FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Evolving Dispatching Rules for Collaborative Environments through Genetic Programming

Rafael Lopes

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Pedro Amorim

Second Supervisor: Cristiane Ferreira

September 21, 2020

Resumo

Como resultado dos recentes avanços tecnológicos, as linhas de produção têm-se tornado mais dinâmicas e flexíveis. Quando combinados como o aumento na procura de produtos mais customizados e com lotes de produção mais pequenos, estes avanços geram uma enorme variação no ambiente de produção. Um dos avanços mais recentes que pode ajudar a responder a esta nova procura são as equipas colaborativas. Estas células de montagem compostas por uma mistura de humanos e robôs colaborativos conseguem alterar facilmente tarefas sem reprogramação dos robôs e vêm com a promessa de aumentar a produtividade da linha enquanto mantêm os custos de produção baixos. O planeamento de operações estático tradicional apresenta dificuldades em gerir este tipo de ambientes dinâmicos nas linhas de montagem. Quando recursos humanos e a sua inerente incerteza nos tempos de processamento são adicionados a estes processos esta dificuldade tende a aumentar. Quando um evento inesperado acontece na linha de produção, uma das possíveis soluções seria voltar a aplicar o método de planeamento estático, mas essa estratégia geralmente não é viável devido ao tempo que este plano demora a ser restruturado.

De forma a lidar melhor com este problema, podem ser utilizadas técnicas de planeamento dinâmico como as regras de despacho. Estes métodos possibilitam o planeamento de operações em tempo real, o que permite a sua adaptação a quaisquer eventos inesperados. No entanto, apesar de apresentarem um tempo de execução bastante rápido, o desenvolvimento humano de regras de despacho é demorado. Uma forma de ultrapassar esta questão é o uso de métodos de aprendizagem computacional como a programação genética de forma a evoluir regras para um conjunto de instâncias.

Este trabalho centra-se na evolução de regras de despacho para uso em ambientes colaborativos com o objetivo de minimizar o *"makespan"* e os tempos de espera. Para melhorar a solução desenvolvida pela programação genética, são utilizadas técnicas de otimização como a seleção de características. As regras de despacho geradas neste trabalho conseguiram superar a performance do *"benchmark"* em todas as instâncias. Das experiências realizadas também foi possível extrair algum conhecimento sobre a informação importante para o problema e a correlação entre os dois índices de desempenho

Abstract

As a result of technological evolution, assembly lines are becoming more flexible and dynamic. Combined with the recent increase in customer demand for lower batch more customized products this shift generates a large amount of variation in production. One of the most recent technological advances that can help answer this granular demand is human-robot teams. These assembly cells composed of humans and collaborative robots can switch tasks easily and come with the promise of increased line productivity while maintaining low costs. Traditional static schedulers have difficulty dealing with this dynamic assembly environment with the difficulty being emphasized when uncertain processing times are added to the mix, which is a common human trait. One possible solution is to reapply the static scheduler when an unexpected event occurs but usually, the time it takes to compute a new schedule makes this strategy unfeasible.

To better deal with this problem, dynamic scheduling techniques such as dispatching rules can be used, these methods schedule the operations in real-time thus easily adapting to any unforeseen events. However, despite having a quick execution time, human development of dispatching rules for a specific set of problems can be time-consuming. One way to overcome this is to use machine learning such as genetic programming to evolve rules for a given set of instances.

This work sets out to evolve dispatching rules for use in collaborative environments with the aim of minimizing makespan and idle times. To enhance the genetic programming solution, optimization techniques such as feature selection are used. The dispatching rules generated were able to outperform the benchmark on all instances. Insights on the information significant to the problem and the correlation between the two performance objectives are also extracted from the experiments.

Acknowledgements

I would like to begin by thanking Cristiane Ferreira for guiding me through this arduous adventure, being understanding and motivating me throughout all the stages of this work. Her explanations enlightened the path through the obscure concepts that genetic programming and scheduling were for me.

To Pedro Amorim for making this dissertation possible and for his specialized advice.

To my family for supporting me during the whole process and enduring my bursts of comedic insanity.

My heartfelt gratitude to my friends who motivated or me through all the worst moments in this process and made last minute arrangements just to hear me vent, your efforts have not been forgotten! Specially those of you that proofread this dissertation.

I would also like to thank everyone who made my academic stay a year longer than it was supposed to be, without you all the years that culminated in this project wouldn't have felt so complete.

Rafael Lopes

"Make things as simple as possible,
but no simpler."

Albert Einstein, 1879

Contents

List of Figures

List of Tables

Abreviaturas e Símbolos

DR	Dispatching Rules
FS	Feature selection
GP	Genetic programming
IT	Idle Time
LSD	Least Successive Diference rule
LTR	Longest Tail Remaining rule
MS	Makespan
MWF	Most Work Following rule

Chapter 1

Introduction

1.1 Context

Industry 4.0 (i4.0) refers to the revolution in the manufacturing industry that is currently ongoing, this new type of production processes rely on cyber-physical systems to network the whole production line allowing for the real-time gathering of data such as machine breakdowns, job alterations, performance metrics and general production status. With this kind of information it is possible to decide and assign tasks in real-time, in what is known as dynamic scheduling, which allows production lines to be more flexible and to manufacture a more varied and customised array of products.

Contrary to the traditional highly specialized, high output machinery that was separated from the human workspace for safety reasons, one big promise of i4.0 is the possibility of collaborative environments where robots work alongside humans by using advanced sensoring technologies and software to coordinate their movements [1].

1.2 Motivation

Collaborative cells have the advantage of being able to perform various assembly tasks without the usual need to stop for machine reprogramming or tool refitting. Furthermore collaboration between humans and robots in an industrial environment is a promising way to increase general line productivity while maintaining low production costs as it synergises the operator's ability to think, plan, decide and quickly react with the strength, precision and repeatability of a robot [2].

There are still two major challenges for the widespread implementation of this manufacturing system that directly impact its efficiency, the first relates to the way the human-robot collaboration is modelled and programmed, which can limit the type of interactions that the robot is able to perform. The second is the effective operation scheduling and resource allocation, which represents the main problem that this work is trying to solve.

Manufacturing systems were already dynamic environments in terms of timelines due to machine failures, the arrival of urgent jobs, due date alterations, etc. With traditional static scheduling

where operations were planned ahead for the next work period, due to these reasons some schedules quickly became unfeasible soon after being released to the production line [3]. With the use of collaborative teams, due to human variability, the uncertainty of processing times is added to the already dynamic environment creating timelines that greatly vary from the plans created by the traditional static scheduling. To combat their drawbacks dynamic scheduling can be used, dynamic scheduling is the term used for scheduling techniques that account for real-time events and adapt the production plan accordingly. These techniques show very promising results for even a small uncertainty in processing times, gaining an exponential advantage over static schedulers as the uncertainty increases [4].

Since the schedules are altered or remade "on the go" dynamic scheduling techniques need to process all available information and produce a solution in the smallest time frame possible which limits the use of search algorithms and heuristics that schedule all the available operations. Some alternatives have been proposed like the use of Petri nets and reachability trees by Casalino et al. [5] that significantly reduces the solution's processing time, but does so by limiting the time horizon of the simulation.

All these techniques greatly depend on the available processing power and time to be successful, but when dealing with a large number of tasks and resources combined with highly dynamic environments where the rescheduling happens very frequently a more expedited technique is needed.

Even if the quality of the solution is not as good, the rescheduling happens so often that its impact is reduced when compared to the time it takes to produce a feasible schedule. Taking this into account dispatching rules seem like a good solution for the problem. They have been in use for a long time and for specific performance objectives there are rules known to have a good performance. They are simple to understand and require a very small amount of processing power, these rules are usually a simple mathematical combination of variables extracted from the state of the assembly line, jobs and operations.

1.3 Aims and Objectives

This project aims to develop a simple and effective set of dispatching rules for a dynamic manufacturing system, optimising the work of the simulated collaborative working cells through machine learning, namely with the use of genetic programming to evolve good performing dispatching rules according to the desired scenarios and performance objectives.

These rules will then be tested and compared in different instances which correspond to different factory setups regarding resources, jobs and performances against the best performing ones from Lawrence et al. [4]. These rules will also be tested against each other to ascertain if there is any correlation between the experimental performance metrics.

For the aforementioned testing a preexisting simulator from the project of Ferreira et al. [6] is intended to be altered for the accommodation of the new experimental requirements such as: new variables to be used by the terminals of the machine learning algorithm, new performance metrics,

different job queuing and handling functions and a large number of small tweaks to preexisting parameters in order to enhance the performance of the process and optimality of the results when applied to the new work environment.

A novel technique termed feature selection will also be utilized on the genetic programming algorithm to verify if there are improvements either on the created rules or on the machine learning process itself.

1.4 Structure and Organization

This dissertation is divided into 5 chapters, with the first being this introduction describing the problem and what is expected of a solution. Chapter 2 includes descriptions of concepts used in this work along with references to recent works and important papers published in the different subjects comprehended in this project. Moreover, a description of the problem being tackled is also provided A proposed solution is presented in Chapter 3 with clarifications on its implementation. In Chapter 4 the obtained results are presented and analyzed. With Chapter 5 formalizing the conclusions and their importance.

Chapter 2

Dynamic Scheduling for Human Robot Teams

With the increase in demand for low volume/high variation production in today's market, human-robot collaborative teams have been getting a big amount of attention due to their high adaptability and reusability in different parts of assembly systems. Depending on the average lot size, hybrid assembly systems can reduce fixed production costs relative to variable costs [7], with the reduction increasing the smaller and more customized the produced lots are.

One of the most common uses for hybrid teams is sequential task division with robots handling simple, more repetitive or strenuous tasks frequently found upstream and more complex tasks found downstream handled by humans with a better ability of differentiating the products. This type of work division has been successfully utilized for decades in assembly lines with good efficiency results [8, 9].

Contrasting to the above-mentioned situation, in this paper's proposed case, some of the tasks can be performed either by humans, robots or in a collaborative manner requiring one human and one robot. The decision of which option is more efficient and when to dispatch that order is made by the developed scheduler.

2.1 Human-Robot Teams

In manufacturing systems collaborative environments or human-robot teams refer to the most advanced implementation of collaborative robots of the four classifications proposed by Muller et al. [10], in which different use cases of collaborative robots are divided into four classifications based on the level of human-robot interaction:

- **Coexistence** refers to a case where the human and the robot share the same area but do not interact, this is usually done when the available space is small or to reduce transport times.
- **Synchronization**, where the robot and human can work on the same product and part of the workspace but do so in an alternating manner further reducing space needs.

- **Cooperation**, where the robot and human are able to perform simultaneous tasks inside the same workspace, these tasks can even be on the same object but they could be done separately.

- **Collaboration**, where the human and robot work together on a single task and each one's actions require a reaction from the other.

In the last type of collaborative environments both human and robot can perform tasks together, for example in a case where the robot must lift and hold an object while the human worker performs some type of operation on it as presented in Zanchettin et al. [11], where the robot would act as a third hand for the worker. There is also another application where the robot can be used as a fatigue reducer and it merely reacts as a direct feedback to the worker's input, in Peternel et al. [12] a sensor placed on the worker's arm would detect muscle activity and from that input the human worker's intention would be inferred, with the robot aiding the expected movement to reduce muscle strain.

Figure 2.1 shows an example of a collaborative working cell containing one human and two robots, a possible solution for a ten job problem is presented in the Gantt chart. There are two possible human-robot teams present in this situation, one composed by the human H and the robot R_1 and the other team composed by the same human H and the second robot R_2. In the presented solution the tasks C is performed in a collaborative mode by H and R_1, with the rest of the tasks being performed by the resources in an individual manner. It is also worthy to note that in this particular schedule when the human H finishes operation A it waits for the robot R_1 in order to begin the collaborative operation C.

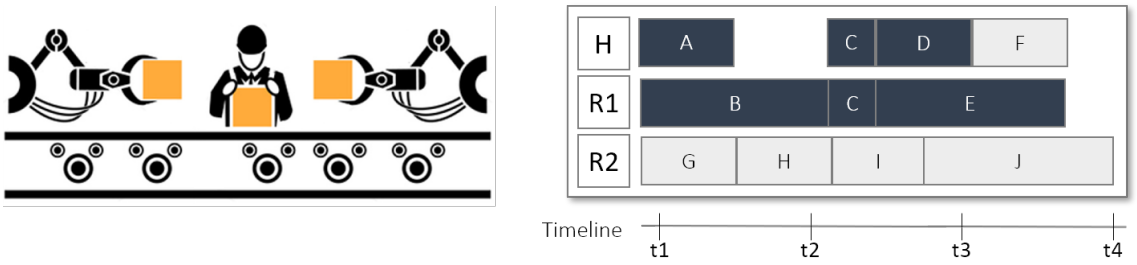

Figure 2.1: Example of a collaborative cell and possible schedule solution

In this method the interaction can be done proactively, where the human is able to signal the system its intentions and based on that information the robot acts towards performing the tasks intended by the human [13] or it can be done re-actively where the scheduler reevaluates its intended actions when the human element creates a change in the assembly line state [14].

Over this last type of interaction some improvements have been done in Casalino et al. [5], where the system also takes into account the predictions of human activity when reevaluating the best path. Besides the interaction synchronization one other element of the process that has a great impact on the performance of these systems is the scheduler, when considering such a varied

workforce with the ability to perform the same tasks in different ways, a good schedule can have massive gains over a simply feasible one.

2.2 Parallel Machine Scheduling

2.2.1 Introduction

Parallel machine scheduling refers to a wide array of problems in which a set of jobs or operations need to be scheduled for processing and have multiple resources to be scheduled on. The time it takes for the job to be accomplished varies depending on what resource the job is scheduled to, with each resource only being able to process one job at a time. In unrestricted parallel machines, any job can be scheduled for any resource with the only restriction being the availability of the resources at the moment the job is scheduled to start.

In restricted parallel machines the difference is that each job has a subset of resources, ranging from the totality of the resources to at least 1 of the resources, and the job can only be scheduled for execution on this particular subset of resources in a process that is termed resource eligibility.

Different eligible resources can display different processing times for the same job. This variability in the length of the individual job processing times makes these types of scheduling problems a particularly interesting for study in terms of actual makespan, total expected completion time and tardiness [15].

In the search of a solution for these types of problems a variety of methods have been employed, Darvish et al. [16] considered AND/OR graphs to model the possible assembly paths from a particular moment in time, with a decision being made based on the expected outcomes of each path. Chen et al.In the realm of meta-heuristics [17] made use of genetic algorithms with the solutions corresponding to different possible schedules and Wu et al. [18] made use of a tabu search algorithm with the objective of dealing with larger size instances while reducing the inherent increase computational inefficiency.

For smaller sized problems some mixed integer programming solutions have been proposed by Wu et al. [18] and Gombolay et al. [19] as a way to model the problem, this method has the advantage of reaching near optimal solutions for the modelled cases.

2.2.2 Dynamic scheduling

When dealing with dynamic situations and faced with unexpected or uncertain events in the production line, traditional static schedulers have the option to wait for the unexpected event to resolve or to make a completely new schedule. Although the second option may be better in terms of optimality it can be prohibitive due to its high computational cost and can be disruptive to other higher-level schedules [3], in an evolution of the aforementioned options to deal with these types of environments, dynamic scheduling strategies were developed.

These strategies are developed from the ground up on the basis that any scheduling or predictions undertaken might not happen as expected, as defined by Aytug et al. [20] three main approaches can be taken:

- **Robust Scheduling** in which very similarly to static scheduling, a predictive schedule is built at the start with the difference being that in its creation a big focus is taken on reducing of the impact that expected disruptions can have on the main performance objective of the schedule. One big challenge of this type of scheduling is that it requires extensive knowledge of the process to reach satisfactory results.
- **Predictive-Reactive scheduling**, this type of scheduling divides the process into two steps. In the first, a predictive schedule is created, as in a static scheduler, which is released to the assembly line. At certain intervals or triggered by unforeseen events this schedule is then adjusted or remade depending on the selected strategy. This technique is usually implemented along with a receding horizon technique with the intention of reducing solution computing times [21][22].
- **Reactive scheduling**, in this approach each resource has a queue of jobs that it is eligible to perform, when a resource becomes available or an unexpected event happens the system calculates the best task to perform next on that particular resource.

In the reactive approach, recent works have taken three distinct strategies to find a solution for this problem. In the first researchers have resorted to search heuristics in which the solution space consists of the ordered tasks for each eligible resource, in Chen et al. [17] genetic algorithms were used to evolve a schedule. This types of strategies are able to consistently achieve optimal or semi-optimal solutions, but as the search space grows the computing times to arrive at a solution can become prohibitive.

Another approach relies on mapping the possible occurrences in the assembly line until a horizon time frame, usually being represented by trees that branch out at points in time where a decision should happen or an uncontrollable event is predicted to occur. In this approach each node represents a possible state of the assembly line, and when the branching is done it accumulates costs or a performance metric from parent to child. When the simulation horizon is reached the metrics or costs are backpropagated from the leaves up to the children of the root node (that represents the current moment and the possible decisions), the decision with the most favourable weighted outcomes is then chosen [16][11].

One particularly interesting application of this approach can be seen in Casalino et al. [5] in which the assembly line is modelled by a Petri net with stochastic timed transitions. A partial reachability tree is then constructed by running a set of Monte Carlo simulations until a certain time horizon with the costs being backpropagated from child to parent until the children of the root node are reached. If the transition represents a controllable situation the parent inherits the cost from the most desirable child (representing a conscious decision towards the most desirable path), if the transition represents an uncontrollable situation the cost representation of the parent is obtained by combining the children's costs.

Although these strategies have the ability to provide good solutions and additionally good insights on the specific problem instances they are applied to, they have a very high computational cost and despite it being exponentially reduced by decreasing the time horizon, this can have very negative effects on the scheduler's performance.

The third strategy is based on the creation of mathematical formulas that are applied to each job on a resource's queue when the latter becomes available. These formulas are usually composed of variables related to job and operation attributes or the state of the assembly line and are used to calculate a priority value for each job. The highest priority job is then selected for processing by that resource. The referred process does not actually develop any kind of predictive schedule per se but rather computes the best action to perform when a resource becomes available or an event alters the assembly line state.

2.2.3 Dispatching rules

The mathematical formulas utilized by the aforementioned method are termed dispatching rules. When compared to other dynamic scheduling methods these mathematical formulas have the advantage of requiring very little processing power, on the other hand, they are somewhat myopic when compared to the capabilities of taking future consequences into account that the previous strategies display. This can be coped with to some extent by implementing certain look ahead strategies as done in Su et al. [23] and Nguyen et al. [24].

Due to their characteristics, dispatching rules are better suited for application in situations where there is considerable uncertainty in the processing times or a large number of disruptions in the production line. In these scenarios the predictive nature of the first two strategies mentioned in Subsection 2.2.2 stops being an advantage since the events will most likely never happen as planned, in addition, if in this situation the rescheduling happens frequently the time it takes for these strategies to compute a solution might become unacceptable. In these cases the dispatching rules' logic simplicity becomes an advantage due to their very light computational cost and consistency in the time it takes to find a solution.

Despite their general shortcomings dispatching rules can perform outstandingly well on most scheduling scenarios taking into account that a suitable rule is built, this rule development for a given scenario can come from intrinsic knowledge of the problem [4] or it can be achieved through machine learning [25] [6][26]. Most recent cases of the latter use evolutionary algorithms that allow for a rule to be evolved to optimize certain performance metrics in a desired set of scenarios.

2.3 Problem definition

The problem presented in this work closely resembles the multiprocessor task scheduling problem presented in Drozdowski et al. [27] and relies on a set of R resources and J jobs subdivided into operations with precedence constraints O_j. Resources can be humans or robots. A mode M is composed of a subset of resources R_m, this subset can be a human, a robot or a collaborative mode

composed by a human-robot team. Each resource can only process one operation at a time and an operation o must be processed in one of its eligible modes M_o, with each mode having its specific processing time p_{om}.The objective is to schedule all operations for processing in a non-preemptive manner. The expected processing time for each mode p_{jm} is stochastic, with the actual processing time only being revealed when the operation is finished.

Chapter 3

Methodology

3.1 Genetic Programming

> " Computers do not program themselves; they need a qualified and experienced programmer who understands the complexity of the task. An alternative to paying a human programmer to design and debug a program, is to build a computer system to evolve a program. This may not only be cheaper, but has the advantage that progress can be made on problem domains where a human programmer may not even have a clear idea of what the programming task is, as there is no formal program specification. Instead, a partial description of the desired program's behaviour could be supplied in terms of its input-output behaviour." Burke et al. [28]

Heuristics are problem solving strategies mostly used when the search for an optimal solution is impracticable or when this solution is impossible to reach. These strategies rely on practical problem information to find an acceptable solution for the problem at the minimum, this solution development usually employs some sort of trial and error with the guidance of a function that is able to discern the quality of a possible result in relation to desired performance metrics. They are usually derived from previous experiences with similar problems but have a wide array of applicable situations.

Genetic programming is a heuristic that borrows ideas from natural evolution and survival of the fittest to evolve a population of individuals towards the fittest ones using concepts such as selection, reproduction and mutation. Due to the abstract nature of the individuals it can evolve, to the fact that it only requires sparse knowledge of the information it is given as an input and also a way to test the solution's performance, the technique can be applied to a wide variety of problems. This type of heuristic also displays the perk of for a given problem creating good performing solutions that are specialized in the subset of instances that it was trained in. When these strategies are applied to scheduling they allow for the development of schedulers that can outperform manually designed ones [6] even when utilized in dynamic scheduling problems of complex understanding [25].

3.1.1 Individuals

Each individual of a population is usually represented by a tree structure, in these structures the leaf nodes, also referred to as terminals, are problem-related variables or constants and they are the inputs of the structure. Non-leaf nodes are referred to as functions or operators and usually consist of mathematical or logic operators that return a value by performing their respective operation on the returned values of the children nodes. In Figure 3.1 an example of an individual can be seen alongside its mathematical representation.

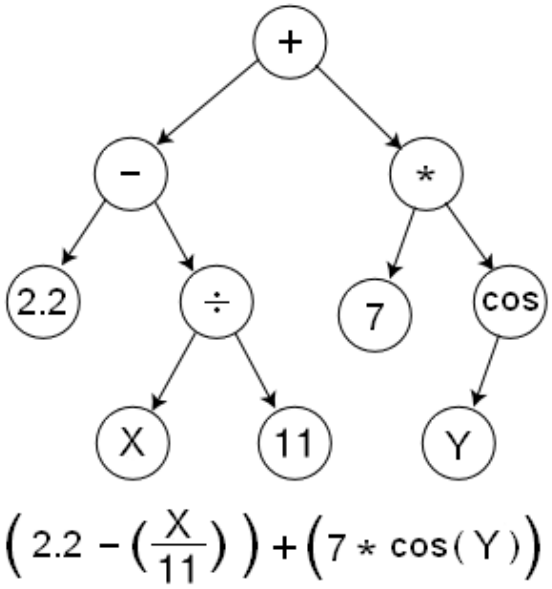

Figure 3.1: Genetic programming individual and equivalent mathematical formula.

3.1.2 Evolution

Evolution creates a new population of individuals based on the previous one, each set of individuals in a population that coexist between evaluations is called a generation. Before starting this process some parameters have to be set such as the number of individuals per generation (what is termed population size), the operators and terminals of the individuals, some values that dictate how the population is evolved: mutation, crossover and reproduction rates, and parameters that dictate which percentage of the population is selected to pass their genes to the next generation.

The first step in the process is to initialize a population of random individuals, this is done by generating the trees with a size comprehended inside a specific range and containing random operators and terminals, the randomness aspect is important in that the more diverse a population is, the bigger is the chance of starting with an individual that has good traits.

The subsequent process, the evolution itself, can be divided into three phases that happen in a repeating manner until the termination condition is activated, which usually corresponds to the number of passed generations:

- **Evaluation** - in this step each individual is evaluated by testing its performance in a set of instances and thus attributing a fitness value to each individual corresponding to their performance in the desired performance metrics. For these tests a simulation environment is usually developed.

- **Selection** - is the process through which the parents of the next generation are selected, being that the individuals of the current generation that have the better fitness value have a higher probability of being selected as parents, although various techniques exist the selection is frequently done through a process called tournament selection in which a number of individuals corresponding to the tournament size are randomly selected from the population with the better performing chosen to undergo the next step, this process is then repeated until the parent size requirement is met.

- **Reproduction** - in this step the individuals of the new generation are created either by crossover in which parts of the parent trees are exchanged by removing a node and its corresponding sub-tree from each parent and attaching it to the other one or they can be created by mutation which modifies an individual by selecting a node and its sub-tree and replacing it with a random one. These processes can be visualized for better understanding in Figure 3.2.One additional strategy that is frequently implemented to avoid performance degradation between generations is elitism in which the best n individuals are allowed to stay unaltered between generations, thus guaranteeing that the performance never decreases as generations pass.

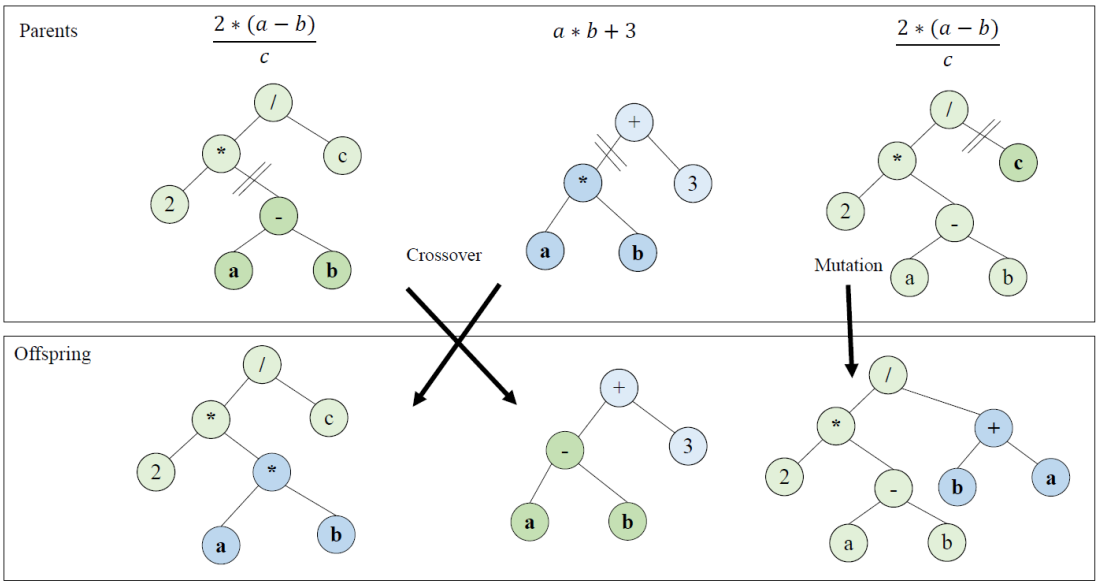

Figure 3.2: Example of the creation of new individuals through crossover and mutation.

3.1.3 Feature selection

Feature selection refers to a strategy that aims to reduce the total pool of possible variables (features) that can be selected for use in the terminals of an individual. The removal of redundant or unimportant information results in a reduced solution space which in turn decreases the training time and improves classification accuracy by decreasing the likeliness of a good performing solution having random unimportant input values.

There are two main approaches for feature selection, wrapper approaches and filter approaches [29], wrapper approaches consist of an extra feature evaluation algorithm in the evaluation step that classifies the impact of the variables on the desired performance metric, these algorithms add immense computational load which can be prohibitive in most manufacturing cases where there is a lot of information about the problem. On the other hand filter approaches require less processing power but have the downfall of not evaluating the terminal's impact on the performance both when isolated and when inserted in a feature subset.

Mei et al. [30] proposed a very simple yet effective way to evaluate the impact of a certain feature on the desired performance metrics in which from the features present in the best rules each one is removed in an alternating manner and the rule is then simulated to evaluate the impact of that removal. This strategy contrasts with previous ones based on appearance frequency analysis [31] in the way that it intrinsically takes into account the fact that a feature can appear several times on an individual but depending on the operators its contribution to the total performance value can be cancelled.

3.2 Proposed Algorithm

Genetic programming has seen extensive use in the design of efficient dispatching rules for scheduling problems [24]. In this work it's used to evolve schedulers for dynamic scheduling in a combination of two uncommon scenarios in the literature, uncertain processing times and resource collaboration.

3.2.1 Terminals and function sets

The individuals evolved in this work are dispatching rules, as these are supposed to be easily understood and executed, the function set is composed of the four basic arithmetic operators. The division operation has been substituted by a protected division, that in the case of a division by zero happening it simply returns zero. The terminal set is composed of the dispatching rules from Lawrence et al. [4] and any factory state and job information that could be of use to the algorithm, as can bee seen in Table 3.1.

A generous amount of data was given to the GP, even some that may not seem useful for two reasons, the first one is so that the GP can "choose" which information is indeed of value for achieving a good performance, secondly if the initial set had a small size the set after feature selection might not be reduced enough for the impact of a smaller search space to be tangible.

Table 3.1: Function and terminal set of GP

Function set	**Description**
+	Addition
-	Subtraction
*	Multiplication
/	Protected division
Terminal set	**Description**
1	constant of value 1
10	constant of value 10
PT	Expected processing time of selected operation
NPT	Processing time of the next operation in the current job
RPt	The sum of the processing times of all unfinished operations in the respective job
PT/RPT	Operation processing time divided by the remaining processing time of the job
QS	Queue size of current execution mode (number of operations in queue)
ptQ	Average processing time in the queue of eligible resources for the remaining operations of the job
QSNN	Queue Size of resources for the next operation in the job
nOpsLeft	Number of operations left to perform in the job
Now	Number of time elapsed since the beginning of the simulation
t	Time elapsed since the order entered the system
Readytime	Moment in time when the operation is expected to be completed
jt	Addition between the time spent in system and the expected processing time
WIQ	Processing times in queue for the current mode
WQS	Work in queue for the machines of the next operations in the job divided by the number of remaining operations for the job
NPossModes	Number of eligible modes for the current operation
NPossMachines	Number of possible resources in which current operation can be executed
nMach	Number of Resources in current mode
j	Number of jobs in the factory
AVGQ	Average queue size of all resources
AvgWIQ	Average processing times in the queue of all resources
RelSpeed	Expected processing time on current mode divided by the expected processing time on the fastest mode
PossAvgWIQ	Average WIQ of the eligible resources for current operation
SDiff	Average processing time of the remaining operations of a job minus the expected processing time of current operation
rSuccPt	Remaining processing time of the job minus the expected processing time of current operation
U	Utilization value for current simulation
t	time the order has been in a queue
USlack	one subtracted by utilization value of current simulation

3.2.2 Population evolution

Following the work done in Ferreira et al. [6], the initialization of the population uses the Ramped Half-and-Half algorithm. This technique combines the methods "Grow" and "Full" with variable maximum depths, contained inside a predefined range in order to create a diverse population. These trees can have all the terminals at the maximum depth when created by the "Full" method or can be unbalanced and not reach maximum depth when built by the "Grow" method such as the tree in Figure 3.3. All nodes are selected randomly from the available sets of functions and terminals.

In order to obtain a good dispatching rule the population needs to be evolved, the pseudo code used for this purpose is shown in Algorithm 1.

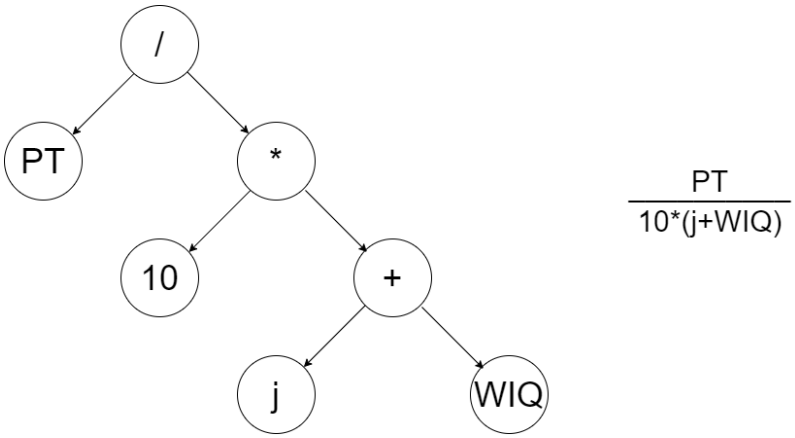

Figure 3.3: Genetic programming individual represented by its tree and the corresponding dispatching rule

In the used GP there is a generational evolution, in which each individual of a population of a given generation P_g is evaluated, with the population being updated after all of its individuals are evaluated. By far the most computationally expensive part of the algorithm is obtaining the fitness value of the individuals, which is even more noticeable due to the need of several replications of the simulator, detailed in Subsection 3.2.4, in order to obtain a reliable estimation of the Dispatching rule's performance.

3.2.3 Fitness function

Given the varied number of possible schedules and the differences of processing times depending on what mode the operation is performed, makespan is expected to be an interesting performance metric due to the high variability of processing times in this problem. This is in line with the performance metrics most recent works on collaborative machines have been using [17, 5, 32], most of these works also use idle time either as a performance metric or as a rule for scheduling, Chen et al. [17] uses it with an associated cost for a human and another for the robot with the human's cost being higher representing the actual cost of operating these resources in a real assembly line.

Algorithm 1: Genetic programming algorithm

```
   Data: Training instance set IS
   Result: Dispatching rule represented by its individual Ind*
1  Ind* ← 0;
2  g ← 1;
3  Population initialization: P_g;
4  while g ≤ g_max do
5      forall Ind in P_g do
6          forall instances i in IS do
7              Perf_Avg ← Average performance metric from n_e runs ;
8          end
9          Calculate Fitness of Ind;
10     end
11     A ← Best individual in P_g;
12     if A better than A* then
13         A* ← A
14     end
15     P_{g+1} ←New population after crossover and mutation
16     g ← g + 1 ;
17 end
```

Makespan is the time in which the last job is finished. In this work, we calculate the idle time as the sum of all time intervals in which a resource is idle. The fitness function also takes into account the size of the trees by adding a size penalty to the performance value. Besides aiding in obtaining simpler, more elegant and more easily interpretable dispatching rules this penalty also combats what is known as over-fitting. This problem occurs when trees become large and adapt too well to a specific problem but at the same time lose performance when they are applied to a set different from the training one.

As shown in Equation 3.1 the penalty value is calculated with the use of a tuning parameter p, the size of the tree S_{Ind} and the number of the current generation g, this last value is taken into account to avoid discarding promising branches early in the evolutive process.

$$f(S_{Ind}, g) = Perf_{Ind} + S_{Ind} * g * p \tag{3.1}$$

3.2.4 Simulator

To evaluate the performance of an individual, lines 7-9 of the Algorithm 1, a simulation is created, this simulation receives a dispatching rule, a set of parameters explained in detail in Subsection 3.3.1 and a set of instances that differ from each other in the number of resources, the relative performance between human and robots and the percentage of how many operations will be eligible for a collaborative mode (Subsection 3.3.2). Both the parameters and the instance sets vary between the training part of the GP and the validation of the created rules.

Before the beginning of the simulation the information relating to the current instance is read from a instance set database such as the amount of resources, their performance and the job characteristics, a more in depth description is given in Subsection 3.3.2. When the simulation starts all operations are queued on the respective machines, then the received dispatching rule is run on all operations of a resource's queue with the one that has the highest priority value being selected for processing. This last process is repeated every time a resource becomes available until there are no more operations on its queue. When there are no more operations on any queue and all machines are available the simulation stops.

When the simulation closes the performance metrics are returned. These can be either the makespan, by calculating the time between the start of the processing of the first operation and the time at which the last operation finished processing, or the idle time implemented by checking when was the previous operation finished every time a new operation starts processing on machine and adding to the idle time counter the difference between those two moments.

The pseudo code for the simulator is presented in Algorithm 2,with the inputs for the simulator instance being a dispatching rule *DR*, the coefficient of processing time variation *cv*, the number of repetitions *nReps*, the number of total jobs *nJobs* and an instance from the instances database such as the ones shown in Appendix A.

3.3 Experimental Design

The genetic programming algorithm is coded in java and makes use of the ECJ library, this evolutionary computation framework is very flexible with the capability of receiving parameters, function sets and terminal sets dynamically through user-specified files. The experiments were done on the Grid FEUP computational infrastructure and on a ryzen 2500u, 8GB RAM personal computer.

3.3.1 Genetic Programming parameters

Several preliminary tests were done to set some basic parameters for the GP. For the population size and the number of generations, the values were selected by balancing an acceptable runtime with a highly varied starting population, the number of generations was then tuned so that the fitness value stabilizes at around $0.7 * g_{max}$. The next step was tuning the tournament size, mutation and crossover rates so that the population stays diversified throughout all generations while improving the fitness value and not stagnating to early.

The chosen values are shown in Table 3.2.

3.3.2 Instance sets

The simulation called by the genetic programming algorithm to test the evolved rules makes use of an instance set database to create distinct simulated assembly line instances. In order to test the generalizability of the evolved rules two different instance sets were created, a smaller one with 16 instances used to evaluate the performance metrics during evolution, Appendix A.1 and a larger

Algorithm 2: Simulator algorithm

```
Data: DR, cv, nReps, nJobs, instance
Result: Performance metrics
Rep ← 0;
Makespan ← 0;
Idletime ← 0;
while Rep ≤ nReps do
    ITmetric ← 0;
    MSmetric ← 0;
    j ← 0;
    while j < nJobs do
        jobs ← create new job;
        j ← j + 1;
        push jobs to machine queues;
    end
    while j>0 do
        if machine completed an operation then
            if job was completed then
                MSmetric ← max(MSMetric, time instant);
                j ← j − 1;
            else
                push job to next machine;
            end
            forall operations in resource's queue do
                OpScore ← result of DR evaluation;
            end
            nextOp ← Op with lowest score;
            ITmetric ← ITMetric + (time instant − time last op finished);
            execute nextOp;
        end
    end
    Rep ← Rep + 1;
    Makespan ← Makespan + MSMetric;
    IdleTime ← IdleTime + ITmetric;
end
Makespan ← Makespan/nReps;
IdleTime ← IdleTime/nReps;
```

one with 32 instances for the validation of the best rule in the evolutive process, Appendix A.2.The use of smaller training set also provides the benefit of lowering the evolutive computational cost of the genetic programming algorithm.

The instances used differ from each other in five key parameters:

- The number of humans (1; 2) and robots (1; 2; 3), these variations were used to make sure the evolved dispatching rule could handle environment settings with a different number of available resources. Represented by nH and nR respectively. The number of resources is

Table 3.2: Parameters used in the Genetic Programming

Population Size	200
Number of generations(g_max)	100
Tournament size	3
Maximum depth of trees	Infinite
Crossover rate	0.7
Mutation rate	0.3

relatively small when compared to parallel machine problems, but follow common sizes used in works referring to collaborative assembly cells.

- The minimum (1) and maximum (100) time of the created operations, represented by *minPT* and *maxPT*, when generating operations the simulator randomly attributes a duration between these values to the expected processing time of the operation.
- *minOps* (2) and *maxOps* (14) represent the minimum and the maximum number of operations in a job, and follow some commonly adopted values present in the literature.
- Robot eligibility (0.25; 1), which informs the simulator on the percentage of generated operations that are eligible to be performed by robots.Symbolized by $RElig$.
- Robot performance (0.5; 2) representing the relative processing speed of the robot in comparison to the human, a relative performance of two signifies that the robot's processing time for a given operation is half of the human's.Characterized by $RPerf$.
- Multi mode eligibility (0; 0.5), this parameter quantifies the percentage of operations that are eligible to be performed on a collaborative mode. Depicted in the Table by $mMode$.

Besides the instance information on the instance set database, the simulator also receives parameters from a user-adjustable function. Following the aforementioned logic, two sets of these parameters were created one for training and one for validation. The used parameters were the following:

Table 3.3: Parameters used the evolutive and validation component of the GP

Component	Training	Validation
ISdb	A.1	A.2
timeCV	0	0; 0.2; 0.4; 0.6; 0.8; 1
nJobs	20	10; 20; 30; 50
Reps	5	100
smallReps	0	-

In the cases where there are various values for a parameter, the simulator runs each instance in the Instance set database (*ISdb*) one time for each of the values in the parameter list. As there are

six *timeCV* and four *njobs* on the validation component each instance in the instance set database represents 24 ($6*4$) sub-instances that are run separately. In the following we present a parameter description:

- The number of repetitions done on each instance is represented by *Reps*, the repetition values do not take into account the first simulation, a *Reps* value of five corresponds to six simulations being done, on the validation component one hundred and one runs are done to get a very accurate estimation is done.*smallReps* is the number of repetitions used in the first (small) iteration of line 5 in Algorithm 1, since it is only used in training there is no value for evaluation.

- *nJobs* represent the number of jobs that are released to the assembly line on that particular sub-instance, as can be seen, the evolution component is done with one set of twenty jobs and the validation has a bigger set of jobs with smaller, equal and higher values. As previously mentioned this reduces the training time while also allowing the verification of generalizability.

- *timeCV* dictates the amount of uncertainty in processing times, a value of zero means the processing times are deterministic and a value of one signifies that the processing times are represented by a gamma distribution with the mean equal to the deterministic processing time Pt_{det}, the standard deviation is calculated through the following formula $\sigma = Pt_{det} * timeCV$.The value of zero in the training part implies that the simulations used the rules' evolution use deterministic processing times, for the testing of the rules' performance this is expanded to 6 cases with varying degrees of uncertainty.

In sum during the evolution process, rules are tested in sixteen different instances, with variation deriving solely from the instance set database, while the evolved rule is evaluated in 768 ($32*6*4$) difference settings that include high uncertainty in processing times, counting the parameters for the instance sets and the user-adjustable function.

3.3.3 Experimental setups

3.3.3.1 Size penalty

The next experiment was produced to test the effects of different size penalties on the evolved rules. Since this project works with two performance objectives, makespan and idle time, a similar setup was produced for each objective.

For the makespan, four size penalties were tested (0; 0.001; 0.01; 0.1), values higher than 0.1 were discarded, in tests of preparation for the experiment, due to producing rules with only two terminals since at the end of the evolution the weight of the penalty would surpass any benefit provided by a proper dispatching rule. The value of 0 was added to the set in the interest of surveying what kind of dispatching rules were evolved when there was no individual size oversight, and ultimately test in this application of GP if some sort of tree depth limiter is ultimately needed.

In the case of the idle time objective three size penalty values were tested (0.001; 0.01; 0.1), the reason for the maximum size penalty being 0.1 is the same as in the makespan objective. In this case the value 0 is not present on the set due to time constraints, and the main objective of using it being almost completely fulfilled by its utilization in the setup for the makespan.

The values for these penalties were chosen by differences in orders of magnitude 10 as a way to test their effects, in more specific applications they can be fine-tuned for the problem at hand. The results of these tests are presented in Section 4.2.1.

To get a point of comparison the results are tested against the best dispatching rule from Lawrence et al. [4], most work following (MWF) as described in the paper:

> "The MWF heuristic selects that job for processing with the longest expected processing time on all remaining successor machines - note that processing time on the current machine is excluded.(...)This can be seen by noting that for any arbitrary makespan M, the due date for a particular operation will be M less the total processing time of all successor operations. We would thus expect MWF to do well in minimizing maximum tardiness relative to M, and since M is arbitrary, to do well in minimizing makespan." Lawrence et al. [4]

.

3.3.3.2 Idle time vs Makespan as performance objective

As mentioned in 2.2.2 some recent works use idle time either as a performance objective or as a variable to be reduced in schedulers working with decision trees based in simulations. In order to investigate if there is a connection between resource idle time and makespan this part of the experiment was designed.

After selecting the best-suited size penalty for each fitness objective, ten runs are done for each one with both setups tracking the two performance indicators on the validation runs of the evolved rule with the final objective being to compare how the makespan fitness function dispatching rule performs in terms of idle time and vice versa.

With this kind of experiment we hope to shed some light on the viability of using idle time as a cost mechanic to help the scheduler decide in collaborative environments.

3.3.3.3 Terminal sets

The last experimental setup relates to the feature selection technique mentioned in Subsection 3.1.3 with the objective of analysing the impact that a smaller terminal set has on the evolutive process of the GP. With this in mind three additional terminal sets were created by removing terminals perceived as ineffective by the genetic programming algorithm. The terminal sets and their selection criteria are presented in Table 3.4 and described below.

- **GPAll** - This set includes all the terminals used throughout the whole project, and has the aim of giving all available information about the jobs and assembly line state to the GP.

Table 3.4: Different terminal sets used to test Feature selection, terminal description can bee seen in Table 3.1

Terminal	GPAll	LFSMS	FSMS	FSIT
1	x			
10	x			
PT	x			
NPT	x			x
RPT	x	x		
PT/RPT	x	x	x	
QS	x			
ptQ	x	x		
QSNN	x			
nOpsLeft	x			
Now	x	x		
t	x			
Readytime	x			
jt	x	x	x	x
WIQ	x			
WQS	x			x
NPossModes	x	x		x
NPossMachines	x	x	x	
nMach	x	x	x	x
j	x			
AVGQ	x			
AVGWIQ	x	x		
RelSpeed	x			
PossAvgWIQ	x	x	x	x
Sdiff	x			
rSuccPt	x			x
U	x			
t	x	x	x	
USlack	x			

- **FSIT** - are the feature selected terminals for the idle time objective. Due to the small size the best evolved rule (2 to 4 terminals)in each of the ten runs with the fitness metric of idle time, the criteria for this terminal set was for the terminal to appear in at least one of the ten evolved rules

- **FSMS** - represents the feature selected terminal set for the makespan objective. The evolved dispatching rules for the best performing size penalty value with the makespan fitness metric had between 4 and 10 terminals, too keep this terminal set restricted terminals were selected when they appeared on at least 3 of the evolved dispatching rules.

- **LFSMS** - is the large feature selected set for the makespan performance objective. Due

to the relatively large size of the trees in the makespan performance objective mentioned in the previous item, a larger feature selected set was created with the criteria of terminals appearing at least once in one of the ten evolved rules for this fitness metric. The different feature selected sets might also help to ascertain that, if a terminal set is too restricted does it have the opposite effect of the one intended with the technique.

Chapter 4

Computational Results

4.1 Parameters setting

To select the number of training jobs a small experiment was conducted, in this experiment a comparison between a set of (20) training jobs and a set of (10;50) training jobs are compared. The initial setup was to run each set of training jobs with varying GP size penalties (0; 0.001; 0.01; 0.1), the genetic algorithm run ten times for each experiment, ensuring that abnormal cases have their impact minimized.

The training set of (10;50) jobs proved unfeasible with the (0; 0.001; 0.01) size penalties in terms of keeping up with work milestones, this happened with the lower penalty values due to these values allowing for the evolution of larger individual trees. Since the most computationally expensive part of the GP algorithm is the simulation, more specifically the computation of the priority values for all operations in machines queue, a larger tree signifies more calculations to obtain each priority value, thus increasing computation time by a large amount.

Furthermore using the obtained results of both training job sets, with the size penalty of (0.1), and averaging the ten runs by instance, when compared the rules created by the GP with 20 jobs performed 15.8% better on the used metric of makespan.

The calculations were done by averaging the ten values for each instance, then each instance was compared between the two training sets in the following manner $(MS_{10e50} - MS_{20})/MS_{20}$ lastly all the values for these instances were averaged giving a positive result of 15.8%, since the makespan values can't be negative, the positive signal signifies that the value of MS_{20} is lower than the value of MS_{10e50}.

As can be seen on Figure 4.1 the relative makespan difference increases as the number of jobs on the instance increases, which favours the training with the 20 jobs as better suited to produce more generalizable rules.

For the two previously mentioned reasons the rest of the experiments were set up with 20 training jobs.

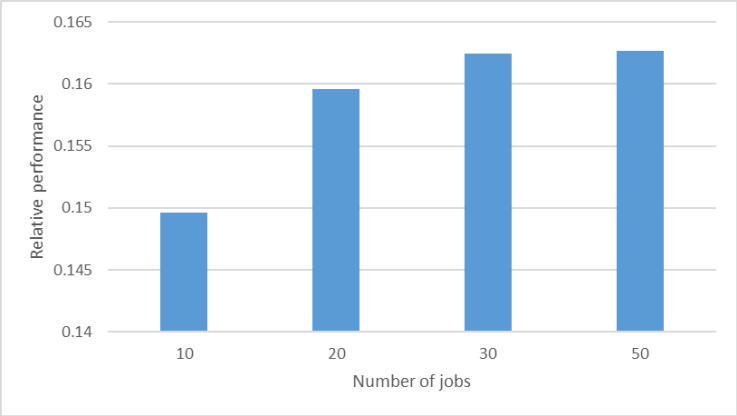

Figure 4.1: Percentual diference between MS_{10e50} and MS_{20} grouped by number of jobs on validation instance (higher means MS_{20} is better)

4.2 Evaluating different GP configurations

To create a performance benchmark the fitness value of the three best rules from Lawrence et al. [4] was calculated in order to select the best performing rule. The most work following (MWF), the least successive difference (LSD) and the longest tail remaining (LTR) rules were executed as the scheduling rule in the validation instances. The MWF rule proved to be the best, as it was in the aforementioned work, with its performance being 27% better when compared to the LTR one and 2% in comparison to the LSD rule. As such MWF will be used through the rest of this dissertation as the fitness value benchmark for each of the instances.

4.2.1 Size penalty

The results of the size penalty experiments are compiled in Figure 4.2, the penalty of 0.01 outperforms the penalties of 0, 0.001 and 0.1 by 6%, 1% and 3% respectively. On Table 4.1 the individual performance gains in comparison to the benchmark are shown, along with how many times the average score of the evolved dispatching rules was better than the MWF benchmark, it is interesting to note that the GP with the penalty of 0.01 managed to get a lower makespan than the benchmark on every instance. The poor performance of penalty 0 can be attributed to the big size of its individuals, these trees become so big with the insertion of random nodes through mutation that it dilutes the performance impact of meaningful terminals.

Table 4.1: Penalties performance in comparison to MWF benchmark rule

Penalty	0.1	0.01	0.001	0
Performance vs. benchmark	+3.9%	+6.2%	5.3%	0.49%
Won vs. benchmark	730	768	729	387
Lost vs. benchmark	38	0	39	381

In terms of the generated individuals' size (number of nodes in the tree) the outcomes were: 3 to 7 rules for 0.1, 4 to 10 for 0.01, 5 to 77 for 0.001 and 102 to 217 for the penalty of 0. This extreme discrepancy in the rules' size led to the experiments with the two lowest penalties taking more than three times longer to run when compared to the two highest penalty ones.

Even with the small performance difference between the two best penalties, the pick of 0.01 can be reinforced by the fact that it produces smaller rules than 0.001, which makes the solution more understandable and as mentioned reduces the computational cost of the evolution process.

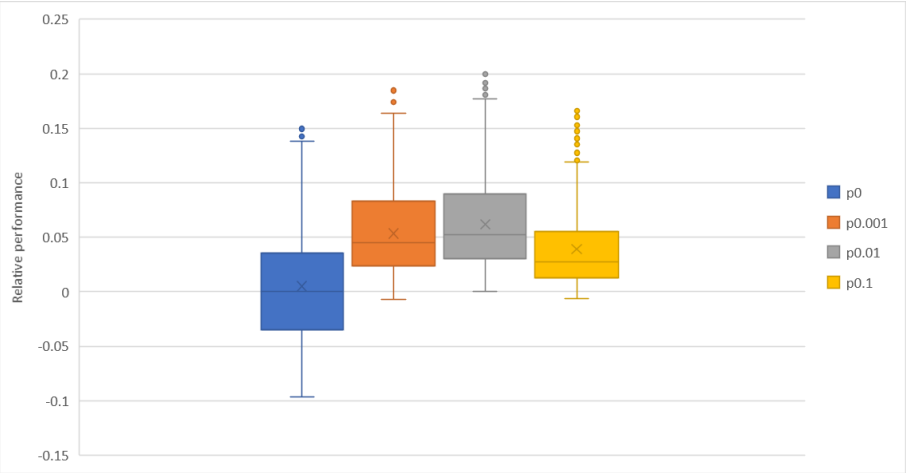

Figure 4.2: Comparison of performances size penalties for makespan, relative performance evaluated in comparison to benchmark rule

In the tests for the fitness value calculated with idle time, the best performing penalty was 0.1, with a 17% improvement over 0.01 and 29% over the 0.001 penalty values. The average size of the evolved rules was 5 for 0.1, 13 for 0.01 and 79 for 0.01 This difference can be explained by the need of less complex dispatching rules in order to obtain a good performance in the idle time objective.

4.2.2 Feature selection

To assess the impact of different size terminal sets in the genetic programming algorithm the experiment in Subsection 3.3.3.3 was executed, with the results presented in this section. Relating to makespan the average performance result of the three terminal set instances (GPAll, LFSMS, FSMS) was very similar with a verified maximum difference of 0.1% being insignificant.

The most impactful difference relates to the evolution of the best rule's fitness value of each generation, as can bee seen in Figure 4.3. This image shows us that although the best rules for all three sets at the end of the evolutive process have similar performance, the sets in which feature selection was implemented arrive at the best performing solution in a much earlier generation.

This improvement can be explained by the reduced search space for the evolutive processes in the algorithms with the feature selected terminal sets. When used in setups developed for

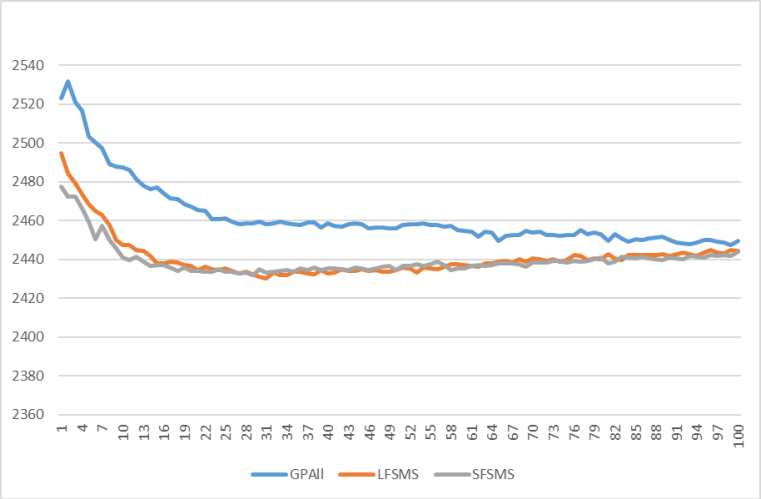

Figure 4.3: Evolution of the average population fitness by generation, performance evaluated through makespan

them, these reduced terminal sets will have the advantage of needing a lower number of maximum generations, thus achieving a good performing solution while requiring only fraction of the computational power when compared to larger terminal sets that contain unimportant information.

The setup for the idle time objective had similar results, shown in Figure 4.4, with the performance of the evolved dispatching rule being 1% better for the instance with no feature selection. As in the makespan case, the feature selected terminal set achieves better performing individuals in earlier generations.

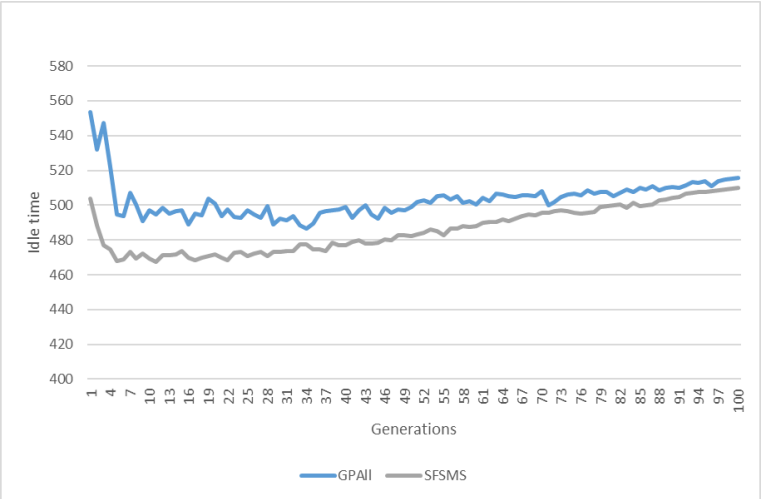

Figure 4.4: Evolution of the average population fitness by generation, idle time as fitness metric

The slight fitness increase after stabilization that is observable in figures 4.3 and 4.4 can be attributed to the size penalty increasing as generations pass.

4.3 Main results

4.3.1 GP vs Literature

From the 30 independent GP runs of penalty 0.01 realized with the various terminal sets, the average makespan of the evolved rules was 6.2% better than the MWF rule. This value is obtained by firstly averaging the makespan value of the 30 runs for each instance, then the averaged GP instance performance and the MWF performance on the same instance are compared, this gives us a relative performance value for each instance. When we average the relative performance gains of all instances we obtain the above mentioned 6.2%.

Table 4.2: The three best evolved rules with the makespan objective, performance in comparison to benchmark rule

Gp instance	Run	Dispatching rule	Size	Perf
GPAll	3	$nPossMachs^4 * PT^2/(RPT * jt)$	13	+7%
LFSMS	5	$nPossMachs^2 * NPossModes * PT^2/(PtQ * jt * RPT)$	13	+6.8%
LFSMS	1	$nPossMachs^3 * PT^2/(RPT * jt)$	11	+6.7%

On Table 4.2 the three best performing rules of all terminal sets are presented, the operation chosen to be performed next corresponds to the lowest value outputted by the dispatching rule. The rules all have similar size and relative performance, but most interestingly they all display some key terminals in comparable positions on the equation:

- *nPossMachs* and *NPossModes*, these terminals makes the scheduler prioritize operations that can be processed on a smaller number of resources, thus balancing the workload by scheduling first the operations that create the biggest imbalance in queue size.

- *PT*, this terminal is equivalent to the shortest processing time rule, which selects for processing jobs with the smallest processing time. This rule has been shown to be a good policy for reducing makespan in stochastic parallel machine related problems by Pinedo et al. [33], with its good performance in dynamic environments being reinforced by the results shown in Lawrence et al. [4].

- PT/RPT is equivalent to the rule known in the literature as the shortest processing time by total remaining work, and has been shown to have good performance in some makespan related problems such as the one in Nguyen et al. [34]

- $1/jt$ with the use of this terminal the dispatching rule gives priority to operations that have been in the queue for the longest time, in problems with precedence constraints this rule has been shown to aid in makespan minimization by releasing successor operations earlier to other resources (Frostig et al. [35], Pinedo et al. [33])

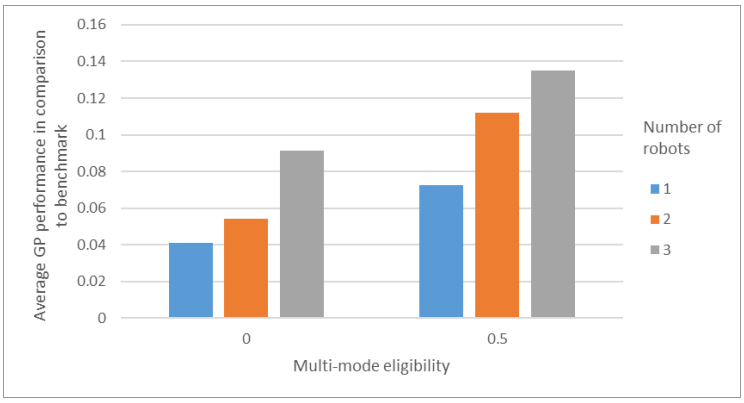

Figure 4.5: Makespan performance improvement in comparison to the benchmark rule (MWF)

The Figure 4.5 shows the average performance improvement of the GP evolved rules when compared to the MWF rule, $(Perf_{MWF} - Perf_{GP})/-Perf_{GP}$. The two most significant factors in performance variation are multi-mode eligibility and number of robots. When multi-mode eligibility is 0.5 the number of possible ways to schedule operations increases and that makes the problem more complex along with decreasing the makespan lower bound of the solution. The addition of more robot resources also increases the number of possible schedules.

With the increase in possible schedules, the possible makespan gap between the boundaries increases consequently allowing for the GP to display a higher relative performance. This performance enhancement could be even higher if the instance sets of the GP were more restricted in terms of instance parameters, such as GP applications to more specific scenarios.

4.3.2 Idle Time vs Makespan

When assessing the rules evolved with the objectives of makespan and idle time in each others' metrics the first apparent result was an uncorrelation between the two with the genetic programming dispatching rules developed. The performance objective of makespan performed 80% worse in the tests with the fitness calculated through idle time and the rules evolved with the idle time objective performing 26% worse in the makespan tests when compared to the objective specific evolved rules.

These results are presented in Figure 4.6, with the relative performances for both tests calculated by for each instance subtracting the makespan's rules performance to the idle time rules' performance and then dividing by the makespan $(P_{MSDR} - P_{ITDR})/P_{MSDR}$. Since both fitness metrics have the aim of minimization, the higher a percentage value is the better the makespan dispatching rules performed and negative values mean the idle dispatching rules performed better.

The three best evolved rules with the idle time objective are presented in Table 4.3, the operation with the highest priority corresponds to the lowest score outputted by the dispatching rule. On these dispatching rules three terminals seem to have some significant impact for a good performance in the idle time objective:

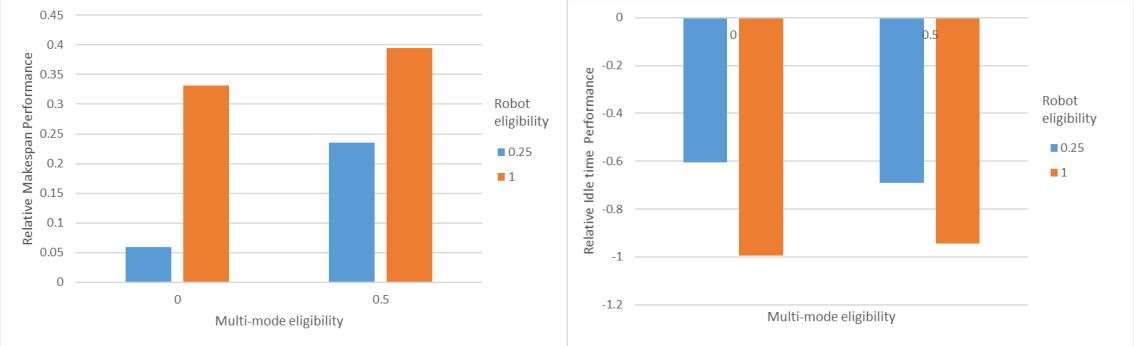

Figure 4.6: Cross comparison of rules evolved with objective functions of makespan and idle time in both objectives

Table 4.3: Three best evolved rules with the idle time objective

Gp instance	Run	Dispatching rule	Size
GPAll	6	$nMachsInMode - WQS * rSuccPt$	5
FSIT	3	$PossAvgWIQ + NPossModes - WQS$	5
FSIT	9	$PossAvgWIQ - WQS - rSuccPt$	5

- *PossAvgWIQ* represents the average work in the queue of the eligible resources for a given operation, by using this terminal the rule gives priority to operations that have shorter queues in their eligible machines. This may seem counter-intuitive but taking into account that the idle time only counts between operations and not when a resource has performed all operations and stops until the end of the simulation, this terminal makes it so that, in the instances where robot eligibility is 0.25, the robots finish all their operations earlier and with little to no idle time. Leaving the operations only eligible to humans for them to perform.

- $-WQS$, with this terminal the dispatching rule makes use of the negative work in queue of the eligible resources for the direct successor operations, thus prioritizing operations that will fill the queues of the human resources, this can be explained by the logic mentioned in the previous terminal.

- $-rSuccPt$, the use of the negative remaining processing time on the job, gives a higher priority value to operations belonging to jobs that have a higher amount of processing time remaining. By doing this the dispatching rule avoids situations where there are a lot of operations remaining to process but they belong to a reduced number of jobs which can cause wait times due to operation precedence constraints inside jobs. With the prioritization of jobs that have a longer remaining processing time, the rule ensures that the operations left at the later stages of the simulation are part of a varied number of jobs thus reducing possible precedence constraints.

The instance set parameters that have a significant impact in the performance correlation are multi-mode eligibility and robot eligibility, with the idle time DRs performing only 6% worse in the makespan performance metric when multi-mode eligibility is 0 and robot eligibility is 0.25. On the other hand makespan rules perform especially poorly on the idle time fitness when robot eligibility is 1.

The differences in performance seen in Figure 4.6 on the rightmost graph can be explained by the makespan rules sometimes making resources wait to perform certain operations in collaborative mode, which happens more often when the multi-mode eligibility is higher or when there are more robots eligible to perform that task.

The performance disparities presented on the graph on the left can be justified by the fact that on almost every occasion one resource will have to wait for another to become available in order to perform an operation in collaborative mode. As the idle time dispatching rules aim to eliminate wait times, they sometimes may choose slower processing modes which negatively impact makespan. An evidence of this is the presence of the terminal *nMachsInMode* (GPAll, Rule 6) in the Table 4.3, that makes the rule select modes with a lower number of resources.

With this information, we can say that for some instances idle time is not a good indicator for makespan nor do they have a general correlation. So in general cases with the objective of optimising makespan, idle time shouldn't immediately be applied as the primary scheduling factor without considering the specific instances at hand.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

By using genetic programming we were able to evolve several good performing dispatching rules in a number of distinct setups. These setups differed from each other in the terminals that were available to the GP algorithm, the instance set characteristics, the GP parameters and the desired performance objective. By analysing and comparing the results obtained in the aforementioned experiments some important conclusions may be extracted:

- Genetic programming parameters play an important role in developing a population that is varied enough and thus obtaining a good solution. These parameters should be tuned for the desired instance sets, with the individual's size penalty value depending heavily on the desired fitness metric and having a large influence not only in the GP performance but also on its computational cost .

- Some instance parameters highly influence the performance of the GP evolved solution, in these cases evolving different dispatching rules for different instance subsets should result in a relative performance improvement.

- The use of feature selection in the GP's terminals might not improve solution performance, but its use drastically reduces the number of generations needed to achieve a solution with a comparable performance to a non feature selected terminal set. If this is taken into account when designing the GP the computational cost needed to evolve the dispatching rules can be reduced by a large amount.

- The dispatching rules evolved in this work were able to outperform the benchmark in all instances, with the benefit of the understandability of its inner workings.

- Although idle time is used in some cases as an indicator or even as a mean to reduce makespan, depending on the used instance this correlation might not always be true.

5.2 Future work

On the feature selection case, future developments could pursue the use of wrapper approaches that estimate the significance of terminals during the evolution process and flexibly select those terminals in order to evolve more simple and significant individuals. One other route to be pursued is the division of a larger instance set into subsets according to certain significant parameters with the aim of evolving and validating GP designed rules in more specific cases, which is expected to achieve better results. Lastly, a more in-depth research should be done to determine what instance parameters decrease or increase the correlation between makespan and idle time.

Appendix A

Instance sets

Table A.1: Instance set used for GP training.

I	nH	nR	minPT	maxPT	minOps	maxops	RElig	HElig	RPerf	mMod
9	1	2	1	100	2	14	0.25	1	0.5	0
10	1	2	1	100	2	14	0.25	1	0.5	0.5
11	1	2	1	100	2	14	0.25	1	2	0
12	1	2	1	100	2	14	0.25	1	2	0.5
13	1	2	1	100	2	14	1	1	0.5	0
14	1	2	1	100	2	14	1	1	0.5	0.5
15	1	2	1	100	2	14	1	1	2	0
16	1	2	1	100	2	14	1	1	2	0.5
25	2	1	1	100	2	14	0.25	1	0.5	0
26	2	1	1	100	2	14	0.25	1	0.5	0.5
27	2	1	1	100	2	14	0.25	1	2	0
28	2	1	1	100	2	14	0.25	1	2	0.5
29	2	1	1	100	2	14	1	1	0.5	0
30	2	1	1	100	2	14	1	1	0.5	0.5
31	2	1	1	100	2	14	1	1	2	0
32	2	1	1	100	2	14	1	1	2	0.5

Table A.2: Instance set used for GP validation.

I	nH	nR	minPT	maxPT	minOps	maxops	RElig	HElig	RPerf	mMod
1	1	1	1	100	2	14	0.25	1	0.5	0
2	1	1	1	100	2	14	0.25	1	0.5	0.5
3	1	1	1	100	2	14	0.25	1	2	0
4	1	1	1	100	2	14	0.25	1	2	0.5
5	1	1	1	100	2	14	1	1	0.5	0
6	1	1	1	100	2	14	1	1	0.5	0.5
7	1	1	1	100	2	14	1	1	2	0
8	1	1	1	100	2	14	1	1	2	0.5
9	1	2	1	100	2	14	0.25	1	0.5	0
10	1	2	1	100	2	14	0.25	1	0.5	0.5
11	1	2	1	100	2	14	0.25	1	2	0
12	1	2	1	100	2	14	0.25	1	2	0.5
13	1	2	1	100	2	14	1	1	0.5	0
14	1	2	1	100	2	14	1	1	0.5	0.5
15	1	2	1	100	2	14	1	1	2	0
16	1	2	1	100	2	14	1	1	2	0.5
17	1	3	1	100	2	14	0.25	1	0.5	0
18	1	3	1	100	2	14	0.25	1	0.5	0.5
19	1	3	1	100	2	14	0.25	1	2	0
20	1	3	1	100	2	14	0.25	1	2	0.5
21	1	3	1	100	2	14	1	1	0.5	0
22	1	3	1	100	2	14	1	1	0.5	0.5
23	1	3	1	100	2	14	1	1	2	0
24	1	3	1	100	2	14	1	1	2	0.5
25	2	1	1	100	2	14	0.25	1	0.5	0
26	2	1	1	100	2	14	0.25	1	0.5	0.5
27	2	1	1	100	2	14	0.25	1	2	0
28	2	1	1	100	2	14	0.25	1	2	0.5
29	2	1	1	100	2	14	1	1	0.5	0
30	2	1	1	100	2	14	1	1	0.5	0.5
31	2	1	1	100	2	14	1	1	2	0
32	2	1	1	100	2	14	1	1	2	0.5

References

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

[2] Eloise Matheson, Riccardo Minto, Emanuele G G Zampieri, Maurizio Faccio, and Giulio Rosati. Human–Robot Collaboration in Manufacturing Applications: A Review. *Robotics*, 8(4):100, dec 2019. URL: https://www.mdpi.com/2218-6581/8/4/100, doi:10.3390/robotics8040100.

[3] Djamila Ouelhadj, Sanja Petrovic, D Ouelhadj, · S Petrovic, and S Petrovic. A survey of dynamic scheduling in manufacturing systems. *J Sched*, 12:417–431, 2009. doi:10.1007/s10951-008-0090-8.

[4] Stephen R. Lawrence and Edward C. Sewell. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. *Journal of Operations Managementem*, 15(1):71–82, 1997. doi:10.1016/s0272-6963(96)00090-3.

[5] Andrea Casalino, Andrea Maria Zanchettin, Luigi Piroddi, and Paolo Rocco. Optimal Scheduling of Human-Robot Collaborative Assembly Operations With Time Petri Nets. *IEEE Transactions on Automation Science and Engineering*, pages 1–15, 2019. doi:10.1109/TASE.2019.2932150.

[6] Cristiane Ferreira, Gonçalo Figueira, and Pedro Amorim. Optimizing Dispatching Rules for Stochastic Job Shop Scheduling. *Advances in Intelligent Systems and Computing*, 923:321–330, 2020. doi:10.1007/978-3-030-14347-3_31.

[7] J. Krüger, T. K. Lien, and A. Verl. Cooperation of human and machines in assembly lines. *CIRP Annals - Manufacturing Technology*, 58(2):628–646, 2009. doi:10.1016/j.cirp.2009.09.009.

[8] H. Bley, G. Reinhart, G. Seliger, M. Bernardi, and T. Korne. Appropriate human involvement in assembly and disassembly. *CIRP Annals - Manufacturing Technology*, 53(2):487–509, 2004. doi:10.1016/S0007-8506(07)60026-2.

[9] J. Krüger, R. Bernhardt, and D. Surdilovic. Intelligent assist systems for flexible assembly. *CIRP Annals - Manufacturing Technology*, 55(1):29–32, 2006. doi:10.1016/S0007-8506(07)60359-X.

[10] Rainer Müller, Matthias Vette, and Aaron Geenen. Skill-based Dynamic Task Allocation in Human-Robot-Cooperation with the Example of Welding Application. *Procedia Manufacturing*, 11(June):13–21, 2017. URL: http://dx.doi.org/10.1016/j.promfg.2017.07.113, doi:10.1016/j.promfg.2017.07.113.

[11] Andrea Maria Zanchettin, Andrea Casalino, Luigi Piroddi, and Paolo Rocco. Prediction of Human Activity Patterns for Human-Robot Collaborative Assembly Tasks. *IEEE Transactions on Industrial Informatics*, 15(7):3934–3942, 2019. doi:10.1109/TII.2018.2882741.

[12] Luka Peternel, Nikos Tsagarakis, Darwin Caldwell, and Arash Ajoudani. Adaptation of robot physical behaviour to human fatigue in human-robot co-manipulation. In *IEEE-RAS International Conference on Humanoid Robots*, pages 489–494. IEEE Computer Society, dec 2016. doi:10.1109/HUMANOIDS.2016.7803320.

[13] Panagiota Tsarouchi, Alexandros-Stereos Matthaiakis, Sotiris Makris, and George Chryssolouris. On a human-robot collaboration in an assembly cell. *International Journal of Computer Integrated Manufacturing*, 30(6):580–589, 2017. URL: https://www.tandfonline.com/action/journalInformation?journalCode=tcim20, doi:10.1080/0951192X.2016.1187297.

[14] Nikolaos Nikolakis, Niki Kousi, George Michalos, and Sotiris Makris. Dynamic scheduling of shared human-robot manufacturing operations. *Procedia CIRP*, 72:9–14, 2018. URL: https://doi.org/10.1016/j.procir.2018.04.007, doi:10.1016/j.procir.2018.04.007.

[15] Michael L. Pinedo. *Scheduling: Theory, algorithms, and systems*. Springer-Verlag, 2008. doi:10.1007/978-0-387-78935-4.

[16] Kourosh Darvish, Francesco Wanderlingh, Barbara Bruno, Enrico Simetti, Fulvio Mastrogiovanni, and Giuseppe Casalino. Flexible human–robot cooperation models for assisted shop-floor tasks. *Mechatronics*, 51(March):97–114, 2018. URL: https://doi.org/10.1016/j.mechatronics.2018.03.006, arXiv:1707.02591, doi:10.1016/j.mechatronics.2018.03.006.

[17] Fei Chen, Kosuke Sekiyama, Ferdinando Cannella, and Toshio Fukuda. Optimal subtask allocation for human and robot collaboration within hybrid assembly system. *IEEE Transactions on Automation Science and Engineering*, 11(4):1065–1075, 2014. doi:10.1109/TASE.2013.2274099.

[18] Lingxiao Wu and Shuaian Wang. Exact and heuristic methods to solve the parallel machine scheduling problem with multi-processor tasks. *International Journal of Production Economics*, 201(February):26–40, 2018. URL: https://doi.org/10.1016/j.ijpe.2018.04.013, doi:10.1016/j.ijpe.2018.04.013.

[19] Matthew Gombolay, Ronald Wilcox, and Julie Shah. Fast Scheduling of Multi-Robot Teams with Temporospatial Constraints. In *Robotics: Science and Systems IX*. Robotics: Science and Systems Foundation, jun 2013. URL: http://www.roboticsproceedings.org/rss09/p49.pdf, doi:10.15607/RSS.2013.IX.049.

[20] Haldun Aytug, Mark A. Lawley, Kenneth McKay, Shantha Mohan, and Reha Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. In *European Journal of Operational Research*, volume 161, pages 86–110. North-Holland, feb 2005. doi:10.1016/j.ejor.2003.08.027.

[21] Ping Lou, Quan Liu, Zude Zhou, Huaiqing Wang, and Sherry Xiaoyun Sun. Multi-agent-based proactive–reactive scheduling for a job shop. *The International Journal of Advanced Manufacturing Technology*, 59(1-4):311–324, mar 2012.

URL: http://link.springer.com/10.1007/s00170-011-3482-4, doi:10.1007/s00170-011-3482-4.

[22] Guilherme E. Vieira, Jeffrey W. Herrmann, and Edward Lin. Predicting the performance of rescheduling strategies for parallel machine systems. *Journal of Manufacturing Systems*, 19(4):256–266, jan 2000. doi:10.1016/S0278-6125(01)80005-4.

[23] Huiqiao Su, Michael Pinedo, and Guohua Wan. Parallel machine scheduling with eligibility constraints: A composite dispatching rule to minimize total weighted tardiness. *Naval Research Logistics (NRL)*, 64(3):249–267, apr 2017. URL: http://www.interscience.wiley.com/jpages/0894-069X/ http://doi.wiley.com/10.1002/nav.21744, doi:10.1002/nav.21744.

[24] Su Nguyen, Yi Mei, and Mengjie Zhang. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems*, 3(1):41–66, 2017. doi:10.1007/s40747-017-0036-x.

[25] Christoph W. Pickardt, Torsten Hildebrandt, Jürgen Branke, Jens Heger, and Bernd Scholz-Reiter. Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. In *International Journal of Production Economics*, volume 145, pages 67–77. Elsevier, sep 2013. doi:10.1016/j.ijpe.2012.10.016.

[26] Domagoj Jakobović and Leo Budin. Dynamic Scheduling with Genetic Programming. In *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin Heidelberg, 2006. URL: http://link.springer.com/10.1007/11729976{_}7, doi:10.1007/11729976_7.

[27] Maciej Drozdowski. Scheduling multiprocessor tasks - An overview. *European Journal of Operational Research*, 94(2):215–230, 1996. doi:10.1016/0377-2217(96)00123-3.

[28] Edmund K. Burke, Mathew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and John R. Woodward. Exploring hyper-heuristic methodologies with genetic programming, 2009. URL: https://link.springer.com/chapter/10.1007/978-3-642-01799-5{_}6, doi:10.1007/978-3-642-01799-5_6.

[29] Bing Xue, Mengjie Zhang, Will N. Browne, and Xin Yao. A Survey on Evolutionary Computation Approaches to Feature Selection. *IEEE Transactions on Evolutionary Computation*, 20(4):606–626, aug 2016. doi:10.1109/TEVC.2015.2504420.

[30] Yi Mei, Mengjie Zhang, and Su Nyugen. Feature selection in evolving job shop dispatching rules with Genetic Programming. *GECCO 2016 - Proceedings of the 2016 Genetic and Evolutionary Computation Conference*, pages 365–372, 2016. doi:10.1145/2908812.2908822.

[31] Anna Friedlander, Kourosh Neshatian, and Mengjie Zhang. Meta-learning and feature ranking using genetic programming for classification: Variable terminal weighting. *2011 IEEE Congress of Evolutionary Computation, CEC 2011*, pages 941–948, 2011. doi:10.1109/CEC.2011.5949719.

[32] Andrea Casalino, Filippo Cividini, Andrea Maria Zanchettin, Luigi Piroddi, and Paolo Rocco. Human-robot collaborative assembly: a use-case application. *IFAC-PapersOnLine*, 51(11):194–199, 2018. URL: http://blog.robotiq.com/

collaborative-robot-ebook.https://linkinghub.elsevier.com/retrieve/pii/S2405896318313818, doi:10.1016/j.ifacol.2018.08.257.

[33] Michael Pinedo and Gideon Weiss. Scheduling Jobs With Exponentially Distributed Processing Times and Intree Precedence Constraints on Two Parallel Machines. *Operations Research*, 33(6):1381–1388, 1985. doi:10.1287/opre.33.6.1381.

[34] Su Nguyen, Mengjie Zhang, Mark Johnston, and Kay Chen Tan. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 17(5):621–639, 2013. doi:10.1109/TEVC.2012.2227326.

[35] Esther Frostig. Stochastic scheduling problem with intree precedence constraints. *Operations Research*, 36(6):937–943, 1988. doi:10.1287/opre.36.6.937.