded via r before hz
S_{lr}	Continuous	Start time of lot l in route r
F_{lr}	Continuous	Finish time of lot l in route r
Z_{lr}	Continuous	Minimum between F_{lr} and hz

In the MIP model, it is not mandatory to plan all lots. The objective function (6.1) aims at maximising the total number of wagons unloaded in the planning horizon hz . The parameter H is an upper bound for the maximum finish time of any planned lot, calculated as $hz + \hat{p}$, where $\hat{p} = \max_{l \in \mathcal{L}, r \in R_l} p_{lr}$. The model is given below.

$$\text{Max} \quad \sum_{l \in \mathcal{L}} \sum_{r \in R_l} N_{lr} \quad (6.1)$$

Subject to:

$$\sum_{r \in R_l} Y_{lr} \leq 1 \quad \forall l \in \mathcal{L} \quad (6.2)$$

$$F_{lr} \geq S_{lr} + p_{lr} - H \cdot (1 - Y_{lr}) \quad \forall l \in \mathcal{L}, \forall r \in R_l \quad (6.3)$$

Constraints (6.2) ensure that each lot is assigned to at most one route and constraints (6.3) calculate the finish time of each lot according to its start time and assigned route.

$$\sum_{(l_2, r_2) \in A_{l_1 r_1}} YP_{l_1 r_1 l_2 r_2}^m \leq Y_{l_1 r_1} \quad \forall l_1 \in \mathcal{L}, \forall r_1 \in R_{l_1}, \forall m \in \mathcal{M}_{r_1} \cup \mathcal{M}_{r_2} \quad (6.4)$$

$$YP_{l_1 r_1 l_2 r_2}^{m_1} = YP_{l_1 r_1 l_2 r_2}^{m_2} \quad \forall l_1 \in \mathcal{L}, \forall r_1 \in R_{l_1}, \forall (l_2, r_2) \in A_{l_1 r_1}, \forall m_1, m_2 \in \mathcal{M}_{r_1} \cup \mathcal{M}_{r_2} \quad (6.5)$$

$$YP_{l_1 r_1 l_2 r_2}^m = YP_{l_2 r_2 l_1 r_1}^m \quad \forall l_1 \in \mathcal{L}, \forall r_1 \in \mathcal{R}_{l_1}, \forall (l_2, r_2) \in A_{l_1 r_1}, \forall m \in \mathcal{M}_{r_1} \cup \mathcal{M}_{r_2} \quad (6.6)$$

If m is a single-capacity machine, $YP_{l_1 r_1 l_2 r_2}^m = 0$ by definition. Constraints (6.4) guarantee that lot l_1 is planned to use route r_1 in a simultaneous operation only if its assigned to route r_1 in variable $Y_{l_1 r_1}$. Constraints (6.5) provide consistent planning for all machines used in simultaneous operations, and constraints (6.6) ensure consistency between variables $YP_{l_1 r_1 l_2 r_2}^m$ and $YP_{l_2 r_2 l_1 r_1}^m$.

$$S_{l_2 r_2} \geq F_{l_1 r_1} + st_{r_1 r_2}^m + H \cdot (Y_{l_1 r_1} + Y_{l_2 r_2} - YP_{l_1 r_1 l_2 r_2}^m + X_{ml_1 l_2} - 3) \quad \forall \langle l_1, r_1, l_2, r_2, m \rangle \in T \quad (6.7)$$

$$S_{l_1 r_1} \geq F_{l_2 r_2} + st_{r_2 r_1}^m + H \cdot (Y_{l_1 r_1} + Y_{l_2 r_2} - YP_{l_1 r_1 l_2 r_2}^m - X_{ml_1 l_2} - 2) \quad \forall \langle l_1, r_1, l_2, r_2, m \rangle \in T \quad (6.8)$$

The set T is composed of all tuples $\langle l_1, r_1, l_2, r_2, m \rangle$ in which r_1 is a route eligible for lot l_1 , r_2 is eligible for l_2 , and machine m is present both in r_1 and r_2 . Formally if $\langle l_1, r_1, l_2, r_2, m \rangle \in T$, $r_1 \in \mathcal{R}_{l_1}$, $r_2 \in \mathcal{R}_{l_2}$ and $m \in \mathcal{M}_{r_1} \cap \mathcal{M}_{r_2}$. Constraints (6.7) and (6.8) ensure that each single capacity machine will not process more that one lot at a time (these are the “no-overlap” constraints). If $YP_{l_1 r_1 l_2 r_2}^m = 1$, all double capacity machines in $\mathcal{M}_{r_1} \cap \mathcal{M}_{r_2}$ may overlap in the operation of l_1 and l_2 .

$$S_{lr} \geq at_l \cdot Y_{lr} \quad \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l \quad (6.9)$$

$$S_{lr} \leq hz \cdot Y_{lr} \quad \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l \quad (6.10)$$

$$Z_{lr} \leq F_{lr} \quad \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l \quad (6.11)$$

$$Z_{lr} \leq hz \cdot Y_{lr} \quad \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l \quad (6.12)$$

The start time of each lot is bounded by its arrival time in Constraints (6.9). The upper bound of any start time is given by the planning horizon in Constraints (6.10). Constraints (6.11) and (6.12) ensure that Z_{lr} is the minimum between the finish time of each lot and the planning horizon hz , or 0 if lot l is not assigned to route r .

$$N_{lr} \leq \frac{w_l}{p_{lr}} \cdot (Z_{lr} - S_{lr}) \quad \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l \quad (6.13)$$

Constraints (6.13) calculate the number of wagons unloaded, which depends on the time taken from S_{lr} and Z_{lr} (the operation time inside the planning horizon).

$$Y_{lr} \leq XMB_{lj} + XMA_{lj} \quad \forall j \in MT, \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l : m_j \in \mathcal{M}_r \quad (6.14)$$

$$sm_j \geq F_{lr} + H \cdot (Y_{lr} + XMB_{lj} - 2) \quad \forall j \in MT, \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l : m_j \in \mathcal{M}_r \quad (6.15)$$

$$S_{lr} \geq fm_j + H \cdot (Y_{lr} + XMA_{lj} - 2) \quad \forall j \in MT, \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l : m_j \in \mathcal{M}_r \quad (6.16)$$

If machine m_j is used in the operation of lot l , variables XMB_{lj} and XMA_{lj} define if the operation is performed before task j . If $Y_{lr} = XMB_{lj} = 1$, constraints (6.15) ensure that lot l is unloaded before j . Otherwise, if $Y_{lr} = XMA_{lj} = 1$, constraints (6.16) ensure that task j is executed before lot l .

$$Y_{lr} \in \{0, 1\} \quad \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l \quad (6.17)$$

$$X_{ml_1l_2} \in \{0, 1\} \quad \forall l_1, l_2 \in \mathcal{L}, \forall m \in \mathcal{M} \quad (6.18)$$

$$Y_{l_1r_1l_2r_2}^m \in \{0, 1\} \quad \forall l_1, l_2 \in \mathcal{L}, \forall r_1 \in \mathcal{R}_{l_1}, \forall r_2 \in \mathcal{R}_{l_2}, \forall m \in \mathcal{M}_{r_1} \cup \mathcal{M}_{r_2} \quad (6.19)$$

$$XMB_{lj}, XMA_{lj} \in \{0, 1\} \quad \forall j \in MT, \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l : m_j \in \mathcal{M}_r \quad (6.20)$$

$$N_{lr} \in \mathbb{Z}^+ \quad \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l \quad (6.21)$$

Constraints (6.17) to (6.21) define the domain of the binary and integer variables. The continuous variables are non-negative: $S_{lr}, F_{lr}, Z_{lr} \geq 0, \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l$.

6.4.2 Constraint Programming model

Constraint Programming (CP) has been recently applied in complex scheduling problems [Gökgür et al., 2018, Åstrand et al., 2020, Lunardi et al., 2020]. The approach is particularly effective in problems with a high number of constraints limiting the search space. In Ferreira et al. [2021b], a CP model is designed for the MMTSP. The algorithm outperforms existing heuristics and motivates the development of a specific model for the current application. An overview of Constraint Programming applied to scheduling problems is found in Baptiste et al. [2001].

CP algorithms solve a Constraint Satisfaction Problem (CSP), composed of variables with a finite set of domain values. Constraints limit the combination of values that the variables set may assume. A feasible solution of a CSP is an assignment of variables to values that satisfy all constraints [Tsang, 1993]. CP algorithms generally have two phases. In the first, constraint propagation procedures filter the variables domain and reduce the search space. Then, an exhaustive search throughout the reduced search space is performed. When solving optimisation problems, the CSP has an objective function f and aims at finding the solution S^* that satisfies the CSP and optimises f . If the algorithm completes the search by enumerating all feasible solutions, S^* is the optimal solution to the problem.

Our CP model solves the problem described in Section 6.3. Table 6.3 presents the variables set. Interval variables represent the period of time of operations, from start to finish. We define four types of interval variables, as shown in the table: L_l , LR_{lr} , $Par_{l_1r_1l_2r_2}$ and $Par2_{l_1r_1l_2r_2}$. Variable L_l model the start and finish time of lot $l \in \mathcal{L}$. If l_1 unloads by the dedicated route r_1 , the value of L_{l_1} is given by variable $LR_{l_1r_1}$. Otherwise, if l_1 unloads in a

simultaneous operation with lot l_2 , the value of L_{l_1} is given by variable $Par_{l_1 r_1 l_2 r_2}$, where r_1 and r_2 are the double-capacity routes involved. In such case, it is not mandatory that lots l_1 and l_2 start and finish at the exact same time. If lot l_1 starts earlier than l_2 , for instance, the start time of $Par_{l_1 r_1 l_2 r_2}$ is earlier than the start time of $Par_{l_2 r_2 l_1 r_1}$. Variable $Par2_{l_1 r_1 l_2 r_2}$ represents the whole interval in which both lots operate.

Table 6.3 – Variables of the Constraint Programming model

Variable	Type	Description
L_l	Interval	Time of lot l unloading
LR_{lr}	Interval	Time of l unloading using route r
$Par_{l_1 r_1 l_2 r_2}$	Interval	Time of l_1 unloading using r_1 in parallel to l_2 unloading using r_2
$Par2_{l_1 r_1 l_2 r_2}$	Interval	Time of both lots l_1 and l_2 unloading using routes r_1 and r_2 in parallel
E_l	Integer	Minimum between the finish time of lot l and hz
W_{lr}	Integer	Number of wagons from lot l unloaded via route r
Seq_m	Sequence	Sequence of operations of machine m

For each machine $m \in \mathcal{M}$, a sequence variable Seq_m is defined. The value of each variable Seq_m is the order in which machine m executes its operations. Thus, Seq_m is defined on the set of interval variables that involve route $r : m \in \mathcal{M}_r$. The objective of this variable is to set the value of each variable LR_{lr} and $Par2_{l_1 r_1 l_2 r_2}$ without overlap in the use of machine m .

The model is described below. Functions $start_of(X)$ and $end_of(X)$ return, respectively, the start and finish time of each interval variable X . The objective function (6.22) is to maximise the number of wagons unloaded before the end of the planning horizon hz .

$$\text{Max} \quad \sum_{l \in \mathcal{L}, r \in \mathcal{R}} W_{lr} \quad (6.22)$$

$$alternative(L_l, LR_{lr} | r \in \mathcal{R}_l \cup Par_{lr l_2 r_2} | r \in \mathcal{R}_l, < l_2, r_2 > \in A_{lr}) \quad \forall l \in \mathcal{L} \quad (6.23)$$

The function $alternative()$ in Constraints (6.23) sets the value of L_l equals to one of the variables LR_{lr} or $Par_{lr l_2 r_2}$. A route r must be selected for lot l . If r is used in dedicated mode, $L_l = LR_{lr}$. Otherwise, l unloads in parallel to lot l_2 using route r_2 and $L_l = Par_{lr l_2 r_2}$.

$$presence_of(Par_{lr l_2 r_2}) \Rightarrow presence_of(Par_{l_2 r_2 lr}) \quad \forall l \in \mathcal{L}, \forall r \in \mathcal{R}_l, \forall < l_2, r_2 > \in A_{lr} \quad (6.24)$$

Constraints (6.24) ensure consistency between variables $Par_{lr l_2 r_2}$ and $Par_{l_2 r_2 lr}$.

$$start_of(L_l) \geq at_l \quad \forall l \in \mathcal{L} \quad (6.25)$$

$$E_l = \min(hz, \text{end_of}(L_l)) \quad \forall l \in \mathcal{L} \quad (6.26)$$

$$(6.27)$$

Constraints (6.25) ensure that l unloads after its arrival time. Constraints (6.26) calculate the value of E_l , which is the minimum between hz and the finish time of lot l . This is necessary to calculate the number of wagons unloaded before hz .

$$\text{presence_of}(LR_{lr}) \Rightarrow W_{lr} = (\text{start_of}(LR_{lr}) - E_l) \cdot \frac{w_l}{p_{lr}} \quad \forall l \in \mathcal{L}, \forall r \in R_l \quad (6.28)$$

$$\text{presence_of}(Par_{lr_1l_2r_2}) \Rightarrow W_{lr} = (\text{start_of}(Par_{lr_1l_2r_2}) - E_l) \cdot \frac{w_l}{p_{lr}} \quad \forall l \in \mathcal{L}, \forall r \in R_l, \forall <l_2, r_2> \in A_{lr} \quad (6.29)$$

In constraints (6.28), the number of wagons W_{lr} is calculated from variable LR_{lr} and in constraints (6.29), variable $Par_{lr_1l_2r_2}$ defines the number of wagons.

$$\text{start_of}(LR_{lr}) \geq fm_j \parallel \text{end_of}(LR_{lr}) \leq sm_j \quad \forall j \in MT, \forall l \in \mathcal{L}, \forall r \in R_l \cap \mathcal{R}_{m_j} \quad (6.30)$$

$$\text{start_of}(Par_{l_1r_1l_2r_2}) \geq fm_j \parallel \text{end_of}(Par_{l_1r_1l_2r_2}) \leq sm_j \quad \forall j \in MT, \forall l_1 \in \mathcal{L}, \forall r_1 \in R_{l_1} \cap \mathcal{R}_{m_j}, \forall <l_2, r_2> \in A_{l_1r_1} \quad (6.31)$$

Constraints (6.30) and (6.31) ensure that the unloading operation of each lot is scheduled before the start or after the finish of each maintenance task $j \in MT$. We define $\mathcal{R}_m \subset \mathcal{R}$ as the subset of routes composed of machine m . Given machine m and lot l , the subset $R_l \cap \mathcal{R}_m$ is composed of all routes that include machine m and are eligible for lot l .

$$\text{start_of}(Par2_{l_1r_1l_2r_2}) \leq \text{start_of}(Par_{l_1r_1l_2r_2}) \quad \forall l_1 \in \mathcal{L}, \forall r_1 \in R_{l_1}, \forall <l_2, r_2> \in A_{l_1r_1} \quad (6.32)$$

$$\text{end_of}(Par2_{l_1r_1l_2r_2}) \geq \text{end_of}(Par_{l_1r_1l_2r_2}) \quad \forall l_1 \in \mathcal{L}, \forall r_1 \in R_{l_1}, \forall <l_2, r_2> \in A_{l_1r_1} \quad (6.33)$$

Variables $Par2_{l_1r_1l_2r_2}$ represent the whole interval when lots l_1 and l_2 uses routes r_1 and r_2 if they operate simultaneously. Constraints (6.32) and (6.33) calculate the start and finish of $Par2_{l_1r_1l_2r_2}$, respectively.

$$\text{no_overlap}(Seq_m, st_{r_1r_2}^m \forall r_1, r_2 \in \mathcal{R}) \quad \forall m \in \mathcal{M} \quad (6.34)$$

In a feasible solution, each single-capacity machine $m \in \mathcal{M}$ can process at most one lot and each double-capacity machine operates at most two lots simultaneously. This is guaranteed by constraints (6.34). By associating variables $LR_{lr} : m \in \mathcal{M}_r$ to the sequence variable Seq_m we impose that no operation in dedicated mode is executed with overlap in machine m . Similarly, if m is a double-capacity machine, we associate variables $Par2_{l_1r_1l_2r_2} : m \in \mathcal{M}_{r_1} \cap \mathcal{M}_{r_2}$ to Seq_m . Thus, operations of lots l_1 and l_2 can overlap in the use of machine m , but no other operation can overlap them. The function `no_overlap()` also accepts the definition of setup times $st_{r_1r_2}^m$ between each pair of operations.

As the constraint programming solvers do not scale well for larger-sized instances, an heuristic approach is also provided. The following section details the GP algorithm developed in this work.

6.5. GP-based solution methods

Genetic Programming has been successfully applied as a hyper-heuristic on the design of new heuristics for optimisation problems. Similarly to Genetic Algorithms, GP is an evolutionary and populational approach. However, instead of evolving solution candidates for a problem, it searches on the space of heuristics that can be created. In GP, individuals are mathematical expressions that must be executed and/or interpreted. The objective is to find the individual with the best ability in solving the problem.

This work applies GP for two purposes. First, the hyper-heuristic generates priority rules for the deterministic version of the problem. A greedy-randomised heuristic (GRH) uses the evolved rules in the generation of static schedules. Then we use GP for evolving dispatching rules for the stochastic version. The GRH heuristic and the specific GP algorithm developed in this work are described in the next sections.

6.5.1 Greedy-randomised heuristic

The premise of the GRH algorithm is that no single rule performs best in all instances. Thus, many rules are applied for prioritising operations and generating schedules. However, instead of always selecting the best priority, GRH employs a greedy-randomised approach. The pseudo-code is provided in Algorithm 8, which receives as input an instance i and a set of priority rules R . Until reaching a stopping criterion, GRH uses each rule $pr \in R$ for a given number of consecutive iterations ($max_iterations_{pr}$) in turn. At each iteration, a complete schedule S is generated (line 10). The general idea is to adapt the value of $max_iterations_{pr}$ to the performance of pr on instance i and let the best-performing rules run for more iterations. Hence, at the beginning of the algorithm, all rules receive the same number of iterations (line 2). Every time pr finds a new incumbent solution S (which throughput τ_S is the best found so far), the value of $max_iterations_{pr}$ is increment by one (lines 11-13). The stopping criterion is reaching a given number of total iterations.

The procedure *greedy_randomised_schedule_generation* in line 10 starts with an empty

Input: Instance i , priority rules R , greed factor α

Result: Best solution found S^*

```

1 foreach priority rule  $pr \in R$  do
2   |  $max\_iterations_{pr} \leftarrow 1$ ;
3 end
4  $S^* \leftarrow \emptyset$ ;
5  $pr \leftarrow first\_rule$ ;
6  $iter \leftarrow 0$ ;
7 while not reached stopping criterion do
8   | if  $iter = max\_iterations_{pr}$  then
9     |    $pr \leftarrow next\_rule$ ;
10    |    $iter \leftarrow 0$ ;
11   | end
12   |  $S \leftarrow greedy\_randomised\_schedule\_generation(\alpha, pr, i)$ ;
13   | if  $\tau_S > \tau_{S^*}$  then
14     |    $S^* \leftarrow S$ ;
15     |    $max\_iterations_{pr} \leftarrow max\_iteration_{pr} + 1$ ;
16   | end
17   |  $iter \leftarrow iter + 1$ ;
18 end

```

Algorithm 8: Greedy randomised constructive algorithm

schedule. Once a machine is available, rule r is applied to select the next pair <lot, route> and insert it in the schedule. Instead of selecting the pair <lot, route> with the best score, a list of candidates is created, containing the best $\alpha\%$ pairs available for execution. Then, the next pair to be executed is randomly selected from the candidate list. Hence, the parameter α indicates the algorithm's greediness (if $\alpha = 1$, a completely random approach is performed). When inserting a new operation using route r in the schedule, the starting time is the earliest time when all machines in r are available.

6.5.2 Genetic Programming

In the deterministic version of the problem, the search power of GP leverages GHR's performance by providing effective rules. In the stochastic case, GP evolves dispatching rules for sequencing the operations in a completely reactive approach. This section describes the specific GP developed in this work, applied in both versions of the Wagon Unloading Problem, which differ on their objectives. In the deterministic case, GP aims to find rules that maximise the throughput. In the version with stochastic processing times, the aim is also to minimise deviations from a baseline schedule.

Figure 6.3 illustrates the general procedure of GP algorithms, which basically comprises a training and a validation stage. During training, the algorithm starts with a randomly created population and evolve it over several generations. Each individual a is evaluated by running a simulator of the scheduling environment on a set of training instances. In

the evaluation of a , the corresponding rule is applied by the simulator. Then a fitness score is assigned to a representing its overall performance on the training set. The best individuals are reproduced and combined by the genetic operators, crossover and mutation. The crossover operator selects two nodes from distinct individuals and swaps the sub-trees rooted by those nodes. Mutation selects a node from any individual tree and replaces the sub-tree rooted by that node with a randomly generated tree. Both genetic operators use tournament on the selection of individuals. Once reaching a stopping criterion, the training stage finishes, and the best individual found is evaluated on the validation instance set. The pseudo-code of the GP algorithm is found in Section 6.A from Appendix.

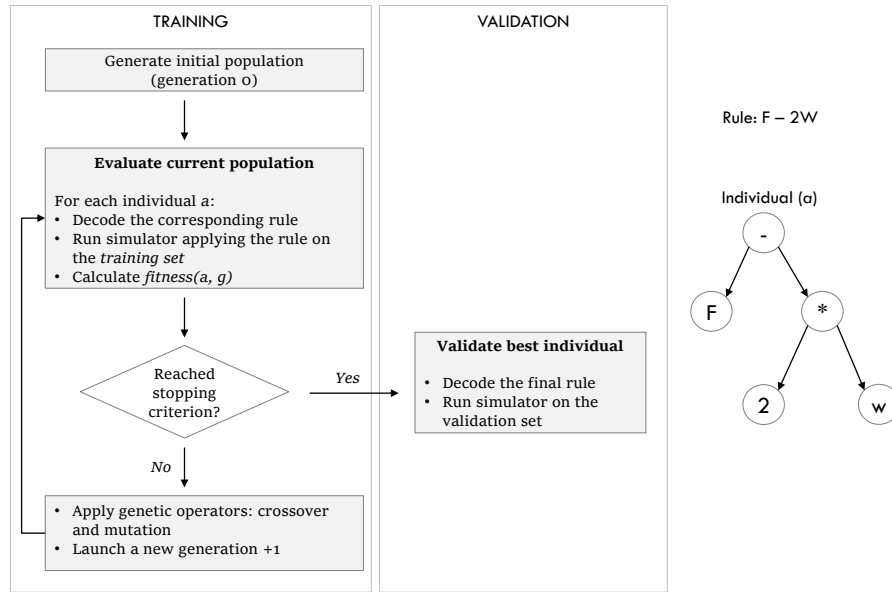


Figure 6.3 – The two stages of GP algorithm (training and validation) and example of a GP individual tree

Figure 6.3 also presents an example of an individual and its corresponding rule. The most common representation are symbolic expression trees, whose leaf nodes are *terminals* related to the specific problem. The internal nodes are *functions* that operate the terminals. In the GP designed in this work, each tree, a , represents a (priority or dispatching) rule whose terminals are parameters from the lots and machines. In the figure, individual a is composed of the terminals 2, F (finish time of operation) and W (number of wagons). The mathematical operators $-$ and $*$ compose the function set. By decoding a , rule $F - 2W$ is obtained.

In the simulation model, there is a single queue for all lots arriving at the buffer yard. Given the interdependencies of the assignment and sequence decisions, they are taken together. Thus, the rule selects the next lot for unloading and the route at the same time. Once at least one route is available, rule r assigns scores for each pair $\langle \text{lot}, \text{route} \rangle$. The procedure goes on until all lots are executed. In the stochastic version of the problem, actual processing times p_{lr} are uncertain and sampled from a Gamma distribution [Lawrence and Sewell,

1997]. The coefficient of variation of the distribution (cv) is an instance parameter.

6.5.2.1 Fitness calculation

The fitness function computes the performance of each individual a on a training set of instances. The simulation procedure applies rule r on each instance i , and returns the throughput τ_{ai} , measured as the percentage of wagons unloaded in the planning horizon hz . The average throughput of a in the training set is given by τ_a . In the stochastic version, we aim to find rules that maximise the throughput and minimise deviations from the baseline schedule. Thus, the performance of rule r also depends on dev_a , the average number of deviations from the baseline per lot. Section 6.3 explains how we calculate the number of deviations from the baseline. As throughput and number of deviations have different unit measures, a factor δ is applied to the latter. The performance of individual a is estimated as $\tau_a + dev_a \cdot \delta$. When running GP for the deterministic version, $\delta = 0$ and only τ_a accounts in individuals performance.

GP trees tend to grow during the search and result in complex rules, an effect known as bloat. Several strategies may be applied to avoid this effect. In our case, the tree size is penalised in the fitness function according to ρ , the bloat control factor. The penalty of tree size depends on the number of nodes $|a|$, the number of generations completed g and the value of ρ . Hence, in addition to the rules' performance, the fitness function also considers a strategy to avoid large trees, as in Equation 6.35.

$$fitness(a, g) = \tau_a + dev_a \cdot \delta + |a| \cdot g \cdot \rho \quad (6.35)$$

6.5.2.2 Functions and terminal set

The function set is composed of the four basic mathematical operations: sum (+), subtraction (/), multiplication (*) and the protected division (/), which returns one if the denominator is zero. Table 6.4 shows the terminal set, composed of parameters from the lots and the machines. Terminal F returns the finish time of the current operation if it starts next, considering maneuver and eventual machines' maintenance. Five terminals refer to eligibility, which by definition vary from 0 to 1. The eligibility of lot l is calculated as $R_l/\mathcal{R}$ (terminal LE). The route eligibility (RE) is the percentage of lots eligible for the route, considering all remaining lots. Terminal REAv returns the route eligibility considering only lots available at the current simulation time. Terminals CE and CEAv are similar to RE and REAv, but referring to car dumpers. DCap indicates if some machine in the route has double capacity. Finally, terminal DEV returns the number of deviations from baseline incurred if the current pair <lot, route> is selected. It applies only to the stochastic case.

Table 6.4 – Terminal set of the GP algorithms

Terminal	Description
PT	Operation processing time
F	Operation finish time
W	Number of wagons on the lot
LE	Lot eligibility
RE	Route eligibility on all remaining lots
REAv	Route eligibility on available lots
CE	Car dumper eligibility on all remaining lots
CEAv	Car dumper eligibility on available lots
DCap	1 if route has double capacity, 0 otherwise
DEV	Deviations from baseline

6.6. Experiments design

We evaluate the proposed methods in real and fictitious ports. This section explain the instance generation and the design of the experiments. We also describe the parameters setting of the algorithms.

6.6.1 Instances

The computational experiments use three instance classes. The first two relate to real ports from Vale, and the third comprises fictitious ports. Table 6.5 shows data from each of the real ports and the randomly generated instances. The value of all time-related parameters is presented in minutes. All instances have a planning horizon of one day ($hz = 1440$ minutes).

Vale is the worldwide leader in iron ore production in terms of volume [Comtois and Slack, 2016]. Three production systems in Brazil export the iron ore overseas for steel companies. Our case study is based on two ports from Vale that appear in the top five iron ore terminals [Merk and Dang, 2012]. This work name them as LargePort and ComplexPort. Among both ports, LargePort is the largest in the number of machines and volume, but the operation of ComplexPort is more complex. In this port, lots have a limited number of eligible piles. Thus, it is challenging to keep all car dumpers operating without conflict in routes. LargePort has four stackers with double capacity, and ComplexPort has three. In the instances considered in this work, both ports have a 24 hours maintenance task for one car dumper, and two stackers have 4-5 hours of maintenance.

Table 6.5 – Instances from real and fictitious ports

	LargePort	ComplexPort	General instances
Setup time	15	20	15
Standard maneuver time	55	20	30
time	55	75	30
Number of car dumpers (<i>cds</i>)	8	5	3 to 7
Number of stackers	12	10	3 to 7
Number of conveyors	133	193	3 to 7
Number of routes	544	1453	7 to 54
Number of products	14	74	2 to 20
Avg. number of of piles	155.5	96.0	20.0
Avg. number of piles/lot	30.4	1.5	1.0 to 10.0
Standard number of wagons/lot (<i>w</i>)	110	86	100
Avg.number of lots/train (<i>lt</i>)	3	3	1
Avg. processing time ($\bar{p}$)	137.4	81.2	75.0

The third class of instances (general instances) include 24 fictitious ports, each with a specific number of machines and products. The aim is to validate the methods in a wide spectrum of ports beyond the large real iron ports. The number of machines and routes is smaller than in the real ports to represent simpler operations. Some parameters such as setup time, maneuver times, and the number of wagons per lot do not vary because preliminary experiments concluded that they do not significantly influence the overall output. Processing times are sampled from $U(50, 100)$. The number of products varies from 2 to 20, and we assume that all ports have 20 piles. Each pile and each lot are assigned to one of the products using a uniform distribution. Thus, in ports with larger number of products, each lot has a smaller number of eligible piles. This is relevant because it influences the overall flexibility of the unloading operations. See Section 6.B from Appendix for a detailed presentation of each of the fictitious ports.

In the three instance classes, random trains are generated considering the characteristics of each port: the standard number of wagons per lot, the average number of lots per train and the average number of eligible piles per lot. Arrivals are sampled from a Poisson distribution which mean λ is calculated as follows:

$$\lambda = \frac{u \cdot cds}{\bar{p} \cdot lt},$$

where *cds* is the number of car dumpers in the port and $\bar{p}$ is the average processing time. As λ refers to train arrivals, we multiply $\bar{p}$ by the average number of lots per train, *lt*. From queuing theory, the parameter *u* is an upper bound for the car dumpers utilisation. To incorporate realistic characteristics to the trains, not all lots have the same number of wagons. 90% of them have the standard number of wagons (*w*); 9% of them are divided into two lots of $w/2$ wagons, and 1% of them into three lots containing $w/2$, $w/4$ and $w/4$ wagons.

Table 6.6 presents the instance sets used by the GP algorithm for training and validation.

The latter is also used for evaluating the MIP model and the CP algorithm. The aim is to simulate distinct scenarios, from days with a limited number of train arrivals to extremely congested ones. Thus, we adopt six values for u during validation. During training, we do not consider the extreme values (0.4 and 1.4). We use 6 fictitious port 6 for training and 18 for validation. The coefficient of variation cv defines the uncertainty level in processing times. The actual processing times are sampled from a gamma distribution with mean $\mathbb{E}[p_{lr}]$ and standard deviation $\sigma_{lr} = \mathbb{E}[p_{lr}] \cdot cv$. The higher the value of cv , the higher the uncertainty level. Five levels of cv are simulated when evaluating strategies for responding to schedule disruptions. In the deterministic version, $cv = 0$. For each combination of u , cv and port, we generate 10 distinct instances for training and 20 for validation (there is no intersection in both sets). The total number of instances result from the full-factorial design of all these parameters.

Table 6.6 – GP training and validation sets

		Training set	Validation set
Machines utilisation (u)		0.6, 0.8, 1.0, 1.2	0.4, 0.6, 0.8, 1.0, 1.2, 1.4
Number of fictitious ports		6	18
Deterministic	Variation in processing times (cv)	0	0
	# Instances from LargePort	$4 \cdot 1 \cdot 10 = 40$	$6 \cdot 1 \cdot 20 = 120$
	# Instances from ComplexPort	$4 \cdot 1 \cdot 10 = 40$	$6 \cdot 1 \cdot 20 = 120$
	# General instances	$4 \cdot 6 \cdot 10 = 240$	$6 \cdot 18 \cdot 20 = 2160$
Stochastic	Variation in processing times (cv)	0.4, 0.8	0.2, 0.4, 0.6, 0.8, 1.0
	# Instances from LargePort	$4 \cdot 1 \cdot 2 \cdot 10 = 80$	$6 \cdot 1 \cdot 5 \cdot 20 = 600$
	# Instances from ComplexPort	$4 \cdot 1 \cdot 2 \cdot 10 = 80$	$6 \cdot 1 \cdot 5 \cdot 20 = 600$
	# General instances	$4 \cdot 6 \cdot 2 \cdot 10 = 480$	$6 \cdot 18 \cdot 5 \cdot 20 = 10800$

6.6.2 Algorithms parameters

Table 6.7 summarises the algorithm parameters adopted after preliminary experiments. The mutation probability is high, 0.8, as it effectively provides diversity in the algorithm search. Conversely, if crossover probability or tournament size is high, the evolution tends to stagnate too early with a homogeneous population. Thus, crossover probability is set to 0.2 and the tournament size to the minimum value, 2. We set the maximum depth of trees to the standard value of 17 for avoiding premature prune of good branches. The probability of selecting a terminal and a non-terminal (0.1 and 0.9, respectively) and the parameters of the Ramped Half-and-Half algorithm (minimum tree depth of 2 and maximum 6) are also set to standard values.

In order to tune the bloat control strategy, we ran 10 executions of GP for each value of $\rho = \{10^{-4}, 10^{-5}, 10^{-6}, 0\}$. We observed that ρ is effective for bloat control and helps to find smaller rules. In the instances from ComplexPort, for instance, the average size of the rules is 1, 7.4, 29.22 and 88.2 for $\rho = \{10^{-4}, 10^{-5}, 10^{-6}, 0\}$, respectively. However, no

Table 6.7 – GP Parameters Setting

Algorithm	Parameter	Adopted values
GP	Population size	800
	Number of generations	100
	Tournament size	2
	Maximum tree depth	17
	Crossover probability	0.2
	Mutation probability	0.8
	Bloat control factor (ρ)	10^{-5}
	Deviation penalty (δ)	0, 0.01, 0.1, 1
GRH	Greed factor (α)	0.2
	Total number of iterations	500

difference in performance was found when using $\rho = \{10^{-5}, 10^{-6}, 0\}$. For $\rho = 10^{-4}$, the algorithm results in worse-performing rules. Similar results are found in the other two instance classes. Thus, we consider that there is a good balance between bloat control and rules performance with $\rho = 10^{-5}$. We perform extensive GP runs using four values of δ . Results are discussed in Section 6.7.1. The greediness of the GRH algorithm decreases as α increases. In 10 experiments using distinct values (0.05, 0.1, 0.15, 0.2, 0.5), the average solution quality is better with $\alpha = 0.2$. The algorithm presents no solution improvement after 400 iterations. Thus, we set its stopping criterion as completing 500 total iterations.

6.7. Experiments results

This section describes the performance comparison of the proposed methods in the validation sets described in Section 6.6.1. We start by explaining the rules obtained by GP. Then we present the results on the deterministic version of the problem. Finally, we explore the performance of seven strategies in responding to disruptions in the planned schedule. All methods were coded in Java and ran on Intel(R) Xeon(R) Processor E5 Family CPU with 2.8 GHz of speed using up to 48 GB of RAM. The GP algorithm uses the evolutionary computation library ECJ [Luke, 2017]. The Mixed Integer Programming and Constraint Programming models use commercial solvers, respectively, ILOG CPLEX and ILOG CP Optimizer (both in version 20.10).

6.7.1 Evolved rules

In the generation of the priority rules, 30 independent GP runs were performed for each instance class. The rules obtained are compared by running the simulator on the validation set (deterministic version). In the instances from real ports, the performance of the rules is quite homogeneous – in LargePort, the evolved rules' throughput (τ_a) varies from 82.7% to 83.5% and in ComplexPort from 67.1% to 67.9%. The experiments on the randomly generated ports resulted in only three distinct rules with performances of 76.9%, 76.4% and 74.6%.

Table 6.8 presents three evolved rules in each instance class and their average throughput. As in LargePort and ComplexPort the rules have low variance in the average performance, we selected three with distinct behaviour on individual instances. The majority of the rules is compact and may be easily interpreted. Two terminals are frequently used, F (finish time of operation) and W (the number of wagons). The use of these terminals prioritises operations that finish earlier and lots with a larger number of wagons. We also observe that the rules of the general instances are quite short and simple, possibly because the machines and routes of the fictitious ports are quite uniform. Thus, GP cannot find relevant parameters for prioritising lots there.

Table 6.8 – Priority rules evolved for the deterministic problem

Instance class	Expression	Throughput (τ_a)
LargePort	$F - (PT/LE) + (DCap+LE) \cdot RE \cdot CDEAv$	83.5%
	$F - W^2 \cdot RE \cdot CDE^2$	83.3%
	$F + RE - (DCap + PT) \cdot PT + REAv / (DCap - RE)$	82.7%
ComplexPort	$F \cdot CDEAv / W$	67.9%
	$LE / (PT - F) - RE$	67.8%
	$DCap + F \cdot CDEAv - W$	67.2%
General	$F - W$	76.9%
	F	76.4%
	F/W	74.6%

In the stochastic case, each simulation has a baseline schedule – the best-known deterministic solution found by the CP model or the algorithm described in Section 6.5.1. The objective is to design dispatching rules that maximise the throughput with minimum deviations from the baseline. Thus, the performance of each rule is measured by $\tau_a + dev_a \cdot \delta$, where dev_a is the average number of deviations per lot and δ is a parameter used to balance the value of τ_a and dev_a . The larger the value of δ , the highest the penalty for deviations from the baseline. We considered four values of δ : 0, 0.01, 0.1 and 1, and ran 30 independent executions of GP for each value, obtaining 120 rules for each instance class.

The charts of Figure 6.4 show the performance of the rules obtained for LargePort and ComplexPort, grouped by the value of δ used during training. When $\delta = 0$, not only the average deviation produced by the rules is much higher than in the other cases, but also the throughput is not noticeably higher (on the contrary). Therefore, using this parameter is important. There is also a large difference in the number of deviations produced in each port. The reason is the limited number of alternative routes for each lot in ComplexPort; several changes in car dumpers and sequences are necessary to effectively accommodate the schedule to disruptions. Moreover, in ComplexPort, penalising the deviation does not affect the average throughput obtained. Indeed, it seems that δ somehow guides the algorithm search to better rules, possibly because these rules tend to reproduce the priority of lots of the original schedule.

The selected dispatching rules are presented in Table 6.9. For each instance class, we took the rule that produced the best throughput and the one with minimum deviation from the baseline. The aim is to analyse the trade-off between both objectives. Thus, two rules are

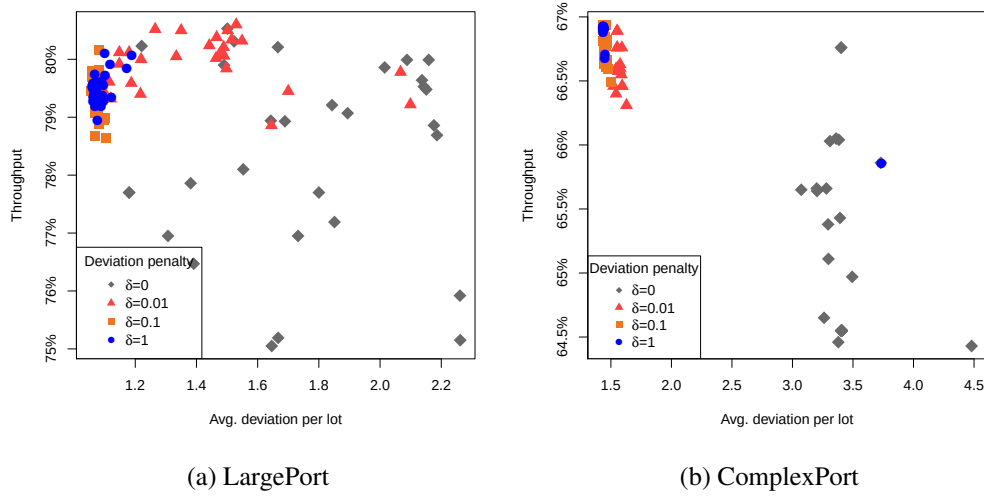


Figure 6.4 – Thought and number of deviations resulted from the evolved rules. Independent runs were performed using distinct penalties for deviations.

selected for LargePort and the General instances. In ComplexPort, a single rule dominates the others in both criteria. As expected, all rules present the terminal DEV, which measures the number of deviations per lot. Observe that in the rules DR_LP1, DR_CP and DR_1 (fewer number of deviations), the value of DEV has a large influence on the score. On the contrary, in rules DR_LP2 and DR_2, DEV is unlikely to determine the rule's behaviour, as it assumes values much smaller than F.

Table 6.9 – Dispatching rules generated for the stochastic problem

Instance class	Rule	Expression	Throughput (τ_a)	Deviations per lot (dev_a)
LargePort	DR_LP1	$1 + DCap + (DEV - RE)/CDEAv$	79.5%	1.06
LargePort	DR_LP2	$DEV + DCap/RE + CDE + F - RE$	80.6%	1.53
ComplexPort	DR_CP	$LE \cdot F / (DEV \cdot RE) + CDE - RE - DCap/LE + DEV \cdot F^2$	66.9%	1.43
General	DR_1	$F \cdot (DEV + LE)$	75.4%	0.93
General	DR_2	$(DEV + F)/W$	76.2%	1.59

6.7.2 Deterministic settings

This section reports the performance of the MIP model, the CP algorithm and the GP combined with GRH. The first two received a time limit of one hour for execution. Table 6.10 summarises the results, grouping instances by the parameter u . The average number of lots, which depends on u , is also reported. Column “GP + GRH” refers to the average of 20 runs of GRH for each instance. Columns “# Instances” present the number of instances for which each method provided feasible solutions. Due to the size of the instances from the real ports, the MIP model did not obtain any integer solution for them. For the same

reason, the CP model found solutions for only 93 out of the 240 instances from these ports. As not all cells refer to the same set of instances, the values in column “Throughput” cannot be directly compared in all cases.

Instances from LargePort result in an average throughput much higher than the ones from ComplexPort, 87.2% on average against 73.4%. This is possibly due to the large number of eligible routes per lot in LargePort. In all instance sets, the throughput tends to decrease as the number of lots increases. This is expected since the throughput is relative to the number of lots, and the number of machines does not vary with u . The MIP model found the optimal solution of 60 instances, but its average performance is far from the other two methods. In general, CP is outperformed by GP+GRH, which also provides a considerable shorter computational time. Analysing only the instances with solutions found by CP, GP+GRH is better in LargePort and the general instances, producing an increased throughput of 2.2% and 2.9% pp when compared to CP, respectively. In ComplexPort, both methods are roughly equivalent.

Table 6.10 – Performance of the proposed approaches on deterministic instances

Instance class	u	#Lots	MIP			CP			GP + GRH		
			#Instances	Throughput	Time (s)	#Instances	Throughput	Time (s)	#Instances	Throughput	Time (s)
LargePort	0.4	20.3	0	-	-	20	93.7%	2233	20	94.6%	3.8
	0.6	29.9	0	-	-	14	87.8%	3145	20	91.2%	6.1
	0.8	42.0	0	-	-	5	87.6%	3020	20	87.9%	11.6
	1	45.3	0	-	-	1	100.0%	346	20	85.3%	20.7
	1.2	57.2	0	-	-	0	-	-	20	81.7%	32.3
	1.4	67.8	0	-	-	0	-	-	20	75.4%	38.2
	Total/Average		0	-	-	40	91.0%	2603	120	86.0%	18.8
ComplexPort	0.4	29.3	0	-	-	19	82.0%	3418	20	81.4%	0.3
	0.6	38.4	0	-	-	19	82.8%	3318	20	81.5%	0.5
	0.8	50.7	0	-	-	13	75.0%	3610	20	75.0%	1.3
	1	67.8	0	-	-	2	61.4%	3622	20	67.5%	2.6
	1.2	74.4	0	-	-	0	-	-	20	63.3%	3.2
	1.4	88.6	0	-	-	0	-	-	20	55.4%	5.4
	Total/Average		0	-	-	53	79.8%	3373	120	70.7%	2.2
General	0.4	28.7	345	78.2%	3141	360	92.2%	3335	360	92.1%	0.3
	0.6	43.3	319	61.4%	3439	360	89.3%	3486	360	89.6%	0.7
	0.8	58.9	254	42.9%	3073	360	83.1%	3583	360	84.8%	1.7
	1	74.0	155	36.7%	3362	360	72.8%	3604	360	76.4%	3.1
	1.2	88.7	129	34.7%	3672	360	61.2%	3606	360	67.6%	5.0
	1.4	104.4	132	28.0%	3811	360	52.5%	3606	360	58.2%	6.3
	Total/Average		1334	47.0%	3416	2160	75.2%	3537	2160	78.1%	2.9

As CP does not scale well to larger instances, its relative performance is better in instances with a limited number of lots. On the other hand, GP+GRH does not perform well in instances with a large number of products – where it is difficult to avoid conflicts. This is the case of ComplexPort. In the class of general instances with $u \leq 0.6$, CP and GP are quite similar (90.9% against 90.8%, respectively). For $u > 0.6$, GP+GRH is much better than CP: the heuristic generate solutions with an average throughput of 71.8%, while CP

results in 67.4%.

We also benchmark the current solution adopted in Vale to the proposed approaches in 12 real instances, eight from LargePort and four from ComplexPort. Results show that GP+GRH provides competitive solutions for both ports compared to the company's algorithm, with an average throughput of 87.1% against 86.3%, respectively. Moreover, the computational time of the priority rules is, on average, four times shorter. This practical aspect is relevant to the company due to the need for rapid responses to unexpected events. Detailed results are presented in Section 6.C from Appendix.

6.7.3 Managing uncertainty

As discussed in Section 6.2, the scheduling literature presents very distinct approaches to deal with unexpected disruptions. The experiments of this section aim at providing guidance on the performance of predictive-reactive strategies under different uncertainty levels. In the application tackled in this work, changes on the original plan incur additional railway maneuvers and overhead. Managers seek a good compromise between maximising throughput and avoiding deviations from the planned schedule. Thus, we also analyse both objectives. Seven strategies are tested in 24-hours simulations. We classify them in three approaches:

- **Follow the baseline schedule:** The original plan is maintained as much as possible. As in Jain and Foley [2016], we evaluate rigid and flexible schemes. The rigid strategy respects the original car dumper and sequence of each lot. Thus, delays in the earlier operations highly affect the subsequent ones. The two flexible strategies may change the assigned car dumper and can anticipate lots. When deciding the next operation, the strategy n -flexible seek lots at most n positions distant in the original schedule. We simulate $n = 1$ and 2.
- **Periodic re-scheduling:** The system periodically re-schedules the remaining lots, and the last schedule is fixed until the next re-scheduling. For generating new partial schedules, the algorithm described in Section 6.5.1 is applied. Three re-scheduling intervals are considered: 12 hours, 6 hours and 3 hours.
- **Use dispatching rules:** We apply the rules evolved with GP to select the next operation once machines are available. When prioritising operations, the rules take into account the original schedule to avoid deviations.

In all experiments, the baseline schedule is the best solution found by any of the proposed methods (MIP, CP or GRH). Each point in the charts represents the average result of 20 replications of each of the 120 instances. This number of replications is sufficient for a 0.05 significance level (guaranteeing an error of at most 0.5 on both metrics).

The charts of Figure 6.5 show results obtained with instances from ComplexPort, and six levels of uncertainty. When $cv = 0$ (processing times are deterministic), no strategy is necessary, and the throughput corresponds to the baseline. The dispatching rule used is DR_CP

(c.f. Section 6.7.1). As expected, maintaining the baseline schedule in a rigid strategy produces the highest deterioration in the throughput but no deviations. At low uncertainty levels, it seems that following the baseline with some flexibility or re-scheduling are both good approaches. As uncertainty increases, the performance of the dispatching rule tends to be more competitive. Overall, re-scheduling with high frequency is the best strategy to maximise the throughput. However, we observed that re-scheduling produces a high number of deviations. This result is consistent with the analysis of Sabuncuoglu and Bayız [2000] for flexible manufacturing systems. At high uncertainty levels, the use of DR_TU reduces the number of changes while providing the second-best result in throughput.

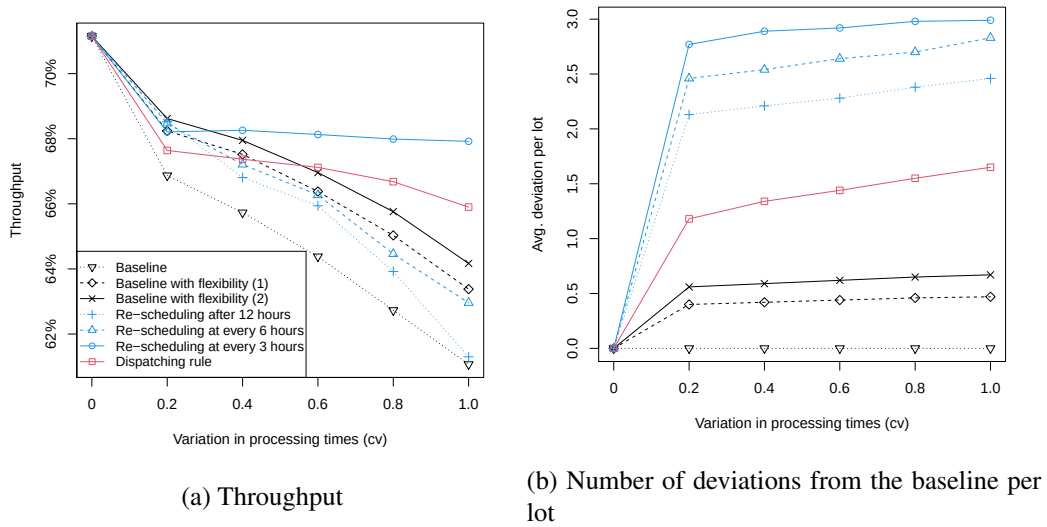


Figure 6.5 – Simulation of wagons unloading with uncertain processing times in instances from ComplexPort

In the charts of Figure 6.6b, the output of instances from LargePort is presented. Two dispatching rules are considered, DR_LP1 and DR_LP2, both shown in Section 6.7.1. In general, the curves are similar to those from the previous experiments, and the overall conclusion is the same. In this case, the re-scheduling approach is even more effective than the others. Surprisingly, in medium-low uncertainty levels re-scheduling at every six hours produces better throughput than following the baseline with flexibility. We suspect that this is due to the several eligible routes of each lot. Even maintaining the original car dumpers and sequence for six hours, one can find alternative routes without overlap. This is not possible in ComplexPort as lots have a limited number of eligible routes there.

We also evaluate the effect of the trains' arrivals on the performance of the strategies. The frequency of train arrivals depends on u , which is an upper bound for car dumpers' utilisation. Instances with low values of u have fewer trains and lots. In Figure 6.7, we present results of three groups of instances, with low, medium and high machines utilisation (all from ComplexPort). In those simulation experiments, the mean machines utilisation

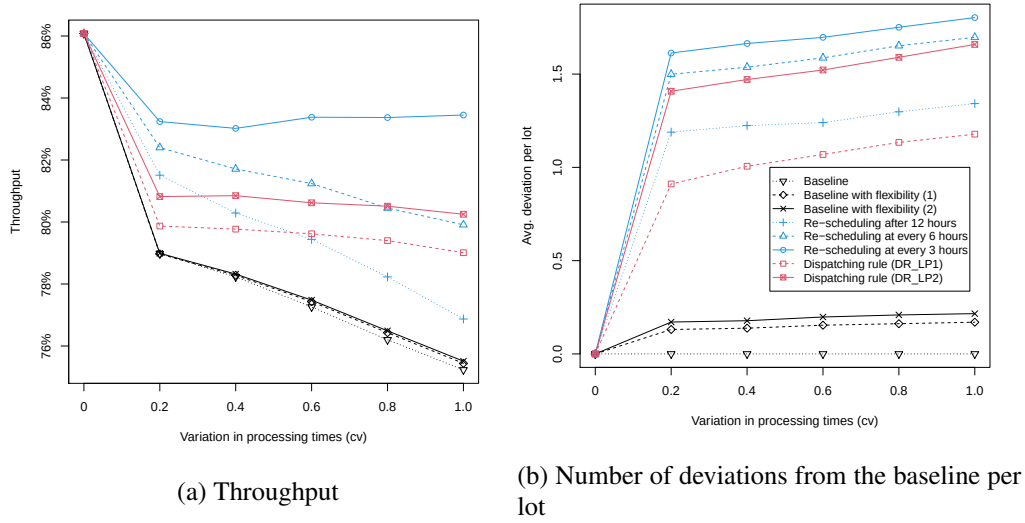


Figure 6.6 – Simulation of wagons unloading with uncertain processing times in instances from LargePort

is 41%, 49% and 59%, respectively for u equals to 0.6, 0.8 and 1.2. Observe that with a smaller number of trains (Figure 6.7a), the difference in the performance of the strategies is not as clear, although re-scheduling at every 3 hours seems the best for maximising the throughput. As the number of trains' arrivals increases, the distinction in the output of the strategies also increases. In congested settings, the performance of the dispatching rule is similar to re-scheduling every three hours. These results suggest that managers should be aware of the system behaviour and adapt strategies accordingly.

6.8. Conclusions and future work

In this paper, we studied product reception in bulk ports. In such process, wagons must be unloaded to stockpiles in a complex operation that requires the synchronisation of the machines involved. Moreover, there is uncertainty in processing times, as unexpected events frequently occur. Our study uses real data from a leading mining company and randomly generated instances to access the performance of several solution approaches. The problem is formalised as a variant of the Multimode Multiprocessor Task Scheduling.

First, we designed two models for the deterministic version of the problem, a MIP and a CP. GP is also employed on the design of priority rules, which are applied by a greedy-randomised procedure, GRH. In the randomly generated instances, the rules, the CP and the MIP provide an average throughput of 78.1%, 75.2% and 47.0%, respectively. The MIP cannot find any integer solution in the real scenarios. These instances are also challenging for the CP algorithm, which finds solutions for only 93 out of 240 instances. CP tends to

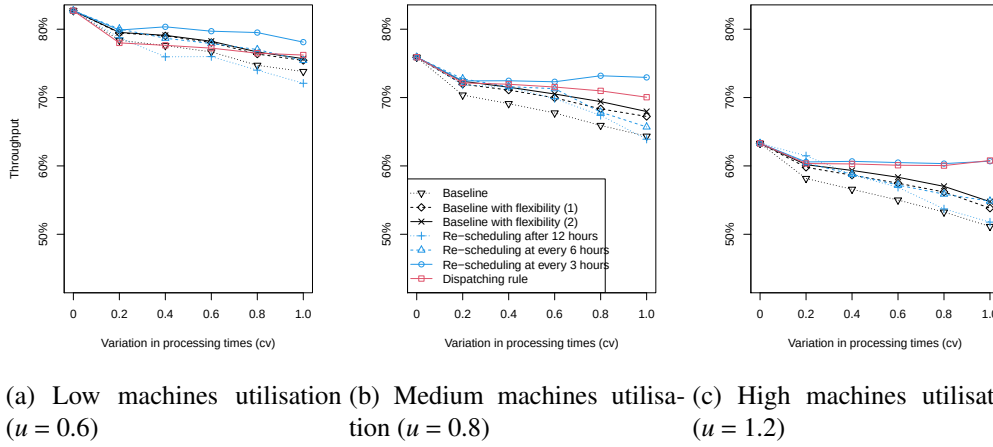


Figure 6.7 – Simulation of distinct machines utilisation and the throughput obtained with each strategy

find better schedules in instances with few train arrivals, and the rules perform better in the opposite case. In comparison to the current solution used by the mining company, the priority rules find similar solutions using only a quarter of the computational time. The overall conclusion is that, in general, GP+GRH outperforms the other methods. Moreover, we highlight the ease of implementing the priority rules, which have better prospects of being wide-adopted in practice than more sophisticated approaches.

Second, we also use GP to evolve dispatching rules for the stochastic version of the problem. The rules were designed with two objectives: maximising the operation's throughput and minimising deviations from a baseline schedule. GP is effective in finding rules that produce fewer deviations from the baseline. Third, we use simulation to evaluate several strategies for managing uncertainty. Three approaches are considered: maintaining the baseline schedule (in rigid and flexible schemes), periodic re-scheduling and the dispatching rules. We concluded that frequent re-scheduling provides better throughput than the other strategies. However, as uncertainty increase the rules generate competitive solutions. This effect is more prominent in settings with a high number of train arrivals. On the other hand, re-scheduling incurs several deviations and the evolved rules present the best compromise between both objectives.

As GRH is a fast constructive algorithm, there is room for improving the solution quality by developing a local search. Future work should also be employed on other predictive-reactive strategies, such as the pairwise exchange heuristic proposed by Jain and Foley [2016]. Moreover, delays in the expected train arrivals influence the performance of the schedule. One should incorporate this stochastic aspect into the problem. Finally, alternative models for counting the number of schedule deviations may be also be considered.

Bibliography

- Aytug, H., Lawley, M.A., McKay, K., Mohan, S., Uzsoy, R., 2005. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research* 161, 86 – 110. doi:<https://doi.org/10.1016/j.ejor.2003.08.027>. iEPM: Focus on Scheduling.
- Baptiste, P., Le Pape, C., Nuijten, W., 2001. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. volume 39. Springer US. doi:[10.1007/978-1-4615-1479-4](https://doi.org/10.1007/978-1-4615-1479-4).
- Barros, V.H., Costa, T.S., Oliveira, A.C., Lorena, L.A., 2011. Model and heuristic for berth allocation in tidal bulk ports with stock level constraints. *Computers & Industrial Engineering* 60, 606–613. doi:<https://doi.org/10.1016/j.cie.2010.12.018>.
- Battaïa, O., Delorme, X., Dolgui, A., Hagemann, J., Horlemann, A., Kovalev, S., Malyutin, S., 2015. Workforce minimization for a mixed-model assembly line in the automotive industry. *International Journal of Production Economics* 170, 489 – 500. doi:[10.1016/j.ijpe.2015.05.038](https://doi.org/10.1016/j.ijpe.2015.05.038). current Research Issues in Production Economics.
- Bianco, L., Dell’Olmo, P., Giordani, S., Speranza, M.G., 1999. Minimizing makespan in a multimode multiprocessor shop scheduling problem. *Naval Research Logistics* 46, 893 – 911. doi:[10.1002/\(SICI\)1520-6750\(199912\)46:8<893::AID-NAV2>3.0.CO;2-7](https://doi.org/10.1002/(SICI)1520-6750(199912)46:8<893::AID-NAV2>3.0.CO;2-7).
- Bilgen, B., Ozkarahan, I., 2007. A mixed-integer linear programming model for bulk grain blending and shipping. *International Journal of Production Economics* 107, 555–571. doi:<https://doi.org/10.1016/j.ijpe.2006.11.008>. operations Management in China.
- Błażewicz, J., Dell’Olmo, P., Drozdowski, M., Speranza, M., 1992. Scheduling multiprocessor tasks on three dedicated processors. *Information Processing Letters* 41, 275 – 280. doi:[10.1016/0020-0190\(92\)90172-R](https://doi.org/10.1016/0020-0190(92)90172-R).
- Burdett, R.L., Corry, P., Eustace, C., Smith, S., 2020. A flexible job shop scheduling approach with operators for coal export terminals – a mature approach. *Computers & Operations Research* 115, 104834. doi:<https://doi.org/10.1016/j.cor.2019.104834>.
- Caramia, M., Giordani, S., 2009. A new approach for scheduling independent tasks with multiple modes. *Journal of Heuristics* 15, 313–329. doi:[10.1007/s10732-007-9062-y](https://doi.org/10.1007/s10732-007-9062-y).
- Caramia, M., Giordani, S., 2010. A fast metaheuristic for scheduling independent tasks with multiple modes. *Computers & Industrial Engineering* 58, 64–69. doi:[10.1016/j.cie.2009.08.004](https://doi.org/10.1016/j.cie.2009.08.004).

- Chen, B., Matis, T.I., 2013. A flexible dispatching rule for minimizing tardiness in job shop scheduling. *International Journal of Production Economics* 141, 360 – 365. doi:<https://doi.org/10.1016/j.ijpe.2012.08.019>. meta-heuristics for manufacturing scheduling and logistics problems.
- Comtois, C., Slack, B., 2016. Dynamic determinants in global iron ore supply chain. CIRRELT, Montréal.
- Drozdowski, M., 2009. Parallel Tasks. Springer London, London. chapter 5. pp. 87–208. doi:[10.1007/978-1-84882-310-5_5](https://doi.org/10.1007/978-1-84882-310-5_5).
- Ferreira, C., Figueira, G., Amorim, P., 2021a. Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning. URL: <https://arxiv.org/abs/2109.03323>, arXiv:2109.03323.
- Ferreira, C., Figueira, G., Amorim, P., 2021b. Scheduling human-robot teams in collaborative working cells. *International Journal of Production Economics* 235, 108094. doi:<https://doi.org/10.1016/j.ijpe.2021.108094>.
- Gökgür, B., Hnich, B., Özpeynirci, S., 2018. Parallel machine scheduling with tool loading: a constraint programming approach. *International Journal of Production Research* 56, 5541–5557. doi:[10.1080/00207543.2017.1421781](https://doi.org/10.1080/00207543.2017.1421781).
- Haupt, R., 1989. A survey of priority rule-based scheduling. *Operations Research - Spektrum* 11, 3–16. doi:[10.1007/BF01721162](https://doi.org/10.1007/BF01721162).
- Jain, S., Foley, W., 2016. Dispatching strategies for managing uncertainties in automated manufacturing systems. *European Journal of Operational Research* 248, 328 – 341. doi:<https://doi.org/10.1016/j.ejor.2015.06.060>.
- Koza, J.R., 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing* 4, 87–112.
- Lawrence, S.R., Sewell, E.C., 1997. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. *Journal of Operations Management* 15, 71 – 82. doi:[https://doi.org/10.1016/S0272-6963\(96\)00090-3](https://doi.org/10.1016/S0272-6963(96)00090-3).
- Liu, L., Zhou, H., 2013. On the identical parallel-machine rescheduling with job rework disruption. *Computers & Industrial Engineering* 66, 186–198. URL: <https://www.sciencedirect.com/science/article/pii/S0360835213000697>, doi:<https://doi.org/10.1016/j.cie.2013.02.018>.
- Lou, P., Liu, Q., Zhou, Z., Wang, H., Sun, S.X., 2012. Multi-agent-based proactive–reactive scheduling for a job shop. *The International Journal of Advanced Manufacturing Technology* 59, 311–324. doi:<https://doi.org/10.1007/s00170-011-3482-4>.
- Luke, S., 2017. Ecj then and now, in: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Association for Computing Machinery, New York, NY, USA. p. 1223–1230. doi:[10.1145/3067695.3082467](https://doi.org/10.1145/3067695.3082467).

- Lunardi, W.T., Birgin, E.G., Laborie, P., Ronconi, D.P., Voos, H., 2020. Mixed integer linear programming and constraint programming models for the on-line printing shop scheduling problem. *Computers & Operations Research* 123, 105020. URL: <https://www.sciencedirect.com/science/article/pii/S0305054820301374>, doi:<https://doi.org/10.1016/j.cor.2020.105020>.
- Menezes, G.C., Mateus, G.R., Ravetti, M.G., 2017. A branch and price algorithm to solve the integrated production planning and scheduling in bulk ports. *European Journal of Operational Research* 258, 926–937. doi:<https://doi.org/10.1016/j.ejor.2016.08.073>.
- Merk, O., Dang, T., 2012. Efficiency of World Ports in Container and Bulk Cargo (oil, coal, ores and grain). Technical Report 09. OECD Publishing. doi:<https://doi.org/10.1787/5k92vgw39zs2-en>.
- Neagoe, M., Taskhiri, M.S., Nguyen, H.O., Hvolby, H.H., Turner1, P., 2018. Exploring congestion impact beyond the bulk cargo terminal gate, in: *Proceedings of the Hamburg International Conference of Logistics (HICL)*, pp. 63–82.
- Nguyen, S., Mei, Y., Zhang, M., 2017. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems* 3, 41–66.
- Ouelhadj, D., Petrovic, S., 2008. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling* 12, 417. doi:[10.1007/s10951-008-0090-8](https://doi.org/10.1007/s10951-008-0090-8).
- Pant, R., Barker, K., Ramirez-Marquez, J.E., Rocco, C.M., 2014. Stochastic measures of resilience and their application to container terminals. *Computers & Industrial Engineering* 70, 183–194. URL: <https://www.sciencedirect.com/science/article/pii/S0360835214000333>, doi:<https://doi.org/10.1016/j.cie.2014.01.017>.
- Ramasesh, R., 1990. Dynamic job shop scheduling: A survey of simulation research. *Omega* 18, 43 – 57. doi:[https://doi.org/10.1016/0305-0483\(90\)90017-4](https://doi.org/10.1016/0305-0483(90)90017-4).
- Robenek, T., Umang, N., Bierlaire, M., Ropke, S., 2014. A branch-and-price algorithm to solve the integrated berth allocation and yard assignment problem in bulk ports. *European Journal of Operational Research* 235, 399–411. URL: <https://www.sciencedirect.com/science/article/pii/S0377221713006802>, doi:<https://doi.org/10.1016/j.ejor.2013.08.015>. maritime Logistics.
- Rocha de Paula, M., Boland, N., Ernst, A.T., Mendes, A., Savelsbergh, M., 2019. Throughput optimisation in a coal export system with multiple terminals and shared resources. *Computers & Industrial Engineering* 134, 37–51. doi:<https://doi.org/10.1016/j.cie.2019.05.021>.
- Sabuncuoglu, I., Bayız, M., 2000. Analysis of reactive scheduling problems in a job shop environment. *European Journal of Operational Research* 126, 567 – 586. doi:[https://doi.org/10.1016/S0377-2217\(99\)00311-2](https://doi.org/10.1016/S0377-2217(99)00311-2).

- Sabuncuoglu, I., Kizilisik, O.B., 2003. Reactive scheduling in a dynamic and stochastic fms environment. *International Journal of Production Research* 41, 4211–4231. doi:[10.1080/0020754031000149202](https://doi.org/10.1080/0020754031000149202).
- Su, H., Pinedo, M., Wan, G., 2017. Parallel machine scheduling with eligibility constraints: A composite dispatching rule to minimize total weighted tardiness. *Naval Research Logistics (NRL)* 64, 249–267. doi:[10.1002/nav.21744](https://doi.org/10.1002/nav.21744).
- Switalski, P., Sredynski, F., 2015. Scheduling parallel batch jobs in grids with evolutionary metaheuristics. *Journal of Scheduling* 18, 345–357. doi:[10.1007/s10951-014-0382-0](https://doi.org/10.1007/s10951-014-0382-0).
- Tsang, E., 1993. *Foundations of Constraint Satisfaction*. 2 ed., Academic Press Limited. doi:[10.1016/C2013-0-07627-X](https://doi.org/10.1016/C2013-0-07627-X).
- Türkoğullar, Y.B., Taşkın, Z.C., Aras, N., Altınel, İ.K., 2014. Optimal berth allocation and time-invariant quay crane assignment in container terminals. *European Journal of Operational Research* 235, 88 – 101. doi:[10.1016/j.ejor.2013.10.015](https://doi.org/10.1016/j.ejor.2013.10.015).
- Umang, N., Bierlaire, M., Vacca, I., 2013. Exact and heuristic methods to solve the berth allocation problem in bulk ports. *Transportation Research Part E: Logistics and Transportation Review* 54, 14–31. doi:<https://doi.org/10.1016/j.tre.2013.03.003>.
- UNCTAD, 2020. *Review of Maritime Transport*. Technical Report. United Nations Conference on Trade and Development. URL: <https://unctad.org/publications>.
- Varadarajan, A., Sarin, S.C., 2006. A survey of dispatching rules for operational control in wafer fabrication. *IFAC Proceedings Volumes* 39, 715 – 726. doi:<https://doi.org/10.3182/20060517-3-FR-2903.00363>. 12th IFAC Symposium on Information Control Problems in Manufacturing.
- Xiong, H., Fan, H., Jiang, G., Li, G., 2017. A simulation-based study of dispatching rules in a dynamic job shop scheduling problem with batch release and extended technical precedence constraints. *European Journal of Operational Research* 257, 13 – 24. doi:<https://doi.org/10.1016/j.ejor.2016.07.030>.
- Xiong, J., ning Xing, L., wu Chen, Y., 2013. Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns. *International Journal of Production Economics* 141, 112–126. URL: <https://www.sciencedirect.com/science/article/pii/S0925527312001739>, doi:<https://doi.org/10.1016/j.ijpe.2012.04.015>. meta-heuristics for manufacturing scheduling and logistics problems.
- Yin, Y., Cheng, T., Wang, D.J., 2016. Rescheduling on identical parallel machines with machine disruptions to minimize total completion time. *European Journal of Operational Research* 252, 737–749. URL: <https://www.sciencedirect.com/science/article/pii/S0377221716000904>, doi:<https://doi.org/10.1016/j.ejor.2016.01.045>.

Åstrand, M., Johansson, M., Zanarini, A., 2020. Underground mine scheduling of mobile machines using constraint programming and large neighborhood search. *Computers & Operations Research* 123, 105036. URL: <https://www.sciencedirect.com/science/article/pii/S0305054820301532>, doi:<https://doi.org/10.1016/j.cor.2020.105036>.

Appendix 6.A Genetic Programming algorithm

The pseudo-code of the proposed GP is provided in Algorithm 9. The algorithm receives as input the training instance set I . The output is the best individual found a^* , from which the best dispatching rule r^* is obtained. In line 1, the initial population is created using the method Ramped Half-and-Half [Koza, 1994]. In this method, trees are randomly created by combining two methods, "full" and "grow", with equal probability. In trees created by the "full" method, the path length from the root to every terminal is equal to a pre-specified maximum depth. The "grow" method instead generates trees with variable shapes and sizes. In lines 2 to 11, the algorithm performs the generational evolution: at each generation g , the population P_g is evaluated and updated. The fitness of each individual a is calculated by simulating the corresponding rule r on the training set (line 6). For each instance $i \in I$, the simulator returns the average throughput obtained (τ_{ai}). The fitness of a combines both metric and the tree size $|a|$ (line 8). As the weight of the penalty size depends on g , in line 9 the algorithm recalculates the fitness of the best individual found so far (a^*).

Input: Training set I and solutions S

Result: Dispatching rule r^*

```

1 Initialise population  $P_g$ ;
2 for  $g = 1$  to  $max\_g$  do
3   foreach individual  $a \in P_g$  do
4      $r \leftarrow \text{decode } a$ ;
5     foreach instance  $i \in I$  do
6        $\tau_{ai} \leftarrow \text{simulate}(r, i, S_i)$ ;
7     end
8      $\tau_a = 1/|I| \cdot \sum_{i \in I} \{\tau_{ai}\}$ ;
9      $\text{fitness}(a, g) = \tau_a + dev_a \cdot \delta + |a| \cdot g \cdot \rho$ ;
10  end
11  Calculate fitness of  $a^*$  in  $g$ ;
12   $a^* \leftarrow \arg \min_{a \in \{P_g \cup a^*\}} \text{fitness}(a, g)$ ;
13   $P_{g+1} \leftarrow \text{Crossover and mutation in } P_g$ ;
14 end

```

Algorithm 9: Genetic Programming algorithm

In line 11, the next generation's population is created from the current population P_g and the genetic operators. There are two breeding pipelines, crossover and mutation, and each

one is applied according to a probability. The crossover pipeline selects two individuals as parents to produce two offspring. This operation selects a node at random in each tree and swaps the two sub-trees rooted by those nodes. Mutation replaces a sub-tree with a randomly generated one, generating a new individual. Both pipelines use tournament for parent selection, returning the best of ts individuals selected at random (ts is said tournament size). The breeding process picks certain kinds of nodes with different probabilities. For example, one can state to pick non-terminals 90% of the time and terminals 10% of the time. The algorithm runs until it reaches a maximum number of generations max_g and returns the best individual found a^* .

Appendix 6.B Fictitious ports

Table 6.11 presents the characteristics of all instances based on fictitious ports. Three distinct settings are considered for training GP and six for the validation of the methods. For each combination of $\langle \#car\ dumpers - \#stackers - \#conveyors - \#routes \rangle$ and number of products, we generated 10 instances for training and 20 for validation. Stochastic instances also consider the cv parameter, which represents the coefficient of variation of processing times.

Table 6.11 – Parameters of the randomly generated ports

	Training set	Validation set
$\langle \#car\ dumpers - \#stackers - \#conveyors - \#routes \rangle$	$\langle 3 - 3 - 3 - 7 \rangle$, $\langle 5 - 5 - 5 - 31 \rangle$, $\langle 7 - 7 - 7 - 54 \rangle$	$\langle 4 - 4 - 4 - 16 \rangle$, $\langle 4 - 6 - 6 - 36 \rangle$, $\langle 5 - 5 - 5 - 31 \rangle$, $\langle 5 - 7 - 7 - 61 \rangle$, $\langle 6 - 6 - 6 - 54 \rangle$, $\langle 6 - 8 - 8 - 96 \rangle$
Number of products	4, 16	2, 10, 20
Machines utilisation (u)	0.6, 0.8, 1.0, 1.2	0.4, 0.6, 0.8, 1.0, 1.2, 1.4
Instances per setting	10	20
# Deterministic instances	240	2160
Variation in processing times (cv)	0.4, 0.8	0.2, 0.4, 0.6, 0.8, 1.0
# Stochastic instances	480	10800

Appendix 6.C Real instances

Table 6.12 presents a performance comparison between the proposed approaches and the current solution adopted by Vale. The algorithm used by the company constructs an initial solution and improves it by using a large 3-opt neighbourhood search. We benchmark the methods using 12 real instances of two ports: LargePort and ComplexPort. Despite the limited number of instances, GP seems competitive to the current algorithm used in the company. The priority rules find similar solutions in a much shorter computational time.

Table 6.12 – Performance comparison on the real instances

Port	Horizon (<i>hz</i>)	#Lots	#Wagons	Current solution		CP		GP+GCH	
				Throughput	Time (s)	Throughput	Time (s)	Throughput	Time (s)
LargePort	2382	83	7798	100.0%	198	-	-	100.0%	46
	1878	89	8446	90.4%	318	-	-	89.7%	67
	1691	85	8500	81.3%	320	-	-	88.8%	97
	1045	65	6046	64.8%	192	-	-	68.2%	24
	1045	57	5338	72.2%	139	-	-	68.0%	73
	542	35	2846	100.0%	94	-	-	100.0%	7
	1598	81	7910	79.8%	305	-	-	89.5%	39
	942	60	5492	64.5%	198	-	-	56.9%	22
Total/Average				81.6%	221	-	-	82.6%	47
ComplexPort	2734	35	2448	100.0%	17	96.6%	3646	100.0%	0
	2734	36	2468	100.0%	13	100.0%	2059	100.0%	1
	1294	32	2190	83.0%	16	84.8%	3625	84.2%	0
	885	28	1784	100.0%	10	100.0%	779	100.0%	1
Total/Average				95.8%	14	95.4%	2527	96.1%	1

Conclusions and future work

The recent advancements in technologies and processes from i4.0 bring more efficiency to operations but also impose new challenges. To grasp the opportunities of i4.0, managers from distinct sectors need to re-design and optimise their processes. In manufacturing, collaborative robots improve performance and flexibility and raise questions about when to employ collaborative tasks and how to schedule HRTs. In logistics, data analytics and artificial intelligence have increased the competitiveness of the sector. Iron ore ports, in particular, have experienced a high increase in demand and need to improve their service levels accordingly. Several sources of uncertainty are present in cobots applications and iron ore ports, which complicate operations management.

This thesis proposes optimisation methods to deal with those questions. Scheduling problems are modelled as the Multimode Multiprocessor Task Scheduling, in which tasks may be executed in different modes. Mixed Integer Programming models, Constraint Programming algorithms and heuristics are proposed to solve the different problems and compared with existing methods. Dynamic scheduling is also tackled with the development of dispatching rules for sequencing operations in real-time. In addition to the challenge of generating well-performing rules, the proposed methods also target interpretability. With that purpose, Genetic programming is applied in a guided empirical approach, combining the computational power of the algorithm and expertise on the problem.

The next sections describe the overall contributions and publications from this thesis. In the following, we answer to the research questions and provide some directions for future work.

7.1. Contributions

The first contribution of this thesis is presented in Chapter 3 and comprises the design of interpretable dispatching rules using GP. Initially, we validated the approach in job shops with static arrivals and setup times. Then we expanded the study by incorporating dynamic arrivals and due dates, which allowed evaluating the methodology in one of the most challenging performance criterion, the mean tardiness. We perform a guided and iterative learning process, combining the computational search power of GP with human intelligence – the algorithm search space is reduced with insights on the problem and results previously obtained. Following this iterative approach, we designed simple and effective rules for minimising the mean tardiness. As a result, the evolved rules outperform existing ones in various load conditions and stochastic operation processing times. From the

aforementioned work, the following research papers have resulted:

- C. Ferreira, G. Figueira, P. Amorim. Optimising dispatching rules for stochastic job shop scheduling. *Hybrid Intelligent Systems*, 2020. https://doi.org/10.1007/978-3-030-14347-3_31
- C. Ferreira, G. Figueira, P. Amorim. Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning. *Submitted to Omega*, 2021.

The second contribution of this thesis comprises the study of HRTs scheduling in collaborative working cells, which is modelled as a particular case of the MMTSP. A new MIP formulation for the problem is devised, and two algorithms are proposed, a Constraint Programming model and a Genetic Algorithm. We also designed a new lower bound for benchmarking the methods. The experiments were conducted on a large set of instances representing a variety of HRTs settings and also general instances of the MMTSP. The proposed methods outperform by far algorithms from the literature. On the HRTs instances, we found optimal solutions for 108 out of 240 instances and tight gaps to lower bounds. Moreover, we derived some managerial insights on the improvement obtained with collaborative tasks. This work resulted in the following paper:

- C. Ferreira, G. Figueira, P. Amorim. Scheduling Human-Robot Teams in collaborative working cells. *IJPE*, 2021. <https://doi.org/10.1016/j.ijpe.2021.108094>

In the third contribution, the dynamic scheduling of HRTs is tackled. The methodology developed for job shops is extended to evolve specific rules for collaborative working cells. In this case, two decisions are taken with the same rule, assignment and sequencing. The work assumes that operations processing times are uncertain when tackled by humans or an HRT. We study settings with static and dynamic arrival of jobs and optimise two common criteria for each case: the makespan, the mean completion time, the mean tardiness and the mean flowtime. In the seek for interpretable rules, we employ Strongly-Typed GP and bloat control. The evolved rules produce better results than the best existing ones, with significant gains in each criterion. We also perform a comparative analysis on the makespan deterioration under uncertain processing times and compare the performance of the evolved rules against good quality deterministic solutions. From the simulations experiments, we conclude that the dispatching rules should be applied in settings where processing times have a coefficient of variation above 0.4. In the opposite case, the best solution is to keep the original machine allocation and adjust the sequence of operations. From this work, the following research paper has resulted:

- C. Ferreira, G. Figueira, P. Amorim. Managing Human-Robot Teams in real-time with dispatching rules. *Submitted to IJPE*, 2021.

Finally, we addressed a scheduling problem found in bulk ports. As the product usually arrives by train, its dumped from the wagons to the stockyard. These income operations

depend on the synchronisation of a car dumper, some conveyor belts and a stacker. In large ports, they are critical to maximising the overall throughput but comprise a challenging problem due to the many machines involved. Moreover, several interruptions occur and lead to uncertain processing times. We have modelled this problem as a variant of the MMTSP and addressed both the deterministic and the stochastic version of the problem. Three approaches are proposed for the deterministic version, a MIP model, a CP algorithm and a constructive heuristic. Here, GP is employed for two distinct purposes: (1) evolving rules that are used by the constructive heuristic, and (2) evolving dispatching rules for the stochastic case. The effectiveness of the proposed methods is analysed using data from two real ports. We also validate the models and algorithms in generated instances to represent a wide spectrum of ports. Moreover, we use simulation to evaluate the use of distinct strategies under uncertain processing times. The results provide insights on the best approaches for maintaining a good throughput and minimising the number of deviations from a baseline schedule. The fourth contribution of this thesis resulted in the following paper:

- C. Ferreira, G. Figueira, P. Amorim, A. Pigatti. Scheduling wagons unloading in bulk ports with uncertain processing times. *Submitted to IJPR*, 2021.

7.2. Answers to the research questions

In this section, we provide the answers to each research question formulated in this thesis.

Research Question Q1

How to design effective and interpretable dispatching rules?

To answer this question, we tackled the well-studied Dynamic Job Shop Scheduling Problem. As performance metric, we have chosen the mean tardiness, which is challenging because the rules' behaviour is highly sensitive to the settings conditions. Our initial hypothesis is that one can improve the algorithmic search by following a guided-learning approach. We confirmed that hypothesis by incorporating insights from domain expertise. For instance, instead of feeding the algorithms with several meaningless parameters, one should provide GP with a small set of carefully selected terminals. We compared the relative performance of the evolved rules against existing ones (the smaller, the better). We obtained an average relative performance of 2.24 on the validation set by reproducing previous studies and using a long list of terminals. By employing a first selection of parameters from the best existing rules, this relative performance is reduced to 1.64. We extended the analysis by incorporating specific terminals that do not consider due date information if jobs are late. A preliminary structure to the rules is also provided, enforcing different behaviour depending on the current slack of jobs. After these modifications, we achieved an overall performance of 0.89. These results were also made possible by carefully selecting representative training instances, which led to rules that are able to generalise their behaviour in unseen scenarios. The evolved rules present a generally good behaviour in

a variety of load conditions and improve the mean tardiness by 19% on the test set, on average. Moreover, a bloat control strategy is also employed, resulting in compact and interpretable rules.

Research Question Q2

When and how to employ collaborative tasks?

The gains obtained with collaborative tasks depend on processing times, precedence constraints and other environmental characteristics. From a performance perspective, the answer to Q2 may be provided by solving the MMTSP, where tasks may be executed in different modes. In the HRTs application, there is a single execution mode representing each element in the team. Duo-processor modes represent the collaborative tasks, which require simultaneous execution by a human and a robot. We solved the MMTSP minimising the cycle time of HRT working cells in a variety of settings, with and without collaborative tasks. Our methods found the optimal solution of 48% of the proposed instances and resulted in an average gap of 3.8% to the lowerbound on the instances with unknown optimal solutions. These results demonstrate the effectiveness of the proposed approach. Moreover, we performed specific experiments to derive some insights about the collaborative tasks. By comparing scenarios with and without the possibility of collaboration, we observe a significant reduction to the cycle time – the larger number of collaborative tasks, the larger improvement in makespan. The reduction is more prominent in settings with several precedence constraints and low robot eligibility.

Research Question Q3

How to sequence tasks to HRTs in real-time?

Some methods have been proposed in the robotics literature to answer this question. However, they generally employ a simple exhaustive search or lack substantial experimental validation. As in dynamic shop-floors, rapid responses must be provided, we adopt a reactive approach using dispatching rules. In this thesis, we evolved specific rules for HRTs and showed that they significantly outperform the best existing rules in four performance criteria. Then we analyse how the makespan of static schedules deteriorate under uncertainty, using near-optimal solutions of the deterministic problem. Simulation shows that under low uncertainty settings, one should strictly follow the original assignment of tasks. In the same conditions, operations should be prioritised according to their sequence in the baseline schedule, allowing bypass of operations when necessary. However, this strategy incurs high makespan deterioration in high uncertainty levels, up to 7% in cells with only one human and 12% in cells with three humans. In such circumstances, it is best to employ the evolved dispatching rules, deciding the next operation to be executed by each element in real-time. This strategy provides, in the worst-case, 6% of increase to the original makespan. Combining the original assignment of tasks with a dispatching rule is also promising in the presence of collaborative tasks.

Research Question Q4

How to effectively sequence wagons for unloading in dry bulk ports?

To answer this question, we tackled the problem of scheduling unloading operations to maximise the overall throughput. GP is employed to generate priority rules used by a greedy-randomised heuristic on the deterministic version of the problem. The experiments showed that this heuristic empowered with GP outperforms the other methods, producing solutions with an average throughput 2.5p.p. better. In the stochastic context, we simulated several strategies using data from two distinct real iron ore ports. As dynamic scheduling strategies, we considered following the baseline schedule in rigid or flexible schemes; re-scheduling with low and high frequencies; and dispatching rules with GP. Results show that re-scheduling every three hours is better to optimise the throughput but provide several undesirable modifications in the baseline schedule. The evolved rules are competitive in high uncertainty levels and incur fewer changes because they consider the baseline for prioritising operations. In low uncertainty levels, re-scheduling every 12 hours also produces fewer modifications and is a good strategy in settings with a few products. Finally, we also observe that the performance of the dispatching rules is similar to re-scheduling every three hours if machines utilisation is higher than 60%.

7.3. Future work

The research conducted in this thesis allows answering all the research questions established. However, several opportunities and literature gaps still remain and deserve further attention by the research community. We highlight here some of the most relevant, as specific directions are found in the end of Chapters 3, 4, 5 and 6. In general, the methods developed should be extended and validated in other problems and industrial sectors. As shown in Chapter 5, the behaviour of the system is completely different in settings where the only decision refers to the sequence of operations (as in job shops) and where the assignment is also relevant (as in HRTs scheduling). Thus, one should study the Flexible Job Shop Scheduling Problem using the guided learning approach adopted in Chapter 3. Beyond scheduling, other problems such as inventory management, vehicle routing and online fulfilment could also be taken. In Chapter 6, we have shown that GP combined with a greed-randomised heuristic is a promising approach. This combination should be applied to other challenging problems, where an effective sequencing of operations is critical. To our knowledge, GP was employed in a real scheduling problem in this thesis for the first time. Thus, future work should employ this methodology in other real problems. Finally, the methods employed in Chapters 4 and 5 should be validated in computational experiments using real-world data or in real verification scenarios.

Dispatching rules for job shops with static arrivals

Optimizing Dispatching Rules for Stochastic Job Shop Scheduling

Cristiane Maria Santos Ferreira^{*†} · Gonalo Figueira^{*†} · Pedro Amorim^{*†}

Published in the book *Hybrid Intelligent Systems*, 2020

Abstract Manufacturing environments commonly present uncertainties and unexpected schedule disruptions. The literature has shown that in these environments simple and fast dynamic dispatching rules are efficient sequencing methods. However, most of the works in the automated designing of these rules have considered deterministic processing times. This work aims to design dispatching rules for problem settings similar to the ones found in real environments such as uncertain processing times and sequence-dependent setup times. We use Genetic Programming to generate efficient rules for stochastic job shops with setup times. We show that the generated rules outperform benchmark dispatching rules, specially in settings with high setup time levels.

Keywords Dynamic Job Shop Scheduling · Stochastic Scheduling · Genetic Programming · Dispatching Rules

A.1. Introduction

Dynamic scheduling refers to environments where the schedule of jobs is subject to unexpected conditions. In dynamic job shops, the sequence of operations must be continuously re-optimized to adapt to the unexpected deviations [Dominic et al. \[2004\]](#). In this work we aim to design efficient heuristics for job shops that incorporate real-world characteristics such as uncertainty in the processing times and setup times.

^{*}Faculdade de Engenharia, Universidade do Porto, Porto, Portugal

[†]INESC TEC - Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Cincia, Porto, Portugal

The dynamic version of the Job Shop Scheduling Problem (JSSP) is defined on a set of n jobs $\{1, \dots, j, \dots, n\}$ and a set of m machines $\{1, \dots, i, \dots, m\}$. Each job j requires one operation to be processed by each machine i , in a given processing time p_{ij} . The sequence of machines to visit may be different among jobs and each machine can only process one operation at a time. The objective is to find a schedule of jobs that minimizes the makespan, i.e., the maximum completion times [Pinedo \[2008\]](#).

Dispatching rules have been shown to be a promising approach for dynamic job shop scheduling [Holthaus and Rajendran \[1997\]](#). They are simple and fast sequencing methods, used to prioritize jobs waiting to be processed. Blackstone et al. [Blackstone et al. \[1982\]](#) and Haupt et al. [Haupt \[1989\]](#) review the literature on rule-based sequencing methods for the dynamic JSSP.

Lawrence and Sewell [Lawrence and Sewell \[1997\]](#) compare the performance of seven simple rules and exact methods for dynamic job shop scheduling. They show that the dynamic heuristics outperform static optimum schedules even with moderate levels of uncertainty in processing times. As uncertainty increases the performance of the dispatching rules converges to that of the exact methods, reinforcing the use of simple scheduling methods in dynamic stochastic environments.

Despite the fact that several rules present good performance in general, the existence of a single rule that performs best in all settings is unlikely. This encourages studying methods that combine several dispatching rules and automated heuristic design [Dominic et al. \[2004\]](#), i.e. a process to explore the “heuristic search space” instead of the solution search space [Shahzad and Mebarki \[2012\]](#).

Genetic Programming (GP) [Koza \[1994\]](#) has been widely used for designing efficient dispatching rules and heuristics for scheduling problems. In the last decade there has been an increasing attention to the use and improvement of this technique [Hildebrandt et al. \[2010\]](#), [Nguyen et al. \[2018\]](#). A recent survey on GP is found in Nguyen et al. [Nguyen et al. \[2017\]](#) and it indicates that most of the works have considered deterministic processing times.

The vast majority of works have not considered aspects found in real-world manufacturing environments. As far as we know, the authors Karunakaran et al. [Karunakaran et al. \[2017\]](#) are the only to deal with dynamic job shop and stochastic processing times. But their work does not include setup times between the operations.

This work analyses the influence of two frequent aspects of real job shops, uncertainty in processing times and the sequence dependent setup times between operations in the same machine. We adapt the instance set used by Lawrence and Swell and extend their work by automating the rules design with GP.

The reminder of this paper is organized as follows: The Genetic Programming algorithm and its parameters are described in Section [A.2](#). In Section [A.3](#) we present the experiments design together with the computational results. We also discuss some insights regarding instances characteristics and the performance evaluation of rules. Finally the paper is concluded with Section [A.4](#).

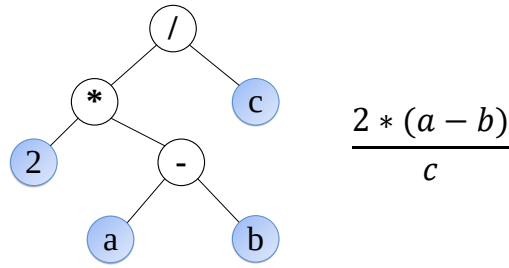


Figure A.1 – A GP individual tree example and its corresponding mathematical expression

A.2. Methodology

In general, evolutionary strategies evolve solution candidates for a problem. In Genetic Programming (GP) individuals are computer programs, which can be interpreted or executed to produce a solution to the original problem. Thus, the fitness value of each individual corresponds to how well its able to perform a specific computational task.

GP algorithms commonly represent individuals as symbolic expression trees. This is due to the fact that in GP the final structure and size of the program is not known in advance. The trees are composed by *terminal* nodes and *functions*. The former include parameters of the problem, which are operated by the latter.

Figure A.1 illustrates an individual, represented by a parsing tree and the corresponding dispatching rule. In the example 2, a , b and c are terminals, possibly related the attributes of the the operation. The function set is composed by the mathematical operators $*$, $/$ and $-$. Details on the specific implementation of GP in this work will be given next.

In this work an individual represents a rule to prioritize awaiting operations. Each rule results in a sequence of operations which will define the schedule for the job shop. The fitness of an individual is the average performance measure of many schedules built by sorting operations according to the corresponding dispatching rule.

A.2.1 Breeding

Breeding is achieved via crossover and reproduction. The crossover operation is standard: it picks two individual trees, selects a node in each tree, and swaps the two sub-trees rooted by those nodes. The reproduction operation simply copies the individuals to be part of the population in the next generation.

The crossover and reproduction operations are chosen according to a likelihood probability. Both processes use tournament selection for selecting individuals. According to the *Tournament Size* t , it returns the best individual from a set of t individuals randomly selected.

The breeding process pick certain kinds of nodes with different probabilities. For example, one can state to pick non-terminals 90% of the time and terminals 10% of the time.

A.2.2 Function and Terminal Sets

The function set is composed by the four basic arithmetic operations, the conditional *If* and the logical operators. The terminal set is based on the seven rules from the work of Lawrence and Sewell. We also incorporated relevant attributes not considered by them such as setup times. Table A.1 describes the complete function and terminal sets.

Table A.1 – Function Set and Terminals of GP

Function Set	Description
+	Addition
-	Subtraction
*	Multiplication
/	Protected Division
If	Conditional
&	Logical operator <i>And</i>
	Logical operator <i>Or</i>
$\geq$	Logical operator <i>Greater than or equal to</i>
$\leq$	Logical operator <i>Less than or equal to</i>
Terminal Set	Description
PT [†]	Expected processing time of current operation
-PT [†]	The negative of the expected processing time of current operation
OR [†]	Number of operations remaining for the current job
WF	Work following: Sum of the expected processing time of all remaining operations of the current job
SWF [†]	Successive work following: Sum of the expected processing time of all remaining operations (excluding the current operation)
SD [†]	Difference between current expected processing time and the expected processing time on the next machine
TR [†]	Right tail of the distribution of the remaining processing time of the current job
FCFS [†]	First-come first-served (Of jobs arrival time)
QSize	Queue size of current machine
RQSize	Sum of the queue size of the remaining machines for the current job
ST	Setup time between the last operation performed by the current machine and the current operation
n	Total number of jobs in the problem instance
m	Total number of machines in the problem instance

[†] Dispatching rule used by Lawrence and Sewell [Lawrence and Sewell \[1997\]](#)

A.2.3 Fitness Evaluation

The primary objective of the fitness function is to guide the evolution process through efficient dispatching rules. Therefore a dynamic job shop simulator was implemented for measuring their performance.

The simulation builds the schedule by sequencing the operations on a machine's queue every time a new job arrives. Once an operation finishes processing, its actual processing time is used to update the schedule, the job moves to the next machine and the dispatching

rule is called again. When all operations are executed, the final makespan is then returned.

The simulator receives as input a dispatching rule and a job shop problem instance and returns the respective makespan C . Given the lowerbound C^* for a problem instance, the *relative performance* is (C/C^*) . The overall performance of an individual i is estimated by $RelPer_i$, the average *relative performance* of the schedules generated by using the corresponding rule. Smaller values of $RelPer_i$ indicate better performances of individual i .

Another aspect considered by the fitness function is the size of trees. Large trees may adapt well for specific problem instances, but lose performance when applied outside of the training set. This is known as over-fitting [Kronberger \[2010\]](#). By promoting short trees, the final result might not only be more elegant, but also have more generalizability.

The most common way of keeping trees short is to limit them to a certain depth. In addition, we defined a parsimony strategy to smoothly penalize the number of nodes. There is a penalty function $h(i, g)$ that depends on the size of the tree $|T_i|$ and the number of generations g and is given by $|T_i| * g^{1.2}$. As we do not want to prematurely prune good branches, which can be critical for the evolution process, h exponentially increases with g .

At each generation all individuals from the population are evaluated by the fitness function with the objective of preserving the best ones for the next generations. Considering the pursuit of efficient rules and the avoidance of very large trees, the fitness $f(i, g)$ of individual i in generation g is calculated as follows:

$$f(i, g) = 1 / (RelPer_i + h(i, g) * 10^{-8})$$

Both the power of 1.2 and the coefficient of 10^{-8} were obtained by conducting some preliminary tests.

A.3. Computational Experiments

The GP algorithm was coded in Java and uses the evolutionary computation library ECJ [Luke \[2017\]](#). All experiments ran on a personal computer using an Intel(R) Core(TM) i7 CPU with 2.8 GHz and 12 GB of RAM. The parameters calibration for the GP were defined in preliminary tests. We consider 100 generations and a population of 200 individuals. We use a tournament size of 3 and the maximum depth of the tree was set to 17. The breeding probabilities are 0.9 and 0.1 for the crossover and reproduction, respectively. At last, the probability of selecting a terminal and a non-terminal node were set to 0.1 and 0.9, respectively. The reported rules are the best found after 10 runs.

In the following we detail the instance generation and the design of the computational experiments.

A.3.1 Experimental Design

The basic problem set is the same as that used by Lawrence and Sewell, which is composed by 53 deterministic job-shop benchmark instances. Problem sizes range from 6 machines and 6 jobs, to 15 machines and 20 jobs. For all instances, all jobs visit all machines.

Lawrence and Sewell introduced uncertainty into the problem set by generating stochastic versions of each instance. Given the processing time of an operation p_{ij} and a coefficient of variation (cv), the actual processing times were found by a gamma distribution with mean $E[p_{ij}]$ and standard deviation $\sigma_{ij} = E[p_{ij}] \times cv$. The expected processing times $E[p_{ij}]$ are the same as in the deterministic version. The coefficient of variation cv determines the uncertainty level (in the deterministic case, $cv = 0$). In this work we use the same distribution with cv in five different levels: 0.2, 0.4, 0.6, 0.8 and 1.0.

As the objective of the study is to approximate the instances characteristics to the ones found in real-world settings, we included sequence-dependent setup times in the basic problem set. These were sampled from a gamma distribution with mean and standard deviation equal to the mean and standard deviation of the instance processing times, multiplied by a factor ϕ (setup time level). For each deterministic instance in the basic problem set there are four variants generated with different setup time levels ($\phi = 0.25, 0.5, 0.75, 1$).

A.3.1.1 Training set

In order to perform a fair evaluation of the resulting rule, we do not use the complete instance set in the GP training. The objective is to evaluate if training the GP with a fraction of the instances is sufficient for it to design good rules for the complete set. Thus the fitness function uses the stochastic version of the benchmark instances with $cv = 0.6$ and setup time levels (ϕ) of 0.25 and 1. The average relative performance refers to 10 replications of each instance. As the benchmark instances compose a set of 53 problems, when evaluating an individual the fitness function calls the simulator $53 \times 2 \times 10 = 1040$ times.

A.3.2 Results and Discussion

To evaluate the quality of the final rules, we report the (*RelPer*) values on the testing set. The latter is composed by 100 replications of each instance type. The lowerbound C^* is the optimal value of the corresponding deterministic instance, provided by Gonçalves and Resende [Gonçalves and Resende \[2011\]](#).

A.3.2.1 Existing Rules

The first set of experiments aimed to evaluate the performance of the seven existing benchmark dispatching rules in the instances with setup times. For the sake of brevity we present only the results for $\phi = 0, 0.5$ and 1.

Table A.2 presents the average *RelPer* for all uncertainty levels and the number of instance types in which each rule presented the best average results (*# Inst.*). On both indicators, the three rules stand out: MOR, LTR and MWF. Given these results we decided to select the best three rules for the comparison with those evolved by GP.

Table A.2 – Results comparison of benchmark rules

Rule	Description	$\phi = 0$		$\phi = 0.5$		$\phi = 1$	
		RelPer	# Inst.	RelPer	# Inst.	RelPer	# Inst.
SPT	Shortest Processing Time	1.34	1	1.79	2	2.25	5
LPT	Longest Processing Time	1.48	0	1.92	0	2.38	0
LSD	Largest Successive Difference	1.30	6	1.79	6	2.23	3
MOR	Most Operations Remaining	1.29	3	1.74	6	2.20	13
LTR	Longest Tail Remaining	1.27	16	1.70	12	2.16	11
FCFS	First-come First-served	1.31	5	1.79	1	2.25	1
MWF	Most Work Following	1.25	22	1.68	23	2.15	20

A.3.2.2 Evolved Rules

We performed three experiment sets with GP which are explained in the following.

Evolved Rule R1 - As observed in Table A.2, the benchmark rules MOR, LTR and MWF present superior performance than the remaining ones for most of the instances with setup times. The first experiments with GP aimed to analyse if there is a combination of these rules suitable to perform better than any of them separately for the complete instance set. We thus ran GP using a terminal set composed only by {OR, TR, SWF}. This resulted in following rule:

$$WF/TR$$

Evolved Rule R2 - In the subsequent experiments we included all the remaining terminals listed in Table A.1 except the setup time (ST). The objective was to verify if simple rules with a poor performance could help producing efficient schedules when combined with other terminals. The result of GP is given bellow:

$$-OR \times ((PT \times n) + 2 \times PT + 2 \times SD)$$

Evolved Rule R3 - In addition to the terminals considered by R2, the last experiment set also considered the setup time terminal (ST). The evolved rule is :

$$\frac{\frac{WF}{ST+PT \times (1-1/(m+RQSize+n))} + m + RQSize + \frac{(2 \times (m+RQSize)+PT+(WF \times m))}{m \times (RQSize+n)}}{m \times (RQSize + ST)}$$

Let us start by analysing the results of the instances without setup times. The chart in Figure A.2 presents the average *RelPer* for all uncertainty levels. As reported in the work

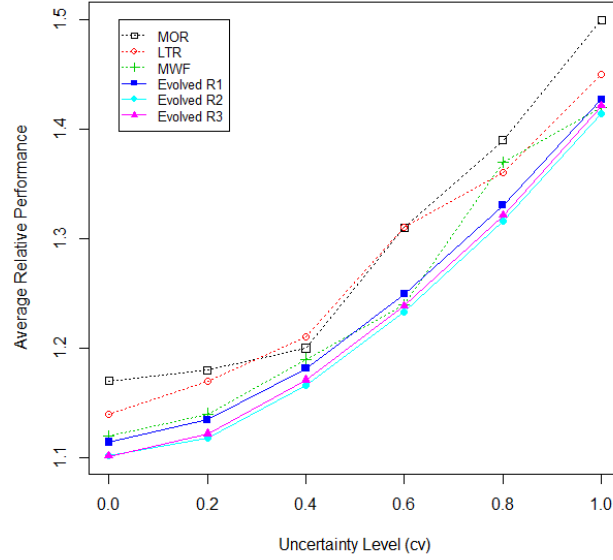


Figure A.2 – Average relative performance of evolved rules in benchmark instances without setup time

of Lawrence and Sewell, the rules tend to deteriorate their performance as the uncertainty level increases.

Observe that GP was able to find an efficient rule from the combination of the best simple benchmark rules. R1 performs better than MOR, LTR and MWF for most of the cases, except when $cv = 0.6$, where MWF is able to outperform it. This is actually surprising, given that this was the cv value used in training.

Regarding the rules R2 and R3, the comparison of their relative performances shows that both resulted in better schedules than all the other rules. Although R3 includes a terminal that is not used in these instance set (ST), its results are competitive. In addition, given that instances with $\phi = 0$ were not present in the training set, it is interesting one that all the three evolved rules were able to perform so well on those instances.

Figure A.3 presents the results obtained for the instances with setup times. Again the average *RelPer* is greater for larger values of cv . As the lowerbound used to calculate it was provided by instances without setup times, the relative performance also increases with the level of ϕ .

The overall results show that in general the three evolved rules performed clearly better than the benchmark ones. The difference in performance becomes greater as the value of ϕ increases. This was expected since the training set included setup times.

Comparing the evolved rules, we notice that the results from R2 and R3 are always better than those produced by R1. We also observe that for low levels of setup time (Figure A.3a) R2 and R3 have similar results. As ϕ increases R3 becomes better. Finally, Figure A.3d shows meaningful differences between the results found by R3 and all the other rules.

A.4. Conclusions and Future Work

In this work we applied Genetic Programming for designing efficient dispatching rules for stochastic job shops. Our main contribution is the analysis of the processing times uncertainty together with setup times between operations in the same machine. Benchmark instances were adapted to incorporate these characteristics.

The proposed method was able to find rules that performed better than the benchmark for the complete instance set. Moreover we observed that the performance of the evolved rules was clearly better for higher setup time levels.

As future research we aim to test the evolved rules in real problem instances and incorporate other characteristics found in real-world environments, such as due dates. Further analysis may also include other scheduling performance measures such as mean and total tardiness. Finally, one may adapt the method to deal with flexible job shops and other variants of scheduling problems.

Acknowledgments

This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme, and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project SAICTPAC/0034/2015- POCI-01-0145-FEDER-016418

Bibliography

- Blackstone, J.H., Phillips, D.T., Hogg, G.L., 1982. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research* 20, 27–45. doi:[10.1080/00207548208947745](https://doi.org/10.1080/00207548208947745).
- Dominic, P.D.D., Kaliyamoorthy, S., Kumar, M.S., 2004. Efficient dispatching rules for dynamic job shop scheduling. *The International Journal of Advanced Manufacturing Technology* 24, 70–75. doi:[10.1007/s00170-002-1534-5](https://doi.org/10.1007/s00170-002-1534-5).
- Gonçalves, J.F., Resende, M.G.C., 2011. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics* 17, 487–525. doi:[10.1007/s10732-010-9143-1](https://doi.org/10.1007/s10732-010-9143-1).
- Haupt, R., 1989. A survey of priority rule-based scheduling. *Operations-Research-Spektrum* 11, 3–16. doi:[10.1007/BF01721162](https://doi.org/10.1007/BF01721162).
- Hildebrandt, T., Heger, J., Scholz-Reiter, B., 2010. Towards improved dispatching rules for complex shop floor scenarios: A genetic programming approach, in: *Proceedings of the*

- 12th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA. pp. 257–264. doi:[10.1145/1830483.1830530](https://doi.org/10.1145/1830483.1830530).
- Holthaus, O., Rajendran, C., 1997. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics* 48, 87 – 105. doi:[https://doi.org/10.1016/S0925-5273\(96\)00068-0](https://doi.org/10.1016/S0925-5273(96)00068-0).
- Karunakaran, D., Mei, Y., Chen, G., Zhang, M., 2017. Evolving dispatching rules for dynamic job shop scheduling with uncertain processing times, in: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 364–371. doi:[10.1109/CEC.2017.7969335](https://doi.org/10.1109/CEC.2017.7969335).
- Koza, J.R., 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing* 4, 87–112. doi:[10.1007/BF00175355](https://doi.org/10.1007/BF00175355).
- Kronberger, G.K., 2010. Symbolic Regression for Knowledge Discovery - Bloat, Overfitting, and Variable Interaction Networks. Ph.D. thesis. Johannes Kepler University. Linz, Austria.
- Lawrence, S.R., Sewell, E.C., 1997. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. *Journal of Operations Management* 15, 71 – 82. doi:[https://doi.org/10.1016/S0272-6963\(96\)00090-3](https://doi.org/10.1016/S0272-6963(96)00090-3).
- Luke, S., 2017. Ecj then and now, in: GECCO.
- Nguyen, S., Mei, Y., Xue, B., Zhang, M., 2018. A hybrid genetic programming algorithm for automated design of dispatching rules. *Evolutionary Computation* , 1–31.
- Nguyen, S., Mei, Y., Zhang, M., 2017. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems* 3, 41–66.
- Pinedo, M.L., 2008. Scheduling - Theory, Algorithms and Systems. 3rd ed., Springer.
- Shahzad, A., Mebarki, N., 2012. Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem. *Engineering Applications of Artificial Intelligence* 25, 1173 – 1181. doi:<https://doi.org/10.1016/j.engappai.2012.04.001>.

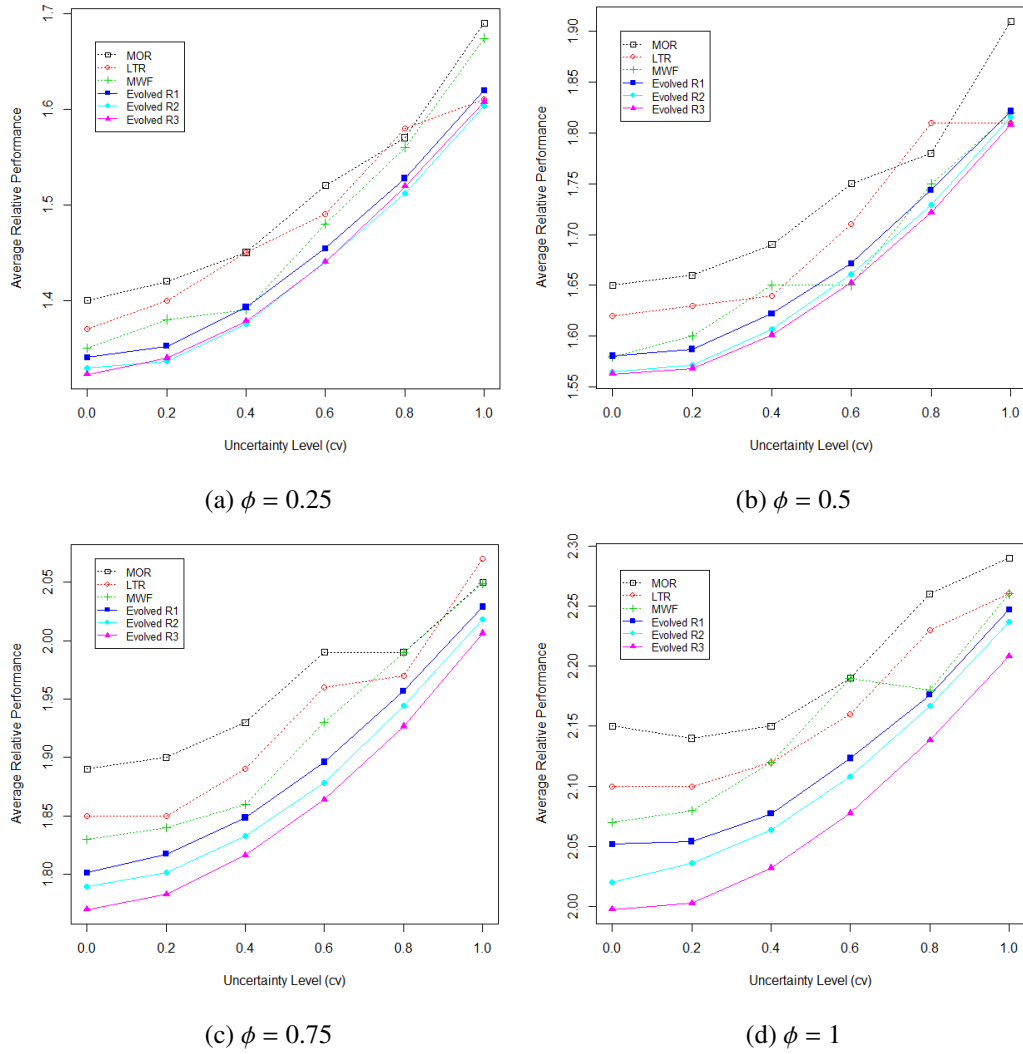


Figure A.3 – Average relative performance of evolved rules in benchmark instances with different setup time levels (ϕ)

