FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Using Dimensionally Aware Genetic Programming to find interpretable Dispatching Rules for the Job Shop Scheduling Problem

Álvaro Manuel Festas Pereira da Silva

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Luís Gonçalo Rodrigues Reis Figueira

Co-Supervisor: Cristiane Maria Santos Ferreira

17 de julho de 2021

Resumo

As regras de despacho (DRs) têm sido usadas em várias aplicações nos sistemas de produção. Elas podem ser expressões matemáticas que têm como parâmetros várias caraterísticas dos trabalhos a serem escalonados, e atribuem uma prioridade a esses trabalhos. Esta atribuição permite escolher qual o próximo trabalho a ser executado. Como criar uma boa regra de despacho é um trabalho complexo, a programação genética (GP) tem sido usada para o fazer. Na GP, várias DRs são desenvolvidas simultaneamente e, através de processos inspirados na natureza (como mutações e *crossovers*), melhoram a sua *performance*. No entanto, na literatura poucos trabalhos tentaram alcançar DRs interpretáveis, sendo muitas vezes os seus tamanhos ignorados. Normalmente as DRs tendem a ser muito longas com cem ou mais termos. O problema concreto que será abordado é a versão dinâmica do *job shop scheduling problem* (DJSS).

Este trabalho utilizou algoritmos baseados em *context-free grammars* (CFG), nomeadamente CFG-GP e *grammar evolution* (GE), para atingir DRs *dimensional aware (DA)*, o que restringe a *syntaxe* das DRs. Por exemplo, numa DR que é *dimensional aware* a adição só pode ser feita entre operandos com as mesmas unidades. Ambos os algoritmos foram testados por terem caraterísticas distintas e nunca terem sido usados para resolver este problema.

O objetivo foi que, ao forçar a sintaxe das DRs a estar correta, seja possível gerar DRs menores e mais interpretáveis, de preferência sem perda de *performance*.

Além disso, foi feito um enumerador que, ao enumerar todas as DRs encontrou as melhores para os tamanhos em que as conseguimos enumerar. Os resultados do enumerador servirão para saber se os algoritmos evolutivos conseguem explorar as pequenas DRs de uma forma ótima ou não, mais ainda, se a resposta for negativa, encontrará melhores DRs do que os algoritmos e trará garantias de otimalidade.

Os resultados mostram uma melhoria significativa de desempenho ao usar métodos do tipo *dimensional aware genetic programming* (DAGP) face a GP normal para este problema. Além disso GP/GE e CFG-GP conseguiram explorar de forma ótima ou quase os espaços que enumeramos, encontrando as melhores DRs.

Concluímos que as DRs geradas pelos métodos DA, em particular CFG-GP, têm uma performance significativamente melhores que as produzidas por GP normal.

Palavras-chave: Regras de despacho, Programação genética, dimensionalmente consistente

Abstract

Dispatching Rules (DRs) have been used in several applications in manufacturing systems. They can be expressions that assign priority to jobs in a machine queue, choosing the next one to be executed. As they are challenging to design, genetic programming (GP) is being used to find better performative DRs. In GP, several different DRs are evolved, and due to some operations and selection processes inspired in nature, the DRs tend to improve. However, little research has been done in trying to reach small and interpretable DRs. Usually, these generated expressions tend to become extremely large, with a couple hundred terms or more.

This work will use context-free grammar (CFG) algorithms, particularly CFG-GP and grammar evolution (GE), for reaching DRs which are dimensionally aware (DA), restricting the syntax of the DRs (e.g. additions can only be made between operands with the same units). These algorithms will be used to solve the dynamic job shop scheduling problem (DJJS), which were never used to solve it.

The objective is that by forcing the syntax of the DRs to be correct, it will be possible to reach smaller and more interpretable DRs without performance loss.

Furthermore, an enumerator was made that found the best possible small DRs, which will serve as a baseline to evaluate how well the different algorithms can explore these spaces and give the best possible DRs for the sizes we could enumerate. Moreover, as the enumerator passes through all DA dispatching rules, it will give optimality guarantees.

The results show a significant performance improvement in using DA methods for this problem. Moreover, GP/GE and CFG-GP can explore the small DRs optimally or close to optimally, managing to find the best small DRs.

We conclude that the DRs generated with DA methods, particularly CFG-GP, had significantly better performance than standard GP for DJJS.

keywords: Dispatching Rules, Genetic Programming, Dimensionally Aware

Agradecimentos

Esta dissertação de Mestrado não poderia ter sido realizada sem o contributo direto e indireto de várias pessoas. Expresso então os meus agradecimentos:

- Ao meu orientador, Professor Doutor Gonçalo Figueira, que me deu contributos valiossímos sobre como investigar, mostrando-se sempre disponível para responder às minhas questões e dúvidas durante todo o processo.
- À minha co-orientadora, Doutora Cristiane Ferreira, pela sua disponibilidade em me ajudar a entender e usar as bibliotecas de simulação e por me motivar a tentar explorar novas dimensões do problema.
- A todos os meus colegas da faculdade, com os quais passei momentos fantásticos. As conversas com eles permitiram-me aproveitar o tempo da faculdade de uma forma muito mais divertida e discutir as aulas com eles sempre foi a melhor forma de as conseguir entender. Particularizo o Bruno Maurício, o David Cunha, o Tiago Sousa e o Tiago Cunha que me acompanharam desde o início até ao fim do curso. Refiro o meu colega Pedro Machado que foi o colega a quem mais duvidas perguntei (obrigado, Pedro, e desculpa por ter sido um chato) e por último o meu colega Eduardo Fonseca que, apesar de termos escolhido ramos diferentes, permanecemos sempre em contacto. Aos colegas que não inumerei fique-se sabendo que sempre me marcaram e vos vou ter presentes, contudo infelizmente a página tem um comprimento finito.
- Ao Núcleo de Estudantes de Engenharia Eletrotécnica e de Computadores (NEEEC) onde fui membro durante o curso todo. Permitiu-me participar em inúmeras atividades e contactar com inúmeras pessoas e aprender fazendo.
- A todos os meus professores que me acompanharam no decorrer destes anos, tanto no ensino superior como no obrigatório. Deixo um especial agradecimento à minha professora de matemática do ensino básico, a professora Fátima Pedroso, por me ter ajudado a descobrir e a desenvolver a minha paixão por números.
- A todos os voluntários da comunidade vicentina de São Vicente de Paulo, da qual sou membro desde há 7 anos. Conviver com o outro, independente da sua posição na vida, ouvindo-o, é um exercicío constante de humildade que me fez desenvolver muito como pessoa.
- À minha namorada Beatriz Almeida que me acompanhou nos últimos anos e tem sido uma presença essencial na minha vida. Ela arranja sempre novas formas de me fazer sorrir.
- Aos meus melhores amigos Rui, Bruno e Leonardo que, mesmo quando afastados por contratempos, como pandemias ou estudos, marcamos sempre atividades para fazer para consolidar esta amizade de longa data.

- Aos meus primos/primas, tios/tias, avôs/avós. Em especial a minha madrinha Clarinda Festas que me ajudou a rever este documento.
- Às minhas irmâs Luísa e Isabel que estão sempre lá para mim, sendo as pessoas com quem passei mais tempo que contribuiram ativamente para todos os momentos serem ótimos.
- Aos meus pais, Álvaro Silva e Maria José, aos quais devo tudo e pelos quais tenho enorme estima, que me moldaram na pessoa que sou hoje.

A todos os meus sinceros agradecimentos

Álvaro Manuel Festas Pereira da Silva

"Success is walking from failure to failure
with no loss of enthusiasm."

Winston Churchill

Conteúdo

Lista de Figuras

Lista de Tabelas

Abbreviations and symbols

A^+/A	Allowance if positive
CFG	Context-free grammar
CFG-GP	Context-free grammar-based genetic programming
CFG-GP-PTC2	CFG-GP using PTC2 as initiator
CFG-GP-RND	CFG-GP using a random algorithm as initiator
CR+/C	Critical ratio if positive
DA	Dimensionally aware
DAGP	Dimensionally aware genetic programming
DJSS	Dynamic version of the job shop scheduling problem
DRs	Dispatching rules / Regras de Despacho
DTAG3P	Developmental tree adjoining grammar-guided genetic programming
EC	Evolutionary computation
ECJ	Java evolutionary computation toolkit
FIFO	First in first out
GE	Grammatical evolution
GE-BAL	GE applied to a balanced grammar
GE-BIG-19	Extended grammar creates DRs between sizes 9-19
GE-BIG-33	Extended grammar creates DRs between sizes 9-33
GE-NBAL	GE applied to a non balanced grammar
GGGP	Grammar-guided genetic programming
GP	Genetic Programming / Programação Genética
GP-AT	GP using only atomic terminals
GP-CR	GP using CR as a terminal
HGE	Hierarchical grammatical evolution
JSS	Job shop scheduling problem
LIFO	Last in first out
NPT/N	Processing time of next operation
O_1 and O_2	Unary and binary operators sets
STGP	Strongly typed genetic programming
PTC2	Initiation algorithm for GGGP
RR	DR by Raghu and Rajendran
RPT/R	Remaining processing time of a job
S^+/S	Slack if positive
SGE	Structured grammatical evolution
SG-GP	Stochastic grammar-based genetic programming
PEEL	Program evolution with explicit learning
PT/P	Processing time
TAG	Tree adjoining grammars
TAG3P	Tree adjoining grammar-guided genetic programming
TWT	Total work content
WHGE	Weighted HGE
WINQ/WinRR/W	Work in next queue
U	Utilization

Capítulo 1

Introduction

Artificial Intelligence has a growing impact on society, having diverse applications from recommendation systems to speech recognition, self-driving cars, to mention a few. However, due to the performance of black-box models, such as neural networks or tree ensembles, a central aspect of the models is neglected: how can a human trust the model outputs if he cannot understand them. One way to make humans understand AI is to use explainable by design models rather than black-box models. Simpler models exist with these characteristics, such as linear models or decision trees, which are simple enough to be understood by their audience. Nonetheless, these simple models come with a significant loss of performance. To change this paradigm and create high-performing and explainable models, Genetic Programming (GP) is a promising tool.

GP is based on natural selection. It maintains a population throughout generations, and in each generation, ranks its expressions (relative to some fitness metric) and selects the best ones (using some selection procedure). Afterwards, GP generates the next population by applying genetic operators to them (e.g. mutations and crossover). This process is repeated until the termination condition is meet (e.g. the maximum number of generations was reached).

However, GP has its problems. The solution space is quite vast as GP has to learn the structure of the expressions and their constituents. This complexity significantly reduces the number of inputs that these algorithms can successfully manage. Also, an effort must be made to keep the solutions small enough to be understandable. No one can understand formulas with hundreds of terms, and they tend to be prone to overfitting. Furthermore, these expressions can have dimensional inconsistencies (e.g. standard GP can evolve an expression that adds meters with time, which is hard to interpret).

One method that tries to diminish the solution space of GP, improving its interpretability, is Dimensionally Aware Genetic Programming (DAGP). In DAGP, the expressions are dimensionally consistent, which means that the expressions take an equivalent form in all systems of units (e.g. in a dimensionally consistent expression, addition is more restricted, only being allowed between operands with the same units). These additional restrictions reduce the number of valid expressions reducing the solution space, and as dimension inconsistencies do not appear, it gives extra interpretability. However, a possible downside of this approach is that if the best possible expression is not dimensionally aware, it will never find it, and its performance can be worse than standard GP.

Furthermore, DAGP can be implemented in several different ways using soft or hard constraints. Soft constraints interpret the original problem as a dual optimisation problem between performance and dimensionality consistency. Here a non-dimensionally consistent expression may exist and is penalised. In contrast, hard constraints make it impossible for non-dimensionally consistent expressions to exist. Therefore, it is only by using hard constraints that the solution space is diminished.

Additionally, even hard constraints have alternative implementations. One option is restricting the operand types a GP node admits, called Strongly-Typed Genetic Programming (STGP). Another is using Grammar-based Genetic Programming (GGGP), which can use a grammar to enforce dimensionally consistent expressions.

A classic optimisation problem that has gained much interest in the GP community in the last couple of decades, with several authors using GP to solve it, is the Job Shop Scheduling Problem (JSS). In JSS, jobs with a predetermined route have to be scheduled to be processed in several machines. Dispatching Rules (DRs) are used as heuristics to solve the dynamic version of the problem (i.e. where part of the input data is unknown) and are applied locally. A DR takes as input characteristics of the job (for instances, its processing time) and the shop itself (for instances, the mean shop utilisation) and attributes a priority to the job, the job in the machine queue that gets the lower value (the highest priority) will be processed next by the machine. In GP, DRs are individuals whose expression is evolved during the generations.

1.1 Motivation

This thesis aims to increase the DRs interpretability and potentially reducing their size by using Dimensionally Aware Genetic Programming (DAGP) enforced by Grammar-Guided Genetic Programming (GGGP) in the JJS problem.

The motivation for approaching JSS with GGGP is that, as many studies have shown, GP can evolve heuristics that outperform manually designed rules [1]. They have improved interpretability compared to Black-Box models and can be integrated into real-time production systems with real-time constraints. After a GP program has evolved and reached a DR, using it only requires computing its score, which is computationally inexpensive especially comparing to other iterative optimisation algorithms.

GGGP appears as a solid line of research because it restricts the standard GP solution space, which rapidly can become too big to search efficiently. Furthermore, some particular grammars can lead to DRs that make more sense a priori (dimensionally aware).

Moreover, in the literature survey provided by [1], only three papers have duelled with this type of algorithms in these problems: In [2], the authors compare three grammars: one composed of standard operations and terminals, other of common heuristics such as (FIFO, LIFO, and others) and another that mixes the previous two. The mixed grammar performed the best and was comparable with the rules COVERT and ATC. In [3] the author uses dimensionally aware genetic programming (DAGP) [4] enforced with strong type genetic programming (STGP) and compares it with standard GP and other algorithms. There were not significant divergences between the two methods in regards to performance. However, as dimensionally aware (DA) dispatching rules are semantically correct, they can be more interpretable. Finally, In [5], one grammar was used to enforce DAGP also using STGP; the standard GP results were better than DAGP, but the dispatching rules less interpretable.

Using complete GGGP methods for this problem fills a lacuna in the literature. The only three papers that applied grammars for the JSS used STGP. The major downside in this approach, which was one of the concerns in [5], is that a simple STGP dimensionally aware implementation only allows dispatching rules of dimension 0 or 1 in all its units (e.g. in [5] an expression that multiplied time with time was invalid). However, if instead of STGP, we use complete grammar methods, like Grammatical evolution (GE) and Context-free grammar-based genetic programming (CFG-GP), and use a grammar that supports expressions with more dimensions (e.g. from -2 to 2), we do not restrict as much the solution space. Hopefully, this can lead to a performance improvement if expressions with those dimensions perform well.

1.2 Objectives and Methodology

This thesis aims to explore and improve GP methods to generate dispatching rules for the job shop scheduling problem that are effective while being interpretable and generalisable.

This work will use DA enforced by GGGP methods. Mainly Grammatical Evolution (GE) and Context-Free-Grammar based Genetic programming (CFG-GP), standard and recognised GGGP methods never used in JSS. Furthermore, our focus is to reach highly interpretable DRs. As small DRs are interpretable, an Enumerator was created to enumerate all possible DRs and evaluate them, allowing to check if GP and GGGP methods explore the small DRs optimally.

Specifically, this thesis will answer three research questions:

- (RQ1) What is the trade-off between performance and interpretability when introducing dimensional awareness?
- (RQ2) Can genetic programming algorithms find (near-)optimal Dispatching Rules (of a given size) for the Job Shop Scheduling problem?
- (RQ3) How performative and explainable are the best Dispatching Rules found?

1.2.1 RQ1

The first objective will be accomplished using a dimensionally aware (DA) grammar in several GGGP algorithms, namely CFG-GP and GE. Thus, dimensionally aware genetic programming (DAGP) is obtained. By comparing their generated DRs to the ones generated by standard GP, we will answer the research question.

Several different variants of all algorithms will be tested. All those are described in Chapter 5. CFG-GP was chosen due to its similarities with GP and proven performance and GE's because it is prevalent in the community and easy to implement. The performance of the algorithms will be compared by comparing its best obtained DRs in each run. A unilateral Mann-Whitney U test will be used to see if the performance differences are significant, allowing us to see which one generated better DRs. If the DAGP algorithms have a significative better performance than standard GP, there would not be a performance drop.

In this question, interpretability is related to the DRs size (for the last one, where we will individually analyse every DR, our interpretation will differ). Other works in the literature also tend to approximate interpretability with size [5]. Thus, if an algorithm generates significantly smaller DRs with better performance than another, it will be more interpretable. However, when the two criteria (size and performance) diverge, plots with both fitness and size of the DRs will be displayed. Here, if the DRs found by an algorithm for all sizes always have better performance than the DRs found by another algorithm for the same sizes, the first algorithm is more interpretable.

1.2.2 RQ2

The second objective will be accomplished by creating an Enumerator of all possible dimensionally aware DRs of a given fixed small size. This total enumeration will allow us to discover the best possible DRs for tiny sizes, allowing us to see if GP and GGGP manage to find optimum or near optimum DRs for those sizes and function/terminal sets.

The enumeration had to restrict itself to the DA dispatching rules because the number of expressions would have been too big to be evaluated without that restriction. The way a DR was classified as unique is further elaborated in Chapter 4. The enumerated DRs were all evaluated in the JSS simulator. By analysing if the algorithms can find the enumerated best DRs will answer the research question. If they did, they explored optimally; if not, they do not explore optimally. Furthermore, if the answer is negative, it will be possible to know the rank of the best-found DR for that algorithm.

1.2.3 RQ3

The last objective will be accomplished by statistically comparing the performance of the best-found DRs in a wide range of job shop instances. This comparison will allow us to see in which scenarios they perform well and validate whether they can generalise to different settings. Furthermore, we will interpret the best DRs by comparing them to several DRs found in the literature.

The third research objective can be divided into two subparts. RQ3.1: How we test a DRs performance? RQ3.2: How we test is interpretability?

A global performance metric is used to answer RQ3.1. This metric was also used in [6], and it was chosen in order for us to have comparable results with this paper. Moreover, statistical tests to compare the DRs performance will be implemented. All these tests and metrics will be described in Chapter 3.

Answering RQ3.2 is a lot more challenging. As interpretability is related to size, we will evaluate it. However, this leaves behind much meaning. We will say that a DR is interpretable if an expert can, just by looking at the DR, make predictions on its behaviour for some shop scenarios. In these two ways, interpretability will be explored, comparing the DR's size and seeing if comparing with other rules (presented in the literature) insights on the rule performance in different scenarios can be made.

Moreover, a statistical analysis on the global metric parameter will be made. This analysis will allow us to see if our chosen metric is robust.

1.3 Structure of the thesis

The organisation of the thesis is the following: chapter 2 does the literature review. It starts with an overview of Production Scheduling, stating related problems and characterising the Job Shop Scheduling problem. Afterwards, it briefly explores the methods used to solve this problem (from proactive approaches to reactive). Then, it explains the origin and behaviour of some DRs, emphasising the DRs we use as a benchmark. Next, it introduces GP and discusses its limitations, hinting at GGGP as a method to culminate one of them (enormous search space). Subsequently, GGGP is introduced, and an expositive section where grammars / and the most used GGGP methods (CFG-GP and GE) are described. Close to the end, it explores Dimensionnaly Aware Genetic programming, where it shows an iterative procedure to generate a DA grammar. Next, it has a small section giving context to expression counting. Finally, the last section binds GP and DRs, exploring what works made DRs using GP, particularising the ones that applied grammar methods.

Chapter 3 details everything related to the problem being solved and the general experimental design. It starts by defining our implemented Job Shop Simulator detailing how we set the jobs due dates and their arrival times. Then, it presents the parameters used for the training and test instances. Afterwards, it explores our fitness metric and how we compare DRs. Finally, it ends with a small section dedicated to studying the variability of our fitness metric.

Chapter 4 describes the Enumerator. Here the mathematical derivation of the total amount of possible expressions is presented for Binary and Unary operators. Afterwards, an algorithm capable of enumerating all these expressions and selecting only the dimensionally aware ones without duplicates is described. Furthermore, this chapter indicates the terminal sets used and the required modifications to enumerate sizes 9 and 11. Finally, the expected performance of a random search on dimensionally aware expressions is defined, which serves in Appendix B as a baseline for comparing the convergence speed of the algorithms.

Chapter 5 describes all the specificities and parameters of the GP algorithms. It starts by briefly defining all algorithms tested and their respective variants. Then it details the experimental setup and enumerates the parameter configurations used in the pre-test phase, ending by choosing the best parameters for each algorithm variant. Finally, it explores the grammars used, the initialisation procedures, and some specific parameters (for example, max tree depth) of all algorithm variants.

Chapter 6 presents the results and the discussion, answering the three research questions. First plots comparing the performance and size of the best DRs are presented, allowing us to answer the first research question: What is the trade-off between performance and interpretability when introducing dimensional awareness? Next, the enumerator results are compared to the best small DRs found by all GP algorithms, answering the second research question: Can genetic programming algorithms find (near-)optimal Dispatching Rules (of a given size) for the Job Shop Scheduling problem? Finally, the best DRs overall, the ones that represent the Pareto frontier in size and performance, are studied. Here their performance is compared in every instance, and they are interpreted. This allows us to answer the third research question: How performative and explainable are the best Dispatching Rules found?

Finally, the last chapter is the conclusion 7. Here is presented a brief resume of the findings and possibilities of further work.

The thesis is complemented with three appendices. Appendix A tests a way to trim down a given terminal/operator set using the Enumerator. Appendix B presents the algorithms training performance during the evolution. Finally, Appendix C shows all the grammars used in the thesis.

Capítulo 2

Literature Review

2.1 Production Scheduling

This section explores and classifies a variety of scheduling problems and variants, as well as some solution approaches, in order to provide the necessary context to our JSS.

Production scheduling [1] is a major topic in operations management. It is required to determine when a job should be processed and in what machine to process. The production setting might be subject to different uncertainties, such as dynamic arrivals of jobs, variations in executing time and machine failures. The objective is to schedule the jobs as optimal as possible concerning some fitness metric.

Our primary problem will be the Job Shop Scheduling Problem (JSS) with dynamic job arrivals, i.e., the release dates are not known a priory.

2.1.1 Related Production Scheduling Problems

[7, p. 13-14] describes a scheduling problem by the triplet $\{\alpha, \beta \gamma\}$, α is the machine environment, β are the process characteristics, and γ is the objective to be optimized. The objective is to schedule a job and allocate a machine to optimize γ, given that all constraints β are satisfied.

The machines enviroments represented by α can be:

- **Single machine:** it is the case of a single machine. It is the simplest case.
- **Identical machines in parallel** (*Pm*)**:** there are M identical parallel machines. A job j requires a single operation and may be processed on any machine or a given subset.
- **Machines in parallel with different speeds** (*Qm*)**:** there are m machines in parallel with different speeds. Machines i speed is denoted by v_i. So the speed only depends on the machine.
- **Unrelated machines in parallel** (*Rm*)**:** there are m different machines in parallel. The machine i can process job j at speed v_{ij}. The speed depends on the machine and the job.

- **Flow shop (Fm):** there are m machines in series. A job has to be processed on each machine. All jobs must follow a given route. A job goes to the next machine queue after its completion on the machine it was.

The previous environments serve as building blocks for our main problems of interest.

2.1.2 JSS Problem Characterization and Definition

Also, in [7, p. 13], the author defines our main environment of interest.

- **Job shop (Jm):** is a job shop with m machines. Each job j has its set of operations O_j to perform, which are made in a predetermined order. Furthermore, each operation must be processed in a predetermined machine and has a specific processing time p_{ij}. The β can distinguish between job shops where a job can only visit a machine once, or it can visit it multiple times. In the latter case, β has the recirculating parameter rcrc. Moreover, a machine can only process a job at a time.

The problem's objective is to make a scheduler that optimizes a given objective function γ for each of the previous machine environments.

[8] lists some common objective functions in production scheduling :

- **Makespan:** completion time of the last job that leaves the system.
- **Maximum flowtime:** maximum flowtime achieved by any of the jobs.
- **Maximum tardiness:** is the maximum tardiness achieved by any of the jobs.
- **Total weighted completion time:** denotes the weighted sum of all completion times.
- **Total weighted tardiness:** is the weighted sum of tardiness values of all jobs.
- **Mean tardiness:** is the mean of the tardiness of all jobs
- **Total flowtime:** is the sum of flow times of all jobs.
- **Weighted number of tardy jobs:** is the weighted sum of all tardy jobs.
- **Weighted earliness, and weighted tardiness:** is the sum of the total weighted tardiness and the total weighted earliness.
- **Machine utilization:** the difference between the maximum utilization and minimum utilization of all machines.

This thesis's problems will focus on the environments (Jm), without recirculation, using a variant of the mean tardiness metric as an objective to be optimized.

2.1.3 JSS Problem Approaches

Scheduling is a major topic in several domains. From operation research, CPU tasks assignments, hospital shift assignments, among others. As it is so important several well-structured variants of scheduling are studied. One of them is the classical optimization problem Job Shop Scheduling Problem (JSS). The problem can be stated as follows: Let M define a set of machines and J a set of jobs that must be processed. Each job ($j \in J$) has multiple operations (set O_j) that must be processed in a specific machine. Moreover each operation ($i \in O_j$) has a specific processing time p_{ij} [7]. Also, we assume that a machine cannot be preempted, and the setup time is included in the processing time. The production setting might be subject to different uncertainties in the dynamic variant, such as dynamic arrivals of jobs, variations in executing time and machine failures. We will not consider variation in processing times but will considerer dynamic arrivals of jobs (i.e. the jobs arrival time are unknown in advance).

Several ways of tackling this problem were used. Proactive approaches produce an offline schedule robust to the variance of executing time events [9]. So, at execution time, the activities (jobs) will be dispatched according to the offline scheduler's defined sequence. However, its performance is highly dependent on the information obtained when producing the offline scheduler. There is hybrid predictive-reactive scheduling, where some component of the production system is continuously adapted during operations. Furthermore, there are reactive approaches that do not take any decision in advance. Instead, when a job in real-time comes, a local decision is made. Usually, this uses simple and constructive heuristics such as Dispatching Rules (DRs) to prioritize jobs. The job with the highest priority is the job that is scheduled next [10].

2.1.4 Dispatching Rules

There are many applications of DRs in manufacturing systems. However, their use can oversimplify and be myopic [11]. A lot has been made in comparing the proactive and reactive approaches, and each strategy has its benefits. More precisely, in [12], it is shown that DRs perform as well as more advanced dynamic heuristics in environments with much uncertainty.

The principal aspects of a DR are: they require low computational resources, their implementation is simple, and they can react in real-time. Furthermore, if the DRs are small enough, they tend to be easily explainable.

DRs have different characteristics. For instance, rule SPT (chooses the job with shortest processing time PT) is an excellent match to optimize Total Flowtime (but can be myotic, can lead to the generation of long queues). Thus, a good DR has to consider the future. This information can be inserted in the DRs using diverse parameters, such as WINQ, the next machine's workload, and NPT, the jobs' next operations' processing time. In this way, in the literature, rules have been evolving, passing from PT, a myotic rule, to PT+WINQ [13] to 2PT+WINQ+NPT [14] which is very good in optimizing mean flowtime. There are two distinct ways to compute WinQ. One of them adds all the processing time of the jobs of the next queue (which is not used in this thesis but appears in appendix A as WQ). Moreover, the other proposed by [13] is an improved version that only sums the processing times of operations that would be assigned a better score than the current operation in this next machines' queue (it is this way of computing that we will use).

In the survey [8], many DRs are compared to the unrelated JSS problem. Although some of them perform well, none outperformed the others in all scenarios. Thus, it is challenging to have a single rule to be highly performative in all shop scenarios as rules have distinct characteristics. For example, processing-time-based rules are good in tight loads, and due-date-based rules are good when the shop-load levels are low [15].

Other authors [16] utilize U, the mean machine utilization, to reach a more sophisticated formula that changes the behaviour depending on the machine's utilization. $PTe^{u} + PT\frac{sla}{RPT}e^{-u} + WINQ$, which, although presenting good performance, is worse than $2PT + WINQ + NPT$ in tight loads. This more sophisticated rule is called RR rule (rule by Raghu and Rajendran). Here RPT represents the remaining processing time of all the job processes, NPT is the processing time of the next process of a job, and S is the slack of a job which is the difference between the job allowance (due date minus current time) and the job remaining processing time. If the slack is inferior to zero, the job is delayed. A common variant of this terminal is its positive version $S^{+} = max(0, S)$, which will be used.

Finally, there is another family of rules that is worth considering. CR+PT and S/RPT+PT were proposed in [17] and introduce a new terminal, the critical ratio (CR). CR is the allowance of a job divided by its remaining processing time (CR = A/RPT). The allowance of a job is the difference between its due date and the current time (it represents the available time to finish the job without delays). Therefore, a CR superior to one means that the job is not delayed and an inferior to one means a delay, furthermore CR=1 means the job is on time. Some useful variants are their positive variants $A^{+} = max(A, 0)$ and $CR^{+} = max(CR, 0)$.

During the thesis, our metric of interest is a relative metric that uses the DRs in [14] and [18] as benchmark rules. Specifically, we evaluate the score obtained by each DR presented in those two papers for every shop instance. After that, we define the benchmark Mean Tardiness for each instance as the lowest mean Tardiness obtained by any of those rules. Having the fitness comparison integrated into the fitness metric will allow us to interpret easier the performance of a rule. If a rule has a performance inferior to one, it is better in general than every rule presented in those papers. Further details on the objective function to be optimized are described in chapter 3. Here follow the rules presented in those works that serve as a benchmark to our performance metric and some nomenclature on Table 2.1.

- **FIFO (First in First Out):** first in first out. The job that enters the queue sooner is scheduled first.

- **AT (Arrival time):** The job with the earliest arrival time is scheduled first.

$$Z_i = T_i \tag{2.1}$$

Tabela 2.1: Symbols used to calculate priorities of common dispatching rules

Sym	**Meaning**
t	Time of the dispatching decision
t_{ij}	Processing time of operation j of job i
o(i)	Number of operation left of job i
D_i	Due-date of job i
W_i	Next queue of job i total work content
T_i	Job arrival Time
Z_i	Priority value assigned to the job at time t

- **EDD (Earliest Due Date):** The job with the earliest due date is scheduled first.

$$Z_i = D_i \tag{2.2}$$

- **S/OPN (Slack over Remaining Operations):** The job with least slack over remaining of operations is scheduled first.

$$Zi_i = \begin{cases} s_i/(o(i)-j+1) & if\, s_i \geq 0 \\ s_i \times (o(i)-j+1) & if\, s_i < 0 \end{cases}. \tag{2.3}$$

where s_i can be calculated as.

$$s_i = D_i - t - \sum_{q=j}^{O(j)} t_{iq} \tag{2.4}$$

- **COVERT (Cost Over Time):** Each job i is atributes a priority index $Z_i = c_i/t_{ij}$ being t_{ij} is processing time on the machine j and WT_i the expected waiting time of the job uncompleted operations. Here the job with the highest Z_i is scheduled first in oposition to the rest of the rules. The computation for the penality c_i is the following:

$$c_i = \begin{cases} (WT_i - s_i)/WT_i & if\, 0 \leq s_i < WT_i \\ 0 & if\, s_i \geq WT_i \\ 1 & if\, s_i < 0 \end{cases}. \tag{2.5}$$

- **RR (rule by Raghu and Rajendran):** The job priority index is determined by the following equation. Where u in the shop utilization, W_{nxt} is the work of the next queue and RPT_i is the remaining processing time of a job.

$$Z_i = s_i \times t_{ij} \times exp(-u)/RPT_i + t_{ij} \times exp(u) + W_{nxt} \tag{2.6}$$

- **SPT (Shortest Process Time):** The job with the shortest processing time is scheduled next.

$$Z_i = t_{ij} \tag{2.7}$$

- **PT+WINQ (Processing Time plus Work-In-Next-Queue):** The priority index in obtain by:

$$Z_i = t_{ij} + W_i \tag{2.8}$$

The following rule were also used as benchmark: $(PT+WINQ+AT)$, $(PT+WINQ+SL)$, $((PT+WINQ)/TIS)$, (PT/TIS), $(AT-RPT)$, $((2PT+WinQ+NPT))$, $(PT+WINQ+NPT+SL)$. Where TIS represents time in shop, AT arrival time and SL is the negative slack ($SL = min(0, Slack)$).

Designing a DR is a complex process. Therefore, Machine Learning is being used to automate this process and find better performative DRs. It follows an overview of machine learning applied to this problem.

2.2 Machine Learning for Production Scheduling

What makes JSS challenging to solve with traditional optimizers is that it is an NP-hard problem [19], this causes optimizers to fail to reach an optimum solution in a viable time [20]. Therefore, scheduling policies are used instead of full optimizers.

Due to this, DRs have gained much interest, as previously shown, but they are tough to design manually. Obtaining a highly performative rule is difficult because DRs tend to be myotic [21].

Therefore, Machine Learning is being used to reach these DRs, overcoming the difficulties in manually designing one. In ML, the learning methods can be divided into supervised learning, unsupervised learning, and reinforcement learning.

In supervised learning, annotated training data exist [22]. In which for each training example, there is associated a result. The objective is to induce a model that maps the training data to its associate results which can be used to classify unlabeled data. Supervised learning is being used for JSS to extract rules from the optimization results [21]. Several supervised algorithms have been used to this problem ranging from logistic regression [23], decision trees [24] and neural networks [25].

Furthermore, GP can also be used in some problems in a supervised way, such as in Symbolic Regression, where you know the result for each data input.

However, using supervised learning for production scheduling is not very practical as it requires a previous model to generate labelled data. For instance, in [25] the author uses Genetic algorithms to make a scheduler to train the network. So before training, a previous good scheduler is required.

In unsupervised learning, the data does not have a label, and there is no need to consider the sample size. Instead, these models try to separate the data in the groups that best describe them [26]. These techniques are not commonly used in Production Scheduling.

In Reinforcement Learning (RL), an agent learns by trial-an error how to interact with a dynamic environment [27], optimizing a given reward function in the process (learning a mapping from states to actions). In [28] the authors trained, in a highly dynamic environment, an RL agent that optimizes DRs in a complex JJS system. Also, another research line is, instead of generating single dispatching rules, to make an algorithm capable of choosing from a range of rules which dispatching rule to use depending on the job conditions. This automated combination of rules was used in [29] where a reinforcement learning agent choose, depending on the state of the job shop and the machine, which dispatching rule to use. The agent was better than all the individual dispatching rules that it could choose. We will not explore these methods as we focus on interpretability. Adding another RL agent introduces another layer of complexity, making it more difficult to understand the local choices made in each machine, being less interpretable than a single rule.

Finally, there is also Evolutionary Computation (EC). EC represents a family of algorithms inspired by biological evolution that solve optimization problems. There are several proposed EC algorithms. From genetic algorithms to evolution strategies, evolutionary programming, and genetic programming (GP) [30] stating some. These algorithms simulate the evolution of data structures (known as individuals) and, through computational operations based on nature's evolution, improve the individuals' quality (performance). This thesis will focus more on GP as it is used as the primary research line for generating DRs. For more information on the other algorithms, [31] gives a good overview. The most common operations are mutation, crossovers, and elitism, defined by their probability of occurrence between other parameters. Usually, the algorithms are susceptible to these parameters values, and care must be taken to choose them. In [32] the author addresses these problems by creating another meta-agent that tunes these parameters, making the original algorithm adaptative. In order to not use bad parameters, we have made pre-tests with several parameter configurations in this thesis.

The GP individuals have a tree representation and directly represent mathematical expressions. In this thesis, the term DRs will refer to the GP individuals in the context of scheduling problems and the term expressions when not.

2.3 Standard Genetic Programming - GP

In GP, unlike supervised methods, there is no need to have a priori a good scheduler. Instead, at first, a set of random DRs are generated, and for each one, a score (known as fitness) is calculated. This fitness results from simulating the expressions and computing how well the specified objective (e.g. makespan) is optimized. To the next generation, pass the ones that perform the best. After that, through crossovers and mutations[33], new DRs are made from the surviving ones, and they tend to become increasingly better. In this way, the better performative DRs survive for the following generation. It is similar to the survival of the fittest in Biology. Figure 2.1 represents in a simplified way how does GP work. In generation X are three DRs (WinQ, PT, and WinQ+PT) with WinQ and PT as terminals. After that, the simulator attributes a fitness to them (1 to WinQ+PT, 2 to WinQ and 3 to PT). Next, through a selection process, some are selected. Afterwards, genetic operators are applied to create new ones that are used for the next generation. Finally, the algorithm ends when the termination condition is met (which can be, for example, reaching a predefined maximum number of generations).

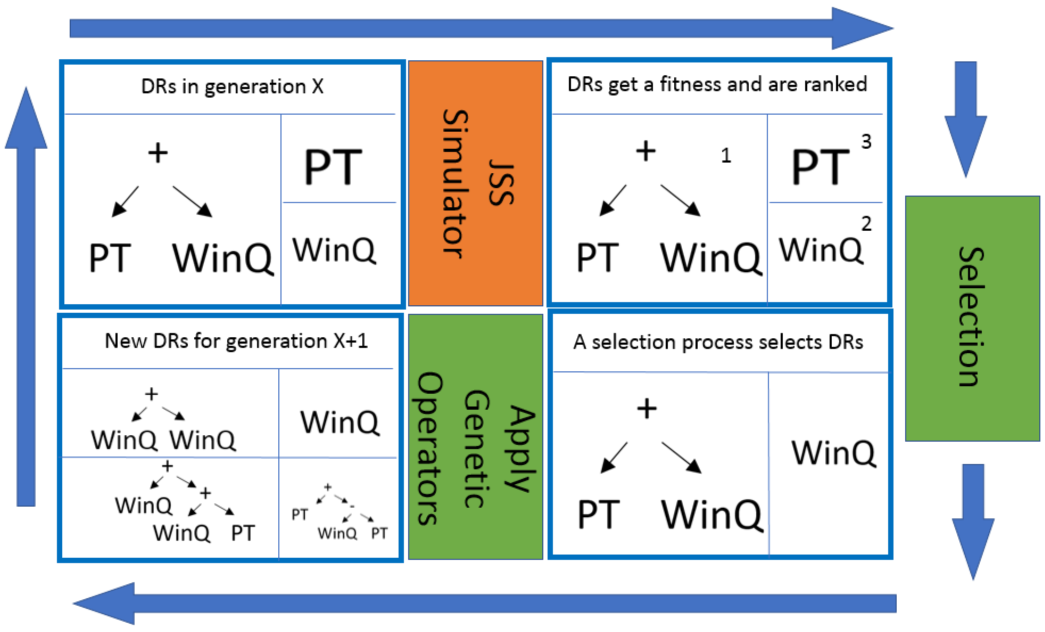

DRs in generation X
PT
WinQ
JSS Simulator
DRs get a fitness and are ranked
1
3
2
Selection
A selection process selects DRs
Apply Genetic Operators
New DRs for generation X+1

Figura 2.1: GP behaviour Diagram. In the top right, square 1, 2, 3 represent the obtained fitness for each expression.

In GP, the user does not need to know the solution structure in advance. [34] state that "GP is an asystematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done."

Still, GP has its limitations: it has a vast domain space and is affected by bloat. Bloat is a point in the evolution of a GP program in which the expressions stop improving due to the population becoming too uniform (redundant) and each expression being too complex.

Several authors tried to cope with those problems in different ways. [35] used dimensional Aware GP (DAGP), enforced with one grammar, to trim down the domain space's size. [36] used an online way to simplify GP training expressions to reduce the redundancies and the sizes of the evolved expressions. Both methods had promising results: DAGP performance was comparable with GP and the expressions smaller; the online simplifier created smaller expressions with similar performance. Both authors applied these methods to a broad range of problems. However, production scheduling was not one of them.

The most common genetic operators are mutation and crossover. In GP, both of them are simple. The simplicity comes from the closure property of GP, which allows any node to be the input and output of the function set [37]. Unfortunately, this characteristic is not present in grammar guided GP (c.f. the next section).

In a mutation, a mutation point is chosen in a GP individual. A random tree replaces the subtree below and the node itself. In the crossover, two parents are chosen, and two random crossover points are chosen, and then the subtrees are swapped between the parents given origin to two new trees. Figures 2.2 and 2.3 represent a GP crossover and a GP mutation respectively.

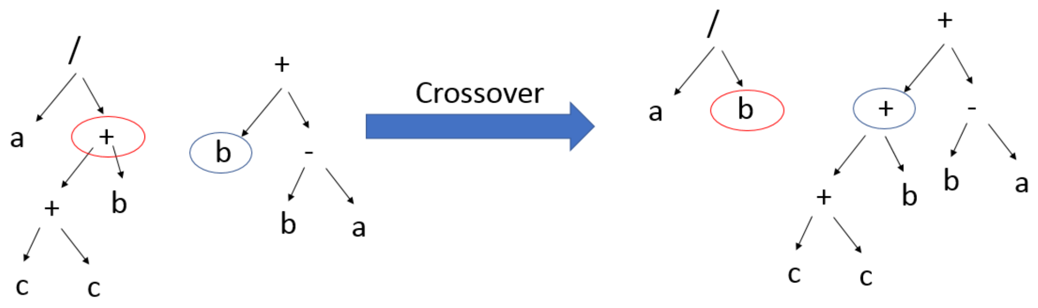


Figura 2.2: Crossover in GP. The circle represent the crossover points.

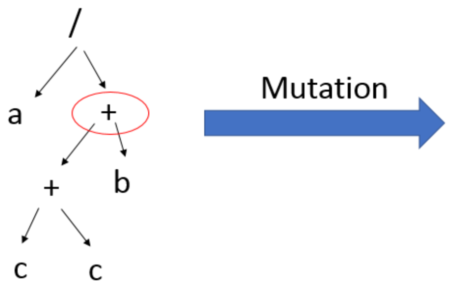


Figura 2.3: Mutation in GP. The circle represent the mutation point.

2.4 Grammar-Guided Genetic Programming - GGGP

One subfield of GP that manages to tackle the first GP limitation is grammar guided GP (GGGP). Grammars are used to represent restrictions on domains; they limit the expressions that can be used. In this way, experts can trim their search spaces like in [35].

Some terminology is needed for the understanding of GGGP. For example, [38], defines that, "genotype refers to the structure operated on by genetic operators such as crossover and mutation", "phenotype refers to the structure that is directly executed to produce the behaviour of an individual"and "the grammar defines the interpretation of the genotype space".

Firstly, the most used Grammar type in GGGP is a context free grammar. A context free grammar can be defined by a four-tuple $\{N, \sum, P, S\}$, where N is the nonterminal alphabet, $\sum$ is the terminal alphabet, S is the start symbol, and P is set of production. The productions P are of the form $x \to y$ where $x \in N$ and $y \in \{\sum \cup N\}^*$. Assume a simple example: $N = \{op, sym, exp\}$, $\sum = \{a, b, c, +, \times\}$ and the production set P can be:

$$\begin{aligned} S &\to exp \\ exp &\to op\ sym\ sym \mid op\ sym\ exp \\ op &\to + \mid \times \\ sym &\to a \mid b \mid c \end{aligned} \tag{2.9}$$

The symbol | represents an "Or". This simple grammar can define the expression in prefix notation $\{\times a + bc\}$ that translates in the usual notation in: $(b+c) \times a$. The derivation of this expression from the grammar is rather simple. First, all expressions start with the initial symbol S. S is translated in exp in the first rule. After that, we have to translate exp. We have two options: translate exp in (op sym sym) or in (op sym exp). In this case (op sym exp) was chosen. Next, we have to choose op between + or $\times$, we have chosen $\times$, and for sym, we can choose between (a,b,c) we have chosen a. In this state, we have the expression $\{\times a\,exp\}$, as exp is not a terminal symbol, we continue the decoding process. In this case, exp is substituted by (op sym sym) the op by + the first symbol by b and the last by c, and we have the final expression $\{\times a + bc\}$.

Having introduced the bases of GGGP, some variants of GGGP can be determined. Firstly [38] in the survey classifies the GGGP methods in terms of their representation. Ones can have tree representations (like CFG-GP) of their solutions, and others can have a linear representation, like in Genetic Evolution (GE) [39]. Both these methods will be further explored in the following sections. The other division is between the type of grammars used: Context-Free Grammars, Context-Sensitive Grammars, and Tree-Adjoining-Grammar (TAG, like in the algorithm TAG3P). Finally, the subsequent separation is between the type of search algorithm used. The most relevant types are probabilistic model-building algorithms, evolutionary developmental systems, meta-evolution, and others. Table 2.2 represents some notorious GGGP algorithms classifying them in terms of representation and search algorithms used.

Tabela 2.2: Grammar-based genetic programming algorithms classification. Divided in genotype representation and search method.

	Search algorithms			
Genotype Representation	Tournment search	Probabilistic model building	Combinatorial optimization	Meta-evolution
Linear	GE, SGE, HGE	TAG3P	Any linear	Meta-GE
Tree	CFG-GP	SG-GP, Ant-TAG	none	PEEL, DTAG3P

Some reference papers that compare GGGP methods are [40] in which TAG3P outperformed standard GP and CFG-GP in symbolic Regression, and [41] that compares GE and its variant SGE with CFG-GP. The conclusion reached is that GE performed poorly, but SGE and CFG-GP had very similar performances (one was superior to the other only in a set of problems). One particular aspect of this field is that many researchers create their variations of main algorithms, for example, [42] with BGBGP-HL, or [43] with HGE. Due to this, keeping up to date with the most recent algorithms is a challenge. A single paper that compares many GGGP algorithms was not found in the thesis research until the date. The comparisons are only made in groups of around four algorithms per paper. Being the papers referenced in this paragraph examples of that.

A choice had to be made in what algorithms to use, and CFG-GP [44] due to its high performance and GE [39] due to its simplicity will be implemented in this thesis to achieve dimensional awareness for the JSS problem. Methods that were not applied yet to this problem in this way.

Although the individuals created in GGGP are more complex than with just GP, it is easy to impose restrictions to the allowed operations, only changing the grammar. For instance, we can quickly write a grammar that only allows us to add the terminals B and C, and the A can be used only in multiplications. Doing this in regular GP requires significant changes to the code itself. This simplicity in making restrictions to the structure of the expressions is crucial in GGGP. A more detailed explanation on how to design grammars to specific problems can be found in [45].

Another relevant difference between these methods is the initialization process. In GGGP, an expression can end with a non-terminal still left to expand if we use a random initiation process (that generates many invalid expressions). So ways of complete non-terminated expressions can be used. In [46] several initiations algorithms are compared, being PTC2 that performs the best. So both random initiation and PTC2 [47] will be tested for grammar methods.

It follows next an overview of CFG-GP and GE.

2.4.1 Context Free Grammar Genetic Programming CFG-GP

[44] used a tree representation and Context-Free Grammar to solve the 6-bit parity problem. In this representation, each evolved solution is represented by a derivation tree (the genotype because it is where the operators operate), which is translated into a "conventionally"GP tree (that is, its phenotype). This method is named CFG-GP.

This derivation process can be represented in a tree form, which is directly evolved in CFG-GP. The CFG-GP tree contrasts with a normal GP tree that performs the same operation as can be seen in Figure 2.4 and 2.5. The CFG-GP is representing an individual of the grammar 2.9.

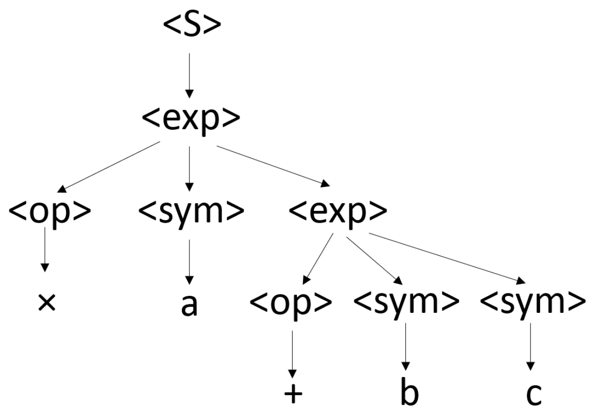


Figura 2.4: Tree in GGGP.

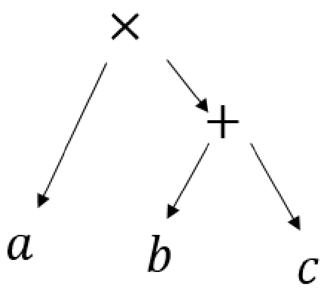


Figura 2.5: Tree in GP.

2.4.1.1 Mutation and Crossover in CFG-GP

Both mutation and crossover have some differences in relationship with GP. Firstly as all terminals have at least one non-terminal above them, the crossover points and the mutation point are located only on non-terminals.

The second one is that CFG-GP is not closed under its operations, so to ensure valid offsprings, care must be taken.

The same non-terminal must be expanded at each crossover site to ensure that the crossover creates a valid solution. First, a non-terminal is chosen in the parent, and if the second parent has the same non-terminal, the subtrees are swapped. If it does not, the first step is repeated until a non-terminal shared by the two parents is found. The worst-case is to swap the entire tree if the non-terminal at the root is the only one in common. Figure 2.6 represents a CFG-GP crossover.

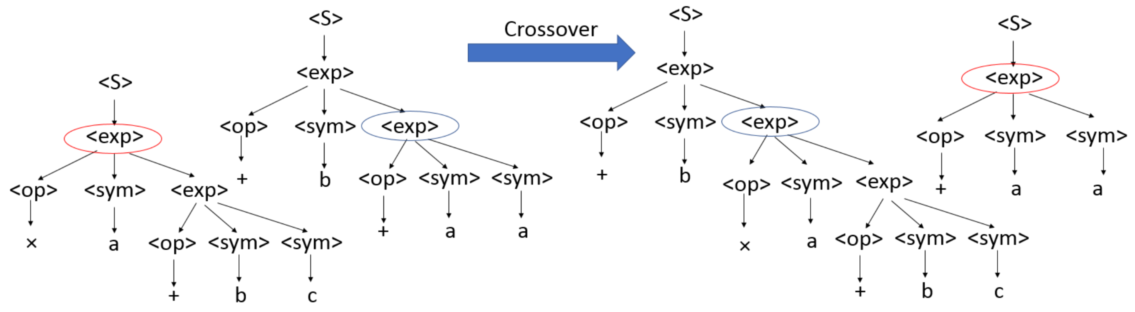


Figura 2.6: Crossover in CFG-GP. The circle represent the crossover points.

If some operation creates an offspring that exceeds the max tree depth, that process is repeated until an offspring respects the maximum depth.

For mutations, a mutation point (a non-terminal) is chosen. The bottom part of the subtree is erased, and a new tree is created by resampling the grammar respecting the maximum depth. Figure 2.7 represents a CFG-GP mutation.

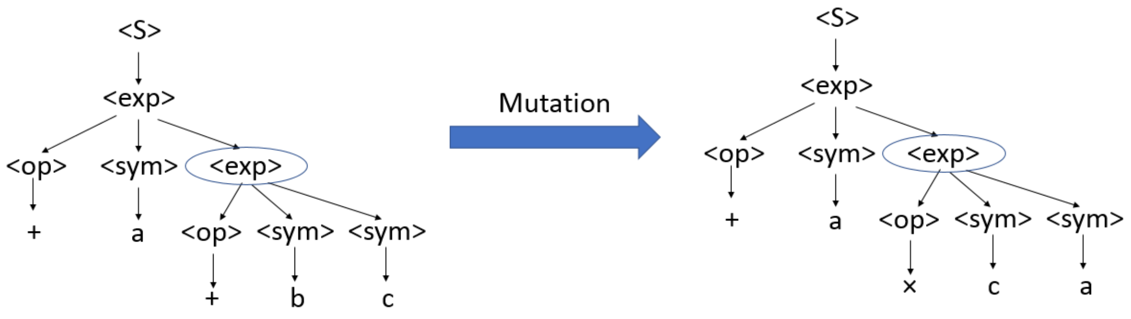


Figura 2.7: Mutation in CFG-GP. The circle represent the mutation point.

2.4.2 Grammatical Evolution - GE

GE, in opposition to GP and GGGP, uses a linear representation. The principal difference between GE and CFG-GP is that its crossover mutation operators are performed in the linear encoded element (genotype), which has to be interpreted back (mapped) to an expression in a phenotype to be run. As in GGGP, it follows an example using the grammar 2.9.

Having a individual in a linear format (genotype) $x = [1,5,3,9,4,6,1,2]$ the first step to run it is translated to its phenotype. Table 2.3 explains detailed the steps required , they will be detailed in text only for the first 3 derivations.

Every individual begins with the initial symbol. Then the first codon 1 is interpreted. It represents which one of the possible expansions of S will replace S. The operation $1\%1 = 0$

chooses the expansion number 0 (S is substituted by exp). Now it is necessary to expand exp using the second codon 5. It has two expansion possibilities, so $5\%2 = 1$ chooses the expansion number 1 (exp is substituted by op sym exp). Now each phenotype is expanded, beginning from the one on the left. So we expand now op, op has two possibilities, $3\%2 = 1$ so it is substituted by $\times$ (at this moment, the phenotype to expand is $\times$ *sym exp*). This process continues until the end: or every non-terminal is expanded, or we reach the end of the codons (i.e. the integer vector).

Tabela 2.3: GE mapping from genotype to phenotype.

I	**phenotype**	**codon**	**used Rule**	**compute**	**chose**
1	S	1	S→exp	1%1=0	expansion 0 exp
2	exp	5	ex p→ op sym sym ǀ op sym exp	5%2=1	expansion 1 op sym exp
3	op sym exp	3	o p →+ǀ×	3%2=1	expansion 1 ×
4	× sym exp	9	sym → a ǀ b ǀ c	9%3=0	expansion 0 a
5	× a exp	4	exp → op sym sym ǀ op sym exp	4%2=0	expansion 0 op sym sym
6	× a op sym sym	6	o p →+ǀ×	6%2=0	expansion 0 +
7	× a + sym sym	1	sym → a ǀ b ǀ c	1%3=1	expansion 1 b
8	× a + b sym	2	sym → a ǀ b ǀ c	2%3=2	expansion 2 c
Final	× a + b c				

Mutations and crossovers in GE work directly in the encoded number string. The mutation erases a part of the string and is substituted by a randomly generated string. The crossover chooses two crossover points and interchanges the parts on the right of the parent strings between the two forming the child.

The problem in GE is that the mutation and crossovers can create invalid individuals [46], so completion techniques can be used to avoid this problem. Another is that as the mutation and crossover operate at the genotype level, small local changes in the genotype can endorse considerable changes in the phenotype, which can be detrimental to evolution [48]. This is known as the locality problem of GE. Figures 2.8 and 2.9 represent a GE crossover and mutation respectively.

The main advantage is that it is versatile with its genotype-phenotype mapping and has a simple implementation.

2.4.2.1 Explosive an unbalanced grammars

A problem that affects GE in a more relevant way than CFG-GP is that some grammars can be explosive [49]. An explosive grammar is one where if there is a choice between non-terminals and terminals, a non-terminal expansion is more likely to be chosen. In these grammars, random initiation processes have a probability close to zero to generate long individuals. In contrast with CFG-GP crossover, GE crossover can generate an invalid individual. If the grammar is explosive,

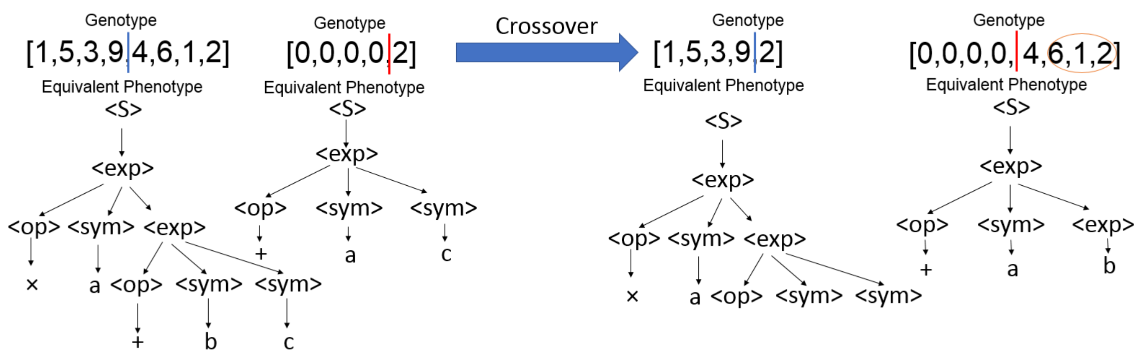


Figura 2.8: Crossover in GE. A bar represents the crossover points. Here one of the individuals is incomplete (it is invalid). The other is valid but has redundant codons that were not used (marked with a circle).

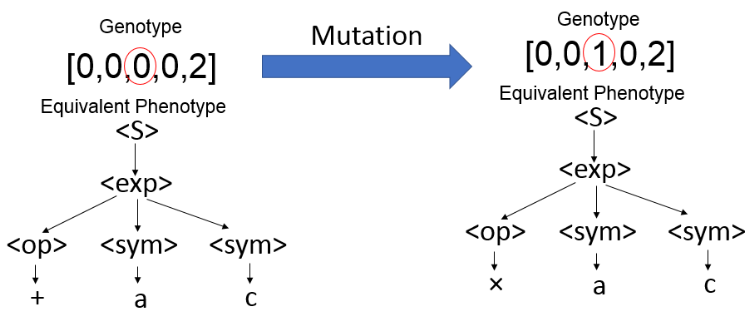


Figura 2.9: Mutation in GE. The circle represent the mutation point.

this generated individual will have a low probability of being complete. So GE will be best applied with not explosive grammars or will have to use ways of complete invalid individuals.

Therefore, other methods of initialization have to be used in these types of grammars. In [50] the author uses an adaptation of PTC2 [47] as an initiation method in GE and shows that PTC2 grants significant improvements face to a random initiation for this type of grammars. So PTC2 will also be used in this thesis.

In PTC2, the user chooses the individual's size probability distribution and the terminals and non-terminals probability distributions. Furthermore, the individual can be slightly bigger than the required size. Due to this, PTC2 only approximates the user-provided tree size distribution.

2.4.3 Dimensionally Aware GP

A particular way to impose restrictions on GP is by enforcing dimensional awareness. GP with this characteristic is called Dimensionally Aware GP (DAGP). In science, the dimensions of the variables act as syntactic constrain on the valid formulas. We cannot sum apples and oranges and expect the result to have a meaning. In contrast, in standard GP, any operation between the terminals is allowed, making the solution space of DAGP smaller than standard GP. Which is a good indicator that DAGP can have a better search efficiency. Several methods to use DAGP have been proposed, from transforming the problem to a dual objective one (the second one is the goodness of fit [4]), or by using a grammar [45]. Both ways have very distinct characteristics. In dual objective, candidates with dimensional inconsistencies may appear but are penalized, contrasting with the grammar method that makes it impossible for them to appear. The downside is that the crossover and mutation operators performed in GGGP are more complex than in GP, which can cause some complexity overhead. In both these papers, GGGP and GP's results were similar, being that GGGP involved smaller expressions in general but could be easier stuck in its evolution procedure.

In DAGP, each variable, v_i belonging to the terminal set, is associated with one dimension vector. Each position represents the exponent of that dimension for that variable. The vector size is equal to the total number of different dimensions in the problem. For instances, if a problem had one variable that measured time, one variable that measures meters, and one variable that measures temperature, the dimension vector would have size three and could be written as $[x_k, y_k, z_k]$. For the variable representing time the vector is $[1,0,0]$, for the one measuring distance is $[0,1,0]$, for an adimensional constant is $[0,0,0]$, and for an acceleration (m/s^2) is $[-2,1,0]$. In DAGP, it is only possible to add or subtract terms with equal dimension vectors. Trigonometric and power operations can only be performed on adimensional terms. Furthermore, when multiplying or dividing, the dimension vectors are added or subtracted, respectively.

2.4.4 Dimensionally Aware grammar

In order to make a context-free grammar that guarantees dimensional consistency, some restrictions have to be made. The set of possible units has to be finite. Furthermore, besides being finite, the dimensions are always integers.

In the thesis, the terminals were dimensionless (a ratio) or represented time. So the units of each variable are represented only with one variable u. For example, the processing time (PT) has dimension 1, and $1/PT$ has dimension -1. In the grammar, N_u represents the production rules referring to expressions with that dimension. We considerer u to have values between [-2 and 2]. As this grammar can become extensive, an automatic procedure (algorithm 1) to generate it similar to [51] was implemented.

For simplicity for each production rule, instead of writing $A := A|X$, the second A was omitted. This omission conforms with the writing conventions of the ECJ (Java evolutionary computation toolkit used to implement the genetic algorithms) grammar parser. Doing that this set of rules: $A := X$ and $A := Y$ is equivalent to $A := X|Y$

Due to multiplication being commutative, the number of multiplications used was reduced. The complete grammars used can be seen in appendix C.

Algorithm 1 Automatic Generation of Production Rules

```
1:  procedure AUTOMATIC GENERATION OF PRODUCTION RULES
2:      $u = \{-2,-1,0,1,2\}$
3:      # innit production set P
4:      P = {}
5:      for dimension in u do
6:          $P = P \cup \{< start >:=< N_{dimension} >\}$
7:      for d in u do
8:          #add addition and subtraction rules
9:          $P = P \cup \{< N_d >:= (+ < N_d >< N_d >)\}$
10:         $P = P \cup \{< N_d >:= (- < N_d >< N_d >)\}$
11:         #multiplication derivations
12:         for each v,p such that v+p=d,v<p do
13:             $P = P \cup \{< N_d >= (\times < N_v >< N_p >)\}$
14:         if d is multiple of two then
15:             $a = d/2$
16:             $P = P \cup \{< N_d >= (\times < N_a >< N_a >)\}$
17:         #divisions derivations
18:         for each v,p such that v-p=d do
19:             $P = P \cup \{< N_d >= (/ < N_v >< N_p >)\}$
20:         #terminals derivations
21:         for each terminal T with dimension d do
22:             $P = P \cup \{< N_d >= (T)\}$
        return P
```

2.5 Expression Enumeration

This small section entails what has been done for enumerating expressions. Many papers explore a way to reduce the space dimensionality in GP. However, none states what was the impact of the changes analytically on the solution space. They only show it empirically.

Nevertheless, some research has been done trying to count words of a given grammar. Moreover, as a grammar can define the usual GP expressions, methods used to count words in grammars can count GP expressions. For an unambiguous, (i.e. there does not exist a string that can have more than one leftmost derivation or parse tree) context-free grammar, the Chomsky–Schützenberger enumeration theorem can be used to estimate asymptotic bounds to the total number of words in a grammar [52]. Unfortunately, our used DA grammars are ambiguous, so we cannot use this theorem. Nevertheless, it is possible to know if the number of words has exponential or polynomial growing [53].

However, as counting all possible mathematical expressions with binary operations is a particular problem, these theorems related to all grammars are unnecessary to apply to reach the intended result. In fact, in [54] it is calculated the number of full binary trees with b+1 leaves and proven that this number is equal to all possible expressions created of size 2b+1 that only use binary operators. Figure 2.10 shows the relationship between full binary trees and GP expressions with binary operators (OP is an operation and Ter is a terminal).

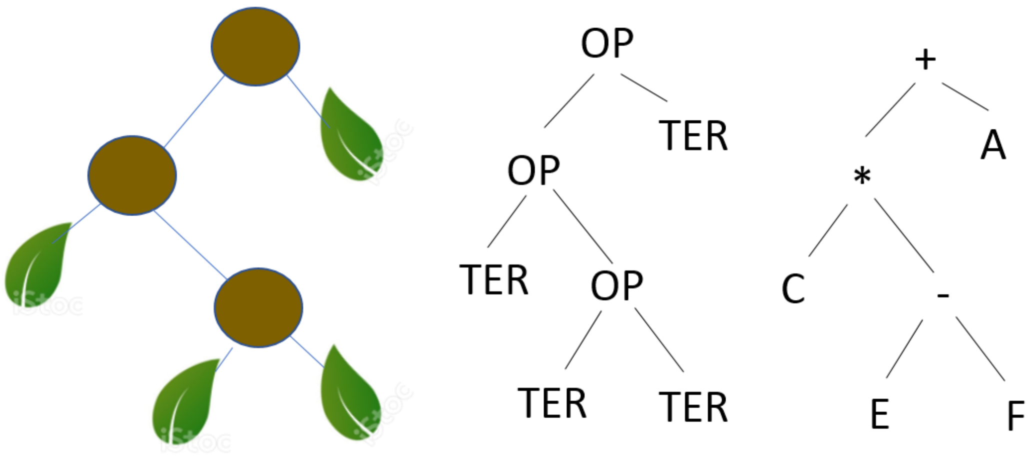


Figura 2.10: GP expression compared with a full binary tree. In the left, the full binary tree in the middle a GP tree with one terminal (TER) and one operator (OP) on the right a GP tree with the same structure representing expression (E-F)*C+A.

2.6 Genetic Programming uses in Job Shop Scheduling

Now that the GP algorithms were presented, this last section briefly reviews the literature that applies them to the job shop scheduling problem.

Making DRs with GP is being heavily studied, and it can, in some instances, outperform manually designed rules. A comprehensive summary can be found in [1].

[55] was probably the first to use GP to evolve dispatching rules for the Job Shop Scheduling problem. In this paper, the obtained rules were effective. In the following years, from 2005 to

2009, researchers got more interest in improving the GP performance on more production scheduling problems. New genetic operators and representations were used in particular problems [56] and studies compared different GP methods [57].

In these lines of research, many things can be evolved with GP. From Dispatching Rules (the most common approach) to Routing Rules [58] or even Due date assignment rules [59] between others.

Other representation of GP were also explored, from Strongly Typed GP [60], Cartesian GP [61], Linear GP [62], and Grammar-Based GP [38].

Also, to speed up training, much research has been done in creating surrogate models. These simpler models substitute the complete simulator in some training instances, allowing to discard bad rules without much computational burden [63].

Finally, multi-objective methods were also used in the literature. In [64] the author aimed to tackle three objectives. The objectives were combined in a single function in a weighted sum. Although this simple approach can give unsatisfactory results, combining multi-objective as a sum is an exciting approach that can, for instance, penalize too big expressions [6] or penalize not dimensional aware expressions [4] without a lot of added complexity.

A fundamental fault in this literature revision is that most works do not compare their results with state-of-the-art methods. Another is that the sizes of the expressions tend to be disregarded. In [65] simplification techniques were used on the expressions, but they still become too big and complex to interpret.

Specifically for GGGP, in the literature survey provided by [1], only three papers have duelled with this type of algorithms in JSS:

[2] created three grammars, one composed of normal operations and standard terminals, other of common heuristics such as (FIFO, LIFO, and others) and another that mixes the both. In this paper, the author compares the results of the three grammars. The combination of both grammars performed the best and was comparable with the rules COVERT and ATC. This use of grammars is distinct from our utilization, as we will try to achieve DA expressions.

In [3] the author uses dimensional aware GP (DAGP) and compares it with GP and other algorithms. There were not significant divergences between the two methods in regards to performance. Therefore, these results indicate that DAGP can be used to generate semantically reasonable solutions without performance loss. Here the author did not use grammars to impose semantical restrictions; instead, restrictions are added to the accepted types of the GP nodes operands, and the output type is also defined (this method is called Strongly Typed GP (STGP)). Thus, for example, this method can define that time cannot be added to an adimensional number and that the product of time and an adimensional number gives time.

In [5], one grammar was used to enforce a specific Dimensional aware GP (DAGP); the standard GP performance was better than DAGP, but the expressions less interpretable (i.e. semantically more challenging to interpret and bigger). Here the author also used STGP to implement dimensional awareness. The most significant difference between using a full DA Grammar and this STGP implementation is that some semantically correct operations were not allowed in the

STGP implementation. For example, the multiplication of time with time is not allowed, but it would be allowed if instead was used a full DA grammar. In fact, in our automatic production of grammar rules described in algorithm 1 if our allowed dimensions were u = [0,1], the generated grammar will represent the same restrains as the ones of the STGP implementation (for our terminals). As we use u = [-2,-1,0,1,2] we restrict less the valid expressions.

This thesis adds to these works in particular [3] and [5] by using DAGP methods to evolve heuristics to the JJS problem entirely based on a full DA Grammar. The main advantage is that our solution space is less restricted than using STGP (we allow higher-order interactions between the terms). The disadvantage is that it is a lot more challenging to implement. Nevertheless, we will use two common GGGP methods to solve this problem using a full DA grammar (CFG-GP and GE).

Capítulo 3

Problem definition and Experimental Design

This chapter explores all the relevant details of our implementation of the problem. Furthermore, it explores our chosen fitness metrics and methods to compare Dispatching Rules (DRs). This includes the simulation model for the job shop, its parameters, how its output is used to compute the expressions' fitness, and the statistical comparison of those expressions.

All the genetic programming algorithms and solution space analysis provided in this thesis are applied to one of the most classical optimization problems: the (dynamic) job shop scheduling. The problem can be stated as follows: Let M define a set of machines and J a set of jobs that must be processed. Each job ($j \in J$) has multiple operations (set O_j) that must be processed in a specific machine. Moreover each operation ($i \in O_j$) has a specific processing time p_{ij} [7]. Also, we assume that a machine cannot be preempted, and the setup time is included in the processing time. The production setting might be subject to different uncertainties in the dynamic variant, such as dynamic arrivals of jobs, variations in executing time and machine failures. We will not consider variation in processing times but will considerer dynamic arrivals of jobs (i.e. the jobs arrival time are unknown in advance).

3.1 Simulation model

The simulation model used follows common literature assumptions [13]. The number of machines is a parameter $|M|$. The number of operations of each job is sampled from a discrete uniform distribution with limits depending on $|M|$. Thus, machines are randomly assigned operations via a uniform distribution. Furthermore, re-circulation is allowed (i.e. a job can have different processes to be executed in the same machine).

The simulation starts without any jobs (empty), and a warm-up time is considered. The fitness is only computed after the warm-up time has passed. After that, the system is assumed to be in a steady-state.

As is common, jobs are numbered in increasing order upon arrival, and the mean tardiness is calculated for the jobs from 501 to 2500 (the first 500 correspond to the warm-up period). The arrival of jobs follows a Poisson distribution with rate λ. This rate depends on the utilization level of the shop (u), the mean number of operations of a job ($\tilde{n}$, in this work $\tilde{n} = 8$), the mean processing time of operations ($\tilde{p}$) and the number of machines ($|M|$). λ is calculated by equation 3.1.

$$\lambda = \frac{u \times |M|}{\tilde{n} \times \tilde{p}} \tag{3.1}$$

The arrival time and release time of each job are the same. When an operation finishes, the job moves to the next machine queue. When a machine is available, the DR assigns a score to each job. The job with the best score (lowest value of the fitness metric) is selected to be processed. Due dates d_j are calculated via the Total Work Content (TWC) [66] depending on the allowance factor (a). It is calculated by equation 3.2.

$$d_j = r_j + a \times \sum_{i \in O_j} p_{ij} \tag{3.2}$$

3.2 Experiments design and parameters

An instance can be defined by the tuple $< |M|, \tilde{p}, a, u >$. The performance of the DRs highly depends on the load conditions, defined by machine utilization (u) and allowance factor (a). In most cases u varies from 85% to 95% and a from 3 to 8 [67], [13]. The training set is used in the algorithm's evolution, and the test set is used to evaluate the best expressions in other instances. Table 3.1 presents all the parameters used for the training set and the test set.

Parameter	Description	Training	Test
$\lvert M\rvert$	Number of machines	10	10
$\tilde{p}$	Mean processing time of operations	50,100	50, 100, 200
a	Due dates allowance factor	2, 3, 4	2, 3, 4, 6, 8
u	Shop utilization level	0.85, 0.90, 0.95	0.80, 0.85, 0.90 ,0.95 ,0.97

Tabela 3.1: Training and test instances.

Several studies have shown that shop size does not affect the performance of the rules relative to the others [68]. Also, it is usual to consider ten machines in the simulations. So both in training and test, only ten machines will be used.

The test set is composed of five values for $a = 2, 3, 4, 6, 8$, five for $u = 0.8, 0.85, 0.90, 0.95, 0.97$ and three for $\tilde{p} = 50, 100, 200$. In this way the performance of a rule in a vast range of shops with different characteristics can be evaluated. More specifically a rule is evaluated in $5 \times 5 \times 3 = 75$ different conditions in the test set.

The training set is composed of three values for $a = 2,3,4$ three for $u = 0.85, 0.90, 0.95$ and two for $\tilde{p} = 50, 100$. More specifically a rule is evaluated in $3 \times 3 \times 2 = 18$ different conditions in the training set.

Furthermore, to provide reliable results in the training set for each tuple (each instance), 10 different independent runs are generated, and for the test set, 100 (usually this value is around 30 in the literature, but for the test set we increase it). Each repetition is called a replica and K_i is the set of all replicas ($k \in K_i$) belonging to instance i.

The simulator was coded in Java which, through the capabilities of ECJ [69] can handle the main evolutionary logic of the algorithms, allowing GP and GE. However, ECJ limits the use of grammars, only allowing GE. Due to that, CFG-GP had to be externally implemented in python. Further considerations on this point will be talked about in the further chapters.

3.3 Algorithms baseline and fitness metric

For the problem JSS, the hole population at generation g, P_g is evaluated at a time. The most expensive part of the algorithm is the fitness computation. Each simulation returns the mean tardiness of instance i in replication k when applied the rule A ($\tilde{T}_{Aik}$). These $\tilde{T}_{Aik}$ are subsequently grouped in a unique fitness metric.

It is challenging to have a single metric grouping the behavior of all instances in a fair and interpretable way. In this subsection will be detailed the chosen metric.

For starters, in each run (each replication k of instance i), a metric must be chosen. The chosen metric is mean tardiness. The tardiness of a job is given by $max(0, c_j - d_j)$, being c_j the completion time and d_j the due date. It measures how much a job was delayed if any. The mean tardiness $\tilde{T}$ of a set of jobs J is calculated as in Equation 3.3.

$$\tilde{T}_{Aik} = \frac{\sum_{j \in J} max(0, c_j - d_j)}{\#J} \tag{3.3}$$

The mean tardiness of an instance is computed as the mean of the mean tardiness of all replicas K_i of the instance i. As the number of replicas is the same for all instances, $\#K_i$ will be written as $\#K$. Equation 3.4 presents the calculation.

$$\tilde{T}_{Ai} = \frac{\sum_{k \in K_i} \tilde{T}_{Aik}}{\#K} \tag{3.4}$$

The problem with $\tilde{T}_{Ai}$ is that it is not easy to interpret/compare. Only by comparing this value with other rules can we know if one rule performed well or not. So we introduced a rule B_i, a benchmark rule. For each instance, one rule B_i was chosen to perform this relative adjustment. In this way, it is easy to see how many times a rule is better or worse than other. The benchmark rules B_i are the rules used in [14] and [18]. The $\tilde{T}_{Bi}$ is then the minor value of mean Tardiness that all the benchmark rules obtained for that instance i (see Subsection 2.1.4 for an overview of the benchmark rules). So the relative mean Tardiness of an instance is computed as in equation 3.5.

$$\tilde{F}_{Ai} = \frac{\tilde{T}_{Ai}}{\tilde{T}_{Bi}} \tag{3.5}$$

The problem lies in how to group this metric associated with several different instances that can have severe disparities in magnitude. For example, if instance x has mean relative tardiness of 100 and instance v has a mean relative tardiness of 1.01, the instance v instance will be almost ignored when grouping the two. As the geometric mean handles better magnitude disparities than the arithmetic mean, we chose it. Furthermore, the geometric mean allows us to rewrite the final fitness equation in a way that allows us to separate the fitness computation of rule A from the benchmark rules, which is not possible with the arithmetic mean. So the fitness associated with rule A can be computed as in equation 3.6.

$$F_A = \sqrt[\#I]{\prod_{i \in I} \tilde{F}_{Ai}} = \frac{\sqrt[\#I]{\prod_{i \in I} \tilde{T}_{Ai}}}{\sqrt[\#I]{\prod_{i \in I} \tilde{T}_{Bi}}} \tag{3.6}$$

3.4 Statistically significance when comparing two rules

The algorithms are evaluated based on the fitness presented in the previous section. However, one important question is to know if the rules generated at the end are statistically different. This section presents the procedure that was followed in order to evaluate that.

3.4.1 Significance for one instance

When comparing two rules A and B in an instance, we have the mean Tardiness off all replicas on that instance for each rule. As the distribution seeds are equal, the same jobs appear for rules A and B for each replica k_i on instance i. Also, their difference represents the exact performance difference in the same job conditions. With these differences, a T-student test can be applied to see the statistical significance of the results.

So our variable of interest Z_i with samples $\{Z_{i1}, Z_{i2}, ... Z_{i\#K_i}\}$ is the variable where we want to apply the statistical test. Being Z characterized as in equation 3.7

$$Z_{ik} = \tilde{T}_{Aik} - \tilde{T}_{Bik}, \forall k \in K_i \tag{3.7}$$

We want to refute the null hypothesis H_0: Z_i has mean 0. The alternative hypothesis H_1: is Z_i has mean superior to 0. So a one side T-test has to be performed. After the test, a p-value is obtained.

With this p-value, it is easy to extrapolate the value of the test in the other direction. If instead, our variable was $Z'_{ik} = \tilde{T}_{Bik} - \tilde{T}_{Aik}$ the sample mean of Z'_{ik} is the symmetric of Z_i (equation 3.8). Furthermore, it is easy to see that the sample variance would be the same (equation 3.9).

$$mean(Z'_{ik}) = \frac{1}{\#K_i} \sum_{k \in K_i} \tilde{T}_{Bik} - \tilde{T}_{Aik} = -\frac{1}{\#K_i} \sum_{k \in K_i} \tilde{T}_{Aik} - \tilde{T}_{Bik} = -mean(Z_{ik}) \tag{3.8}$$

$$var(Z'_{ik}) = \frac{1}{\#K_i} \sum_{k \in K_i} (\tilde{T}_{Bik} - \tilde{T}_{Aik} - mean(Z'_{ik}))^2 = \frac{1}{\#K_i} \sum_{k \in K_i} ((-1)(\tilde{T}_{Aik} - \tilde{T}_{Bik} - mean(Z_{ik}))^2 = var(Z_{ik}) \tag{3.9}$$

With this in mind, the p-value associated with the inverse direction can be associated with the one obtained in the direct direction.

$$\begin{aligned} p' = Pr(Z' \geq mean(Z') \mid H0) = Pr(-Z \geq -mean(Z) \mid H0) = \\ Pr(Z < mean(Z) \mid H0) = 1 - Pr(Z \geq mean(Z)) = 1 - p \end{aligned} \tag{3.10}$$

In the rest of the thesis, only the test in one direction will be presented and its respective p-value. A p-value between 0 and 0.5 will be represented in graphs with green (stating rule B is better than A with that significance level) and from 0.5 to 1 with red (stating rule A is better than C with a 1-p significance level).

For instance, if the p-value is 0.03 and our significance level is 0.05, we reject the null hypothesis, and so the mean of $\tilde{T}_{Aik} - \tilde{T}_{Cik}$ is significantly greater than 0, so rule C is better than rule A (has lower mean tardiness). If the value is 0.98, this is equivalent to the p-value in the reverse direction being 0.02, so it is significant the opposite, and in this case, rule A is better than rule C in that instance.

3.4.2 Extending to all the instances

Assigning a single value representing the test significance between two rules is difficult, especially considering a score that aggregates several instances in a single metric. So two ways of comparing will be presented. One of them is a bar plot representing how many instances a rule is significantly better than the other in function of the desired confidence level, allowing to see if a rule is dominated by the other. The other is an alteration to the previous method that uses another test variable X with samples $\{X_{11}, X_{12} ... X_{1\#K_1} ..., X_{\#I\#K_{\#I}}\}$ that will be discussed here.

A naive approach to group the tardiness of all replicas of all instances in a single variable X could be: ignoring that the replicas are associated with the instance and simply considering the values of all the replicas independently of the instance.

$$X_{ik} = \tilde{T}_{Aik} - \tilde{T}_{Cik}, \forall i \in I, \forall k \in K_i \tag{3.11}$$

The problem with this approach is that an instance with big mean tardiness will have a lot more weight than an instance that does not have big mean tardiness. The instances with mean tardiness close to zero would be almost completely ignored from the analyses, especially comparing to a bigger one.

Due to that, a relative factor that takes the mean tardiness into account has to be considered. A good metric will guarantee that: if rule A performs 5 times better than rule B in instances 1 and 2, they will have the same relevance in the overall metric despite the differences in the mean tardiness of rule 1 and rule 2.

A way to do this is by dividing the previous variable X_{ik} by the mean tardiness of the replicas of the rule A and B, for that instance. So the new variable X_{ik} will represent how much a rule is better than the other in that replica with respect to the mean tardiness, for that instance.

$$X_{ik} = \frac{\tilde{T}_{Aik} - \tilde{T}_{Cik}}{1/(2\#K_i) \times (\sum_{u \in K_i} \tilde{T}_{Aiu} + \sum_{u \in K_i} \tilde{T}_{Ciu})}, \forall i \in I, \forall k \in K_i \tag{3.12}$$

The same test explained in the previous section is performed with this new variable, giving a single p-value indicating if rule C is better than A.

3.4.3 Variance of the fitness metric

In addition to comparing the performance of two expressions, it is important to understand the confidence around our fitness indicator, i.e., how much would the fitness metric change when repeating the experiments with different seeds? This question is essential to understand if it is clear that one DR with fitness x is better than the another with fitness x+e without the need to do a specific test. Standard genetic algorithms only compare fitness values. We should also consider how variability discrepancies between instances impact global fitness and how can we prevent that.

As we are using the geometric mean to compute our score we need two describe to things: The effect that multiplying random variables has in the result and the effect of subsequently taking the root.

Due to the fitness indicator around some scenarios being with confidence close to zero any log transformation to the data did not work. This log transformation could allow us to transform the geometric mean in a arithmetic which is easier to analyse.

Furthermore the method to compute the mean and variance of the product independent random variables did work to do that, but given that $E(x^{1/n}) \neq E(x)^{1/n}$ and its variance is to high, the second order approximation of $E(x^{1/n})$ was not restricted enough and did not represent the reality of the data. In fact, being x the product of the $\tilde{F}_{Ai}$ (for all i), being λ_i and σ_i^2 the mean and variance of each $\tilde{F}_{Ai}$, $E(x) = \lambda$ and $Var(x) = \sigma^2$. These two moments can be easily computed by the folowing formulas (3.13, 3.14) which are easy to demonstrate (as the scores of the instances are independent):

$$\lambda = \prod_i \lambda_i \tag{3.13}$$

$$\sigma^2 = \prod_i (\sigma_i^2 + \lambda_i^2) - \prod_i \lambda_i^2 \tag{3.14}$$

Defining a new variable d as the deviation of x in respect to its mean ($d = x - \lambda$) we have that $E(d) = 0$ and $E(d^2) = \sigma^2$ which allow us to aproximate $E(x^{1/n})$, our main score metric by its second order taylor series around λ. Also by calling our aproximation λ_p we get:

$$E(x^{1/n}) = E((\lambda + d)^{1/n}) \approx \lambda^{1/n} + \frac{1}{n}\lambda^{(1/n)-1}E(d) + \frac{1}{2}\frac{1}{n}(\frac{1}{n} - 1)\lambda^{(1/n)-2}E(d^2) \tag{3.15}$$

$$E(x^{1/n}) \approx \lambda^{1/n} + \frac{1}{2}\frac{1}{n}(\frac{1}{n} - 1)\lambda^{(1/n)-2}\sigma^2 = \lambda_p \tag{3.16}$$

Suppose that the second term of the approximation is negligible ($\sigma^2 << \lambda$), then the first-order approximation (which is the one we use when computing the score) is an excellent approximation to the expected score. An interesting observation is that our estimate for F_A is always superior to the expected value of F_A (we are using the first-order approximation).

If the second-order approximation is good enough, and λ_p is close enough to the actual value we can aproximate the estimator variance:

$$Var(x^{1/n}) \approx E((x^{1/n} - \lambda_p)^2) = E(x^{2/n}) - \lambda_p^2 \approx \lambda^{2/n} + \frac{2}{n}\lambda^{(2/n)-1}E(d) + \frac{1}{2}\frac{2}{n}(\frac{2}{n} - 1)\lambda^{(2/n)-2}E(d^2) - \lambda_p^2 \tag{3.17}$$

$$Var(x^{1/n}) \approx \lambda^{2/n} + \frac{1}{2}\frac{2}{n}(\frac{2}{n} - 1)\lambda^{(2/n)-2}\sigma^2 - \lambda_p^2 = \frac{\lambda^{(2/n-2)}\sigma^2}{n^2}[1 - \frac{1}{4}\frac{(1-n)^2}{n^2}\lambda^{-2}\sigma^2] \tag{3.18}$$

Unfortunately, if this condition does not hold, and worse, the other terms of the series are not negligible using this method to compute the variance of $E(x^{1/n})$ by just using first and second-order moments do not work (it underestimates the interval). Unfortunately, this was the case in our results. The second-order approximation for the confidence intervals of the indicator was two times smaller than the actual intervals (for some of our DRs). Therefore, the distribution of the global fitness score was computed directly from the definition as the other methods did not provide good enough approximations.

It is assumed that each $\tilde{F}_{Ai}$ (mean tardiness of an instance) can be modeled by a Truncated Gaussian distribution. Truncated Gaussian represents the maximum entropy probability distribution for variables with fixed means and variances, constrained to be in the interval [a,b]. In our case, the mean tardiness is constrained to be non-negative, i.e, in the interval $[0,+\infty]$.

The distribution of $C_A = \prod_i \tilde{F}_{Ai}$ was computed directly by the definition of the distribution of the product of random variables. If $Z = XY$,X and Y are always greater or equal to 0,the CDF of Y is F_Y,the PDF of X is f_X then the CDF of z (F_Z) can be computed as follows:

$$F_z(z) = P(z < Z) = \int_{x=0}^{+\infty} f_X(x) F_y(z/x) dx$$

This expression allows us to generate the next CDF used for the next product. Finally having the variable $z = C_A$ it only remains to characterize $F_A = y = C_A^{1/n}$. As $x^{1/n}$ is monotonically increasing, we have that: $P(z < Z) = P(y < Z^{1/n})$. This simple relationship allows to get the CDF of F_A and with that, getting confidence intervals is straighforward (the mean was also computed after calculating the PDF).

With the confidence intervals around a fitness, we can now just look at the fitness to know what rules are statistically better than the others. Furthermore, we can see which scenarios are undermining the score and how.

Capítulo 4

Expressions Enumerator

This chapter describes all the work done to create the expression enumerator. It is subdivided into three main parts: i) Expression Counting, where the mathematical derivations that allow us to compute the total number of possible expressions of a given size are shown; ii) Expression Enumeration, which explains how the DA enumerator was created; iii): Random Enumeration Analysys, where a simple approach to estimate the expected computational time evolution of the Enumerator is presented. This derivation is what is used in appendix B to generate the time performance graphs of the random Enumerator for comparison purposes. The Enumerator will reveal if GP and GGGP methods explore optimally small size expressions. If they do not, the Enumerator will also find better expressions.

4.1 Expression Counting

4.1.1 Problem definition

Let EXP^n_{TO} equals to every expression of length n using the terminals of set T and operations of set O, where O is subdivided in operators types, unitary operators O_1, binary operators O_2 etc. ($O = \{O_1, O_2, O_3\}$). For example let $O = \{\emptyset, \{+, -, *, /\}, \emptyset/\}$ and $T = \{a, b, 3\}$, an expression of the set EXP^5_{TO} can be in prefix notation [-c+ab] that translates in c- (a+b). The first question would be: what is the dimension of EXP^n_{TO} ($\#EXP^n_{TO}$)?

4.1.2 Analysis

4.1.2.1 Upper bound

It follows here the calculation analysis of these spaces. Let S, the set containing N symbols in which any sequence of operation and symbols would be valid (all strings are valid in this case): for instance, the [–+-a] (this is an example with N=5) is valid in this first analysis. We know that for each of the N symbols to choose, the number of different values that can be there is equal to the number of terminals plus the number of operators. In conclusion, we know that the number of strings with this form is (this will be an upper bound to all of our subsequent analysis):

$$\#S = (\#O_1 + \#O_2 + \#O_3 + \#T)^N \tag{4.1}$$

4.1.2.2 Analysis for O_2

From [54] it is shown that the number of full binary trees with b+1 leaves is the Catalan number C_b. Furthermore, a full binary can represent the applications of successive binary operators. A binary tree is the standard representation of GP (with only binary operators); each leave is a terminal, and the nodes are the binary operators. Moreover, b+1 leaves require b nodes for the tree to be completed, so the total size of the tree is n = 2b+1 (total size of the expression). After attributing a terminal to the b+1 terminals and an operator to the b operators of each tree, we can reach a complete formula for the number of expressions. There are $\#T^{b+1} = \#T^{(n+1)/2} =$ ways of attributing terminals to b+1 terminals. And there are $\#O_2^b = \#O_2^{(n-1)/2}$ ways of attributing a operators to b+1 operators. This analysis allows to reach the following formulas for n odd, for n even there are 0 expressions:

$$\#EXP^n_{TO_2} = Cat(\frac{n-1}{2})\#T^{(n+1)/2}\#O_2^{(n-1)/2} \tag{4.2}$$

$$\#EXP^n_{TO_2} = \frac{2}{n+1}\binom{n-1}{\frac{n-1}{2}}\#T^{(n+1)/2}\#O_2^{(n-1)/2} \tag{4.3}$$

Asymptotically using Stirling approximation:

$$\#EXP^n_{TO_2} \asymp \frac{2^n\sqrt{2}}{(n-1)^{3/2}\sqrt{\pi}}\#T^{(n+1)/2}\#O_2^{(n-1)/2} \tag{4.4}$$

Exploring the expression, going from n to n+2 causes a $4\#T\#O_2$ increase in the total number of expressions.

The product $(\#T\#O_2)$ is essential to keep the solution space manageable. A more in-depth analysis can be performed and making $\#T + \#O_2 = K$ (a constant), the values that give the biggest number of possible solutions is when $\#T = \#O_2 + 1$, this can be seen by differentiating the first expression in regards to $\#O_2$ (substituting first $\#T$ by $K - \#O_2$) and finding when the derivate is 0. This is an exciting result that says that if the number of possible terminals and possible operations to choose is close to each other, the solution space is bigger than when they are very different. It follows a numeric example:

Considering the first example: $O_2 = \{+,-,/,\times,max,min\}$, $T = \{a,b\}$. And the second example: $O_2 = \{+,-,/,\times\}$, $T = \{a,b,c,d\}$,$O_2 = \{+,-,/,\times,a,b,c,d\}$. Figure 4.1 compares the sizes of the two examples in a log scale and their relative size. Interestingly the space for n=20 of the second example is 25 times larger than the first one. So, keeping the number of operators as different as possible than the number of terminals can be useful.

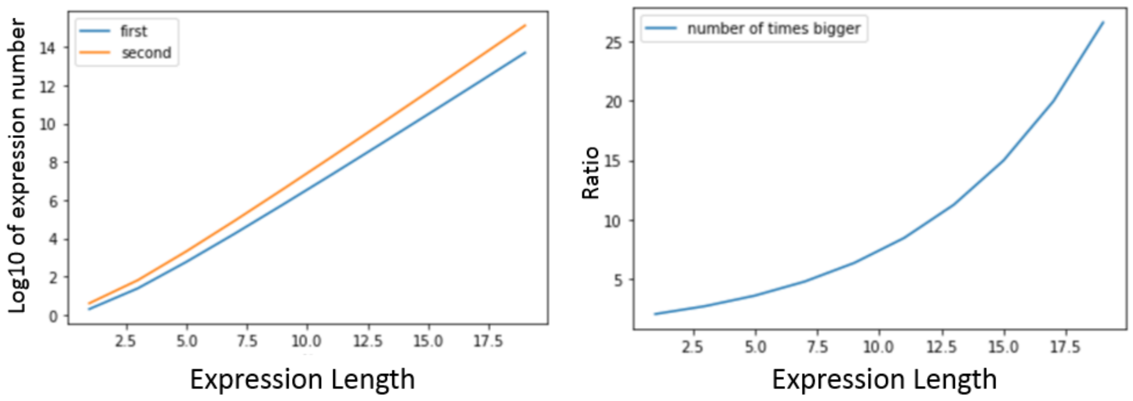


Figura 4.1: Comparison of the solution space for the $\#O_2$ case. The left graph first represents the total number of expressions using 6 operators and 2 terminals, and second the total number of expressions using 4 operators and 4 terminals. On the right is presented the division of second over first.

4.1.2.3 Analysis for O_2 and O_1

To help us counting this space, we will base our explanation in Figure 4.2. In this example, expressions of length 7 are being generated from expressions only with binary operators with dimension 5 and 3. One important consideration is that the last element of a valid expression cannot ever be a unitary operator. In the first example, we can also see that we have to choose 2 places out of 6 to put the unitary operators and in the second one 4 out of six. Generally being i the length of the originator expression only with binary operators we see than the number of possibilities to choose the positions of the unitary operators is $\binom{n-1}{n-i+1} = \binom{n-1}{i-1}$. These considerations allow us to write the final expression 4.6.

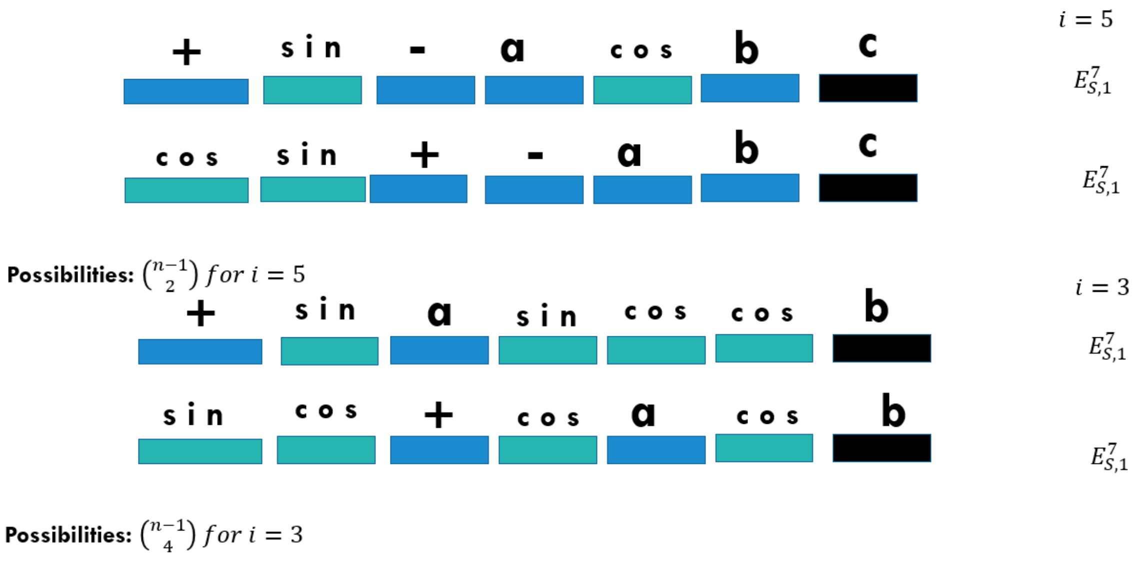


Figura 4.2: Counting expressions with unitary operators.

$$\#EXP_T^n O_{2,1} = \sum_{i=1}^{n} \#EXP_{TO_2}^i \binom{n-1}{i-1} u^{n-i} \tag{4.5}$$

$$\#EXP_T^n O_{2,1} = \sum_{i=3,odd}^{n} \frac{2}{i-1} \binom{i-1}{\frac{i-1}{2}} \#T^{(i+1)/2} \#O_2^{(i-1)/2} \binom{n-1}{i-1} u^{n-i} \tag{4.6}$$

It is not clear if there are fewer or more expressions when using unitary operators than with only binary operators. Numerical examples and charts can show that there is an inversion in the behavior of the expression.

Here are some concrete examples that demonstrate this inversion and behavior. Observing Figure 4.3 is clear that for expression lengths inferior to 25, there are fewer expressions with 7 unitary operators and 13 binary operators than with 20 binary operators. With bigger lengths, the opposite conclusion can be drawn. There is an inversion. Figure 4.4 uses only 4 operators. The number of expressions with only binary operators is superior to the one with unitary in all instances.

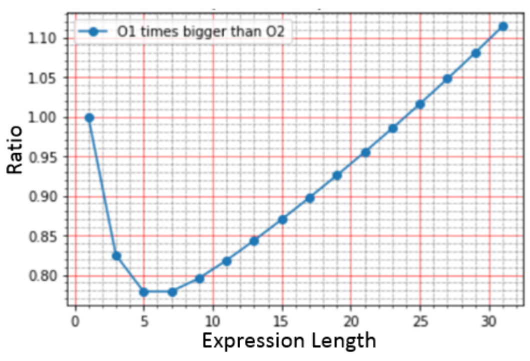


Figura 4.3: Comparison of the number of expressions for 20 operators. Test O2 has only binary operators (20 binary operators), Test O1 has binary and unary operators (13 binary operators and 7 unitary).

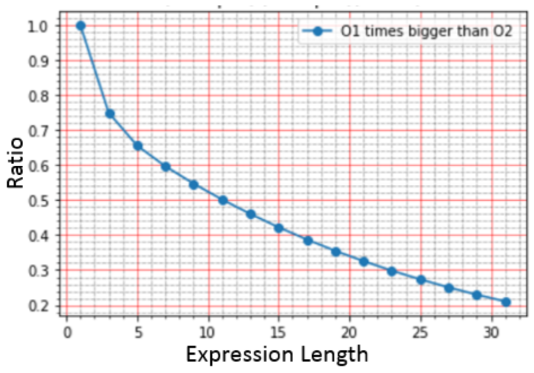


Figura 4.4: Comparison of the number of expressions for 4 operators. Test O2 has only binary operators (4 binary operators), Test O1 has binary and unary operators (2 binary operators and 2 unitary).

4.2 Expression Enumeration

4.2.1 Enumeration algorithm

The algorithm was developed in python, and to check if it was working correctly, the number of expressions it enumerated had to be equal to the previous analytical analysis. The initial enumeration was made in the following way: from the previous section, the following equation can be subdivided into three main terms, $Cat(\frac{n-1}{2})$ referring to the possible basic structures of one expression (for size five, there are only two basic structures [Op Op Ter Ter Ter] or [Op Ter Op Ter Ter]). The term $\#T^{(n+1)/2}$ represents how many ways the terminals can be inserted in a basic structure. Furthermore, the term $\#O_2^{(n-1)/2}$ represents how many ways the operand can be inserted in a basic structure.

$$\#EXP_{TO_2}^{n} = Cat(\frac{n-1}{2})\#T^{(n+1)/2}\#O_2^{(n-1)/2} \tag{4.7}$$

First, to perform the enumeration, a recursive procedure (algorithm 2) that generates all the basic structures was created.

Algorithm 2 Enumeratic Basic Structure length n

```
1:  $expressionList \leftarrow [\,]$
2:  $expression \leftarrow [\,]$
3:  $sizeToadd \leftarrow n$
4:  **procedure** RECURSIVEENUMERATION(sizeToadd,expressionList )
5:      **if** $sizeToadd = 1$ **then**
6:          $newExpressions \leftarrow [\,]$
7:          **for all** exp in expressionList **do**
8:              $newExpressions.append(exp.append(Ter))$
            **return** newExpressions
9:      **if** $sizeToadd! = 1$ **then**
10:         $newExpressions \leftarrow [\,]$
11:         **for all** exp in expressionList **do**
12:             exp.append(Op)
13:         **for** $i = 1; i < sizeToadd; i+ = 2$ **do**
14:             $FirsExpParts = RecursiveEnumeration(i, expressionList)$
15:             $CompleteExpPart = RecursiveEnumeration(sizeToadd - i - 1, FirsExpParts)$
16:             $newExpressions.append(CompleteExpPart)$
            **return** newExpressions
```

The algorithm is recursive, and, at each point of expansion, it has a list of the previously completed part of all expressions and a *sizeToadd* (the size to add of terminals and operators) for instances assume that during the algorithm, the expression list is empty [[]] and the *sizeToadd* is 5. The algorithm begins by inserting as the first element an operand [[Op]]. Now there are two possibilities, or the first operand has length 1 and the second operand length 3, or the first operand has length 3 and the second operand length 1. *RecursiveEnumeration* is called two times for each possibility, the first to give all possible structures for the first operand and the second to complete the expression. So for the first case (the first operand has length 1 and the second operand length 3), the algorithm would return *FirsExpParts* = [[op ter]], and then the second call would complete with the only possible variation of size 3. *CompleteExpPart* = [[Op Ter Op Ter Ter]]. To the second variation, the opposite would occur, and the final expression would be [Op Op Ter Ter Ter]. Adding these two together gives the final result: *newExpressions* = [[Op Op Ter Ter Ter],[Op Ter Op Ter Ter]]. This algorithm produces expressions in the prefix notation format.

With the enumeration of the basic structures, it is now necessary to complete the enumeration with a given set of terminals and operators. For this, a simple procedure was developed. Having one of the basic structures, for instance [Op Ter Op Ter Ter]. We know that we can substitute Ter in $\#T^{(n+1)/2}$ ($n = 5$) ways and Op in $\#O^{(n-1)/2}$ ($n = 5$) ways. So two numbers are sufficient to characterize any expression of a basic structure. T_i referring to the chosen combination for all the Ter (there are $\#T^{(n+1)/2}$ possibilities), and O_i referring to the chosen combination of Op (there are $\#O^{(n-1)/2}$ possibilities). Now it is necessary to map these numbers to expressions. One possible mapping is the following (algorithm 3), which is equivalent to how to represent a number in a given base.

Algorithm 3 Mapping

```
1:  procedure MAPPING(lenght,terminals,operands,Ti,Oi )
2:      chosenOpList ← []
3:      chosenTerList ← []
4:      numOp ← length(chosenOp)
5:      numTer ← length(chosenTer)
6:      for $i = 0; i < (lenght - 1)/2; i++$ do
7:          $id = (Oi \div (numOp^i)) \mod numOp$
8:          $chosenOp = operands[id]$
9:          $chosenOpList.append(chosenOp)$
10:     for $i = 0; i < (lenght + 1)/2; i++$ do
11:         $id = (Ti \div (numTer^i)) \mod numTer$
12:         $chosenTer = terminals[id]$
13:         $chosenTerList.append(chosenTer)$
        return [chosenOpList,chosenTerList]
```

For example suppose that $terminals = [P, U]$, $numTer$=2 and $operands = [+, -, /]$, $numOp$=3 and we want to find the expression of lenght 5 refering to $Ti = 3$ and $Oi = 5$. So if it has lenght 5 there are 3 Terminals to chose, the first is the less signficant and can be calculated as $3\%2 = 1$ so it is chosen U as the first terminal($terminals[1] = U$) , the second terminal id is computed as: $(3/2)\%2 = 1$ so it is chosen U as well, and finnaly the final terminal id is $(3/2^2)\%2 = 0$, so it is referred to P. so the Terminals chosen are [U,U,P]. Simmilary for the operands we have to choose two operands, the first id is $5\%3 = 2$ refering to the operator ($operators[2] = /$) , and the second one is $5/3\%3 = 1$ refering to the operator ($operators[1] = -$). So the procedure returns $[[UUP], [/-]]$.

Finally, the procedure to merge a basic structure to the terminals chosen and operators for that expression is immediate. It is only necessary to transverse the expression and substitute for the chosen terminals and operators. For example, assume that the basic structure is [Op Ter Op Ter Ter]. Merging with the previous example's chosen elements gives $[/U - UP]$. The first ter was substituted by the first chosen terminal of the terminal list, the second ter by the second terminal until every ter is substituted. Repeating this also for the op will complete the merge.

So the Complete algorithm to return all the expressions is algorithm 4:

Algorithm 4 CompleteAlgoEnum

```
procedure ENUMERATION(n,terminals,operands)
    numOp ← length(chosenOp)
    numTer ← length(chosenTer)
    $maxTi = numTer^{(n+1)/2}$
    $maxOi = numOp^{(n-1)/2}$
    $BasicStructures = RecursiveEnumeration(n, [])$
    ChosenExpressions ← []
    for all struct in BasicStructures do
        for $Ti = 0; Ti < maxTi; Ti++$ do
            for $Oi = 0; Oi < maxOi; Oi++$ do
                $chosenElements \leftarrow Mapping(n, terminals, operands, Ti, Oi)$
                $exp \leftarrow Merge(chosenElements, struct)$
                if We wanna keep this expression then $ChosenExpressions.append(exp)$
     return ChosenExpressions
```

4.2.2 Filtering out expressions

This section aims to find all expressions, dimensionally aware expressions, unique expressions, and unique and dimensionally aware expressions. All these different behaviors can be represented by adding logic to line 13 of algorithm 4.

Firstly, by not restricting any expression by making line 13 equal to $if(1)$, all expressions are saved and returned.

The following method was used to trim down which expressions were Dimensional Aware. First, a dimensional vector was attributed to each terminal. For example: if $T = \{Tim, D\}$ and if Tim is a variable that represents time and D a variable that represents the distance, a possible dimensional vector for Tim can be [1, 0] and for D can be [0,1], if there were a terminal that is a velocity its dimensional vector is [-1, 1]. The dimensional vector's length is equal to the number of different dimensions considered (in this case, we had only 2: distance and time). In evaluating an expression, if the operation performed was a multiplication, the operands' dimensional vectors were added. If it was a division, they were subtracted. If the operation was a sum or a subtraction, the operation was performed if the dimension vectors were equal. If not, the expression is not Dimensional Aware, and 0 is returned.

The following method was used to find which expressions were unique. Each expression was evaluated for 1 set of distinct float inputs, and if the result is not on the hash table, the result is added to it. The inputs were chosen to be extremely difficult for two non-equivalent expressions to give the same result when evaluated. After Computing the result of an expression with the inputs, this result is rounded up at the eighth decimal place, and two things can occur:

- The hash table already contains an entry with that result. In that case, the table already contains a saved equivalent expression, so this one is discarded. Furthermore, 0 is returned.

- The hash table does not contain an entry with that result. In that case, no expression on the table is equivalent to this one. Furthermore, we save that expression in that hash entry, and 1 is returned.

This method does not guarantee to have found every different expression. However, it is improbable for two different expressions to give the same result with the same inputs up to the eighth decimal place. What happens more often is two equivalent expressions to give different results due to rounding errors (this is why the hash entry is rounded to the eighth decimal place to decrease this error). However, this error is not very important because it is essential to test all / almost all expressions. If the same expression is several times evaluated is not detrimental to the results.

This method's advantage is that the complexity of checking if an expression is unique is O(1) and can be made in a single inexpensive hash table access. This simplicity is essential when analyzing millions of expressions.

4.2.3 Terminals and operators selection

The number of terminals and operators can be large in GP. Due to that, it is necessary to trim them down if we want to use an enumerator. We will base out chosen terminal/operator set on the terminals/operators chosen in [6]. The operators were the common addition, subtraction, multiplication and protected division [+,-,*,/], and the terminals will be developed next.

In this thesis, only atomic terminals will be considered, except for the critical ratio if positive (CR^+). An atomic terminal is a fundamental terminal, i.e., not a composition of other atomic terminals. This exception was made because for tiny expressions (length 5-7), the Terminal CR^+ appears to add the necessary expression complexity to learn good rules. Almost all the best rules have a CR^+ in them in those ranges, as we will further see. The CR^+ can be obtained by the division between the Allowance if positive (A^+) and the remaining processing time (RPT).

Table 4.1 presents all the atomic terminals considerer in [6] as well as their abbreviation used in these work when needed. However, as eight terminals are still too much, further pruning was performed. As the best subset of terminals used in Table 4.1 did not use NPT and U as terminals, they were removed from our analysis.

When the expressions' length became too large to be computationally viable to do the enumeration with those parameters, some terminals and even operators were removed. The removal criterion was if they seem to not appear in the best solutions. The parameters used in the enumerations for all expression sizes can be seen in Table 4.2.

Also, in Annex A, a method is tested that uses the enumerator to trim down systematically bigger sets of initial terminals. The advantage of using one instead of GP is that, unlike GP, the enumerator explores better and more uniformly and does not repeat expressions.

Tabela 4.1: List of atomic terminals and their used abbreviation.

Terminal	**Abbreviation**	**Description**	**Units**
PT	P	Processing time of current operation	time
WINQ	W	Work in the next queue	time
NPT	N	Processing time of next operation	time
U	U	Shop utilisation	adimensional
S^+	S	Slack if positive	time
A^+	A	Allowance if positive	time
CR^+	C	Critical ratio if positive	adimensional
RPT	R	Remaining processing time	time

Tabela 4.2: Terminals and operators used in the enumeration

Terminals	**Expression Sizes**						**Operators**	**Expression Sizes**					
	1	3	5	7	9	11		1	3	5	7	9	11
PT	x	x	x	x	x	x	+	x	x	x	x	x	x
WinQ	x	x	x	x	x	x	*	x	x	x	x	x	x
S^+	x	x	x	x	x	x	/	x	x	x	x	x	x
RPT	x	x	x	x	x	x	-	x	x	x	x		
CR^+	x	x	x	x	x								
A^+	x	x	x	x									

4.2.4 Defining feasible sizes

For the small sizes, 1-7, all six terminals, and four operators were used (c.f Table 4.2). It follows in Table 4.3 the total number of expressions for each size and each expression characteristic. There is much redundancy in the total expression number. It is evident by the difference in relation to *ALLEXP* and *UNIQUE*. Interestingly, there is a big difference for the expression length 9 of considering the intersection between the UNIQUE and DIM AWARE expressions and only considering the UNIQUE ones. The number of expressions to evaluate goes from 1.5 million to 264 thousand. So considering expressions both *UNIQUE* and *DIMAWARE* is a lot less expensive than considering *UNIQUE* expressions. This indicates that dimensional awareness can help to decrease search spaces. Another intake is that the growth rates from one size to the next are exponential even with the restrictions, and the growth rate of Unique and Dimensional Aware expressions is a lot smaller than the global rate. Therefore, for big expression lengths, the enumeration algorithm will be slow. As we have to transverse all expressions, the time complexity is on the order of *ALLEXP* and not of *UNIQUE* and *DIMAWARE* expressions. So the enumeration will eventually, for big enough expressions, be slower than the actual simulator. It was not the case as the bottleneck was the simulator that took around 6 minutes to evaluate an expression on the test instances.

Unfortunately, evaluating 260 thousand expressions is still too expensive. So further pruning was performed for size 9. Seeing the best expressions for the enumeration of sizes 7 in the test set (Table 4.8 and Table 4.7) it becomes clear that the operator - rarely appears in the best expressions.

Tabela 4.3: Number of expressions (N) and grow rate (GR) for the smallest expressions sizes. M stand for millions, m for thousands.

EXP Length	9		7		5		3		1	
Type	N	GR	N	GR	N	GR	N	GR	N	GR
All	27.8M	67.2	414m	60	6.91m	48	144	24	6	-
Unique	1.50M	25.87	57.8m	24.90	2.32m	22.33	104	17.33	6	-
Dim Aware	9.32M	48,40	192m	43.83	4.39m	35.42	124	20.67	6	-
Unique Dim Aware	264m	15.04	17.7m	14.45	1.23m	13.82	89	14.83	6	-

Further analyses were made, and in Table 4.4 is presented which symbols the y% better expressions used. From this Table, it is clear that the number of minuses appearing in the top 1%,10% is inferior to the other operands. Due to that, this operator was removed from the operator set for the bigger sizes. Further pruning was performed, and A^+ was removed from the terminal set because it was the terminal that appeared fewer times in the best ten expressions. It also appeared to have similar behavior with S^+. Very similar formulas that only changed S^+ per A^+ have a close score, so the amount of information lost is not too significant.

Tabela 4.4: Number of symbols on the top x% best expressions of size 7 in the the test set.

Size 7	Symbols									
	+	*	/	-	P	W	R	S	C	A
1%	202	224	87	**21**	259	108	60	86	123	76
5%	1079	856	503	**229**	813	648	409	550	650	486
10%	2020	1531	1217	**566**	1178	1114	1007	1207	1371	1235
20%	3485	3100	2632	**1448**	2116	2314	1973	2564	2502	2751
50%	6807	6541	8699	**4611**	5607	6243	5341	6180	5616	6557
100%	10857	11416	17469	13568	11933	11928	11883	11833	10624	12879

With this parameters for size 9 the number of expressions are presented in Table 4.5:

Tabela 4.5: Number of expressions of length 9.

	Size 9
All	3543750
Dim Aware Unique	24891

Further pruning is required for size 11 (because even with the last parameters, there would be around 300 thousand expressions). CR^+ was removed due to being a compound terminal, and length 11 begins to be long enough to reach the necessary complexity without requiring a CR^+ as terminal. Furthermore, the best two expressions do not use it.

With this parameters for size 11 the number of expressions are presented in Table 4.6:

Tabela 4.6: Number of expressions length 11.

	Size 11
All	41803776
Dim Aware Unique	60020

4.2.5 Best Expressions found

It follows the best-obtained expressions on the test and training sets for each expression length. The terminal will be substitute with their smaller abbreviatures in the following Tables. In Tables (4.7, 4.8 and 4.9) the best expression of the enumerative method can be seen. Also the histograms with the fitness distributions for all sizes can be seen in Figures (4.5, 4.6, 4.7, 4.8, 4.9, 4.10).

Size 1 is not sufficient to create good rules.

Size 3 has one rule that is a lot better than the others (C*P). However, CR^+ is a compound terminal, so size 3 is insufficient to learn good rules. The number of expressions is still insufficient to understand the behavior of the histogram.

Size 5 has one rule that is better than the others (C*P + W). Here the terminal CR^+ seems crucial as it appears in almost all best expressions. The histogram is bimodal, with a mode in 4.4 and the other in 18. The mean fitness increased in relationship with the previous case.

For Size 7, the best rule is (C*P+W+P). Here the terminal CR^+ seems crucial as it appears in almost all best expressions. The histogram is bimodal, with a mode in 4.2 and the other in 19. The mean fitness increased in relationship with the previous case. As we will see when dropping the -, the histogram ceases to be bimodal.

For Size 9, the best rule is ((S+R)*(W+P)/R). Here the terminal CR^+ seems to lose relevance as many expressions in the top 10 do not use it. The histogram now has only one mode, mean fitness 5.95 (decreasing in relationship with the other). So removing the - as operator increase the mean performance of the expressions.

For Size 11, the best rule is (S+R)(W+P)/(R+W). The mean increased concerning the other case. Interestingly, having WinQ + something tends to be the most usual expression format from the best ones. However, the best one does not follow this pattern. In chapter 5, when evaluating and comparing, and explaining the best expressions found by the different methods, these expressions will be further analyzed.

Furthermore, there were expressions in the ten best expressions of the test set for all sizes on the training set. Moreover, in particular, for sizes 5 and 7, the best expression is the same. This means that an evolutionary algorithm exploring the training set optimally would also give the optimal results on the test set for these sizes. Furthermore, the best expression in the training set for all sizes also appears in the top 10 in the test set, and the score in the test set will represent the cap that an algorithm learning optimally on the training could reach in the test set. For example, the best expression of size 11 in the training set appears in position 10 on the test set, so exploring the training set optimally would discover that expression in the best exploratory scenario and its score would be 0.8168.

Tabela 4.7: Top 10 expressions for sizes 1, 3, 5 in the training and test sets. In bold expressions that belong to the top 10 both in the test and training sets

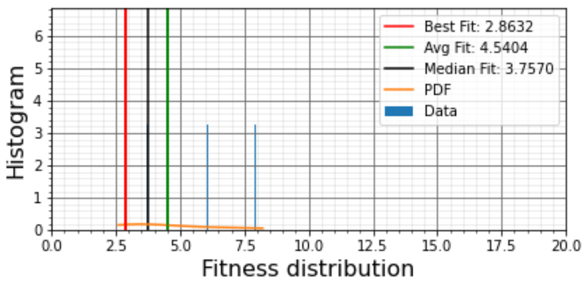
Test Set		**Training Set**		**Test Set**		**Training Set**		**Test Set**		**Training Set**	
Best Exp	Fit	Best Exp	Fit	Best Exp	Fit	Best Exp	Fit	Best Exp	Fit	Best Exp	Fit
C	2.86	**W**	1.63	**C*P**	1.47	**P+W**	1.20	**C*P+W**	0.928	**C*P+W**	0.924
W	2.87	**C**	2.60	**S+W**	2.35	P*W	1.60	**C*P+P**	1.102	**C*P+P**	1.130
S	3.70	**P**	2.81	**P+W**	2.36	R*W	1.62	**C*(P+W)**	1.276	P+W+W	1.194
A	3.82	**A**	3.27	S*P	2.46	W*W	1.63	P*A/R	1.474	P+P+W	1.209
P	6.03	**S**	3.33	A*P	2.52	W+W	1.63	C*(P+P)	1.474	P*(P+W)	1.299
R	7.96	**R**	4.01	S+P	2.61	**C*P**	1.67	C*C*P	1.554	P+C*W	1.304
				A+W	2.75	**S+W**	2.07	C*P*P	1.607	**C*(P+W)**	1.401
				R*W	2.86	W/P	2.13	S*P/R	1.622	R*(P+W)	1.461
				C*C	2.86	S+P	2.15	P*(S+P)	1.663	P*(S+R)	1.522
				C+C	2.86	C*W	2.23	P*(S+W)	1.674	W+P/C	1.534

Tabela 4.8: Top 10 expressions for sizes 7,9 in the training and test sets. In bold expressions that belong to the top 10 both in the test and training sets.

Test Set		**Training Set**		**Test Set**		**Training Set**	
Best Exp	Fit	Best Exp	Fit	Best Exp	Fit	Best Exp	Fit
C*P+W+P	0.850	**C*P+W+P**	0.785	**(S+R)*(W+P)/R**	0.810	**C*P+W+P+P**	0.778
C*P+W+W	0.903	**C*P+W+W**	0.866	**W+P+S*P/R**	0.816	**(W+P)*(P+C*P)**	0.780
P*(C+W/P)	0.925	**P*A/R+W**	0.927	**C*P+W+W+P**	0.833	**P*(C+P/P)+W**	0.786
P*A/R+W	0.927	**P*(C+W/P)**	0.933	**C*(W+P)+W+P**	0.844	**C*P+W+W+P**	0.786
C*(W+P)+W	0.951	**C*(W+P)+P**	0.943	**P*(C+P/P)+W**	0.852	**W+P+S*P/R**	0.789
P*S/R+W	0.961	**P*(C*P+W)**	0.944	**C*P+W+P+P**	0.864	**(S+R)*(W+P)/R**	0.790
C*(W+P)+P	0.968	**C*(W+P)+W**	0.949	**P*(C+P/R)+W**	0.872	**C*(W+P)+W+P**	0.791
C*(P+P)+W	0.984	**P*S/R+W**	0.968	**P*(S+P)/R+W**	0.893	**P*(C+P/R)+W**	0.832
P*(C*P+ W)	1.045	**C*(P+P)+W**	1.039	C*(P+P)+P+W	0.901	(W+P)*(W+C*P)	0.840
P+P*S/R	1.074	(S+R)*(W+P)	1.041	**P*(C+W/P)+W**	0.911	**P*(S+P)/R+W**	0.847

Tabela 4.9: Top 10 expressions for sizes 11 in the training and test sets. In bold expressions that belong to the top 10 both in the test and training sets.

Test Set		**Training Set**	
Best Exp	Fit	Best Exp	Fit
(S+R)*(W+P)/(R+W)	0.7920	**W+P+S*P/(R+R)**	0.7615
P*(S+R)/(R+R)+W	0.8079	**(S+R)*(W+P)/(R+W)**	0.7628
S*P/R+W+W+P	0.8089	**S*P/R+W+W+P**	0.7648
S*(W+T)/R+W+P	0.8100	**P*(S+R)/(R+R)+W**	0.7660
W+P*(S+R+W)/R	0.8101	(S+R)*(W+P)/R+P	0.7682
(S+R)*(W+P)/(R+R)	0.8104	**W+P*(S+R+W)/R**	0.7719
(S+R)*(W+P+P)/R	0.8105	P*(S+R)*(W+P)/R	0.7722
W+P*(S/R+P/P)	0.8130	W+P+P+S*P/R	0.7778
(W+P)*(S+S+R)/R	0.8150	W+(S+R)*(W+P)/R	0.7809
W+P+S*P/(R+R)	0.8168	W+P+S*(W+P)/R	0.7828



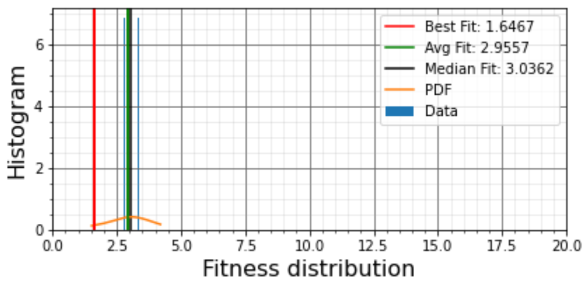
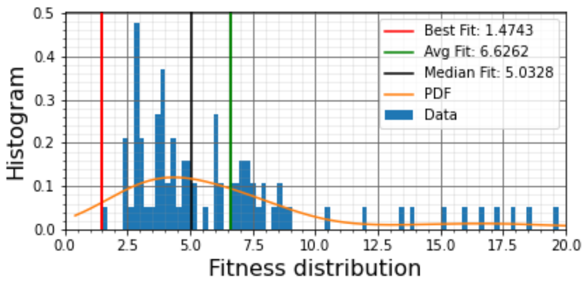


Figura 4.5: Fitness distributions for expressions of size 1, test on the left training on the right.



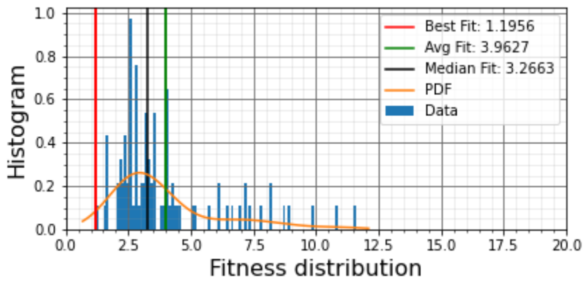
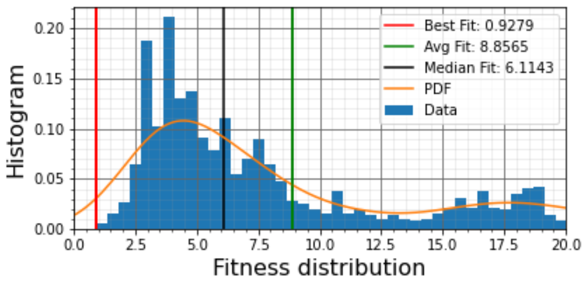


Figura 4.6: Fitness distributions for expressions of size 3, test on the left training on the right.



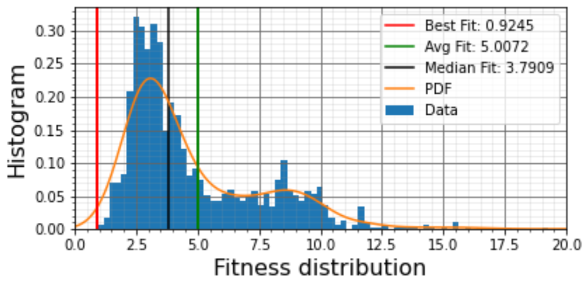
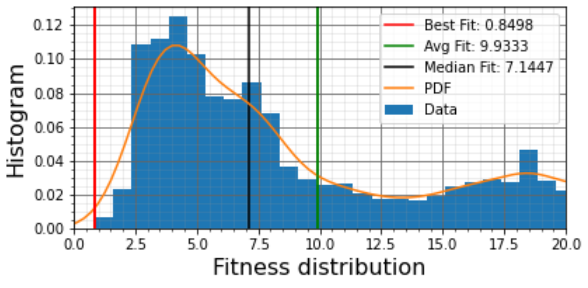


Figura 4.7: Fitness distributions for expressions of size 5, test on the left training on the right.



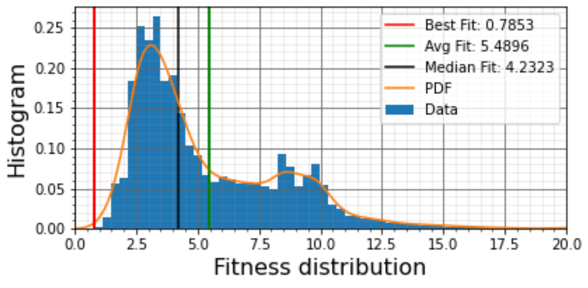


Figura 4.8: Fitness distributions for expressions of size 7, test on the left training on the right.

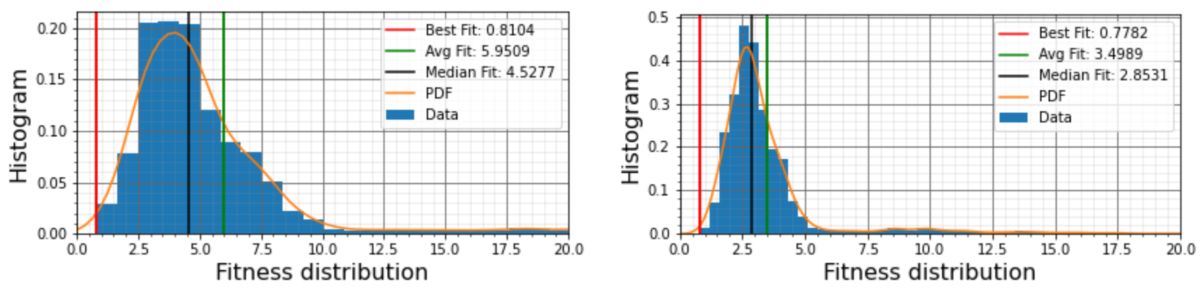


Figura 4.9: Fitness distributions for expressions of size 9, test on the left training on the right.

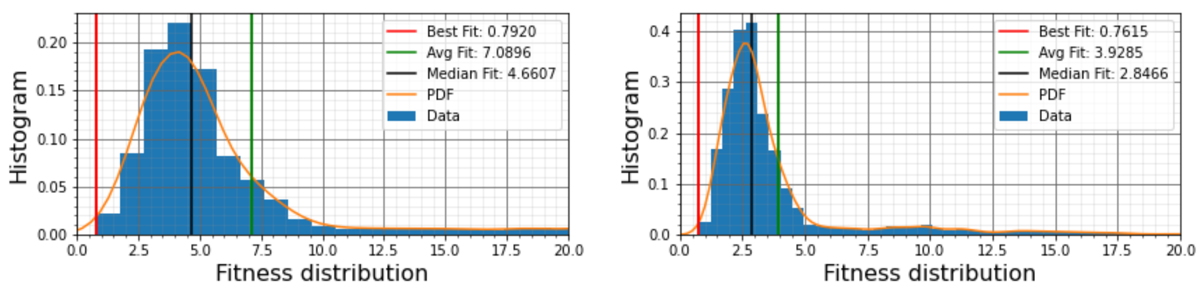


Figura 4.10: Fitness distributions for expressions of size 11, test on the left training on the right.

4.3 Random Enumeration Analysis

The analysis would only considerer the expressions with length 11 enumerated previously. The main reasons is that this is the test with more expressions (60021), which achieved the best results.

With the fitness list for all expressions of size 11, a cumulative distribution will be created $F_X(fit)$, which allows creating the cumulative fitness distribution of the best expressions for a given number of expressions n, $F_X(fit,n)$.

With this information, it is possible to get for each value of n the confidence interval for the fitness of the best expression. By plotting these intervals in function of n (number of evaluated expressions until the moment), we get a chart representing the expected score of the best expressions during the evolution.

This analysis is done entirely on the enumeration of the training dataset because we want to study how the evolution speed varies between the different algorithms. Furthermore, this will be used as a benchmark for the rest of the algorithms.

4.3.1 Cumulative fitness distribution of the best expressions

$F_X(fit,n)$ will depend on our sample technic. We will considerer the simplest case, in which we can re-chose previously picked expressions (sample with replacement).

F_n is the fitness of the n'th expression evaluated, and it is a random variable with cumulative function distribution $F_X(fit)$. The main objective is study the behaviout the expression with best fit (the minor fit). So the variable of interest is study $M_n \sim F_M(fit,i)$ and $M_n = min(F_1, F_2.....F_n)$. The question is now what is $F_M(fit,i)$.

$F_M(fit,i)$ can be easily related with $F_X(fit)$, assuming independence between draws (no replacement) in the following way.

$F_M(fit,n) = P(M < fit) = 1 - P(M \geq fit) = 1 - P(F_1 \geq fit \cap F_2 \geq fit..\cap F_n \geq fit) = 1 - P(F_1 \geq fit)^n = 1 - (1 - P(F_1 < fit))^n = 1 - (1 - F_X(fit))^n$.

With the cumulative distribution getting the 5% and 95% confidence intervals around the expected best fitness is straightforward.

4.3.2 Algorithms convergence

In Figure 4.11 the cumulative distribution for the best fitness is plotted for a different number of chosen expressions. Note that for $n = 1$, this is equivalent to the actual cumulative distribution of the fitness for one expression. In Table 4.10 and 4.11, the expected value of the best fitness and the probability of the best fitness is inferior to one is also presented for the training and test set, respectively.

As we can see by sampling 200 expressions, it is expected that the best found expression to have a fitness inferior to one with 33% of probability. With 10000 sampled expressions, this probability is almost one. In Figure 4.12 we can see 90% confidence intervals for the expected evolution of the fitness of the best expressions.

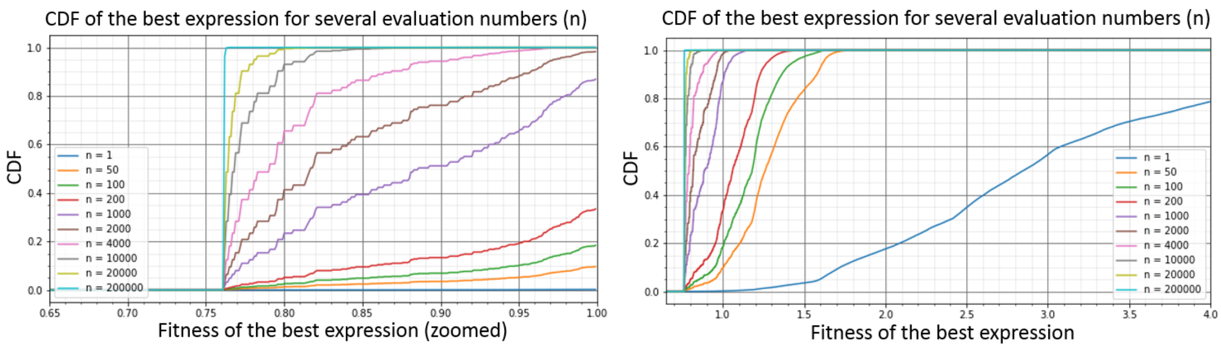


Figura 4.11: Cumulative distribution of the fitness depending on the number of evaluated expressions (n). On the left only shows until fitness 1 on the right until 4.

Tabela 4.10: Some metrics of the random enumerator in function of the number of evaluated expressions in the test set

	number of evaluated expressions							
	1	50	100	200	1000	2000	4000	10000
Expected Value	7.090	1.577	1.328	1.147	0.926	0.875	0,839	0,813
Probability Best Fitness <1	0.16%	7,99%	15,35%	28,35%	81,11%	96,43%	99,87%	99,99%

Tabela 4.11: Some metrics of the random enumerator in function of the number of evaluated expressions in the training set

	number of expressions							
	1	50	100	200	1000	2000	4000	10000
Expected Value	3.929	1.2741	1.1572	1.0624	0.8947	0.8418	0.8025	0.7753
Probability Best Fitness <1	0.2%	9,67%	18,41%	33,43%	86,92%	98,29%	99,97%	99,99%

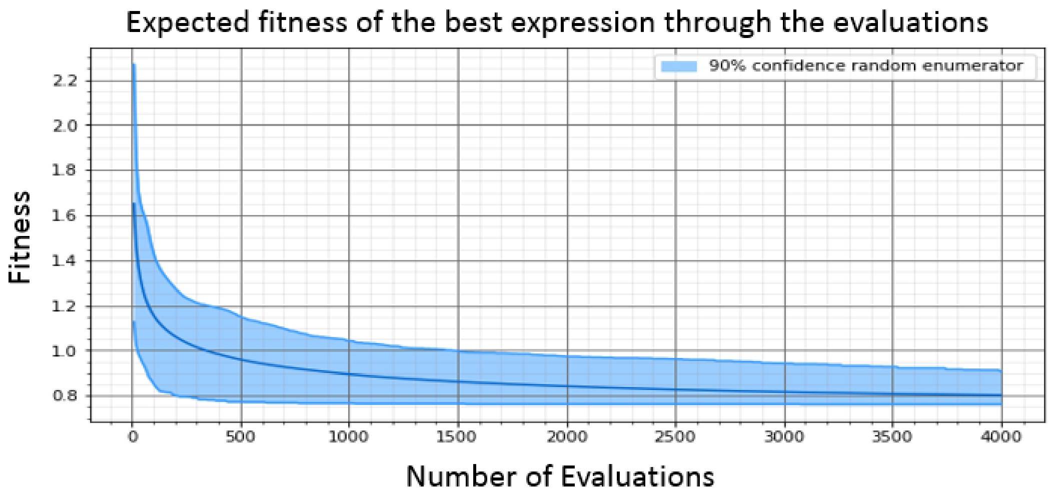


Figura 4.12: Expected Fitness for the random enumerator through the number of evaluations in the test set.

This graph representing the confidence interval for the score in the function of the number of expressions tested will be used as a comparison baseline.

4.4 Chapter summary

Close formulas to the number of expressions were reached to unary and binary operators. The worst case is when the number of terminals is one plus the number of operators, increasing $4\#T\#O_2$. Another key intake is that adding a unary operator has (for some cases) more or less the same impact as a binary one.

These intakes can be further used to help selecting the set of terminals and operators to use. Although an analysis for O_1 is presented, only O_2 operators were used in this work. This reduction was simpler than considerating only O_1, and it was the limit of what could be enumerated.

The enumeration applied to a small set of terminals and operators can find good, explainable, and simple solutions for this problem. The best expressions found in the test set will be used to compare the best expressions found in the test set with GP. Furthermore, the way GP evolves will be compared to the behavior of the enumerator in the training set. In this way, we will see if GP searches optimally tiny expressions.

It is expected that a suitable evolution method will have to find optimal solutions for these small sizes. If it does not, it would be better to enumerate all possible expressions as we did for this case to get the most explainable and most performative solutions.

Capítulo 5

Genetic Programming

This section will describe all the parameters and implementations used for each of the different algorithms tested. Furthermore, it will tell all the characteristics of the experimental setup. On a further note, all the grammars used are presented in appendix C.

Several different algorithms with different variants were tried. We implemented two different GP variants, GP-CR were the same terminals as the other algorithms were used and GP-AT where all except the CR^{+} terminal were used. For CFG-GP, we implemented two different variants: CFG-GP-PTC2, which uses the PTC2 algorithm to build the initial population, and CFG-GP-RND, which uses a more straightforward random procedure to build the initial population. For GE, we implemented two very distinct methods: GE-BIG and GE-PURE. GE-BIG is a method that uses a non-recursive-grammar fixing the size of the expressions. We implemented two variants: GE-BIG-19, which allows expressions between sizes 9-19, and GE-BIG-33, which allows expressions between sizes 9-33. These two variants can appear with a more compact name (GE-19 and GE-33, respectively). GE-PURE is a method that tries two types of grammars: GE-PURE-BAL, which uses a balanced grammar, and GE-PURE-NBAL, which uses the usual DA grammar. These two variants can appear with a more compact name (GE-BAL and GE-NBAL, respectively).

A summary of all the algorithm configurations tested can be seen in Table 5.1.

Tabela 5.1: Implemented algorithms summary.

GP		**CFG-GP**		**GE-BIG**		**GE-PURE**	
CR	AT	PTC2	RND	19	33	BAL	NBAL

5.1 Experimental Setup

As seen previously, GP/GE and CFG-GP evolve a population of computer programs (expressions) that solves a particular problem. The objective is to find the best individual (with the best fitness score) in the space delimited by its terminals/operands and size. Both algorithms are highly flexible, learning both the structure of the solution and the individual parameters. In every generation, all individuals are evaluated, getting a score. Afterwards, a new generation is created by combining the best ones from the previous generation.

Three different pipelines were created for all the algorithms: an elitism pipeline, a mutation pipeline, and a crossover pipeline. Note that both mutation and crossover used tournament selection for picking its parents. A tournament of size t first picks t individuals from the global population at random, and it returns the best one of them. Keep in mind that there exist differences in mutations/crossovers between these algorithms as previously developed.

In all the algorithms that use trees GP and CFG-GP, a max tree depth is specified as 6; this depth is usually used when the focus is interpretability [5]. For the GE methods, the limit was not depth but expression length. It was imposed as 127 $(2^7 - 1)$ for the GE, which allows it to represent all the expressions with depth 6 that the previous methods could represent. We are considering the first node to be at a depth of 0. For GP and CFG being comparable, CFG-GP depth limitation was not imposed in its genotype tree but rather in its equivalent GP tree.

Three options of parameters were tried for each algorithm, being five runs for each option made. The objective was not to choose the best statistically better parameters but rather to avoid choosing parameters that would halt the evolution. This small pre-test phase was needed because we did not have enough resources to do many runs for each parameter configuration in all algorithms, so compromises were made.

Three parameter configurations were tested. The first was one used in [5]. (mutation 0.1, crossover 0.85 and elitism 0.05, and tournament size of 7). As in [70], the author uses a higher mutation rate of 0.5; we tried one case with a similar mutation rate (the author did not use elitism, so a small adaptation was made). The tournament size was also decreased to give more randomness to the chosen operations (mutation 0.475, crossover 0.475, elitism 0.05, and tournament size of 4, in pre-tests, this seems a good alteration). Finally, the extreme opposite of the first parameters was tested (mutation 0.85, crossover 0.1, elitism 0.05, and tournament size of 2).

Tabela 5.2: Configurations tested in pre-tests phase.

	Config 1	**Config 2**	**Config 3**
Mutation rate	0,10	0,475	0,85
Crossover rate	0,85	0,475	0,10
Elitism rate	0,05	0,05	0,05
Tournament size	7	4	2

For every algorithm, there are four operators $\{+,-,\times,/\}$ and six terminals $\{PT, WinQ, RPT, A^+, S^+, CR^+\}$ except for GP-AT were CR^+ was not used.

Each algorithm was run for each parameter configuration with population sizes of 200 during 50 generations, and five independent runs were made. The objective is to choose some suitable algorithm parameters for each configuration as the behaviour of the algorithms can be very different when changing parameters. It is not to find the optimal parameters for running (so five runs seemed enough). For each algorithm, the parameter configuration that generated the best expressions overall (F_{min} minimum fitness) and had the best expressions in average (F_{avg} average fitness of the best expression per run) was chosen for the final runs. If these two criteria did not agree with each other further considerations, applied when needed, were made. Furthermore, the mean size of the generated best expressions will also be displayed. The summary of the three testing configurations can be seen in Table 5.2.

The results for the different configurations can be seen in Table 5.3. The parameters chosen to do the 20 runs were Par 1 for GP-AT GE-PURE-BAL and GE-PURE-NBAL, Par 2 for GP-CR CFG-GP-PTC2, CFG-GP-RND, GE-BIG-19, and GE-BIG-33. The explanation for every choice is given below.

After having chosen the parameters for each algorithm, 15 independent runs are made for each one. The best DRs generated for each run (in total, 5(pre runs)+15(final runs)= 20 runs) are saved and evaluated in the testing set.

Tabela 5.3: Configurations result for the pre-test phase. Several names are compacted: CFG-R stands for CFG-GP-RND, CFG-P for CFG-GP-PTC2, GE-B for GE-BAL, GE-NB for GE-NBAL, GE-19 for GE-BIG-19 and GE-33 for GE-BIG-33.

Algs	Config 1			Config 2			Config 3			BEST
	F_{min}	F_{avg}	DR_{size}	F_{min}	F_{avg}	DR_{size}	F_{min}	F_{avg}	DR_{size}	
GP-AT	**0,71**	**0,74**	26	0,78	0,84	25	0,74	0,75	21	Config 1
GP-CR	0,73	0,77	21	**0,72**	**0,73**	30	0,73	0,76	19	Config 2
CFG-P	0,71	0,73	27	**0,69**	**0,70**	28	0,71	0,73	22	Config 2
CFG-R	0,71	0,73	26	**0,69**	**0,70**	28	0,71	0,73	23	Config 2
GE-B	**0,71**	**0,72**	24	0,72	0,74	16	0,74	0,75	12	Config 1
GE-NB	**0,69**	**0,72**	22	0,72	0,73	18	0,76	0,82	18	Config 1
GE-19	**0,77**	0,82	12	0,78	**0,80**	11	0,78	0,86	12	Config 2
GE-33	0,76	**0,81**	20	**0,71**	0,89	20	0,77	0,91	17	Config 2

5.1.1 Chosen running configurations

Configuration 1 was chosen for the GP-AT variant because it had the minimum overall fitness and the minimum mean best fitness. By the same explanation, configuration 2 was chosen for the GP-CR variant.

For both the CFG-GP-PTC2 and the CFG-GP-RND variants, configuration 2 was chosen. The reason is that it has the minimum overall fitness and the minimum mean best fitness.

For the GE-BIG-19 variant, configuration 2 was chosen. The reasoning is that even though Par 1 had the best min fitness, a difference of 0.2 in the mean is more significant than 0.2 in the min fitness. We only consider this because looking for all the five values for each run, there was no clear outlier that significantly worsens the mean results of a run. For the GE-BIG-33 variant, configuration 2 was chosen. The reasoning is that it has by far the best min fitness of the other configurations, and the bad performance in the mean parameter is explained due to an outlier. The entire five runs for configuration 2 gave: [0.7058, 0.8768, 0.7824, 0.8514, 1.2539], 1.2539 is a clear outlier, and removing that one gives a mean 0.8041 what is close to the one for configuration 1.

For both the GE-PURE-BAL and GE-PURE-NBAL variants, configuration 1 was chosen. The reason is that it has the minimum overall fitness and the minimum mean best fitness.

5.2 Specific parameters for each algorithm

5.2.1 GP variants

For GP, the probability of selecting a terminal was set to 0.1, and to select a non-terminal to 0.9. For initiation, the Ramped Half-and-Half algorithm was used with a minimum tree depth of two and a maximum of 6. The maximum tree depth in the evolution process was set to 6.

5.2.2 CFG-GP variants

For CFG-GP, the max equivalent GP tree depth was also set to 6. The grammar used is the grammar exposed in the literature review (section 2.4.4) in appendix C. The mutations and crossovers are applied to the CFG-GP tree structure. The algorithm used to generate the CFG-GP initial population of the variant CFG-GP-RND will be described next.

5.2.2.1 Initialization Procedures

We chose to implement the same process used in [35]. This process guarantees a valid individual with a bounded tree max depth D_{max}.

Each non-terminal (<NT>) symbol in a grammar is paired with a integer d(<NT>) such that this symbol represent the depth of the smallest tree required to expand <NT> into a valid individual (only terminals). The depth of each terminal symbol is 1. d(<NT>) are recursively computed in the same way as in [35]:

$$d(<Op><NTa><NTb>) = 1 + max(d(<NTa>), d(<NTb>)$$

$$d(<NTi>) = min_j\{d(deriv_j)\}\ for\ <NTi> = deriv_1|deriv_2....deriv_n$$

With this, when creating a tree, if we are at a depth D, we only sample the non-terminals with d(NTi) equal or inferior to $D_{max} - D$. This initiation we called RND because it has less control on the depth of the individuals than PTC2. In this regard, it is random. However, a complete real random initiation did not work well with this grammar being incapable of evolving individuals of size 5 or bigger.

5.2.3 GE-BIG variants

The GE-Big variants use a grammar without recursive parts, which guarantees that any sufficient long vector will be mapped to a valid solution solving our problem of having invalid individuals.

For this method, PTC2 was not used because the PTC2 guarantees uniformity in tree structure initiation, but this grammar does that by definition. The chosen initiation method is a uniform random integer sequence generator. Moreover, to represent all derivation rules, the min value and the max value of the generation procedure were set distant enough. Furthermore, the length of the vector was set long enough to represent the longest individual of this grammar.

We only evolve expressions longer than size 9 because when we allow inferior sizes, the vast majority of the runs ended with expressions of size 1,3, never exploring bigger sizes.

All the operations of crossover/mutation are applied directly in the encoded integer vector of the GE representation. The grammar used for this algorithm will be described next.

5.2.3.1 Non recursive Grammars

It is not difficult to write a grammar that only generates expressions of a given size, a simple procedure to generate a grammar that only produces expressions of size N can be seen in Algorithm 5. The feeling behind this approach is relatively simple. For example, an expression of size 9 (<exp9>) can be computed with operations between an expression of size 7 (<exp7>) and size 1 (<exp1>), one of size 5 (<exp5>) and size 3 (<exp3>), one of size 3 and size 5 and finally one of size 1 and size 7. A small example is presented in Table 5.4 where a small grammar for expressions of size 5 is displayed (CR^+ was classified as size one because it is represented by only one symbol, we could instead have classified CR^+ with size three considering it is a compound terminal).

Algorithm 5 Automatic Generation of Production Rules of a fixed size.

```
1:  procedure AUTOMATIC GENERATION OF PRODUCTION RULES OF A FIXED SIZE(size)
2:      # innit production set P
3:      P = { }
4:      for s=1 to s= size do
5:          $P = P \cup \{< start >:=< exp_s >\}$
6:      for s=1 to s= size s+=2 do
7:          # size to create production rules s
8:          for p=1 to p=s-2 p+=2 do
9:              $P = P \cup \{< exp_s >= (\times < exp_p >< exp_{s-p-1} >)\}$
10:             $P = P \cup \{< exp_s >= (/ < exp_p >< exp_{s-p-1} >)\}$
11:             $P = P \cup \{< exp_s >= (+ < exp_p >< exp_{s-p-1} >)\}$
12:             $P = P \cup \{< exp_s >= (- < exp_p >< exp_{s-p-1} >)\}$
13:     for each terminal T of size u do
14:         $P = P \cup \{< exp_u >= (T)\}$
        return P
```

Tabela 5.4: Fixed grammar of size 5.

Production Rules		
<start>::= <ex1>	<ex5>::= (+ <ex1><ex3>)	<ex5>::= (/ <ex3><ex1>)
<start>::= <ex3	<ex5>::= (- <ex1><ex3>)	<ex1>::= (A^+)
<start>::= <ex5>	<ex5>::= (* <ex1><ex3>)	<ex1>::= (PT)
<ex3>::= (+ <ex1><ex1>)	<ex5>::= (/ <ex1><ex3>)	<ex1>::= ($WinQ$)
<ex3>::= (- <ex1><ex1>)	<ex5>::= (+ <ex3><ex1>)	<ex1>::= (RPT)
<ex3>::= (* <ex1><ex1>)	<ex5>::= (- <ex3><ex1>)	<ex1>::= (S^+)
<ex3>::= (/ <ex1><ex1>)	<ex5>::= (* <ex3><ex1>)	<ex1>::= (CR^+)

The idea is now merging this grammar to the one that enforces dimensional awareness. That is not difficult for instances if we want to merge the rule ($< exp7 >:= + < exp5 >< exp1 >$) with the dimensionally aware grammar, we have to see where does the + operation appear on the dimensionally aware grammar and for each one of the production rules merges them. If the + operator appeared only in the rules : $< N1 >:= + < N1 >< N1 >$ and $< N0 >:= + < N0 >< N0 >$ we could merge it giving as results rules: $< exp7N1 >:= + < exp5N1 >< exp1N1 >$ and $< exp7N0 >:= + < exp5N0 >< exp1N0 >$. The meaning of the last rule is: an expression of size 7 dimension 0 can be made by the addition between one expression of size 5 and dimension 0 and one of size 1 and dimension 0. Doing this for all rules would transform the grammar into a not recursive one.

However, we still need to remove some invalid rules from the grammar to avoid the generation of invalid expressions. For instance, if all our terminals have dimension time, it is impossible to have the following factor $< exp1N2 >$ there is no terminal with dimension 2 with size 1. So after the first grammar creation, an iterative approach has to be taken that starting with size 1 and going to the top checks if every production rule can be made. Every production rule that needs $< exp1N2 >$ in our previous example will be removed from the grammar set. If a non-terminal has all its production rules removed, this non-terminal is removed from the grammar. A small example of the final dimensionally aware grammar can be seen for size 3 in Table 5.5.

Tabela 5.5: Fixes grammar of size 3 merged with a dimensionally aware grammar.

Production Rules		
<start>::= <ex3N2>	<ex3N1>::= (* <ex1N1><ex1N0>)	<start>::= <ex1N0>
<start>::= <ex3N1>	<ex3N1>::= (- <ex1N1><ex1N1>)	<ex1N1>::= (S^+)
<start>::= <ex3N0>	<ex3N1>::= (+ <ex1N1><ex1N1>)	<ex1N1>::= (RPT)
<start>::= <ex3Nn1>	<ex3N0>::= (/ <ex1N1><ex1N1>)	<ex1N1>::= ($WinQ$)
<start>::= <ex1N1>	<ex3N0>::= (/ <ex1N0><ex1N0>)	<ex1N1>::= (PT)
<ex3Nn1>::= (/ <ex1N0><ex1N1>)	<ex3N0>::= (* <ex1N0><ex1N0>)	<ex1N1>::= (A^+)
<ex3N2>::= (* <ex1N1><ex1N1>)	<ex3N0>::= (- <ex1N0><ex1N0>)	<ex1N0>::= (CR^+)
<ex3N1>::= (/ <ex1N1><ex1N0>)	<ex3N0>::= (+ <ex1N0><ex1N0>)	

This approach has some benefits as it can use simpler algorithms: it does not have invalid individuals and does not need advanced initiation methods. However, this approach pays the price by only exploring a subset of the expression sizes and having an extensive grammar.

5.2.4 GE-PURE variants

For GE-PURE, several things will be further explored. First, we used PTC2 as initializing method. PTC2 was chosen because [49] shows PTC2 to be the best initiation procedure from the several he tried. The depth of the PTC2 initialization was set between 2 and 6 for the equivalent GP trees. After initiation, the individual is translated to the GE representation (integer vector), and after that, all crossovers/mutations are only applied to this vector. Moreover, we complete invalid individuals. In this way, it is fairer as all algorithms evaluate the same number of expressions. Furthermore, two different grammars were tested the standard DA grammar for variant GE-PURE-NBAL and the balanced grammar for variant GE-PURE-BAL. Finally, the max expression size was set to 127, representing all expressions that a tree of depth 6 can represent ($2^7 - 1$).

5.2.4.1 Non-explosive and balanced grammars:

[49] exposes in this paper the terms explosive grammar and balanced grammar. In an explosive grammar derivation, it is more likely to choose a non-terminal than a terminal. Therefore, after a certain number of expanded non-terminals, the probability of the phenotype termination tends to zero.

Using the same example in [49]. Assume the following grammar 5.1. In this grammar, the non-terminal v has four possible expansions out of five that transform v in two v's, and one terminates the expansion. So the expansion factor when in node v is : $2 \times 4/5 + 0 \times 1/5 = 133\%$. So if we are assuming a random choice, every v will be, on average, replaced by 1.33v, which will bring the probability of a valid individual being generated before the codons interpretation ends very low.

$$\begin{aligned}
&< S > \rightarrow < v > \\
&< v > \rightarrow + \ < v > < v > \ | - \ < v > < v > \ | \times \ < v > < v > \ | \ / \ < v > < v > \\
&< v > \rightarrow x
\end{aligned} \tag{5.1}$$

The impacts can be seen if we generate the initial population randomly, have mutations that change the genome at a certain point, and use crossovers that can cause codons to be reinterpreted for each one of these scenarios in an explosive grammar any operation as a high probability of rendering the individual invalid.

A grammar is balanced [49] if, for every non-terminal X, it is more likely that they expand for others non-terminals than for itself (X), and there is at least one non-terminal in every parse of the grammar.

So the challenge is now in turning the initial DA grammar into a non-explosive one that is balanced, with an expansion factor close to one or inferior. For each non-terminal expansion, an expansion rule with only terminals is added, or by repeating terminal rules or adding new rules. In this way, in each derivation, the expansion factor will be 1. As our DA grammar only uses binary operators, this method guarantees the grammar to be also balanced.

As the terminals with dimensions -1 and -2 seldomly were being used, they were removed from this grammar (in this way, fewer complex terminals rules had to be added). Furthermore, it is still possible to reach complex terms as $PT * WinQ/RPT$, which can be made by the rule $< N1 >::= (/ < N2 >< N1 >)$.

The exact steps of balancing were:

- Balance the operands, all operators should appear as often as the other operators
- Adding Terminal rules in a way that the number of terminal rules is the same as the number of non-terminal in each derivation
- Small changes are made to balance the two previous restrictions and add some prior knowledge.

The derivation procedure of expanding non-terminal N1 can be seen in Figure 5.1. Note that *X indicates that the rule was repeated X times.

In the middle grammar, the rules with "+", -"and "*"operators were repeated to have the same number as rules as the rules with "/". In the third grammar, terminal rules were added to be almost equal in number to the non-terminal rules. After that, there were two non-terminals rules lesser than terminal rules, so two were added (one multiplication (* N1 N0) and one division (/ N2 N1). We chose them because they were the most performative non-terminals rules to repeat by analyzing previous good expressions. With that, the operators become a little bit unbalanced but not in a critical way.

$$\begin{aligned}
&< N1 > ::= (A^{+}) \\
&< N1 > ::= (PT) \\
&< N1 > ::= (WinQ) \\
&< N1 > ::= (RPT) \\
&< N1 > ::= (S^{+}) \\
&< N1 > ::= (+ < N1 >< N1 >) \\
&< N1 > ::= (- < N1 >< N1 >) \\
&< N1 > ::= (* < N1 >< N0 >) \\
&< N1 > ::= (/ < N1 >< N0 >) \\
&< N1 > ::= (/ < N2 >< N1 >)
\end{aligned} \tag{5.2}$$

$$\begin{aligned}
&< N1 > ::= (A^{+}) \\
&< N1 > ::= (PT) \\
&< N1 > ::= (WinQ) \\
&< N1 > ::= (RPT) \\
&< N1 > ::= (S^{+}) \\
&< N1 > ::= (+ < N1 >< N1 >) * 2 \\
&< N1 > ::= (- < N1 >< N1 >) * 2 \\
&< N1 > ::= (* < N1 >< N0 >) * 2 \\
&< N1 > ::= (/ < N1 >< N0 >) \\
&< N1 > ::= (/ < N2 >< N1 >)
\end{aligned} \tag{5.3}$$

$$\begin{aligned}
&< N1 > ::= (A^{+}) * 2 \\
&< N1 > ::= (PT) * 2 \\
&< N1 > ::= (WinQ) * 2 \\
&< N1 > ::= (RPT) * 2 \\
&< N1 > ::= (S^{+}) * 2 \\
&< N1 > ::= (+ < N1 >< N1 >) * 2 \\
&< N1 > ::= (- < N1 >< N1 >) * 2 \\
&< N1 > ::= (* < N1 >< N0 >) * 3 \\
&< N1 > ::= (/ < N1 >< N0 >) \\
&< N1 > ::= (/ < N2 >< N1 >) * 2
\end{aligned} \tag{5.4}$$

Figura 5.1: Balance N1 derivation in the balanced DA grammar. On the left is the original grammmar, in the middle is the balanced grammar on operators on the right is the final balanced grammar

The greatest difficulty was doing the same for <N2>. As it does not have a single terminal rule, we have to create and add some. In Figure 5.2 we can see the derivation procedure.

In the middle grammar, the operators were balanced, and rules with "+",-", and "/"were repeated, totalizing 8 non-terminal rules. The most interesting operation that forms the N2 non-terminal that can be computed only using terminals are products of N1 per N1. So, we added 10 (5 choose 2) of these rules using all the combinations between terminal symbols. Finally, 2 non-terminal rules have to be added to make the number of terminal rules being equal to the number of non-terminal rules, and (/ <N2> <N0>) and (* <N1> <N1>) were chosen.

The balancing for <N0> is similar to <N1>, so it will no be developed here. The complete grammar can be seen in Appendix C.

$$\begin{aligned}
<N2> &::= (+<N2><N2>) \\
<N2> &::= (-<N2><N2>) \\
<N2> &::= (*<N1><N1>) \\
<N2> &::= (*<N2><N0>) \\
<N2> &::= (/<N2><N0>) \\
<N2> &::= (*PT\ A^{+})*0 \\
<N2> &::= (*PT\ WinQ)*0 \\
<N2> &::= (*PT\ RPT)*0 \\
<N2> &::= (*PT\ S^{+})*0 \\
<N2> &::= (*A^{+}\ WinQ)*0 \\
<N2> &::= (*A^{+}\ RPT)*0 \\
<N2> &::= (*A^{+}\ S^{+})*0 \\
<N2> &::= (*WinQ\ RPT)*0 \\
<N2> &::= (*WinQ\ S^{+})*0 \\
<N2> &::= (*RPT\ S^{+})*0
\end{aligned} \tag{5.5}$$

$$\begin{aligned}
<N2> &::= (+<N2><N2>)*2 \\
<N2> &::= (-<N2><N2>)*2 \\
<N2> &::= (*<N1><N1>) \\
<N2> &::= (*<N2><N0>) \\
<N2> &::= (/<N2><N0>)*2 \\
<N2> &::= (*PT\ A^{+})*0 \\
<N2> &::= (*PT\ WinQ)*0 \\
<N2> &::= (*PT\ RPT)*0 \\
<N2> &::= (*PT\ S^{+})*0 \\
<N2> &::= (*A^{+}\ WinQ)*0 \\
<N2> &::= (*A^{+}\ RPT)*0 \\
<N2> &::= (*A^{+}\ S^{+})*0 \\
<N2> &::= (*WinQ\ RPT)*0 \\
<N2> &::= (*WinQ\ S^{+})*0 \\
<N2> &::= (*RPT\ S^{+})*0
\end{aligned} \tag{5.6}$$

$$\begin{aligned}
<N2> &::= (+<N2><N2>)*2 \\
<N2> &::= (-<N2><N2>)*2 \\
<N2> &::= (*<N1><N1>)*2 \\
<N2> &::= (*<N2><N0>) \\
<N2> &::= (/<N2><N0>)*3 \\
<N2> &::= (*PT\ A^{+})*1 \\
<N2> &::= (*PT\ WinQ)*1 \\
<N2> &::= (*PT\ RPT)*1 \\
<N2> &::= (*PT\ S^{+})*1 \\
<N2> &::= (*A^{+}\ WinQ)*1 \\
<N2> &::= (*A^{+}\ RPT)*1 \\
<N2> &::= (*A^{+}\ S^{+})*1 \\
<N2> &::= (*WinQ\ RPT)*1 \\
<N2> &::= (*WinQ\ S^{+})*1 \\
<N2> &::= (*RPT\ S^{+})*1
\end{aligned} \tag{5.7}$$

Figura 5.2: Balance N2 derivation in the balanced DA grammar. On the left is the original grammar, in the middle is the balanced grammar on operators on the right is the final balanced grammar

5.2.4.2 Completion methods of invalid individuals GE

When a mutation or crossover causes the child to be invalid, the following procedure was done to complete the individual. The required individual length is chosen: for mutations, it was chosen to be equal to the parent size; for crossovers, it was the average parent size +1.

After that, the following is made. The full integer vector (of size S) is placed, and the first u integers are interpreted starting from u = 1 until u = S. For each of these S integer vectors (one for each value of u), its minimum valid individual is computed. The u integers are mapped to a phenotype, and each <NT> is completed.

The completion is simmilar to the initiation in CFG-GP but with one difference, as in GE the length and not the depth of the individual is what matters it is the length of the minimum expansion of the <NT> that is recorder. So a integer f(<NT>) is assossiated with every NT representing the minimum expression length required to finish that non-terminal. The f(<NT>) are computed in the following way:

$$f(<Op><NTa><NTb>) = 1 + f(<NTa>) + f(<NTb>)$$

$$f(<NTi>) = min_j\{f(deriv_j)\}\ for\ <NTi> = deriv_1|deriv_2....deriv_n$$

So a <NT> is expanded only with the rules that will give it the minimum expansion size. The rules that give length f(<NT>). This expansion is translated into integers, and all are appended to u vector of integers. Therefore, S new integer vectors S' all mapping to valid individuals are created. Finally, it is chosen from S', the integer vector that maps the individual with a size closer to the required. If a tie happens, a random one that is tied is picked. Also, the child is always a valid individual, and its size is correlated to its parent's size.

Capítulo 6

Results and Discussion

This section will answer all research questions. It is subdivided into three parts, each dedicated to answering one of them. Also, a small section is dedicated to studying the variability of our fitness metric in the end.

The first subsection gives the relevant information to answer the first research question (**RQ1**). What is the trade-off between performance and interpretability when introducing dimensional awareness? In this subsection, the results obtained by the different algorithms will be compared. Specifically, the distributions of the performance and sizes of the best expressions are compared. To these distributions, we then applied a Mann-Whitney U Test (n1=n2=20, P<0.05, one-tailed) to see the existence of significative differences between them. We will see if there is a performance drop in the DAGP methods. Furthermore, comparing the sizes gives insights into the expressions' interpretability.

Also, a random search procedure was implemented (called in the graphs Live). It is similar to the enumerator algorithm. The first part of the enumerator algorithm generates the basic structures. After that, two numbers, T_i, O_i (see Section 4.2), and one of the basic structures, are picked, which defines an expression. If that expression is dimensionally aware and unique (checked precisely as in the main algorithm), it is evaluated in the simulator. If not, another one is chosen. The allowed size of the expressions were 11, 13, 15, 17, 19, 21, 23, 29. These sizes were chosen because they are bigger than the enumerated sizes and are at the limit of what can be easily interpreted. This random search will serve as a performance baseline for the algorithms. All algorithms are expected to be better than it.

The second subsection answers the second research question (**RQ2**): Can GP find (near-)optimal Dispatching Rules (of a given size) for the Job Shop Scheduling problem? In this subsection, the algorithms' best expressions are compared to the best expressions overall (found by the Enumerator). By comparing them, we can answer the question.

Finally, the third subsection gives the relevant information to answer the third research question (**RQ3**). How performative and explainable are the best Dispatching Rules found? In this subsection, we compare the best expressions overall. We will see if the differences are significant between the expression, which ones performed the best and interpret them. An expression is said to be more interpretable than others in this specific analysis if the size is smaller than others, and an expert can infer the behaviour of the expression in different scenarios just by looking and comparing it with others. This part is subdivided into two analyses: one that evaluates each expression and explains it; another compares them with the best overall expression.

6.1 RQ1 - What is the trade-off between performance and interpretability when introducing dimensional awareness?

6.1.1 Algorithms performance

The following results refer to the performance of all the algorithms in the 20 runs (5 pre-runs plus 15 runs).

Figure 6.1 shows violin plots for the fitness distribution of the best expressions of the 20 runs for each algorithm in the training set. Furthermore, inside the violin plot, a more usual box plot can also be seen. The advantage of the violin plot is that it is possible to see the distribution of all the data, giving more insights into the results. Moreover, in Table 6.1 a Mann-Whitney U Test (n1=n2=20, P<0.05, one-tailed) is presented. This test aims to see if there are significant divergences between the performance of the different algorithms in the training set. We present the training results because they are a good indicator of how well the algorithms manage to explore the training set. The graph on the left (Live) is the performance of the random expression generator in 20 runs for 10000 expressions (the same number of expressions evaluated in 50 generations with a population of 200 for the genetic algorithms 50*200=10000). It serves as a baseline performance comparison: an algorithm has to be at least better than a random one.

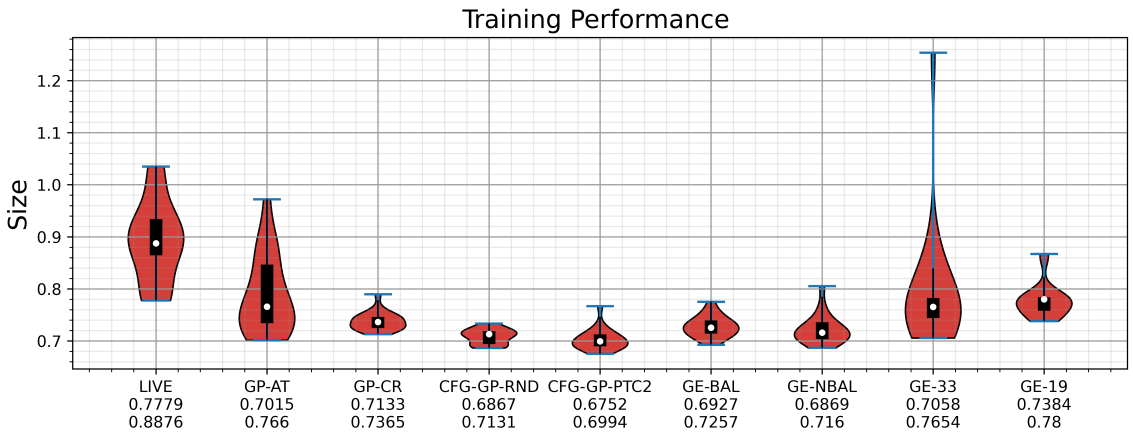


Figura 6.1: DRs performance for each algorithm in the training set. The first number below the algorithms name is the best fitness score; the other is the median.

All the algorithms performed significantly better than the random generator. Moreover, GE-BIG-33 and GE-BIG-19 performed significantly worse than the rest. GE-BIG-33 performance per

Tabela 6.1: Comparison between the performance of the different algorithms in the training set using a Man Whitney U tests. Every point of the table is representing the single side U statistic. Rejecting the null hypothesis (the algorithms are the same) is achieved with a U inferior to 138 (with 95% confidence). Every entry in bold (U inferior to 138) means that the algorithm on the left performs better (lower fitness) than the algorithm on top with 95% confidence

U Stat	LIVE	GP-AT	GP-CR	CFG-GP PTC2	CFG-GP RND	GE BAL	GE NBAL	GE-33	GE-19
LIVE	*	333.0	397.0	400.0	400.0	400.0	395.0	359.0	378.0
GP-AT	**67.0**	*	273.0	369.0	358.0	306.0	321.0	208.0	183.0
GP-CR	**3.0**	**127.0**	*	371.0	368.0	262.0	304.0	**98.0**	**39.0**
CFG-GP PTC2	**0**	**31.0**	**29.0**	*	**137.0**	**61.0**	**98.5**	**23.0**	**8.5**
CFG-GP RND	**0**	**42.0**	**32.0**	263.0	*	**92.0**	145.0	**24.0**	**0**
GE-BAL	**0**	**94.0**	**138.0**	339.0	308.0	*	249.0	**73.0**	**21.0**
GE-NBAL	**5.0**	**79.0**	**96.0**	301.5	255.0	151.0	*	**60.0**	**31.5**
GE-33	**41.0**	192.0	302.0	377.0	376.0	327.0	340.0	*	156.5
GE-19	**22.0**	217.0	361.0	391.5	400	379.0	368.5	243.5	*

run was unstable. GE-BIG-19 was incapable of finding highly performative expressions on the training set.

The algorithms that performed better were CFG-GP-RND and CFG-GP-PTC2 that had tiny variations between runs and got the best performative runs. GP-CR is comparable to GE-BAL and GE-NBAL. Finally, GP-AT manages to find suitable expressions in some runs but was also unstable, having high results dispersion. The dispersion was probably caused because GP-AT was the only GP tree-based algorithm that uses configuration 1 as parameters, and the results for the 5 pre-runs did not represent what we observed in the other 15 runs (there was a disparity between the pre-runs and the runs). However, it is unclear if running with the other parameters would give better scores.

CFG-GP-PTC2 was significantly better than all the algorithms in the training results (Table 6.1). Therefore, PTC2 helped the algorithm explore reaching better expressions than RND in the training set. Moreover, there were no significant differences between GE-BAL and GE-NBAL. Finally, GP-CR was the best GP algorithm, but it was significantly worse than all the CFG-GP and GE-PURE (GE-BAL and GE-NBAL) variants.

In Figure 6.2, one can see the fitness distribution of the best expressions of the 20 runs for each algorithm in the test set. Table 6.2 is analogous to Table 6.1 but applied to the test set.

Firstly, the distributions from the training set to the test set did not suffer significant changes, which is a good indicator that the test set is enough correlated with the training set. However, there were some differences. In particular, CFG-GP-PTC2 expression did not generalize well comparing to the CFG-GP-RND expressions, the statistical test comparing them passed from conclusive to inconclusive in the test results. Also, note that although GP-AT had much disparity between rounds, it also found expressions with good performance, one with a score of 0.787, comparable to the best methods CFG-GP PTC2 and CFG-GP-RND. However, the same can not be said to

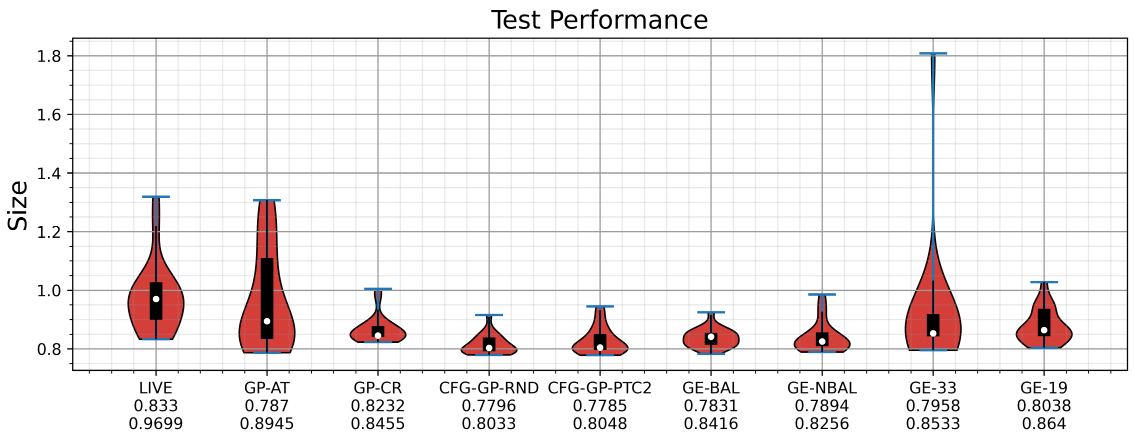


Figura 6.2: DRs performance for each algorithm in the test set. The first number below the algorithms name is the best fitness score; the other is the median.

Tabela 6.2: Comparison between the performance of the different algorithms in the test set using a Man Whitney U tests. Every point of the table is representing the single side U statistic. Rejecting the null hypothesis (the algorithms are the same) is achieved with a U inferior to 138 (with 95% confidence). Every entry in bold (U inferior to 138) means that the algorithm on the left performs better (lower fitness) than the algorithm on top with 95% confidence.

U Stat	Live	GP-AT	GP-CR	CFG-GP PTC2	CFG-GP RND	GE BAL	GE NBAL	GE 33	GE 19
Live	*	242	331	370	386	362	363	306	312
GP-AT	158	*	236	314	335	283	297	228	233
GP-CR	**69**	164	*	308	347	281	307	175	181
CFG-GP PTC2	**30**	**86**	**92**	*	238	143	170	**92**	**85**
CFG-GP RND	**14**	**65**	**53**	162	*	**109**	**137**	**61**	**55**
GE-BAL	**38**	**117**	**119**	257	291	*	232	**119**	**117**
GE-NBAL	**37**	**103**	**93**	230	263	168	*	**104**	**105**
GE 33	**94**	172	225	308	339	281	296	*	195
GE 19	**88**	167	219	315	345	283	295	205	*

GP-CR that, although not dispersed, did not find promising solutions.

Analyzing the table, we see that every algorithm except GP-AT performed significantly better than the random search (LIVE). Furthermore, GP-CR is beaten by the CFG-GP-RND, CFG-GP-PTC2, GE-BAL, and GE-NBAL variants. The GE-BIG-33 and GE-BIG-19 performed comparably with GP-CR. The best algorithms were the CFG variants, especially the CFG-GP-RND. CFG-GP-RND was statistically better than all the other algorithms (except for CFG-GP-PTC2, in which the test is inconclusive).

Finally, there were no significant differences between the two CFG-GP algorithms (CFG-GP-PTC2 and CFG-GP-RND) and between the two GE algorithms (GE-BAL and GE-NBAL). Therefore, it is impossible to tell confidently which CFG variant performed the best and which GE variant performed the best.

6.1.2 Size of Generated Dispatching Rules (DRs)

It follows on Figure 6.3 the sizes comparison of the best expressions for each one of the algorithms. Also, in Table 6.3 is made an analogous test to the ones of the previous section. The difference is that it is the DRs size and not the DRs performance the test variable.

Note that GE-BIG and Live had a significant size restriction, so they are expected to evolve smaller size expressions. GE-BIG-19, Live, and GE-BIG-33, as expected, generated the smallest expressions. Furthermore, GP-CR generated significantly smaller expressions than the other methods besides (those previously three). CFG-GP-RND generated significantly smaller expressions than GE-NBAL.

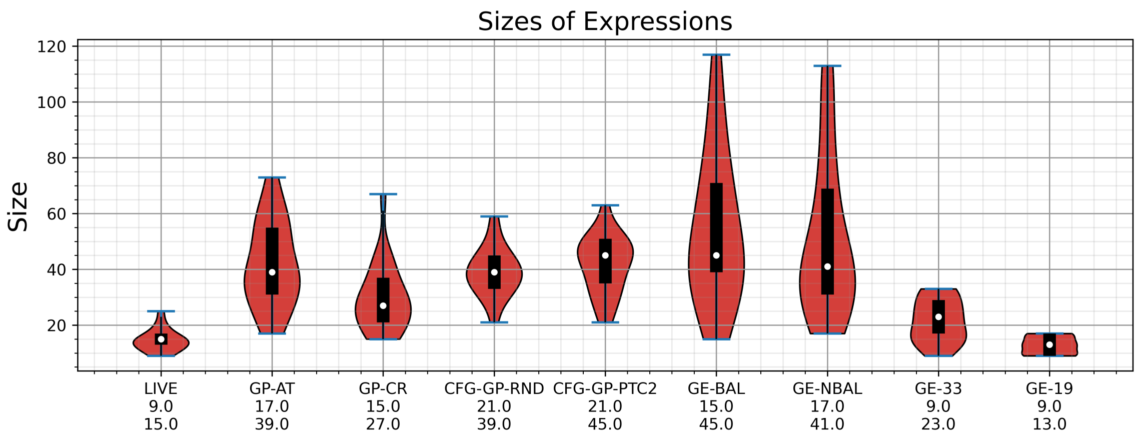


Figura 6.3: Sizes of the best DRs for each algorithm. The value below the name is the minimum size. The other is the median size

As GE-NBAL and GE-BAL had a significantly worse performance than CFG-GP-RND and GE-NBAL/GE-BAL generated significantly longer DRs, we can say that CFG-GP-RND expressions offer a better trade-off between interpretability and performance (as they have better performance and smaller size).

Tabela 6.3: Comparison between the DR sizes generated by the different algorithms using a Man Whitney U tests. Every point of the table is representing the single side U statistic. Rejecting the null hypothesis (the algorithms are the same) is achieved with a U inferior to 138 (with 95% confidence). Every entry in bold (U inferior to 138) means that the algorithm on the left generate smaller expressions than the algorithm on top with 95% confidence.

U Stat	Live	GP-AT	GP-CR	CFG-GP PTC2	CFG-GP RND	GE BAL	GE NBAL	GE 33	GE 19
Live	*	**6.5**	**25**	**2**	**1.5**	**9**	**4.5**	**89.5**	244.5
GP-AT	393.5	*	294.5	192.5	208	141	181.5	355	397
GP-CR	375	**105.5**	*	**76.5**	**88.5**	**82.0**	**94.5**	281.5	388
CFG-GP PTC2	398	207.5	324.0	*	237.0	157.5	195.0	373.0	400
CFG-GP RND	398.5	192	311	163	*	**130.5**	180.5	276.0	400
GE-BAL	391	259	318	242.5	269.5	*	224.0	363.5	394.0
GE-NBAL	395.5	218.5	305.5	204.5	219.5	176.0	*	359.5	397.5
GE 33	310.5	**45.0**	**118.5**	**27.0**	**24.0**	**36.5**	**40.5**	*	335
GE 19	155.5	**2.5**	**12**	**0**	**0**	**6.0**	**2.5**	**65.0**	*

However, the analysis comparing with GP-CR is not as straightforward. CFG-GP-RND got better performance than GP-CR, but as the generated expressions are significantly larger, there is no clear winner in interpretability. Interpretability is a trade-off between performance and simplicity, we would say. Keep in mind that GP-CR and CFG-GP had the same size restrictions of the expressions (depth 6), so they both could generate individuals up to size 127. In order to understand this trade-off, the next part will compare both size and performance simultaneously, which is necessary.

6.1.3 Size vs performance of the generated expressions

Comparing size and performance separately can be misleading, especially if there is not a clear dominance. For example, comparing GP-CR and CFG, one criterion indicates GP-CR as better than the CFG, and the other criterion the opposite. Thus, in order to be fairer, we have to compare them simultaneously.

Also, it is not enough to only evaluate the best expression in each run for the algorithms, as it considers performance but ignores interpretability and size. Often, the algorithms would find bigger DRs with a slightly better score, resulting in very large DRs in the last generation. This phenomenon is known as bloat. For illustration, assume two algorithms, algorithm A and B. Assume that algorithm A found better small expressions during evolution than algorithm B. However, afterwards, it found longer expressions with better performance ending the evolution with longer and highly performative expressions. In this scenario, the two criteria will diverge. Algorithm A expressions are longer but more performative than algorithm B. This disregards the fact that algorithm A found even for those small sizes better expressions than algorithm B during evolution.

Therefore, other DRs besides the best on each run were also evaluated. The non-dominated expressions of each algorithm were chosen. The two objective functions are to minimize the DR size and minimize the DR fitness (maximizing the score). A solution is called non-dominated, Pareto optimal, if none of the objective functions can be improved in value without degrading some of the other objective values. In our case, an expression of size s ($e_{s,alg}$) is said to be non-dominated (belongs to set N_{alg}) if, for all the expressions produced by that algorithm (alg) in all the 20 runs, there are no expressions of equal or smaller size that manage to get a better score (lower fitness value) in the training set than itself. If $E_{s,alg}$ represents the set of all expressions of size s produced by algorithm alg Equation 6.1 defines the non-dominated expressions. All non-dominated expressions were also evaluated in the test set.

$$e_{s,alg} \in N_{alg} \rightarrow \forall p \leq s; \forall b \in E_{p,alg}; b \neq e_{s,alg}; \ fit(e_{p,alg}) > fit(e_{s,alg}) \tag{6.1}$$

In Figure 6.4, we can see the fitness in the training set of all evaluated expressions (non-dominated + best of all runs). Here we can see that GP-AT had many bad runs and that CFG variants after size 21 always found better expressions than the GP variants.

Figures 6.6 and 6.5 depicts the scores of all evaluated expressions on the test set (non-dominated + best of all runs) in one, the y-axis is not cropped in the other it is. It is also shown the Pareto frontier overall, regardless of the algorithm, representing the expression we will further analyze on question 3. Figure 6.6 crop the y axis, hence not showing the best expression scores of sizes 1 and 3, allowing better observation of the scores for the other sizes. In Figure 6.5 we can see the complete non-cropped graph where we see that the fitness for expressions of size 1 and 3 is very high in comparison with the other sizes and also that GP-AT had a lot of bad runs.

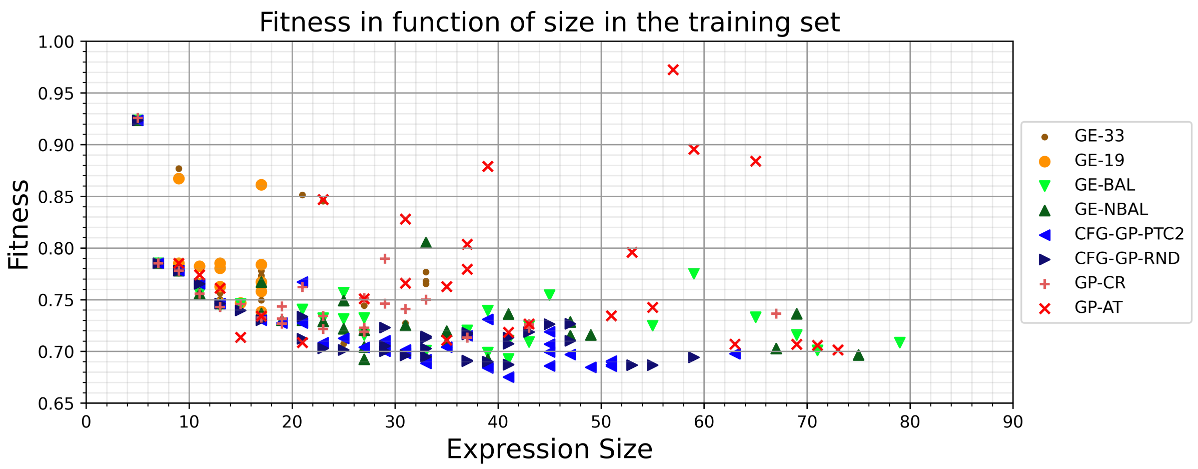


Figura 6.4: Fitness and size of the non-dominated DRs and the best DRs of the 20 runs in the training set.

In Figure 6.6 we see that GP-CR has indeed generated smaller size expressions than CFG methods, but even for small-sized DRs (sizes 17 and bigger), the expressions found by CFG methods are always better than the expressions found by GP-CR for the same sizes. Nevertheless,

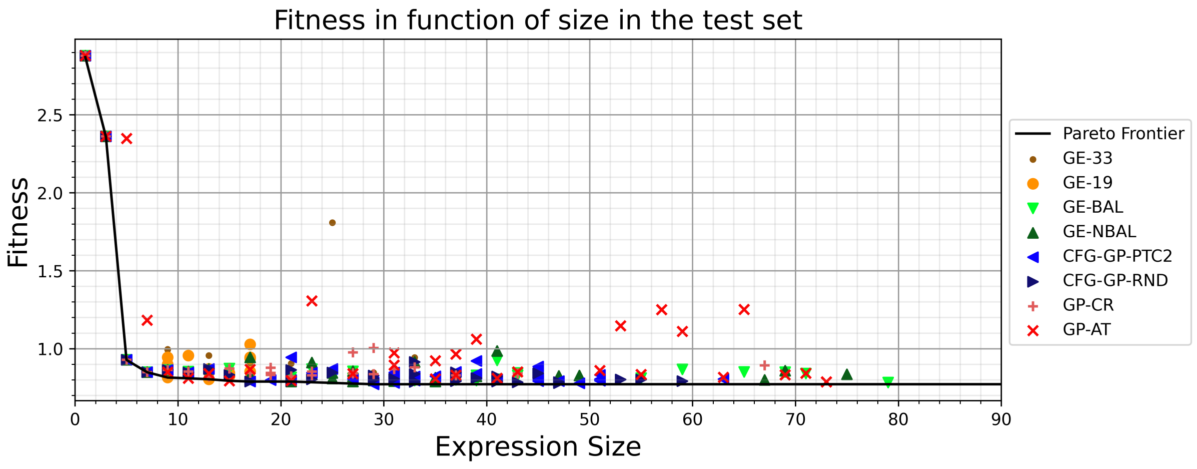


Figura 6.5: Fitness and size of the non-dominated DRs and the best DRs of the 20 runs in the test set. It is also represented the DRs belonging to the Pareto frontier of all algorithms.

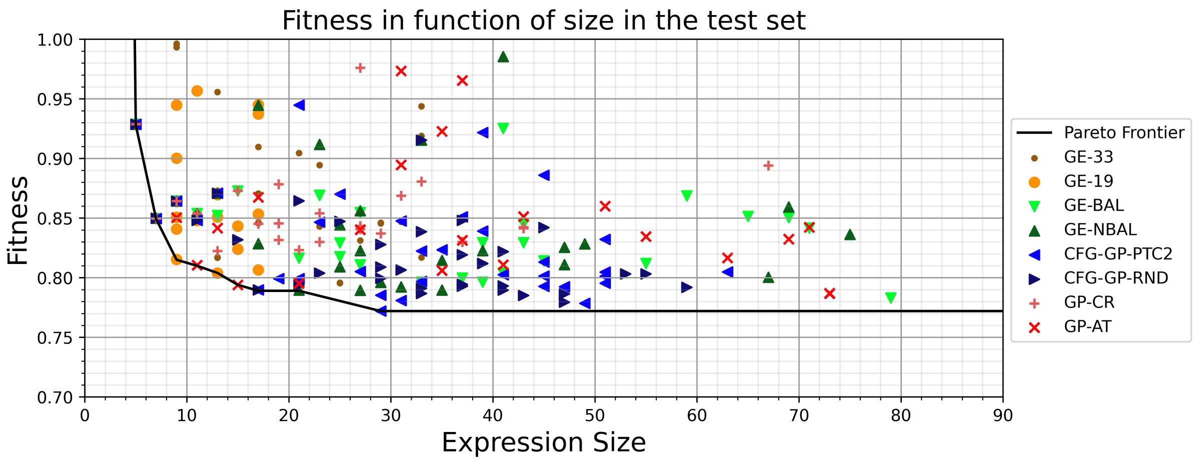


Figura 6.6: Fitness and size of the best evaluated DRs in the test set until fitness of 1 (non-dominated + best of each run). It is also represented the DRs belonging to the Pareto frontier of all algorithms.

for sizes between 11-15, the opposite is true. Also, considering the non-dominated expressions puts GP-AT in a more competitive light, it still has many bad runs (clearly seen by really long expressions with bad scores), but the algorithm found better expressions for sizes 11 and 15. Interestingly, also by analyzing Figure 6.4, we can see that GP-AT found an expression of size 15 that is considerably better. However, as GP-AT does not have CR^+ as a terminal is not as comparable to the other algorithms. Nevertheless, the absence of the terminal CR^+ for this size seemed advantageous, the other algorithms were exploring expressions with that terminal, and GP-AT was not.

Also, analyzing Figure 6.4 we can see that the methods are equilibrated until size 19 in generating expressions, but after that, the main grammar methods managed to find solutions that were a lot better than the GP variants in the training set.

In conclusion, even though the two criteria (size and performance) independently diverge between GP-CR and CFG algorithms (CFG DRs are longer and more performative), looking at the Pareto frontier, we can see that GP-CR is dominated by CFG methods for expressions of size 17 or bigger, being a close contender for smaller sizes. Finally, CFG methods did not have a performance drop. Moreover, in expressions of size 17 and superior, CFG has better interpretability than GP methods (as they are better in both criteria for these sizes). All algorithms have similar performance for DR sizes between 1-9. For sizes between 11-17, GE-19 and GP-AT were the most performative.

6.1.4 Discussion of RQ1

There was no performance loss between dimensionally aware algorithms and regular GP. On the contrary, the results indicate an apparent performance increase for dimensionally aware methods, especially for the CFG-GP, in both its variants CFG-GP-RND and CFG-GP-PTC2. These methods manage to found expressions with fitnesses (0.7785) in the last generation of the evolution, which are better than the ones found for GP-CR (0.823) and the ones found for GP-AT (0.787). Furthermore, it was statistically significant that the expressions generated in the 20 runs for these methods have better fitness than the ones generated for GP-CR and GP-AT. Also, when introducing the non-dominant expressions into the analysis, the CFG methods found for each size greater than 15 better expressions than those found in GP, which indicates that the interpretability was improved for medium sizes.

Some explanations for this can be: i) the smaller solution space of DA helped explore new expressions. It has less expressions to explore, so it is less likely to be stuck in sub-optimum regions, and with the same number of evaluated expressions will explore a more diverse area. ii) it seems that the best expressions are dimensional aware, so we can explore more efficiently when we take this into account. iii) our DA implementation was less restricted than a simple STGP implementation allowing dimensions between [-2, 2] compared to [0, 1], which could have helped to reach better DRs.

Comparing with [5] and [3] where DA did not bring a performance improvement for the JSS, the factor that probably causes this change was iii), the STGP implementation probably restricted too much the dimensions of the operands. Moreover, as the best DRs we found had dimension two, this seems to be the case.

Also, when [4] applied a soft constrain method to impose DAGP (i.e. penalizing non-dimensional consistent expressions in the score), he obtained similar results to standard GP for diverse problems (none being production scheduling). Here we would say that the principal difference is i). It is i) because imposing DA as soft constraints does not reduce the search space. Nevertheless, it has some advantages. For example, if the best possible expressions had dimension three, our implemented method could not generate it (unless we used a DA grammar allowing those dimensions), but soft constraints could.

Finally, applying DAGP in a problem where the solutions were known to be dimensional aware [70] (symbolic regressions of the Feynman equations) helped the authors to navigate more efficiently the solution space finding more easily better solutions. Almost all expressions in the Pareto frontier were dimensionally aware in our results, as we will see in question RQ3 (note that the GP variants could evolve freely non-dimensionally aware expressions). Therefore, it seems that the best performative DRs for the JSS are dimensionally aware (our explanation ii makes sense). So we navigated better the search space finding better solutions than standard GP, similarly as [70], because the best expressions for the JSS are probably dimensional aware.

Regarding the others DA methods tried, there were no significant differences between CFG-GP-PTC2 and CFG-GP-RND in the final results. This was unexpected because in [46] PTC2 was a lot better than the other initiation methods. Explanations for this divergence are: Our initial population size is very restricted (depth 4), which does not allow PTC2 to differentiate itself from the RND initiation; Our random initiation is a bit more sophisticated than a usual random initiation procedure. Nevertheless, in the training set, PTC2 was better and got an edge at the beginning of the evolution (as can be seen in appendix B), conforming with the findings in [46].

There was no significant difference between balanced grammar and non-balanced grammar for GE-PURE, which was also unexpected. In [49] the use of a balanced grammar was better than a non-balanced one. We believe that our implemented methods to complete invalid solutions disguised the explosive grammar problems, allowing individuals to be generated with arbitrary length. Also, GE did not work so well as CFG, both in performance and expression sizes. In [41] GE also had worse performance than CFG-GP.

The GE-BIG grammars were disappointing. Only the test 9-19 was competitive, but 19 as the maximum size is too low to get performative solutions. The major downside was for size 33 that did not even manage to be better than the random search. This behaviour can be explained by the giant required grammars to implement these methods, which greatly affected the locality and also the exploration of this method. Remember that the locality problem in GE means that small local changes in the genotype (integer string) can endorse considerable changes in the phenotype (the evaluated expression), which can be detrimental to evolution (see Section 2.4.2). The locality was affected because the individual could be reinterpreted entirely by changing an integer in the GE genotype. Therefore, any local change in the genotype would have a significant impact on the phenotype. Furthermore, when reaching a point where all the individuals in a generation had the same size, it would be almost impossible to explore other sizes. The only way would have been a random change in the first integer (the integer that defines the expression length), and the subsequent reinterpretation of the entire individual (which is equivalent to generating a random new individual) would generate a better expression than the current best ones, which is extremely unlikely. Similar concerns are detailed in [48] using other grammars.

6.2 RQ2 - Can GP find (near-)optimal Dispatching Rules (of a given size) for the Job-Shop Scheduling problem?

We will answer this research question by comparing the non-dominated expressions found by the algorithm with the results of the enumerator. As the enumerator explores a given size optimally, it will find the best expressions for that size on the training set, which will correspond to its non-dominated expression for that size (in the enumerator, the best expression for size x was always worse than the best expression for sizes x + k). Also, remember the notation used to represent the terminals (Table 4.1).

6.2.1 Best small expressions for all algorithms

Table 6.4 presents the results of the complete enumeration. Here the column on the right represents the score an algorithm that found the best expression in the training set would have, and in the left, the minimal score overall possible in the test set. Keep in mind that the left part is not a lower bound for sizes 9 and 11 (because they operate in a subset of terminals), and also the right part does not represent the expression an optimum algorithm could reach in the training set (because it does not consider CR^+ as a terminal), nevertheless they can be seen as benchmarks for these sizes.

It follows in Tables: 6.5, 6.6, the results for the non-dominated expressions for each algorithm. Note also that GE-BIG tests do not generate expressions with a size inferior to 9. Moreover, GE-33 did not manage to find an expression of length 11 better than the one it found for length 9 (all its expressions of length 11 were dominated).

The following paragraphs will only refer to the algorithms that have all the terminals (GP-AT will be further ahead detailed). By analyzing the tables, all the algorithms found the best possible expression for sizes 7 and inferior found in the training set, so they explored these sizes optimally. Furthermore, all algorithms found the benchmark expression for size nine, so they also seemed to have optimally explored that size. However, for size 11, some algorithms found the expression (P + (P + ((W + P) * C))) + W and others W+P+P + W + (C*P), as there is variance in the expression found the algorithms are not able to find the optimum for these sizes.

More precisely, the benchmark rule has a training performance of 0.7615, rule (P + (P + ((W + P) * C))) + W has a training performance of 0.756, and rule W+P+P+W+(C*P) has a training performance of 0.7653. Therefore, the algorithms that found the expression W+P+P+W+(C*P) did not explore the space optimally (CFG-GP-PTC2, CFG-GP-RND, GE-BIG-19); they do not surpass the benchmark rule. Nevertheless, the same cannot be said for the other case. They explored the space better, having reached an expression with better fitness on the training than the benchmark rule (GE-BAL, GE-NBAL, GP-CR). However, that expression does not generalize as well as the benchmark rule in the test set.

For the GP-AT, a more precise analysis can be made because GP-AT does not have CR^+ as a terminal. In Table 6.4 can be seen the best results expected without CR^+ as a terminal. In this case, GP-AT managed to find the optimal training expression until size 5. Then, for size 7, it has found an expression that is the 10th best expression for that size (see Table 4.8). Next, for size 9, it has found an expression with a training score (0.785) which has a better score on the training set than the benchmark rule (W+P+S*P/R)(0.789), so it explored well this size. Finally, for size 11, it has found the 6th best expression (W+P +(W+S)*P/R) the Enumerator has found (W+P*(S+R+W)/R) (see Table 4.9). So GP-AT found optimal or close to optimal solutions in the training set. However, for sizes 7 and 11, it did not manage to find a solution as good as the best ones found by the enumerator.

In conclusion, all algorithms manage to find optimal or near-optimal expressions for the sizes we could enumerate. GE-33 was the only exception, having a terrible performance.

Tabela 6.4: Best expressions found in the enumerator. On the left, using only atomic terminals on the right using CR^+. Here is only presented the test score, so the score presenting in the training columns are the test score of the expressions that performed the best in the training set. Expression with * indicates that this result is an upper bound as only a subset of terminals was used for those sizes.

Enumerator test AT		**Enumerator training AT**		**Enumerator test CR**		**Enumerator training CR**	
Fitness	Expression	Fitness	Expression	Fitness	Expression	Fitness	Expression
2.879	W	2.879	W	2.879	W	2.879	W
2,361	W+P	2.361	W+P	2.361	W+P	2.361	W+P
1.474	P*A/R	2.348	W+W+P	0.928	W+P*C	0.928	W+P*C
0.927	P*A/R+W	0.927	P*A/R+W	0.849	W+P+P*C	0.849	W+P+P*C
0.810*	(S+R)*(W+P)/R	0.816*	W+P+S*P/R	0.810*	(S+R)*(W+P)/R	0.864*	W+P+P+C*P
0.792*	(S+R)*(W+P)/(R+W)	0.817*	W+P+S*P/(R+R)	0.792*	(S+R)*(W+P)/(R+W)	0.817*	W+P+S*P/(R+R)

Tabela 6.5: Best expressions found by the GP and CFG algorithms.

GP-AT		**GP-CR**		**CFG-RND**		**CFG-PTC2**	
Fitness	Expression	Fitness	Expression	Fitness	Expression	Fitness	Expression
2.879	W	2.879	W	2.879	W	2.879	W
2,361	W+P	2.361	W+P	2.361	W+P	2.361	W+P
2.348	W+P+P	0.928	W+P*C	0.928	W+P*C	0.928	W+P*C
1.183	(S+R)*(P+W)	0.849	W+P+P*C	0.849	W+P+P*C	0.849	W+P+P*C
0.850	W+P + A*P/R	0.864	W+P+P+C*P	0.864	W+P+P+C*P	0.864	W+P+P+C*P
0.810	W+P +(W+S)*P/R	0.853	W+P+P+C*(P+W)	0.848	W+P+P+W+C*P	0.848	W+P+P+W+C*P

6.2.2 Discussion of RQ2

As we can see by the results, the answer is yes concerning the sizes we could enumerate. GP and DAGP methods both reached optimal or close to optimal solutions for these sizes. In fact, for

Tabela 6.6: Expressions found by the GE algorithms. GE-BIG methods did not evolve expressions of sizes inferior to 9 by definition, so an X is in the table. Furthermore, all expressions of size 11 were worse than the best expression of size 9 in GE-BIG-33. Therefore, that size was dominated, and an X is there.

GE-BAL		**GE-NBAL**		**GE-BIG-19**		**GE-BIG-33**	
Fitness	Expression	Fitness	Expression	Fitness	Expression	Fitness	Expression
2.879	W	2.879	W	X	X	X	X
2,361	W+P	2.361	W+P	X	X	X	X
0.928	W+P+P	0.928	W+P*C	X	X	X	X
0.849	W+P+P*C	0.849	W+P+P*C	X	X	X	X
0.864	W+P+P+(C*P)	0.864	W+P+P+C*P	0.864	W+P+P+C*P	0.99	R*W+P*P*C
0.853	W+P+P+C*(P+W)	0.853	W+P+P+C*(P+W)	0.848	W+P+P+W+C*P	X	size dominated

sizes 1-7, they found the optimal solutions, and for sizes 9-11, they found really close solutions to the best one we could enumerate or even better; however, they stop reaching the same solution systematically after these sizes. Even though the best expression found is the same in relationship with the enumerator, some insights can be drawn by it:

The enumerator allows us to have guarantees of optimality which GP does not provide. Furthermore, using the enumerator on the training set and evaluating the best ten expressions would suffice to find the best expression on the test set (see Tables 4.8, 4.9, 4.7, here the best expression in the test set were always one of the best 10 in the training set). Moreover, as we saw, there were significant differences between the best expression on the training set and the best on the test set.

Also, the enumeration on the training set could have been performed to bigger DRs sizes 13 and 15 (because evaluating an expression on the training set is almost 100 times less expensive than evaluating on the test set). However, we did not do it in this work because we also wanted to see if the expressions on the training set would be optimum in the test set, which would be lost if we only enumerated the training set.

Furthermore, as the Enumerator only considerer DA expressions and did not count replicates, it was expected to have an edge at the beginning of the evolutions, and it did. However, after generation 5, GP manages to be better than the expected fitness generated by a random search (see Appendix B).

6.3 RQ3 - How performative and explainable are the best Dispatching Rules found?

Having analyzed all the algorithms, it is necessary now to aboard the last point of this thesis. Were the generated expressions explainable and performative?

This section will analyze the non-dominated expressions found on the test set, disregarding the algorithm that generated them, statistically comparing them with the best DR, and explaining its behaviour for different shop conditions.

6.3.1 Analyses of the best-found expressions overall

In this section, the best expressions found overall will be evaluated. We choose all the non-dominated expressions to analyze, disregarding the algorithm that generated them. Exceptions were made for sizes 11 and inferior, where other exquisite expressions were also included. In particular, we also included the best expressions of the Enumerator, both for the test and the training sets.

The entire expression selection can be seen in Table 6.7. Moreover, this selection corresponds to the expressions of the Pareto frontier of Figure 6.6. For every rule A in the tables, its performance in all test instances will be shown. In each cell of the next tables is written the performance (F_{Ai}) for a given instance i. Also, (A) represents the allowances, (U) the utilizations and (M) the max machine processing times. Moreover, values of F_{Ai} inferior to one mean that the rule performed better for that entry than all benchmark rules.

Tabela 6.7: Expressions we chose to interpret to answer RQ3. all*(1) means that all algorithms found it, all*(2) means that all algorithms except GE-33 found it. Sev* refers to CFG-GP-PTC2 and CFG-GP-RND.

			Best Expressions	
ID	Size	Fit	Expression	Algorithm
0	1	2.87	W	all*(1)
1	3	2.36	W+P	all*(1)
2	5	0.928	W+P*C	all*(1)
3	7	0.849	W+P+P*C	all*(1)
4	9	0.864	W+P+P+P*C	all*(2)
5	9	0.815	W+P+S*P/R	GE-19
6	11	0.817	W+P+S*P/(R+R)	Enum Trai
7	11	0.810	W+P+((W+S)*P)/R)	GP-AT
8	11	0.792	(S+R)*(W+P)/(W+R)	Enum Test
9	13	0.804	P-((W-S)*P-R*W)/R)	GE-19
10	15	0.794	W+P-(S+P * S)/(P-R-R)	GP-AT
11	17	0.789	(P+W)*P*C*C+P*P+W*R	Sev*
12	21	0.789	W*R+P*P+(P*P+W*P)*A*C/R	GE-NBAL
13	29	0.772	(R+P)*(W+P)+P*(C+C)*(W+P)*(C/R)*(P*C+W+W)	CFG-PTC2

The first rule to be analyzed is the rule (id 0): "W", known in the literature as WinQ. The idea behind this rule [71] is to give preference to jobs that would be less halted in a congested queue afterwards (with the least backlog). This rule was the most performative of size 1. Surpassing greatly SPT (shortest processing time first, "P"). In Table 6.8 we can see its performance on the test set. It is clear that the rule has a good performance for high workload shop situations (allowance 2,3, and high utilization), but for high allowances, it is many times worse than the benchmark rules. This behaviour was expected because this rule does not consider the slack or allowance of a job to prioritize it.

Next, we have the rule (id 1) "W+P"also known in the literature. First proposed by [13]. This rule is additive conjunction of the WinQ and SPT rule and managed to be the best rule to minimize

Tabela 6.8: Test performance of rule (W, id 0). For high allowances, this rule is many times worse than the benchmark rules (e.g. for (U) 0.8 (A) 8 and (M) 100, it is 1010 times worse). For low allowances ((A) = 2 and 3), it is around 1.2 times worse.

		(A)														
		2			3			4			6			8		
	(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
	0.80	1.37	1.37	1.38	1.99	2.00	2.01	3.86	3.95	3.98	36.3	36.2	37.1	1010	204	220
	0.85	1.30	1.30	1.31	1.42	1.42	1.45	2.14	2.16	2.16	8.39	9.21	9.91	118	108	122
(U)	0.90	1.24	1.24	1.24	1.20	1.24	1.21	1.32	1.38	1.31	2.86	2.90	2.93	9.81	10.3	10.9
	0.95	1.13	1.14	1.14	1.09	1.12	1.12	1.06	1.07	1.05	1.37	1.40	1.47	2.13	2.29	2.26
	0.97	1.07	1.06	1.09	1.05	1.04	1.07	1.00	0.99	1.02	1.22	1.22	1.21	1.44	1.67	1.38

mean flow time in [13]. With this rule, both throughput maximization at the current machine and the waiting time minimization of a job for the subsequent operation are considered. This rule is the best possible with size 3 and those specific terminals for these scenarios in our simulations. In Table 6.9 we can see its performance on the test set. This rule greatly improved the performance on high workload instances. However, for high allowances, it presents the same defects as the previous rule.

Tabela 6.9: Test performance of rule (W+P, id 1). For high allowances, this rule is many times worse than the benchmark rules (e.g. for (U) 0.8 (A) 8 and (M) 100, it is 1920 times worse). For low allowances ((A) = 2 and 3), it is more or less equal (score around 1) which is an improvement in relation to rule 0.

		(A)														
		2			3			4			6			8		
	(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
	0.80	0.94	0.94	0.95	1.25	1.25	1.24	2.48	2.48	2.52	38.4	40.4	40.9	1920	383	447
	0.85	0.96	0.96	0.96	0.94	0.94	0.95	1.36	1.36	1.38	6.34	6.94	7.33	132	121	150
(U)	0.90	0.96	0.98	0.97	0.91	0.91	0.89	0.94	0.94	0.97	1.87	2.02	1.98	7.47	7.94	8.73
	0.95	0.97	0.99	0.97	0.94	0.96	0.94	0.84	0.85	0.88	1.04	1.10	1.11	1.74	1.71	1.90
	0.97	1.03	1.01	0.99	1.01	0.96	0.96	0.96	0.91	0.88	1.08	1.04	1.03	1.31	1.25	1.25

The following rule (id 2) is "W+P*C", which does not appear in the literature. Moreover, in Table 6.10 we can see its performance on the test set. As it uses the critical ratio, this rule manages to respond well in high allowance situations. However, the behaviour of high-load shops deteriorated in relationship with the previous rules. This phenomenon happens because, for extremely high load scenarios, the critical ratio tends to be more often zero, and the rule becomes similar to "W", which is worse for these scenarios than "W+P". Using the critical ratio had a noticeable impact on the behavior of the jobs for high allowances. Specifically, as we minimize tardiness, a rule to behave well in those scenarios has to consider some parameter that uses the slack or the allowance of a job, which is the case with the critical ratio. For the same W on the next queue, this rule prioritizes delayed jobs (CR<1).

For rule "W+P+P*C"(id 3), a slight adjustment is made. In Table 6.11 we can see its performance on the test set. This rule is similar to RR rule (but without the exponent terms on utilization

Tabela 6.10: Test performance of rule (W+P*C, id 2). For high allowances, this rule is a lot better than the others (scores around 0.8). However, its behaviour for low allowances ((A) 2 and 3) is similar to rule id 0 (1.2 times worse than the benchmark rules).

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	1.21	1.22	1.22	1.09	1.09	1.10	0.78	0.79	0.79	0.29	0.29	0.32	1.00	1.00	1.00
0.85	1.24	1.24	1.24	1.14	1.11	1.15	1.05	1.07	1.02	0.51	0.51	0.57	0.25	0.22	0.20
(U) 0.90	1.24	1.25	1.22	1.16	1.22	1.16	1.16	1.15	1.16	1.10	1.08	1.05	0.67	0.66	0.70
0.95	1.12	1.13	1.14	1.13	1.11	1.12	1.02	1.03	1.06	1.10	1.13	1.15	1.16	1.21	1.26
0.97	1.09	1.07	1.08	1.07	1.07	1.08	1.02	1.02	1.01	1.12	1.05	1.09	1.09	1.14	0.96

and CR^{+} instead of Slack/RPT). In high congested shops, this rule behaves as "W+P", which is better than "W", solving the problem of "W+P*C". Furthermore, it responds well to high allowances scenarios for the same previous reason.

Tabela 6.11: Test performance of rule (W+P+P*C, id 3). For high allowances, this rule is similar to rule id 2. Moreover, for low allowances ((A) 2 and 3), it is similar to the benchmark rules (score around 1), so it is better than rule 2 for these scenarios.

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.99	1.00	1.00	0.93	0.92	0.93	0.72	0.74	0.74	0.36	0.39	0.36	1.00	1.00	1.00
0.85	1.03	1.02	1.02	0.92	0.91	0.95	0.88	0.90	0.86	0.52	0.53	0.54	0.25	0.33	0.33
(U) 0.90	1.02	1.04	1.02	1.00	1.01	0.99	0.88	0.93	0.93	0.95	0.95	0.91	0.65	0.64	0.62
0.95	1.01	1.01	1.00	1.00	1.00	1.02	0.93	0.94	0.92	0.99	0.96	0.95	1.05	0.97	1.00
0.97	1.04	1.04	1.01	1.03	1.06	1.04	0.97	0.94	0.96	1.04	1.00	1.01	0.99	1.04	0.84

For rule "W+P+P+P*C"(id 4), the same considerations are made: it is similar to the RR rule. In Table 6.12 we can see its performance on the test set. This rule represents a slight change from the previous rule, which did not introduce meaningful improvements. The test results even deteriorated. However, this rule is more similar to the RR rule due to the 2PT instead of PT term. In fact, in the RR rule for these scenarios u = [0.85-0.97], the exponential factor on P varies from [2.33 - 2.63], which is closer to 2 than 1.

Tabela 6.12: Test performance of rule (W+P+P+P*C, id 4). There are no meaningful differences between this rule and rule 3.

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.96	0.96	0.97	0.91	0.91	0.94	0.79	0.78	0.78	0.43	0.46	0.52	1.00	1.00	1.00
0.85	0.99	1.00	1.00	0.87	0.88	0.88	0.86	0.86	0.84	0.59	0.62	0.63	0.50	0.44	0.40
(U) 0.90	1.00	1.01	0.99	0.92	0.96	0.93	0.87	0.91	0.91	0.91	0.90	0.88	0.68	0.78	0.76
0.95	0.99	1.00	1.00	0.97	1.01	0.99	0.91	0.85	0.91	0.88	0.91	0.89	1.01	0.85	1.11
0.97	1.03	1.02	1.03	0.99	1.01	1.00	0.98	0.94	0.96	1.04	1.00	1.02	0.90	0.96	0.86

The next rule (id 5) is $W + (R + S) * P/R = W + P + S * P/R$, which is very similar with the RR rule: $PT * exp(u) + PT * S/RPT * exp(-u) + WINQ$ but without the exponentials . In Table 6.13 we can see its performance on the test set. This rule is highly similar to W + P + C*P = W*P + S*A/R but instead of allowance uses the slack, these two options are usually interchangeable in DR design, and in this scenario, using the slack appears to have worked better. Although looking very similar to the RR rule, there are some minor changes to the terminals: we use the positive slack max(0, slack), and the RR rule uses the usual slack.

Tabela 6.13: Test performance of rule (W+(R+S)*P/R, id 5). This rule had generally better performance for utilizations 0.8 and 0.85 than rule 4.

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.95	0.96	0.96	0.89	0.88	0.87	0.65	0.67	0.67	0.29	0.29	0.34	1.00	1.00	1.00
0.85	0.99	0.97	0.98	0.88	0.86	0.86	0.84	0.85	0.81	0.46	0.48	0.50	0.25	0.22	0.27
(U) 0.90	1.00	1.02	1.00	0.94	0.94	0.94	0.91	0.90	0.91	0.93	0.90	0.89	0.60	0.54	0.58
0.95	0.99	1.00	0.98	0.97	0.98	1.01	0.89	0.90	0.91	0.94	0.96	0.95	1.05	1.08	1.15
0.97	1.05	1.01	0.98	1.06	1.04	1.02	1.02	1.01	0.99	1.13	1.01	1.07	0.91	0.99	0.88

Rule (id 6) "W+P+S*P/(2R)"is highly similar to the previous rule, but with a factor of 1/2, it did not bring significant differences concerning the other rule. In Table 6.14 we can see its performance on the test set. This rule is even more similar to the RR rule. In RR, the term S*P/R is affected by exp(-u), which has a value between [0.37-0.427] for our utilization. Moreover, this term is affected by a multiplication per 0.5 (closer to the RR rule). Moreover, an insight can be made. In the limit, if we reduce the factor that multiplies the slack term, we get "W+P"when this factor is 0, which behaves very poorly for high allowances. So we can induce that reducing the weight of this term will slightly worsen the shop performance for high allowances and improve for small allowances (which seems to happen).

Tabela 6.14: Test performance of rule (W+P+S*P/(2R), id 6). For allowances 6 utilization 0.8 and 0.85 this rule is worse than rule 5. However, for allowances 2 utilization 0.8 and 0.85 this rule is better than rule 5.

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.94	0.93	0.94	0.87	0.85	0.85	0.72	0.72	0.72	0.43	0.39	0.43	1.00	1.00	1.00
0.85	0.97	0.95	0.95	0.83	0.83	0.84	0.81	0.81	0.79	0.53	0.58	0.60	0.25	0.22	0.33
(U) 0.90	0.98	0.99	0.98	0.89	0.89	0.89	0.88	0.86	0.84	0.88	0.87	0.87	0.59	0.65	0.59
0.95	0.98	0.99	1.01	0.94	0.91	0.99	0.84	0.87	0.88	0.91	0.92	0.92	1.00	0.94	0.93
0.97	1.02	0.97	0.99	0.98	0.96	0.98	0.97	0.97	0.94	1.08	1.00	1.00	1.05	1.02	0.94

Rule (id 7) W+P+((W+S)*P)/R)= W+P+S*P/R+W*P/R. In Table 6.15 we can see its performance on the test set. It works as the former rule excepting the additional term W*P/R, which gives more relevance to jobs with smaller next machine queues. Interestingly, there was a marginal improvement in the global fitness metric comparing to the previous one. However, when seeing the global p-value case comparison test, it indicates that the other is better than it. Moreover, this divergence shows a case where the global fitness indicator hid much important information in the score indicator, being incapable of taking the right conclusions.

Tabela 6.15: Test performance of rule (W+P+(W+S)*P/R, id 7). This rule had almost always worse score in every instance than rule 6. Except for allowance 8; allowance 4 and 6 for utilization 0.8; and allowances 6 for utilization of 0.85.

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.95	0.96	0.95	0.88	0.87	0.88	0.67	0.67	0.65	0.29	0.29	0.30	1.00	1.00	1.00
0.85	0.99	0.99	0.99	0.86	0.85	0.86	0.82	0.84	0.80	0.47	0.47	0.52	0.25	0.11	0.27
(U) 0.90	1.01	1.00	0.99	0.95	0.96	0.92	0.91	0.91	0.86	0.90	0.87	0.89	0.64	0.58	0.58
0.95	1.00	1.03	1.02	0.99	1.01	0.98	0.95	0.90	0.90	0.98	0.96	0.93	1.04	1.06	0.89
0.97	1.06	1.06	1.05	1.08	1.07	1.04	1.02	1.02	1.02	1.11	1.10	1.05	1.03	1.12	0.88

Rule (id 8) "(S+R)*(W+P)/(R+W)"is different from the previous rules. In Table 6.16 we can see its performance on the test set, and there we can see clearly that this rule has an edge for utilization 0.97 in relationship with the rest of the rules. However, its pairwise tests against rule 6 indicate that none rule has a significant edge, but rule 6 tens to be better as the global-pvalue of Table 6.22 indicates. This DR is a lot harder to interpret. However, for high allowances where the W is usually inferior to R (the queues are small), this DR can be approximate to (S+R)*(W+P)/(R) = W + P + S*P/R + S*W/R. This derivation is similar to rule 5 but with an additional term S*W/R, focusing more on the slack. Moreover, this additional weight on the slack term is expected to give a small edge, in relationship with rule 5, for high allowances, which happens slightly. A similar argument was used to explain rule W+P+ S*P/(2R), which removes weight to the S term. This rule performs well for super high utilization. In this situation, we know that "W+P"is a good rule and that the slack term tends to be near-zero (as usually, the jobs are late, especially in the later machines). This can be seen by the fact that there is not a performance increase for allowance 2 and 3 high utilizations (0.95 and superior) between "W+P"and "W+P+S*P/(2R)". Therefore, by disregarding the slack, this rule becomes equal to (W+P)*R/(R+W) = (W+P)*(1/(1+W/R)). So it is similar to "W+P"but with the extra weight "(1/(1+W/R)". This difference makes it prefer jobs with higher W/R. So it prefers jobs that are close to ending (low remaining processing time), the W on this term is compensated by the initial term (W+P), so the conclusion is not straightforward of which queue lengths it prefers. Interestingly this behaviour also is presented in rule 13 (being similar with "W+P"with a preference for jobs with low remaining processing time, as we will further see).

Rule (id 9) P-((W-S)*P-R*W)/R) = P-((W-S)*P/R-W) = P+W-(W-S)*P/R = P+W * S*P/R - W*P/R. So this rule is similar to W+P+((W+S)*P)/R) (rule 7). However, the extra factor W*P/R has a different sign. This difference will make it prefer jobs that are more close to completion

Tabela 6.16: Test performance of rule ((S+R)*(W+P)/(R+W), id 8). For very high utilization ((u) 0.97) this rule performed better than rule 6.

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.95	0.96	0.96	0.91	0.89	0.89	0.67	0.69	0.70	0.21	0.29	0.29	1.00	1.00	1.00
0.85	0.98	0.99	0.97	0.89	0.88	0.90	0.87	0.85	0.84	0.46	0.43	0.47	0.25	0.22	0.27
(U) 0.90	0.99	1.00	0.98	0.95	0.96	0.94	0.94	0.95	0.93	0.92	0.92	0.89	0.61	0.55	0.54
0.95	0.94	0.94	0.94	0.94	0.96	0.93	0.88	0.86	0.91	0.94	0.98	0.98	0.94	1.04	0.87
0.97	0.94	0.91	0.90	0.95	0.94	0.94	0.88	0.90	0.91	1.04	0.94	1.03	0.89	0.96	0.85

in being executed. In Table 6.17 we can see its performance on the test set. This small change makes the rule better in high load scenarios than the similar rule, but the other is better in high allowance scenarios (however, the global test p-value to compare the two rules gives 0.58, so it is inconclusive, which is better overall).

Tabela 6.17: Test performance of rule ((P-((W-S)*P-R*W)/R), id 9). For low allowances (especially (a) 2), this rule performed better than rule 7.

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.95	0.95	0.96	0.87	0.88	0.89	0.66	0.66	0.67	0.21	0.29	0.30	1.00	1.00	1.00
0.85	0.97	0.99	0.98	0.88	0.87	0.87	0.83	0.85	0.83	0.48	0.47	0.52	0.25	0.22	0.20
(U) 0.90	1.00	1.00	0.99	0.93	0.95	0.95	0.94	0.91	0.92	0.93	0.94	0.91	0.69	0.62	0.61
0.95	0.97	0.98	0.97	0.95	0.97	0.94	0.87	0.87	0.88	0.94	0.98	0.95	0.95	1.00	1.18
0.97	0.97	0.99	0.98	0.98	0.96	0.98	0.92	0.97	0.93	1.07	1.02	0.98	1.01	1.05	0.90

Rule (id 10) W+P-(S+P*S)/(P-R-R) is equal to W+P-S(1+P)/(P-2R). Interestingly it was the only non DA expression to belong to the Pareto frontier. Nevertheless, by noticing that P is 95% of the times a lot greater than 1, this approximates to W+P-S*P/(P-2R). This approximation indicates that the expression is almost DA, which indicates that enforcing dimensional awareness is suitable for this problem. As R>P all the time, the expression becomes equal to W+P+S*P/(2R-P). This rule is similar to rule W+P+S*P/(2R) with a difference of -P in the denominator. Assume two jobs R1<R2 all else equal, the first rule assigns priority: W+P+S*P/2R1 and W+P+S*P/2R2 and the second W+P+S*P/(2R1-P) and W+P+S*P/(2R2-P). This will imply that the second priority rule assigns lower priority values for both cases, increasing the impact of the slack term. However, as the diminishing is dependent on P, it is challenging to predict its impact. Nevertheless, increasing the slack impact tends to improve DR performance in high allowance scenarios and prejudice in low allowance scenarios, which seemed to happen. Furthermore, this was the first rule to be statistically better than rule 6. In Table 6.18 we can see its performance on the test set. This rule behaves significantly better than rule 6.

Rule (id 11) "(P+W)*P*C*C+P*P+W*R"is a lot more challenging to interpret because it is very different from the usual formats in the literate. For starters, it has dimension time squared.

Tabela 6.18: Test performance of rule ((W+P-S(1+P)/(P-2R), id 10). For allowances of 4, 6, 8 this rule tend to be better than rule 6 (except for utilization 0.8).

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.94	0.93	0.93	0.83	0.83	0.82	0.67	0.67	0.69	0.43	0.43	0.45	1.00	1.00	1.00
0.85	0.97	0.95	0.97	0.82	0.82	0.82	0.80	0.79	0.77	0.51	0.49	0.54	0.25	0.22	0.33
(U) 0.90	0.98	0.99	0.98	0.88	0.90	0.89	0.85	0.80	0.83	0.82	0.80	0.76	0.52	0.49	0.50
0.95	0.97	1.00	0.98	0.93	0.95	0.98	0.88	0.88	0.88	0.83	0.85	0.90	1.01	0.98	0.99
0.97	1.03	1.03	1.01	1.02	0.96	0.99	0.94	0.93	0.93	1.01	0.97	0.96	0.91	0.89	0.80

However, we can see that when a job is with allowance negative, its priority is given by P*P+W*R, which has some resemblance with P+W. Furthermore, for very late jobs C < 1, C square makes the job even more urgent than only with C, which we can postulate to be beneficiary in high utilization scenarios. However, it is not easy to understand this DR behaviour. In Table 6.19 we can see its performance on the test set. This expression is always better than the benchmarks rules except for utilization 0.9 allow 2. Moreover, it performed well for utilization 0.97.

Tabela 6.19: Test performance of rule ((P+W)*P*C*C+P*P+W*R, id 11). For ultra-high utilizations ((u) 0.97), this rule is better than rule 6. However, it is worse for allowances 2 and 3 when the utilization is inferior to 0.95.

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.96	0.96	0.97	0.87	0.88	0.89	0.70	0.68	0.71	0.36	0.29	0.34	1.00	1.00	1.00
0.85	0.98	0.99	0.98	0.87	0.88	0.88	0.84	0.83	0.81	0.49	0.45	0.49	0.25	0.33	0.33
(U) 0.90	0.99	1.01	1.00	0.93	0.94	0.94	0.89	0.90	0.88	0.87	0.88	0.85	0.58	0.54	0.53
0.95	0.95	0.96	0.97	0.91	0.91	0.92	0.83	0.86	0.86	0.86	0.85	0.82	0.92	0.85	1.01
0.97	0.97	0.91	0.92	0.91	0.88	0.92	0.90	0.88	0.89	0.91	0.86	0.96	0.86	0.95	0.82

Rule (id 12) "W*R+P*P+(P*P+W*P)*A*C/R"does not seem but is equal to the previous rule. The striking similarity is shown in Table 6.20. Almost all the values are the same. It follows now that derivation of expression equality: W*R+P*P+(P*P+W*P)*A*C/R = (P*P+W*P)*A*C/R + P*P + W*R = (P+W)*P*A*C/R + P*P + W*R = (P+W)*P*C*C + P*P + W*R. So both expressions are equal. Differences in the tables are due to internal calculations roundings.

Rule (id 13) "(R+P)*(W+P)+P*(C+C)*(W+P)*(C/R)*(P*C+W+W)", is the most performative expression from all. Moreover, it is the most challenging DR to understand, and it is significantly better than all the other DRs. When a job is delayed, the expression becomes equal to "(R+P)(W+P)", which is essentially the solid rule W+P multiplied by R+P. This difference benefits greatly jobs that are close to ending (small R, remaining processing time). The C penalty also appears in a squared way as in the previous two rules. Therefore, it is probably well behaved for high utilizations. However, as this formula is complex is difficult to take more insights when dealing with a non-zero CR. In Table 6.21 we can see its performance on the test set. This expression

Tabela 6.20: Test performance of rule (W*R+P*P+(P*P+W*P)*A*C/R), id 12). This rule is equal to rule 11.

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.96	0.96	0.97	0.87	0.88	0.89	0.70	0.68	0.71	0.36	0.29	0.34	1.00	1.00	1.00
0.85	0.99	0.99	0.98	0.87	0.88	0.88	0.84	0.83	0.81	0.49	0.45	0.49	0.25	0.33	0.33
(U) 0.90	0.99	1.01	1.00	0.93	0.94	0.94	0.89	0.90	0.88	0.87	0.88	0.85	0.58	0.54	0.53
0.95	0.95	0.96	0.97	0.91	0.91	0.92	0.84	0.86	0.86	0.86	0.85	0.82	0.92	0.85	1.01
0.97	0.97	0.91	0.92	0.91	0.88	0.92	0.90	0.88	0.89	0.91	0.86	0.96	0.86	0.95	0.82

was highly performative in all scenarios with some problems in allowance 2 for utilization of 0.85 and 0.9. Moreover, it excelled in high and ultra-high utilizations.

Tabela 6.21: Test performance of rule ((R+P)*(W+P)+P*(C+C)*(W+P)*(C/R)*(P*C+W+W), id 13). This rule excelled in high and ultra-high utilizations ((u) of 0.97 and 0.95). It is better than all others for these conditions. Its weakness is for allowance 2 for low utilizations.

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.94	0.95	0.95	0.83	0.83	0.84	0.69	0.68	0.67	0.50	0.46	0.57	1.00	1.00	1.00
0.85	1.00	1.00	0.98	0.84	0.82	0.83	0.74	0.75	0.69	0.47	0.52	0.47	0.25	0.22	0.40
(U) 0.90	1.01	1.02	1.01	0.89	0.92	0.90	0.82	0.83	0.81	0.75	0.73	0.77	0.55	0.50	0.48
0.95	0.97	0.96	0.97	0.92	0.91	0.93	0.81	0.81	0.83	0.85	0.82	0.79	0.78	0.84	0.85
0.97	0.92	0.90	0.92	0.93	0.88	0.90	0.86	0.86	0.84	0.94	0.87	0.88	0.82	0.81	0.71

It follows a Table 6.22 comparing if the results are significant between all rules following the methodology described in 3.4.2 to reach a unique p-value for each pairwise test. In each entry is represented a p-value comparing the rule on the left to the one on top. For example, if we choose a 95% confidence level, when the value is inferior to 5%, the DR on the left is better than the DR on the top with significance. The p-values are colour coded to be easier to interpret the table between green (pvalue 0) and red (pvalue 1).

Thus, the overall structure of the table was expected. When an expression has a better score is expected to be significantly better than the others, which happens in the vast majority of the scenarios (and creates that pyramid look on the figure). However, there is a clear case where this does not happen in comparing expressions 10 to 11 and 12. It appears that in the statistical test, expression 10 is better than 11 and 12, and this enters in contradiction with the chosen metric, which gives that rule 12 and 11 are better than 10. This difference is relevant and will be further elaborated on in the discussion. Also, comparing expressions 6 and 8, we can see that the 0.02 difference in the score did not bring significant differences for these two expressions. On the contrary, it seems that expression 6 is better than 8. Finally, it is worth mentioning that expression 13 was significantly better than all other expressions, followed by expression 10.

Tabela 6.22: Comparision between all chosen rules to answer RQ3 using the global p-value indicator.

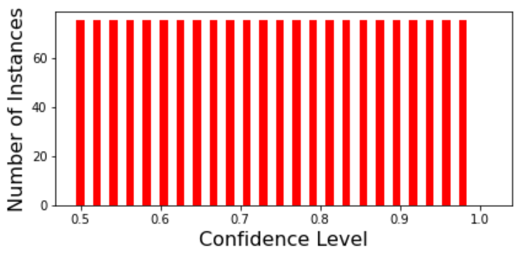
		Expressions IDs													
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
Expressions IDs	0	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	1	0.00	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	0.00	0.00	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	0.00	0.00	0.00	0.50	0.53	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	4	0.00	0.00	0.00	0.47	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	5	0.00	0.00	0.00	0.00	0.00	0.50	1.00	0.72	0.95	0.71	1.00	1.00	1.00	1.00
	6	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.01	0.27	0.02	1.00	0.90	0.90	1.00
	7	0.00	0.00	0.00	0.00	0.00	0.28	0.99	0.50	0.90	0.58	1.00	1.00	1.00	1.00
	8	0.00	0.00	0.00	0.00	0.00	0.05	0.73	0.10	0.50	0.31	1.00	0.95	0.94	1.00
	9	0.00	0.00	0.00	0.00	0.00	0.29	0.98	0.42	0.69	0.50	1.00	1.00	1.00	1.00
	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.00	0.00	0.99
	11	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.00	0.05	0.00	1.00	0.50	0.50	1.00
	12	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.00	0.06	0.00	1.00	0.50	0.50	1.00
	13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.50

6.3.2 Comparison with the best overall rule

It follows here in this section a more exhaustive comparison between the best rule and all the previous rules. This section will clarify in which scenarios the best rule is worse than the other expressions. As stated in 3.4.1, a statistics test was performed, and here are the results. The p-value for the alternative hypothesis: Rule 13 is better than Rule x is shown for each of the shop scenarios. A value inferior to 5% means that rule 13 is better with confidence 95 % than the other, a p-value greater than 95% means, as discussed in section x, that the other rule better than 13 with a 95% confidence level. The figures are colour-coded green being low values of p and red being high values of p. Furthermore, a bar plot is represented showing, depending on the confidence level, how many scenarios is rule 13 better than the other rule (red bars) and how many scenarios the opposite occurs (yellow bars, the other rule is better than rule 13) (keep in mind that the total number of scenarios is 75).

		(A)														
		2			3			4			6			8		
	(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.85	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
(U)	0.90	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.95	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.97	0.00	0.00	0.00	0.02	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00

(a) T-Student test



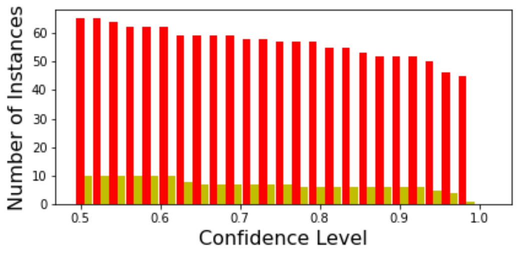
(b) Bar Plot

Figura 6.7: DR 13 comparison with DR 0

Rule 0 was completely dominated by rule 13, even with 98% confidence intervals for all scenarios rule 13 is better (Figure 6.7).

		(A)														
		2			3			4			6			8		
	(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
	0.80	0.45	0.75	0.61	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.85	0.96	0.97	0.92	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
(U)	0.90	0.98	1.00	0.98	0.39	0.61	0.63	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.95	0.46	0.20	0.45	0.38	0.16	0.38	0.30	0.25	0.15	0.02	0.01	0.00	0.00	0.00	0.00
	0.97	0.00	0.01	0.06	0.05	0.04	0.08	0.06	0.19	0.16	0.07	0.01	0.05	0.00	0.00	0.00

(a) T-Student test



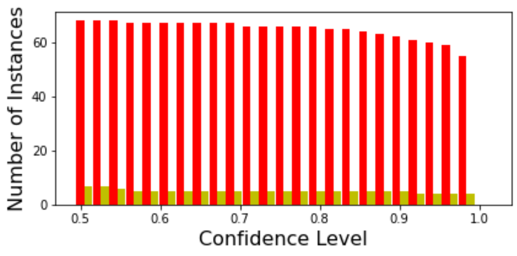
(b) Bar Plot

Figura 6.8: DR 13 comparison with DR 1

Rule 1 was worse than rule 13 in almost all scenarios. However, it manages to be better with 95% confidence for utilization 0.85-0,9 for allowance 2 (Figure 6.8). This is a reminder that it is challenging to have a rule better than all others for all instances. Even a simple rule manages to be better than the best rule in a couple of scenarios.

		(A)														
		2			3			4			6			8		
	(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	1.00	1.00	1.00	0.09	0.45	0.56
	0.85	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.19	0.53	0.08	0.30	0.91	1.00
(U)	0.90	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.15	0.13	0.12
	0.95	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.01	0.02	0.02
	0.97	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.03	0.03	0.03	0.06	0.03	0.01

(a) T-Student test



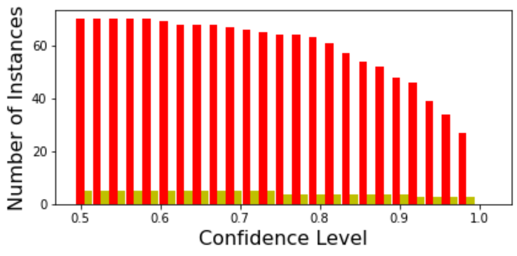
(b) Bar Plot

Figura 6.9: DR 13 comparison with DR 2

Rule 2 was vastly dominated by rule 13. However, it manages to be better with 95% confidence for utilization 0.8 for allowance 6. Nevertheless, as we can see by the bar plot rule, 13 is always in a lot more scenarios significantly better (Figure 6.9). Nevertheless, it was a bit unexpected for rule 13 to lose in that specific scenario. For allowance 4, it is considerably better, and for allowance 8, it is equally performative to rule 2.

		(A)														
		2			3			4			6			8		
	(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
	0.80	0.04	0.03	0.02	0.00	0.00	0.00	0.16	0.00	0.00	1.00	0.99	1.00	0.38	0.25	0.91
	0.85	0.10	0.14	0.06	0.04	0.00	0.00	0.02	0.00	0.00	0.18	0.41	0.11	0.22	0.07	0.73
(U)	0.90	0.27	0.30	0.33	0.02	0.04	0.02	0.19	0.01	0.06	0.01	0.00	0.00	0.17	0.07	0.07
	0.95	0.20	0.07	0.16	0.07	0.00	0.05	0.02	0.01	0.04	0.11	0.10	0.02	0.04	0.17	0.18
	0.97	0.00	0.00	0.02	0.01	0.00	0.01	0.02	0.07	0.02	0.11	0.06	0.09	0.13	0.04	0.17

(a) T-Student test



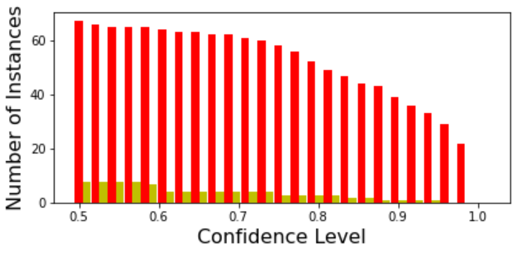
(b) Bar Plot

Figura 6.10: DR 13 comparison with DR 3

For rule 3 the same conclusions reached for rule 2 can be made. In Figure 6.10 we can see its comparison to rule 13.

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.19	0.40	0.21	0.00	0.00	0.00	0.00	0.00	0.00	0.95	0.58	0.86	0.25	0.16	0.82
0.85	0.60	0.48	0.25	0.23	0.01	0.01	0.02	0.00	0.00	0.01	0.03	0.00	0.16	0.03	0.34
(U) 0.90	0.59	0.60	0.74	0.27	0.21	0.19	0.18	0.13	0.10	0.03	0.02	0.00	0.01	0.03	0.03
0.95	0.30	0.11	0.23	0.18	0.01	0.09	0.05	0.20	0.07	0.39	0.21	0.15	0.10	0.47	0.06
0.97	0.00	0.00	0.01	0.06	0.01	0.04	0.03	0.06	0.02	0.09	0.06	0.08	0.29	0.11	0.12

(a) T-Student test



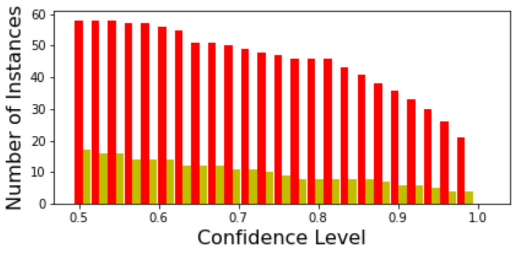
(b) Bar Plot

Figura 6.11: DR 13 comparison with DR 4

Rule 4 was had a similar comparison to rule 13 as rules 3 and 2. Moreover, It only manages to be better in one scenario with 95% confidence (utilization 0.8 allowance 6)(Figure 6.11).

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.36	0.45	0.42	0.00	0.02	0.13	0.95	0.61	0.54	1.00	1.00	1.00	0.09	0.36	0.56
0.85	0.76	0.87	0.62	0.11	0.00	0.10	0.03	0.00	0.00	0.72	0.88	0.36	0.37	0.92	0.99
(U) 0.90	0.68	0.52	0.75	0.19	0.32	0.17	0.01	0.01	0.00	0.01	0.00	0.00	0.25	0.30	0.18
0.95	0.29	0.15	0.39	0.17	0.06	0.03	0.08	0.06	0.04	0.13	0.11	0.05	0.03	0.08	0.05
0.97	0.00	0.01	0.06	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.03	0.01	0.23	0.07	0.13

(a) T-Student test



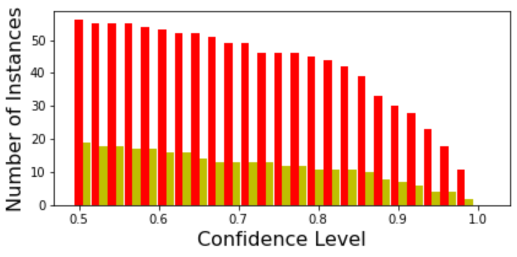
(b) Bar Plot

Figura 6.12: DR 13 comparison with DR 5

Rule 5 was more competitive than the other, managing to be better for allowance 6 in utilization 0.8 and being competitive for allowance 2 with median-low utilization. The comparative bar plot shows that this rule is better in around 1/4 of the cases around all significance levels (Figure 6.10).

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.55	0.91	0.79	0.02	0.09	0.34	0.09	0.00	0.00	0.99	0.97	1.00	0.85	0.27	0.65
0.85	0.93	0.98	0.93	0.60	0.42	0.11	0.08	0.03	0.00	0.13	0.03	0.00	0.16	0.43	0.65
(U) 0.90	0.86	0.89	0.87	0.51	0.73	0.63	0.00	0.13	0.18	0.03	0.02	0.00	0.14	0.01	0.08
0.95	0.38	0.17	0.15	0.33	0.49	0.12	0.29	0.15	0.15	0.21	0.20	0.14	0.13	0.28	0.32
0.97	0.01	0.03	0.04	0.12	0.04	0.03	0.02	0.02	0.05	0.08	0.06	0.08	0.05	0.07	0.05

(a) T-Student test



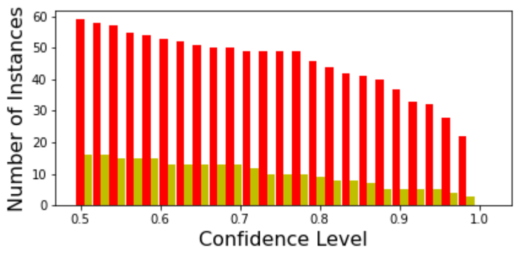
(b) Bar Plot

Figura 6.13: DR 13 comparison with DR 6

Rule 6 was had a similar comparison to rule 13 as rules 5. However, it was even better for allowance 2 lower utilizations than the previous rule, being better than rule 13. (Figure 6.13).

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.34	0.39	0.49	0.02	0.04	0.08	0.84	0.73	0.85	1.00	0.99	1.00	0.45	0.44	0.81
0.85	0.58	0.71	0.46	0.30	0.01	0.00	0.05	0.01	0.00	0.59	0.87	0.17	0.52	0.96	0.95
(U) 0.90	0.44	0.69	0.78	0.15	0.20	0.36	0.02	0.01	0.20	0.03	0.03	0.00	0.22	0.17	0.21
0.95	0.23	0.02	0.13	0.06	0.01	0.12	0.01	0.03	0.09	0.09	0.12	0.04	0.04	0.11	0.40
0.97	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.03	0.01	0.03	0.10	0.02	0.08

(a) T-Student test



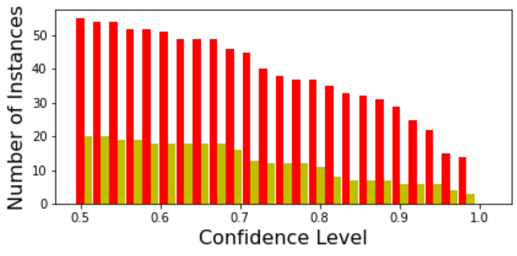
(b) Bar Plot

Figura 6.14: DR 13 comparison with DR 7

Rule 7 managed to be better than rule 13 for high allowances with low utilization. Although having a higher score than rule 6, it is not as competitive as the last one in relationship with the best rule. It loses in more scenarios, as can be seen by comparing its bar plot with the previous one (Figure 6.14).

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.28	0.40	0.38	0.00	0.00	0.02	0.80	0.28	0.13	1.00	1.00	1.00	0.95	0.07	0.58
0.85	0.71	0.72	0.69	0.06	0.00	0.00	0.01	0.00	0.00	0.68	0.96	0.54	0.45	0.29	0.95
(U) 0.90	0.82	0.79	0.88	0.10	0.16	0.12	0.00	0.00	0.00	0.00	0.01	0.01	0.26	0.27	0.28
0.95	0.80	0.69	0.80	0.31	0.08	0.50	0.05	0.17	0.05	0.17	0.08	0.05	0.11	0.02	0.45
0.97	0.27	0.33	0.68	0.32	0.07	0.19	0.39	0.24	0.06	0.09	0.19	0.04	0.33	0.09	0.05

(a) T-Student test



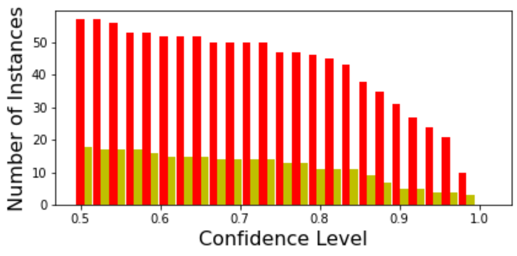
(b) Bar Plot

Figura 6.15: DR 13 comparison with DR 8

Rule 8 managed to be better with 95% confidence for allowances 6 for low utilization than rule 13. Furthermore, it is very competitive for allowances 2. None of the rules has an edge with 95% confidence in the relationship with the other for this allowance (Figure 6.15).

	(A)														
	2			3			4			6			8		
(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
0.80	0.34	0.60	0.41	0.01	0.03	0.05	0.92	0.84	0.57	0.97	0.99	1.00	0.89	0.19	0.44
0.85	0.86	0.74	0.65	0.10	0.03	0.00	0.03	0.00	0.00	0.45	0.84	0.16	0.47	0.88	0.98
(U) 0.90	0.78	0.79	0.87	0.22	0.27	0.11	0.00	0.00	0.00	0.00	0.00	0.00	0.16	0.12	0.19
0.95	0.50	0.25	0.45	0.26	0.05	0.35	0.16	0.11	0.15	0.13	0.07	0.07	0.14	0.17	0.02
0.97	0.10	0.03	0.07	0.13	0.04	0.04	0.10	0.04	0.02	0.10	0.04	0.16	0.12	0.03	0.02

(a) T-Student test



(b) Bar Plot

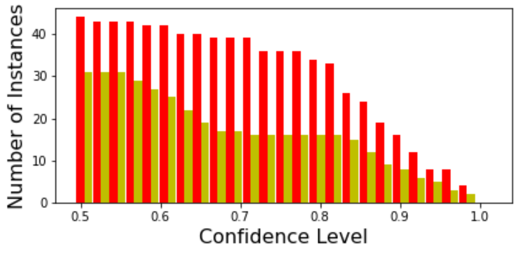
Figura 6.16: DR 13 comparison with DR 9

Rule 9 was better with 95% confidence for utilization 0.8 and allowance 6. Thus, this rule is even more competitive than the previous rule for low allowances except for very high utilizations

(0.97), where rule 13 always has an edge (Figure 6.16).

		(A)														
		2			3			4			6			8		
	(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
	0.80	0.61	0.88	0.83	0.57	0.55	0.84	0.86	0.61	0.11	1.00	0.94	0.99	0.84	0.11	0.22
	0.85	0.91	0.98	0.85	0.69	0.66	0.60	0.14	0.11	0.03	0.09	0.93	0.07	0.13	0.22	0.57
(U)	0.90	0.86	0.91	0.87	0.59	0.64	0.63	0.10	0.66	0.34	0.18	0.14	0.56	0.95	0.63	0.27
	0.95	0.50	0.15	0.38	0.42	0.17	0.18	0.13	0.13	0.18	0.59	0.39	0.07	0.06	0.18	0.20
	0.97	0.00	0.00	0.01	0.03	0.04	0.02	0.09	0.09	0.04	0.17	0.08	0.18	0.28	0.29	0.15

(a) T-Student test



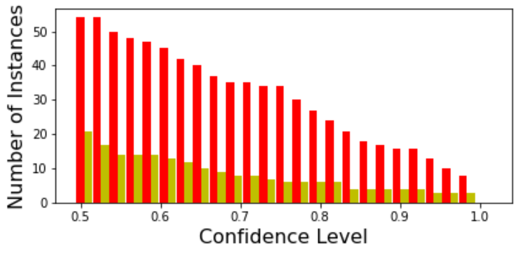
(b) Bar Plot

Figura 6.17: DR 13 comparison with DR 10

Rule 10 was very competitive with rule 13. It was the most competitive rule overall (trivial to see by analyzing the bar plots, it is the rule that manages to win in more scenarios). However, even for this rule, there was no confidence level in which the rule won in more scenarios than rule 13. Also, we can see a similar behaviour as the previous rule (at least it excels in the same scenarios) but more clearly than the other rule (Figure 6.17). Also, besides this rule having a worse score than rules 11 and 12, the global p-value test indicated it as being significantly better than those rules. Observing the bar-plot and comparing the corresponding one for rules 11 and 12 assure us that this rule is better than those two. This indicates that our global fitness score is not as good as the global p-value. The limitations of the global fitness score will be further seen in the discussion.

		(A)														
		2			3			4			6			8		
	(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
	0.80	0.24	0.44	0.23	0.02	0.01	0.03	0.41	0.46	0.05	1.00	0.99	1.00	0.81	0.08	0.51
	0.85	0.69	0.64	0.50	0.18	0.00	0.05	0.02	0.01	0.01	0.38	0.92	0.38	0.32	0.22	0.82
(U)	0.90	0.74	0.59	0.72	0.22	0.33	0.18	0.06	0.02	0.08	0.06	0.01	0.01	0.23	0.23	0.27
	0.95	0.67	0.45	0.52	0.54	0.47	0.54	0.36	0.16	0.21	0.47	0.37	0.38	0.16	0.47	0.14
	0.97	0.11	0.35	0.42	0.63	0.52	0.35	0.20	0.35	0.17	0.61	0.53	0.20	0.40	0.19	0.18

(a) T-Student test



(b) Bar Plot

Figura 6.18: DR 13 comparison with DR 11 and DR 12

Rules 11 and 12 are the same, so they were combined in the analysis. Rule 11/12 was vastly dominated by rule 13. It only has an edge for allowance 6 utilization 0.8. Nevertheless, this rule manages to have equilibrated scores being worse in only 16 scenarios with a confidence of 90%. (Figure 6.18).

6.3.3 Discussion of RQ3

The best expressions found have good behaviour in all shop scenarios, being the better rule overall (DR 13) significantly better than all the other tested rules. Nevertheless, even expressions with a weak fitness like (W+P, fitness 2.39) manage to be better in some scenarios than DR 13.

This paradox is a reminder that it is tough to have a single rule better than all the others in all shop conditions as stated in [8].

Also, this is a hint that it would be helpful to guarantee diversity in order to preserve a well-balanced population in all job shop scenarios. This diversity could be done by preserving expressions for future generations that are the best ones in one job shop scenario independently of the overall score.

We could explain the majority of the rules because they resemble some rules in the literature mainly PT+WINQ [13] and $PTe^{u} + PT\frac{S}{RPT}e^{-u} + WINQ$ (the RR rule [16]). However, our rules had very slight differences (our rule closer to PT+WinQ is P+W, which although looking equal to the other, the work in the next queue is computed in a slightly different way). There are two distinct ways to compute WinQ. One of them adds all the processing time of the jobs of the next queue (which is used in PT+WinQ). Another is an improved version that only sums the processing times of operations that would be assigned a better score than the current operation in this next machines' queue (it is the one we use in P+W first proposed in [13]). Our findings also concluded that this method to compute the work in the next queue is better. By seeing appendix A there is a preference between WinRR (the improved version) in relationship to the normal WinQ. For the RR rule, the difference is in the slack. The RR rule uses the normal slack, and we use the positive version of the slack max(0, slack). The DRs could be interpreted with ease until size 15. However, the explanation of the longer DRs (ids 13, 12, 11) was vaguer than the others. This difficulty is expectable. The longer the expression, the more challenging it is to explain (except for DR 8, which was also difficult to explain).

Our best rules behave better than all the benchmark rules, especially DR 13 that was 23% better than all the benchmark rules (global fitness of 0.772). So this rule was better performative than the RR and Covert and all the other rules we used as a benchmark and enumerated. Finally, using a DA grammar instead of STGP showed promising results. The best expression found was significantly better than the others and had dimension time square. This expression would be invalid in a simple STGP implementation. Restricting less the solution space appears to have worked well.

6.4 Global fitness metric limitations

The pairwise comparison between the rules revealed a particular limitation concerning our chosen global fitness indicator metric. Small changes in the indicator are sometimes unreliable and tend to induce us to the wrong conclusions, as seen in comparing rule 10 vs 11 and rule 6 vs rule 8. Using the methodology presented in Section 3.4.3, the 90% confidence intervals for the scores will be shown for DR 13. The objective is to see the minimum score differences between this rule and other rules that allow us to know with significance which rule is better. We will expose the variability of the global fitness metric, giving the 95% confidence intervals around the most probable value for the global fitness and infer a conclusion using that. Keep in mind that as discussed in 3.4.3 our fitness calculation always overestimates the most probable fitness value.

In order to evaluate the score variability, we will focus on a single rule (id 13). Table 6.23 shows rule 13 performance and confidence interval for that performance. In each cell is written the performance (F_{Ai}) and the deviation (z_{Ai}) for a given instance i (assuming that F_{Ai} can be modelled as a truncated Gaussian distribution). The 90% confidence interval for the performance in each instance is $F_{Ai} \pm z_{Ai}$.

Tabela 6.23: Test performance and confidence intervals of rule id 13. The confidence intervals are larger for allowances 6 and 8 than 2, 3, 4 (0.04 for allowances 2 up to 0.29 for allowances 8).

		(A)														
		2			3			4			6			8		
	(M)	50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
	0.80	0.94±0.04	0.95±0.04	0.95±0.04	0.83±0.07	0.83±0.07	0.84±0.07	0.69±0.12	0.68±0.11	0.67±0.11	0.50±0.29	0.46±0.28	0.57±0.32	1.00±0.29	1.00±0.34	1.00±0.27
	0.85	1.00±0.04	1.00±0.04	0.98±0.04	0.84±0.07	0.82±0.06	0.83±0.06	0.74±0.10	0.75±0.10	0.69±0.09	0.47±0.16	0.52±0.18	0.47±0.16	0.25±0.14	0.22±0.13	0.40±0.19
(U)	0.90	1.01±0.04	1.02±0.05	1.01±0.04	0.89±0.06	0.92±0.06	0.90±0.06	0.82±0.09	0.83±0.09	0.81±0.09	0.75±0.21	0.73±0.19	0.77±0.20	0.55±0.27	0.50±0.25	0.48±0.23
	0.95	0.97±0.05	0.96±0.04	0.97±0.05	0.92±0.06	0.91±0.05	0.93±0.05	0.81±0.07	0.81±0.06	0.83±0.06	0.85±0.13	0.82±0.13	0.79±0.13	0.78±0.21	0.84±0.22	0.85±0.23
	0.97	0.92±0.05	0.90±0.05	0.92±0.05	0.93±0.06	0.88±0.06	0.90±0.06	0.86±0.07	0.86±0.07	0.84±0.07	0.94±0.11	0.87±0.11	0.88±0.10	0.82±0.16	0.81±0.17	0.71±0.15

For rule 13, the obtained 95% confidence interval for the most probable value of the geometric mean was: [0.740 0.787], and the expected global fitness is 0.762. Our estimation of the score is 0.772. This implies that the global confidence interval has a length of 0.047. So using only the indicator, we can only say with a confidence of 95% that DR 13 is better than other DR x if our estimation of the global fitness score for DR x is at least (0.772 + 0.047/2) = 0.7955. Note that the global fitness metric hides some complexity. For example, if one DR have mean tardiness [8.1, 8.2, 8.3, 8.4] and the other have mean tardiness [8.2, 8.3, 8.4, 8.5], the first DR is always better than the second so that the pairwise test will give a significant result, but the geometric mean will be similar. Nevertheless, it is essential to see how the global score tends to vary because this is the only metric that GP uses when choosing which expressions to keep.

As the tardiness varies greatly for high allowance scenarios, we also checked how the metric varies if we disregarded the higher variance scenarios (allowance 6 and 8). The interval passes to [0.858 0.879], and the expected global fitness is 0.867. Moreover, our estimator is 0.870 (a lot closer than in the previous case). This closeness implies that the global confidence interval has a length of 0.021 (two times smaller than the previous case). Therefore, the global indicator varies a lot less, and with smaller score differences, significant results can be taken. This difference indicates that making more replicas for higher allowances values will be an excellent method to increase the confidence around our global fitness indicator, as it will reduce the score estimator variability. This will diminish the variability of those instances, protecting ourselves from rushed conclusions when the GP program is evolving.

Also, it is worth mentioning that the global-pvalue and the global fitness indicator disagreed on which expression was better in some cases. We can explain this disagreement because the two ways of calculation are very different. One uses the geometric mean on the instances, the other the arithmetic mean and analyses each replica individually. As the statistical tests are more robust when applying arithmetic means, we had to choose it to test the results. Nevertheless, all the points regarding why we choose the geometric mean as fitness are still valid, being the added interpretability of the score a clear benefit of this chosen metric.

Capítulo 7

Conclusion

This thesis explores three main research questions:

- **RQ1)** What is the trade-off between performance and interpretability when introducing dimensional awareness?
- **RQ2)** Can genetic programming algorithms find (near-)optimal Dispatching Rules (of a given size) for the Job Shop Scheduling problem?
- **RQ3)** How performative and explainable are the best Dispatching Rules found?

For **RQ1**, we implemented Dimensionally Aware Genetic Programming by using prevalent Grammar-Guided Genetic Programmmins algorithms (CFG-GP and GE) and compared them to standard GP. By analyzing the best expressions found by the algorithms, we saw a performance increase when using dimensionally aware algorithms, especially CFG-GP. Moreover, although the size of the expression was, on average bigger than standard GP when comparing both size and performance simultaneously, it is clear that for expressions of size 17 or bigger, CFG-GP always found more performative expressions than GP. For this question, if the DRs found by an algorithm for all sizes always have better performance than the DRs found by another algorithm for the same sizes, the first algorithm is more interpretable. Therefore, CFG-GP appears to be more interpretable than GP for DRs of size 17 or larger. Moreover, until size 11, the methods do not diverge, and between 11 and 15, one variant of GP and GE seem to be better. Interestingly enough, the GP expressions were dimensionally aware until size 11, which indicates that restricting expressions to be dimensionally aware does not prejudice the overall score, being a good heuristic.

Several things can be further studied for this question: first, as CFG-GP performed well, it would be interesting to apply them to other scheduling problems (e.g. Flexible Job Shop Scheduling Problem). Moreover, it would be interesting to compare it directly with STGP on an equal footing (i.e. representing the same restrictions). Also, further work can be done to understand if the DRs performance differences are mainly due to the algorithms uses or the DA's expressions for themselves. However, we postulate that DA expressions probably can generalize better than non DA expressions. Also, the GE methods did not seem promising. Nevertheless, other mapping procedures as used in GE variants SGE, HGE could increase the performance of GE in these problems.

For **RQ2**, we implemented an enumerator that evaluated all possible expressions with those function/terminal sets of a given size (restrict them to be dimensionally aware and unique). By ranking them and finding the best possible expressions on the training set, we found the optimal expression for a given size. If an algorithm explores the training set optimally, it will find that expression. Therefore, by comparing which expression the algorithms found, we could answer this question. GP and several GGGP methods explore optimally the spaces that we could enumerate and test, reaching if not the optimum very close expressions to the optimum for sizes inferior to 11. For size 11, we had to restrict the enumerator function/terminal set, and the enumerator did not enumerate all expressions. Nevertheless, it found a high performing one, serving as a benchmark. Three algorithms found expressions better than the benchmark rule, and three found expressions worse than the benchmark rule.

The enumerator was more feasible than initial considerations indicated, managing to enumerate expressions for a reasonable size and a limited function/terminal set. Several aspects of the enumerator can be explorer further: As it is 100 times less expensive to evaluate expressions in the training set than in the test set, the enumerator could be used in the training set for bigger sizes expressions (size 13 or even size 15) what could bring even better results. Furthermore, surrogate models can be implemented, vastly increasing the speed of evaluating the enumerated expressions. It would be easier to speed up the enumerator than a regular GP with a surrogate. Because GP has to distinguish between expressions with close scores, but as in the enumerator, most expressions are poorly performative; it only has to distinguish them from the good ones.

For **RQ3**, we evaluate the best expressions found overall (i.e. the ones that belonged to the Pareto frontier having to minimize expressions size and to maximize performance as objectives). We based our interpretations on rules in the literature (e.g. PT+WinQ and RR rules). We successfully explained all of the expressions except the three longest (sizes: 29, 21, 17). In those expressions, only minor considerations were made for some scenarios. Also, the best expressions found overall was significantly better than all other expressions, and as our benchmark rule incorporates several known rules in the literature (RR rule and Covert); it was also better than those. Moreover, using a complete DA grammar appears to have been promising as it allowed to generate expressions that could not be generated with a simple STGP approach (expressions with time square that were indeed the best performative expressions).

Furthermore, it is also worth understanding what affects our chosen performance metric. If considerations on this are not taken, expressions can be classified as better than others by luck. One way to reduce this effect would be by making more replicas for the instances where the variance is considerable. Also, the best overall rules can be more deeply compared to the RR and Covert rules and other offline meta heuristics both in terms of overall performance and execution speed.

Anexo A

Method to choose terminals

The same method exposed in Chapter 4 Section 4.2.2 will be used here to trim down a bigger set of terminals. The two-terminal lists are called GP-SMALL and GP-BIG and are exposed in Tables A.1 and A.2. Note that the "W"terminal in GP-SMALL corresponds to "WR"in GP-BIG.

Tabela A.1: Terminals used in GP-SMALL

Terminal	**Description**	**Units**
P	Processing time of current operation	time
W	Work in the next queue	time
N	Processing time of next operation	time
U	Shop utilisation	adimensional
$S+$	Slack if positive	time
$A+$	Allowance if positive	time
S	Slack	time
A	Allowance	time
C	Critical ratio if positive	adimensional
R	Remaining processing time	time

We use the Enumerator to generate expressions of sizes 3, 5, 7, 9 for each one of the terminal sets. In Table A.3 we can see how often do the symbols appear in the best x% expressions for GP-SMALL.

As we can see, the operation -"was the least used in the best expressions. Furthermore, all the terminal symbols are uniformly distributed in the 100% tag. Also, the positive variants of "S"and "A"("S+", "A+") appear more often. Moreover, "P", "W"and "C"are always the most used terminals, followed by U, R, and N.

In conclusion, we would remove -", "A", and "S"from the terminals list if we needed to trim down this terminal set further. Next, it would follow "A+"and the finalized terminal list would be: ("+", "*", "/", "P", "W", "R", "S+", "C", "U", "N"). This selection has its similarities with the terminals chosen in [6]. Their non-negative counterparts substituted the slack and the allowance. The significant difference is that using this method, U and N would be chosen.

Tabela A.2: Terminals used in GP-BIG. There are some terminals ptQ and QS that have as dimensions a number of jobs. However, as our implementation was only thought for one dimension, we will consider it adimensional in the calculations. It only cannot be added with time. Because of this, in the Units appears adimensional *.

Terminal	**Description**	**Units**
PT	Processing time of current operation	time
WR	Work in the next queue improved (sum of processin time of operations that would be assigned a better score than the current operation in the next machines' queue)	time
WQ	Work in the next queue (sum of the processing time of all operations in the next machines' queue)	time
WO	Work in all successive machines' queues	time
NP	Processing time of next operation	time
U	Shop utilisation	adimensional
$S+$	Slack if positive	time
$A+$	Allowance if positive	time
S	Slack	time
A	Allowance	time
RP	Remaining processing time	time
CR	Critical ratio if positive	adimensional
OCR	Critical ratio	adimensional
MD	Maximum due date time among jobs in queue	time
MP	Maximum processing time among operations in queue	time
mD	Minimum due date time among jobs in queue	time
mP	Minimum processing time among operations in queue	time
jt	Processing time of current operation	time
t	Time in the system	time
j	Number of jobs	adimensional*
ptQ	Processing time of queue	time
QS	Queue Size	adimensional*

Making the same analysis but to an even more extensive set of terminals GP-BIG, we get the results of Table A.4. Here the minus operation could also be discarded. For the terminals, the order by decreasing use is: ["WR", "PT", "WO", "mP", "MD", "MP", "U", "j", "ptQ", "MD", "QS", "WQ", "CR", "RP", "OCR", "NP", "S+", "A+", "S", "A", "jt", "t"]. If we needed to choose only 6 terminals, the list would be this one: "WR"(the next queue indicator that appears more often between "WR", "WO", and "WQ"), PT, MP, MD, MP, U. Here, the terminals chosen would be different from GP-SMALL except for "PT"and the "WR". Moreover, this selection would exclude terminals that give slack information ("S", "S+", "A", "A+", "C", "OCR"). Therefore, it would not generate the best rules. In conclusion, this selection method for this vast number of terminals did not work as well. Nevertheless, some similar intakes were also retrieved (A+ appears more times than A and S+ than S, and S+ more than A+), and "PT"and "WR"were chosen.

Finally, using an Enumerator to perform this task brings some advantages in relationship to GP. It is fair (it uses any symbol the same amount of times). GP would tend to overly explore a set of terminals and repeat many rules making the 100% distribution very different from a uniform one.

Tabela A.3: Number of symbols on the top x% best expressions of GP-SMALL. The Symbols names were compacted: P-PT, W-WinQ. R-RPT, S+-Slack+, C-CR, A+-Aplus ,U , N-NPT, S-Slack, A-Aplus

	Symbols													
	+	*	/	-	P	W	R	S+	C	A+	U	N	S	A
1%	2210	1341	696	332	1627	2122	399	270	786	212	445	379	157	183
5%	9648	7971	5779	3012	5642	10031	2957	3021	2768	1876	3327	3466	3021	1876
10%	18169	20150	16122	5922	10959	18867	7023	6740	7079	7490	6298	6653	4175	5000
20%	37133	39158	35896	13267	21567	30690	15584	16125	15488	15326	13664	12777	11238	15326
50%	87280	96628	91782	40046	44017	51363	37253	46780	37470	45291	34365	37531	38609	43058
100%	152589	189962	199814	137635	91616	91897	91571	92314	72593	92718	72753	91680	91475	91383

Tabela A.4: Number of symbols on the top x% best expressions of GP-BIG

Size 7	Symbols													
	+	*	/	-	PT	WR	WO	WQ	CR	OCR	MD	MP	mD	mP
1%	1966	1895	990	529	817	1588	880	232	220	159	318	298	274	358
5%	8443	9292	6348	3042	2104	5783	4884	1990	833	845	1852	1673	1724	1769
10%	16858	20431	15345	6625	4158	8887	6748	5533	2812	4218	3551	3271	3448	3410
100%	136143	202947	213398	127512	38969	39105	38912	38784	30513	30335	39050	38838	38805	38808
	A	A+	S	S+	jt	t	RP	NP	U	j	ptQ	QS		
1%	75	120	73	130	80	59	177	156	297	282	279	252		
5%	492	721	424	731	860	698	970	1204	1332	1430	1749	1373		
10%	1823	2744	1562	2050	2004	1708	2683	2715	2731	2812	3323	2731		
100%	39369	39159	38540	38324	38826	39102	38931	39139	30251	30620	39606	30533		

Anexo B

Algorithms performance evolution during training

This appendix explores how did the algorithms evolve during the training. The Enumerator, if picking the expressions randomly, will do the maximum exploration rather than exploitation. Therefore, it will probably be a point during the evolution in which the algorithms surpass the Enumerator. To see when this transition occurs is a good indicator of how fast does an algorithm converges. As the live Enumerator performed (random Search for sizes between 11-29) was worse than the expected performance of the random Enumerator (enumerating the small size expressions), the baseline for comparison will be the expected performance of the random Enumerator. In Figure B.1 both can be seen.

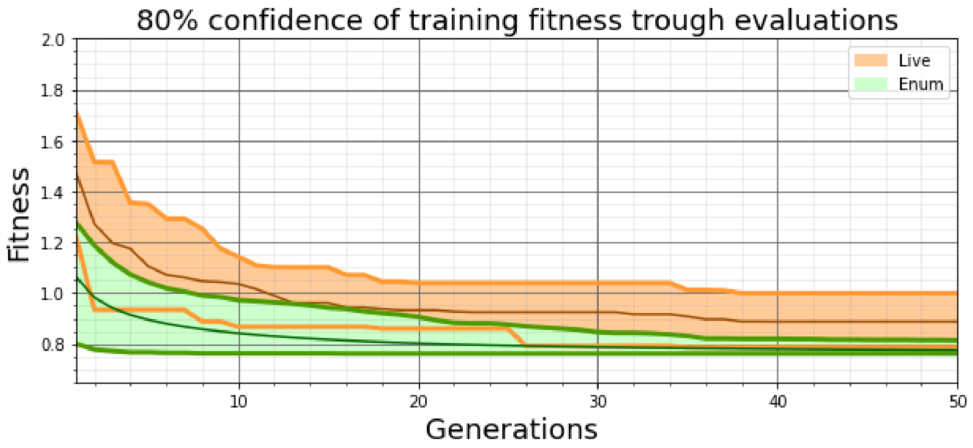


Figura B.1: Live expression sampler and Enumerator algorithms 80% confidence interval of training performance during the generations

It follows here the pairwise comparison of the training scores during evolution for the different algorithms. Being represented a graph for each algorithm type. One for GP (CR, AT, in Figure B.2) one for CFG-GP (RND, PTC2 in Figure B.3) one for GE-PURE (BAL, NBAL in Figure B.4) and one for GE-BIG (19,33 in Figure B.5). Care was taken to plot every graph with the same scale.

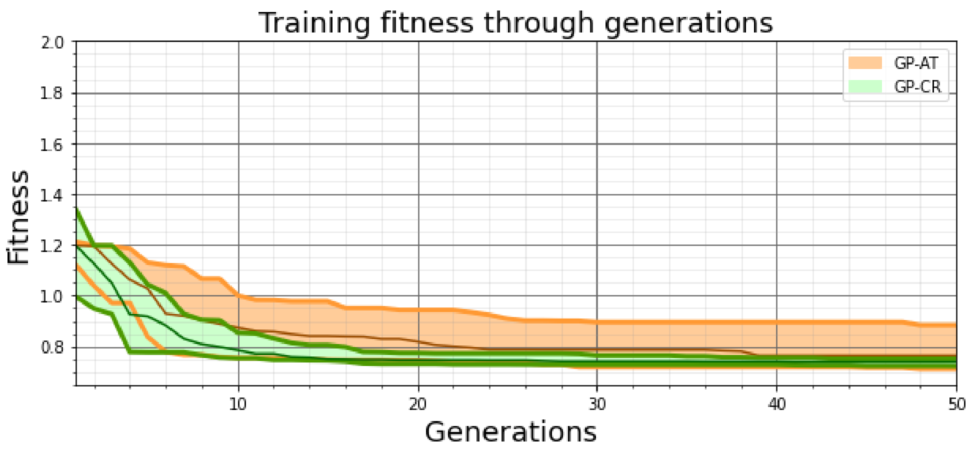


Figura B.2: GP algorithms 80% confidence interval of training performance during the generations

GP-CR dominates GP-AT in terms of continuous evolution. It is also noticeable that around generation 20, GP-CR had almost finished its evolution process. Nevertheless, it is also interesting that in the end, it is GP-AT that has the lowest lower bound for the best expression (its lower orange line is below the lower green line in the end). As GP-CR was better than GP-AT, it would be that one to be used for further comparisons.

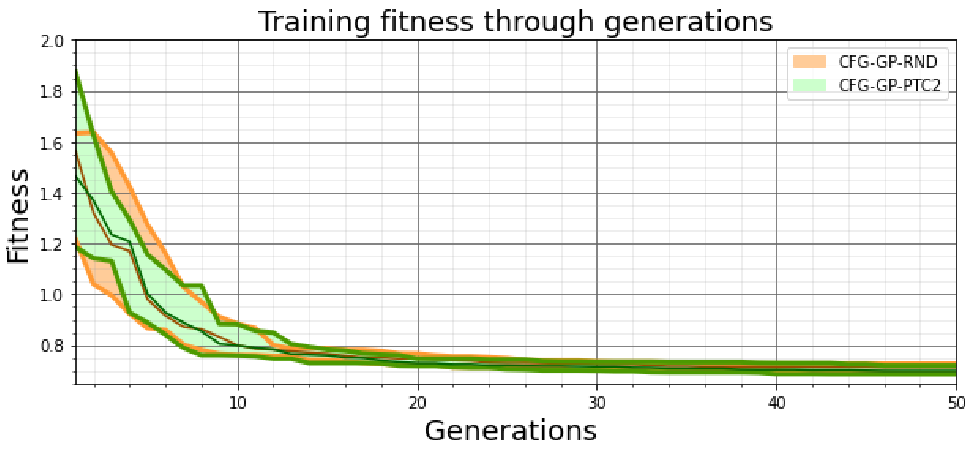


Figura B.3: CFG-GP algorithms 80% confidence interval of training performance during evolution generations

CFG-GP-RND and CFG-GP-PTC2 had very similar performances in this metric. Although for the rounds 2-8 rounds, the better initiation of PTC2 seemed to give an edge in relationship with CFG-RND in the evolution (the green part is below the orange one). However, this difference quickly faded around round 10, when the graphs became strikingly similar and consistently decreasing. As is necessary for only one to pass to further analysis (for plot perception), CFG-RND was chosen because it was the algorithm with the best overall fitness.

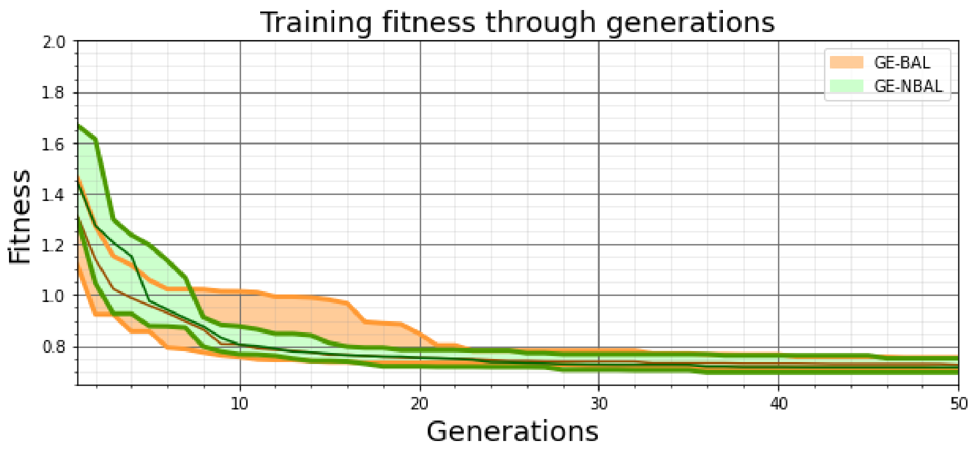


Figura B.4: GE-PURE algorithms 80% confidence interval of training performance during the generations

Comparing GE-BAL and GE-NBAL, we can see that GE-NBAL converged a lot faster consistently to reasonable solutions. Moreover, after that, GE-BAL catches it at around generation 23, and they become similar. So GE-NBAL was chosen for further testing.

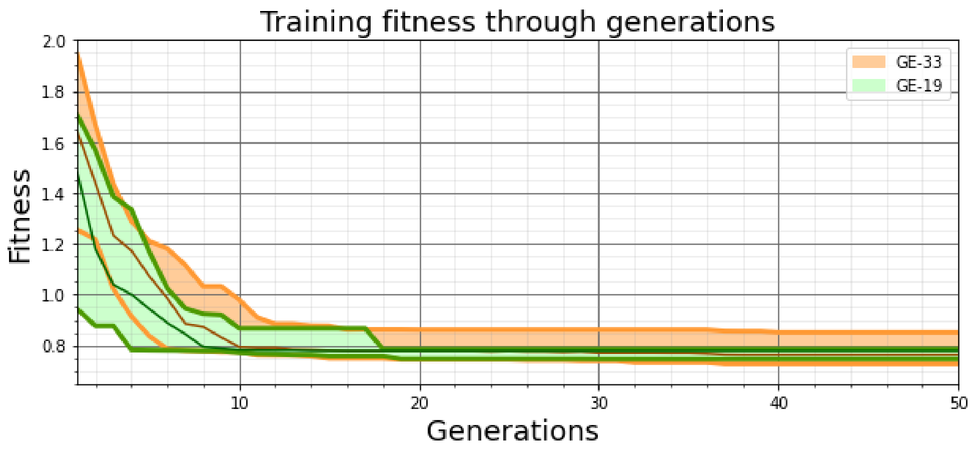


Figura B.5: GE-BIG algorithms 80% confidence interval of training performance during the generations

GE-BIG-19 was always below GE-BIG-33 (except for the top best-found solutions). Due to this fact and due to faster convergence, GE-BIG-19 will be used as a further comparison.

It is presented in Figure B.6 the evolution comparison between 3 grammar methods (CFG-GP-RND, GE-NBAL, GE-19).

Every method has more or less the same evolution curve initially, but to the end, CFG-GP-RND is consistently lower than GE-NBAL, which is consistently lower than GE-BIG-19.

It is presented in Figure B.7 the evolution comparison between the expected Enumerator, the CFG-GP-RND, and the GP-CR. This image intends to see two things. If there are severe divergences in the evolution format of GP-CR and CFG-GP-RND, furthermore to see when it is done

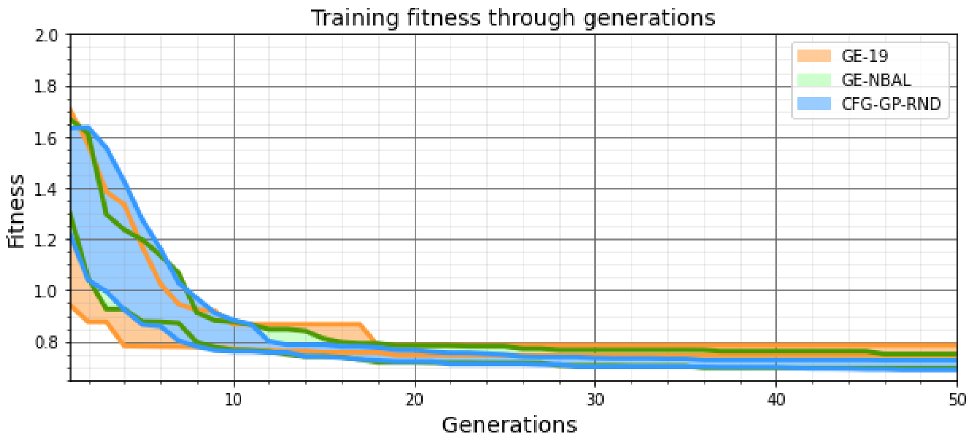


Figura B.6: Best Grammar algorithms 80% confidence interval of training performance during the generations

the crossover point in which the Enumerator starts losing for the other two methods.

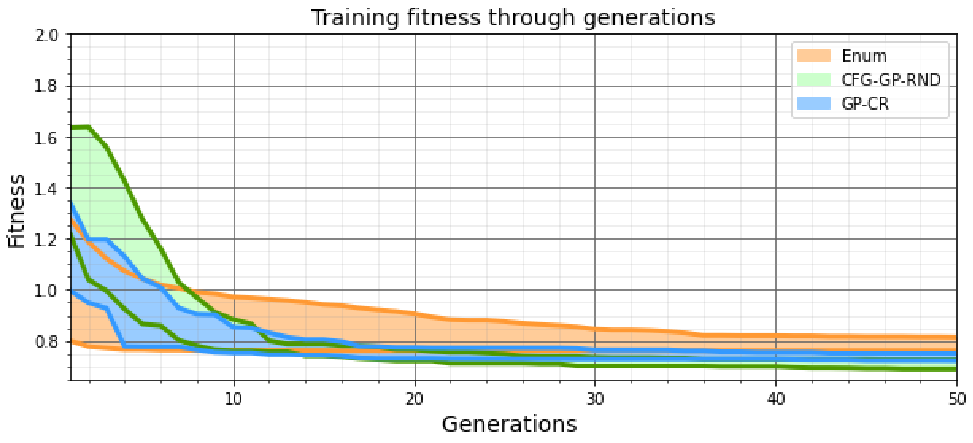


Figura B.7: Best algorithms 80% confidence interval of training performance during the generations

CFG-GP-RND stars in a much higher value than the others in the early evolutionary stage. Nevertheless, after the initial generations, it starts to improve faster.

We can see that the crossover point for the GP-CR is around generation 5, indicating that bigger initial populations (to a maximum of 1000= 200*5) will not prejudice the initial evolution steps. Furthermore, the crossover point of CFG-GP-RND is at generation 8, managing to be equilibrated with GP-CR after generation 10. It is clear that around generation 30, CFG-GP-RND managed to cross GP-CR, being consistently better after that point.

Anexo C

Grammars used

In this appendix, some grammars used are presented. In Figure C.1 the grammar used in CFG-GP and GE-NBAL is displayed. In Figure C.2 the grammar used in GE-BAL is displayed. The grammars for GE-BIG-19 and GE-BIG-33 are too big to be here (one has 1371 rules, the other 4670 rules). So a smaller example representing size GE-BIG-7 is represented in full in Figure C.3.

Figura C.1: Grammar of CFG-GP and GE-NBAL

$< start >::=< Nn2 >$
$< start >::=< Nn1 >$
$< start >::=< N0 >$
$< start >::=< N1 >$
$< start >::=< N2 >$
$< Nn2 >::= (+ < Nn2 >< Nn2 >)$
$< Nn2 >::= (- < Nn2 >< Nn2 >)$
$< Nn2 >::= (* < Nn1 >< Nn1 >)$
$< Nn2 >::= (* < N0 >< Nn2 >)$
$< Nn2 >::= (/ < Nn2 >< N0 >)$
$< Nn2 >::= (/ < Nn1 >< N1 >)$
$< Nn2 >::= (/ < N0 >< N2 >)$
$< Nn1 >::= (+ < Nn1 >< Nn1 >)$
$< Nn1 >::= (- < Nn1 >< Nn1 >)$
$< Nn1 >::= (* < N0 >< Nn1 >)$
$< Nn1 >::= (* < N1 >< Nn2 >)$
$< Nn1 >::= (/ < Nn2 >< Nn1 >)$
$< Nn1 >::= (/ < Nn1 >< N0 >)$
$< Nn1 >::= (/ < N0 >< N1 >)$
$< Nn1 >::= (/ < N1 >< N2 >)$
$< N0 >::= (+ < N0 >< N0 >)$
$< N0 >::= (- < N0 >< N0 >)$
$< N0 >::= (CR)$
$< N0 >::= (* < N0 >< N0 >)$
$< N0 >::= (* < N1 >< Nn1 >)$
$< N0 >::= (* < N2 >< Nn2 >)$
$< N0 >::= (/ < Nn2 >< Nn2 >)$
$< N0 >::= (/ < Nn1 >< Nn1 >)$
$< N0 >::= (/ < N0 >< N0 >)$
$< N0 >::= (/ < N1 >< N1 >)$
$< N0 >::= (/ < N2 >< N2 >)$
$< N1 >::= (+ < N1 >< N1 >)$
$< N1 >::= (- < N1 >< N1 >)$
$< N1 >::= (A^+)$
$< N1 >::= (PT)$
$< N1 >::= (WinQ)$
$< N1 >::= (RPT)$
$< N1 >::= (S^+)$
$< N1 >::= (* < N1 >< N0 >)$
$< N1 >::= (* < N2 >< Nn1 >)$
$< N1 >::= (/ < Nn1 >< Nn2 >)$
$< N1 >::= (/ < N0 >< Nn1 >)$
$< N1 >::= (/ < N1 >< N0 >)$
$< N1 >::= (/ < N2 >< N1 >)$
$< N2 >::= (+ < N2 >< N2 >)$
$< N2 >::= (- < N2 >< N2 >)$
$< N2 >::= (* < N1 >< N1 >)$
$< N2 >::= (* < N2 >< N0 >)$
$< N2 >::= (/ < N0 >< Nn2 >)$
$< N2 >::= (/ < N1 >< Nn1 >)$
$< N2 >::= (/ < N2 >< N0 >)$

Figura C.2: Grammar of GE-BAL

```
< start >::=< N0 >
< start >::=< N1 >
< start >::=< N2 >
< N0 >::= (+ < N0 >< N0 >)
< N0 >::= (+ < N0 >< N0 >)
< N0 >::= (+ < N0 >< N0 >)
< N0 >::= (− < N0 >< N0 >)
< N0 >::= (− < N0 >< N0 >)
< N0 >::= (− < N0 >< N0 >)
< N0 >::= (∗ < N0 >< N0 >)
< N0 >::= (∗ < N0 >< N0 >)
< N0 >::= (∗ < N0 >< N0 >)
< N0 >::= (/ < N0 >< N0 >)
< N0 >::= (/ < N1 >< N1 >)
< N0 >::= (/ < N2 >< N2 >)
< N0 >::= (CR)
< N0 >::= (CR)
< N0 >::= (CR)
< N0 >::= (CR)
< N0 >::= (CR)
< N0 >::= (CR)
< N0 >::= (CR)
< N0 >::= (CR)
< N0 >::= (CR)
< N0 >::= (CR)
< N0 >::= (CR)
< N0 >::= (CR)
< N1 >::= (+ < N1 >< N1 >)
< N1 >::= (+ < N1 >< N1 >)
< N1 >::= (− < N1 >< N1 >)
< N1 >::= (− < N1 >< N1 >)
< N1 >::= (∗ < N1 >< N0 >)
< N1 >::= (∗ < N1 >< N0 >)
< N1 >::= (∗ < N1 >< N0 >)
< N1 >::= (/ < N1 >< N0 >)
< N1 >::= (/ < N2 >< N1 >)
< N1 >::= (/ < N2 >< N1 >)
< N1 >::= (WinQ)
< N1 >::= (PT)
< N1 >::= (RPT)
< N1 >::= (A^+)
< N1 >::= (S^+)
< N1 >::= (WinQ)
< N1 >::= (PT)
< N1 >::= (RPT)
< N1 >::= (A^+)
< N1 >::= (S^+)
< N2 >::= (+ < N2 >< N2 >)
< N2 >::= (+ < N2 >< N2 >)
< N2 >::= (− < N2 >< N2 >)
< N2 >::= (− < N2 >< N2 >)
< N2 >::= (∗ < N1 >< N1 >)
< N2 >::= (∗ < N1 >< N1 >)
< N2 >::= (∗ < N2 >< N0 >)
< N2 >::= (/ < N2 >< N0 >)
< N2 >::= (/ < N2 >< N0 >)
< N2 >::= (/ < N2 >< N0 >)
< N2 >::= (∗ < PTt >< Aplust >)
< N2 >::= (∗ < PTt >< WinRRt >)
< N2 >::= (∗ < PTt >< rPtt >)
< N2 >::= (∗ < PTt >< Slackt >)
< N2 >::= (∗ < Aplust >< WinRRt >)
< N2 >::= (∗ < Aplust >< rPtt >)
< N2 >::= (∗ < Aplust >< Slackt >)
< N2 >::= (∗ < WinRRt >< rPtt >)
< N2 >::= (∗ < WinRRt >< Slackt >)
< N2 >::= (∗ < rPtt >< Slackt >)
< PTt >::= (PT)
< Aplust >::= (A^+)
< WinRRt >::= (WinQ)
< rPtt >::= (RPT)
< Slackt >::= (S^+)
```

Figura C.3: Grammar of GE-BIG-7

```
< start >::=< ex7N2 >
< start >::=< ex7N1 >
< start >::=< ex7N0 >
< start >::=< ex7Nn1 >
< start >::=< ex7Nn2 >
< start >::=< ex5N2 >
< start >::=< ex5N1 >
< start >::=< ex5N0 >
< start >::=< ex5Nn1 >
< start >::=< ex5Nn2 >
< start >::=< ex3N2 >
< start >::=< ex3N1 >
< start >::=< ex3N0 >
< start >::=< ex3Nn1 >
< start >::=< ex1N1 >
< start >::=< ex1N0 >
< ex7N2 >::= (/ < ex5N2 >< ex1N0 >)
< ex7N2 >::= (∗ < ex5N2 >< ex1N0 >)
< ex7N2 >::= (∗ < ex5N1 >< ex1N1 >)
< ex7N2 >::= (/ < ex3N2 >< ex3N0 >)
< ex7N2 >::= (/ < ex3N1 >< ex3Nn1 >)
< ex7N2 >::= (∗ < ex3N2 >< ex3N0 >)
< ex7N2 >::= (∗ < ex3N1 >< ex3N1 >)
< ex7N2 >::= (− < ex3N2 >< ex3N2 >)
< ex7N2 >::= (+ < ex3N2 >< ex3N2 >)
< ex7N2 >::= (/ < ex1N1 >< ex5Nn1 >)
< ex7N2 >::= (/ < ex1N0 >< ex5Nn2 >)
< ex7N2 >::= (∗ < ex1N1 >< ex5N1 >)
< ex7N1 >::= (/ < ex5N2 >< ex1N1 >)
< ex7N1 >::= (/ < ex5N1 >< ex1N0 >)
< ex7N1 >::= (∗ < ex5N1 >< ex1N0 >)
< ex7N1 >::= (− < ex5N1 >< ex1N1 >)
< ex7N1 >::= (+ < ex5N1 >< ex1N1 >)
< ex7N1 >::= (/ < ex3N2 >< ex3N1 >)
< ex7N1 >::= (/ < ex3N1 >< ex3N0 >)
< ex7N1 >::= (/ < ex3N0 >< ex3Nn1 >)
< ex7N1 >::= (∗ < ex3N2 >< ex3Nn1 >)
< ex7N1 >::= (∗ < ex3N1 >< ex3N0 >)
< ex7N1 >::= (− < ex3N1 >< ex3N1 >)
< ex7N1 >::= (+ < ex3N1 >< ex3N1 >)
< ex7N1 >::= (/ < ex1N1 >< ex5N0 >)
< ex7N1 >::= (/ < ex1N0 >< ex5Nn1 >)
< ex7N1 >::= (∗ < ex1N1 >< ex5N0 >)
< ex7N1 >::= (− < ex1N1 >< ex5N1 >)
< ex7N1 >::= (+ < ex1N1 >< ex5N1 >)
< ex7N0 >::= (/ < ex5N1 >< ex1N1 >)
< ex7N0 >::= (/ < ex5N0 >< ex1N0 >)
< ex7N0 >::= (∗ < ex5N0 >< ex1N0 >)
< ex7N0 >::= (− < ex5N0 >< ex1N0 >)
< ex7N0 >::= (+ < ex5N0 >< ex1N0 >)
< ex7N0 >::= (/ < ex3N2 >< ex3N2 >)
< ex7N0 >::= (/ < ex3N1 >< ex3N1 >)
< ex7N0 >::= (/ < ex3N0 >< ex3N0 >)
< ex7N0 >::= (/ < ex3Nn1 >< ex3Nn1 >)
< ex7N0 >::= (∗ < ex3N1 >< ex3Nn1 >)
< ex7N0 >::= (∗ < ex3N0 >< ex3N0 >)
< ex7N0 >::= (− < ex3N0 >< ex3N0 >)
< ex7N0 >::= (+ < ex3N0 >< ex3N0 >)
< ex7N0 >::= (/ < ex1N1 >< ex5N1 >)
< ex7N0 >::= (/ < ex1N0 >< ex5N0 >)
< ex7N0 >::= (∗ < ex1N1 >< ex5Nn1 >)
< ex7N0 >::= (∗ < ex1N0 >< ex5N0 >)
< ex7N0 >::= (− < ex1N0 >< ex5N0 >)
< ex7N0 >::= (+ < ex1N0 >< ex5N0 >)
< ex7Nn1 >::= (/ < ex5N0 >< ex1N1 >)
< ex7Nn1 >::= (/ < ex5Nn1 >< ex1N0 >)
< ex7Nn1 >::= (/ < ex3N1 >< ex3N2 >)
< ex7Nn1 >::= (/ < ex3N0 >< ex3N1 >)
< ex7Nn1 >::= (/ < ex3Nn1 >< ex3N0 >)
< ex7Nn1 >::= (∗ < ex3N0 >< ex3Nn1 >)
< ex7Nn1 >::= (− < ex3Nn1 >< ex3Nn1 >)
< ex7Nn1 >::= (+ < ex3Nn1 >< ex3Nn1 >)
< ex7Nn1 >::= (/ < ex1N1 >< ex5N2 >)
< ex7Nn1 >::= (/ < ex1N0 >< ex5N1 >)
< ex7Nn1 >::= (∗ < ex1N1 >< ex5Nn2 >)
< ex7Nn1 >::= (∗ < ex1N0 >< ex5Nn1 >)
< ex7Nn2 >::= (/ < ex5Nn1 >< ex1N1 >)
< ex7Nn2 >::= (/ < ex5Nn2 >< ex1N0 >)
< ex7Nn2 >::= (/ < ex3N0 >< ex3N2 >)
< ex7Nn2 >::= (/ < ex3Nn1 >< ex3N1 >)
< ex7Nn2 >::= (∗ < ex3Nn1 >< ex3Nn1 >)
< ex7Nn2 >::= (/ < ex1N0 >< ex5N2 >)
< ex7Nn2 >::= (∗ < ex1N0 >< ex5Nn2 >)
< ex5N2 >::= (/ < ex3N2 >< ex1N0 >)
< ex5N2 >::= (∗ < ex3N2 >< ex1N0 >)
< ex5N2 >::= (∗ < ex3N1 >< ex1N1 >)
< ex5N2 >::= (/ < ex1N1 >< ex3Nn1 >)
< ex5N2 >::= (∗ < ex1N1 >< ex3N1 >)
< ex5N1 >::= (/ < ex3N2 >< ex1N1 >)
< ex5N1 >::= (/ < ex3N1 >< ex1N0 >)
< ex5N1 >::= (∗ < ex3N1 >< ex1N0 >)
< ex5N1 >::= (− < ex3N1 >< ex1N1 >)
< ex5N1 >::= (+ < ex3N1 >< ex1N1 >)
< ex5N1 >::= (/ < ex1N1 >< ex3N0 >)
< ex5N1 >::= (/ < ex1N0 >< ex3Nn1 >)
< ex5N1 >::= (∗ < ex1N1 >< ex3N0 >)
< ex5N1 >::= (− < ex1N1 >< ex3N1 >)
< ex5N1 >::= (+ < ex1N1 >< ex3N1 >)
< ex5N0 >::= (/ < ex3N1 >< ex1N1 >)
< ex5N0 >::= (/ < ex3N0 >< ex1N0 >)
< ex5N0 >::= (∗ < ex3N0 >< ex1N0 >)
< ex5N0 >::= (− < ex3N0 >< ex1N0 >)
< ex5N0 >::= (+ < ex3N0 >< ex1N0 >)
< ex5N0 >::= (/ < ex1N1 >< ex3N1 >)
< ex5N0 >::= (/ < ex1N0 >< ex3N0 >)
< ex5N0 >::= (∗ < ex1N1 >< ex3Nn1 >)
< ex5N0 >::= (∗ < ex1N0 >< ex3N0 >)
< ex5N0 >::= (− < ex1N0 >< ex3N0 >)
< ex5N0 >::= (+ < ex1N0 >< ex3N0 >)
< ex5Nn1 >::= (/ < ex3N0 >< ex1N1 >)
< ex5Nn1 >::= (/ < ex3Nn1 >< ex1N0 >)
< ex5Nn1 >::= (/ < ex1N1 >< ex3N2 >)
< ex5Nn1 >::= (/ < ex1N0 >< ex3N1 >)
< ex5Nn1 >::= (∗ < ex1N0 >< ex3Nn1 >)
< ex5Nn2 >::= (/ < ex3Nn1 >< ex1N1 >)
< ex5Nn2 >::= (/ < ex1N0 >< ex3N2 >)
< ex3N2 >::= (∗ < ex1N1 >< ex1N1 >)
< ex3N1 >::= (/ < ex1N1 >< ex1N0 >)
< ex3N1 >::= (∗ < ex1N1 >< ex1N0 >)
< ex3N1 >::= (− < ex1N1 >< ex1N1 >)
< ex3N1 >::= (+ < ex1N1 >< ex1N1 >)
< ex3N0 >::= (/ < ex1N1 >< ex1N1 >)
< ex3N0 >::= (/ < ex1N0 >< ex1N0 >)
< ex3N0 >::= (∗ < ex1N0 >< ex1N0 >)
< ex3N0 >::= (− < ex1N0 >< ex1N0 >)
< ex3N0 >::= (+ < ex1N0 >< ex1N0 >)
< ex3Nn1 >::= (/ < ex1N0 >< ex1N1 >)
< ex1N1 >::= (S^+)
< ex1N1 >::= (RPT)
< ex1N1 >::= (WinQ)
< ex1N1 >::= (PT)
< ex1N1 >::= (A^+)
< ex1N0 >::= (CR)
```

Referências

[1] Su Nguyen, Yi Mei, e Mengjie Zhang. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems*, 3(1):41–66, 2017.

[2] Su Nguyen, Mengjie Zhang, Mark Johnston, e 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, 2012.

[3] Marko Đurasević, Domagoj Jakobović, e Karlo Knežević. Adaptive scheduling on unrelated machines with genetic programming. *Applied Soft Computing*, 48:419–430, 2016.

[4] Maarten Keijzer e Vladan Babovic. Dimensionally aware genetic programming. Em *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, páginas 1069–1076, 1999.

[5] Rachel Joanne Hunt, Mark Richard Johnston, e Mengjie Zhang. *Evolving dispatching rules with greater understandability for dynamic job shop scheduling*. Citeseer, 2016.

[6] Cristiane Ferreira, Gonçalo Figueira, e Pedro Amorim. Optimizing dispatching rules for stochastic job shop scheduling. Em *International Conference on Hybrid Intelligent Systems*, páginas 321–330. Springer, 2018.

[7] Michael Pinedo. *Scheduling*, volume 29. Springer, 2012.

[8] Marko Đurasević e Domagoj Jakobović. A survey of dispatching rules for the dynamic unrelated machines environment. *Expert Systems with Applications*, 113:555–569, 2018.

[9] J Christopher Beck e Nic Wilson. Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research*, 28:183–232, 2007.

[10] Djamila Ouelhadj e Sanja Petrovic. A survey of dynamic scheduling in manufacturing systems. *Journal of scheduling*, 12(4):417–431, 2009.

[11] Haldun Aytug, Mark A Lawley, Kenneth McKay, Shantha Mohan, e Reha Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161(1):86–110, 2005.

[12] Stephen R Lawrence e Edward C Sewell. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. *Journal of Operations Management*, 15(1):71–82, 1997.

[13] Oliver Holthaus e Chandrasekharan Rajendran. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics*, 48(1):87–105, 1997.

[14] Chandrasekharan Rajendran e Oliver Holthaus. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European journal of operational research*, 116(1):156–170, 1999.

[15] R Ramasesh. Dynamic job shop scheduling: a survey of simulation research. *Omega*, 18(1):43–57, 1990.

[16] TS Raghu e Chandrasekharan Rajendran. An efficient dynamic dispatching rule for scheduling in a job shop. *International Journal of Production Economics*, 32(3):301–313, 1993.

[17] EJ Anderson e JC Nyirenda. Two new rules to minimize tardiness in a job shop. *The International Journal of Production Research*, 28(12):2277–2292, 1990.

[18] Oliver Holthaus e Chandrasekharan Rajendran. Efficient jobshop dispatching rules: further developments. *Production Planning & Control*, 11(2):171–178, 2000.

[19] Michael R Garey, David S Johnson, e Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129, 1976.

[20] David H Wolpert e William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[21] Yong Zhou, Jian-jun Yang, e Zhuang Huang. Automatic design of scheduling policies for dynamic flexible job shop scheduling via surrogate-assisted cooperative co-evolution genetic programming. *International Journal of Production Research*, 58(9):2561–2580, 2020.

[22] Pádraig Cunningham, Matthieu Cord, e Sarah Jane Delany. Supervised learning. Em *Machine learning techniques for multimedia*, páginas 21–49. Springer, 2008.

[23] Helga Ingimundardottir e Thomas Philip Runarsson. Discovering dispatching rules from data using imitation learning: A case study for the job-shop problem. *Journal of Scheduling*, 21(4):413–428, 2018.

[24] Xiaonan Li e Sigurdur Olafsson. Discovering dispatching rules using data mining. *Journal of Scheduling*, 8(6):515–527, 2005.

[25] Gary R Weckman, Chandrasekhar V Ganduri, e David A Koonce. A neural network job-shop scheduler. *Journal of Intelligent Manufacturing*, 19(2):191–201, 2008.

[26] Jonathan J Oliver, Rohan A Baxter, e Chris S Wallace. Unsupervised learning using mml. Em *ICML*, páginas 364–372. Citeseer, 1996.

[27] Leslie Pack Kaelbling, Michael L Littman, e Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[28] Bernd Waschneck, André Reichstaller, Lenz Belzner, Thomas Altenmüller, Thomas Bauernhansl, Alexander Knapp, e Andreas Kyek. Optimization of global production scheduling with deep reinforcement learning. *Procedia Cirp*, 72:1264–1269, 2018.

[29] Shu Luo, Linxuan Zhang, e Yushun Fan. Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning. *Computers & Industrial Engineering*, página 107489, 2021.

[30] John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

[31] Joao Pedro Pedroso Ramos Dos Santos et al. Niche search and evolutionary optimisation. Relatório técnico, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), 1996.

[32] João Pedro Pedroso. Control of search parameters in evolutionary algorithms. Em *Proceedings of the International Symposium on Nonlinear Theory and its Applications*, páginas 1265–1268, 1997.

[33] Sean Luke e Lee Spector. A comparison of crossover and mutation in genetic programming. *Genetic Programming*, 97:240–248, 1997.

[34] Riccardo Poli, William B Langdon, Nicholas F McPhee, e John R Koza. *A field guide to genetic programming*. Lulu. com, 2008.

[35] Alain Ratle e Michèle Sebag. Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. Em *International Conference on Parallel Problem Solving from Nature*, páginas 211–220. Springer, 2000.

[36] Phillip Wong e Mengjie Zhang. Algebraic simplification of gp programs during evolution. Em *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, páginas 927–934, 2006.

[37] Peter A Whigham. Inductive bias and genetic programming. *IET*, 1995.

[38] Robert I Mckay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, e Michael O'neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3-4):365–396, 2010.

[39] Michael O'Neill e Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

[40] Nguyen Xuan Hoai, Robert Ian McKay, D Essam, e R Chau. Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: The comparative results. Em *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 2, páginas 1326–1331. IEEE, 2002.

[41] Nuno Lourenço, Joaquim Ferrer, Francisco B Pereira, e Ernesto Costa. A comparative study of different grammar-based genetic programming approaches. Em *European Conference on Genetic Programming*, páginas 311–325. Springer, 2017.

[42] Pak-Kan Wong, Man-Leung Wong, e Kwong-Sak Leung. Hierarchical knowledge in self-improving grammar-based genetic programming. Em *International Conference on Parallel Problem Solving from Nature*, páginas 270–280. Springer, 2016.

[43] Eric Medvet. Hierarchical grammatical evolution. Em *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, páginas 249–250, 2017.

[44] Peter A Whigham et al. Grammatically-based genetic programming. Em *Proceedings of the workshop on genetic programming: from theory to real-world applications*, volume 16, páginas 33–41, 1995.

[45] Gabriel Kronberger e Michael Kommenda. Search strategies for grammatical optimization problems—alternatives to grammar-guided genetic programming. Em *Computational Intelligence and Efficiency in Engineering Systems*, páginas 89–102. Springer, 2015.

[46] Miguel Nicolau. Understanding grammatical evolution: initialisation. *Genetic Programming and Evolvable Machines*, 18(4):467–507, 2017.

[47] Sean Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283, 2000.

[48] Franz Rothlauf e Marie Oetzel. On the locality of grammatical evolution. Em *European Conference on Genetic Programming*, páginas 320–330. Springer, 2006.

[49] Robin Harper. Ge, explosive grammars and the lasting legacy of bad initialisation. Em *IEEE Congress on Evolutionary Computation*, páginas 1–8. IEEE, 2010.

[50] Miguel Nicolau, Michael O'Neill, e Anthony Brabazon. Termination in grammatical evolution: Grammar design, wrapping, and tails. Em *2012 IEEE Congress on Evolutionary Computation*, páginas 1–8. IEEE, 2012.

[51] Alain Ratle e Michele Sebag. Grammar-guided genetic programming and dimensional consistency: application to non-parametric identification in mechanics. *Applied Soft Computing*, 1(1):105–118, 2001.

[52] Noam Chomsky e Marcel P Schützenberger. The algebraic theory of context-free languages. Em *Studies in Logic and the Foundations of Mathematics*, volume 26, páginas 118–161. Elsevier, 1959.

[53] Paweł Gawrychowski, Dalia Krieger, Narad Rampersad, e Jeffrey Shallit. Finding the growth rate of a regular or context-free language in polynomial time. *International Journal of Foundations of Computer Science*, 21(04):597–618, 2010.

[54] Elena Barcucci e M Cecilia Verri. Some more properties of catalan numbers. *Discrete mathematics*, 102(3):229–237, 1992.

[55] Kazuo Miyashita. Job-shop scheduling with genetic programming. Em *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, páginas 505–512, 2000.

[56] Christopher D Geiger, Reha Uzsoy, e Haldun Aytuğ. Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *Journal of Scheduling*, 9(1):7–34, 2006.

[57] Domagoj Jakobović e Leo Budin. Dynamic scheduling with genetic programming. Em *European Conference on Genetic Programming*, páginas 73–84. Springer, 2006.

[58] Li Nie, Yuewei Bai, Xiaogang Wang, e Kai Liu. Discover scheduling strategies with gene expression programming for dynamic flexible job shop scheduling problem. Em *International Conference in Swarm Intelligence*, páginas 383–390. Springer, 2012.

[59] Adil Baykasoğlu e Mustafa Göçken. Gene expression programming based due date assignment in a simulated job shop. *Expert Systems with Applications*, 36(10):12143–12150, 2009.

[60] David J Montana. Strongly typed genetic programming. *Evolutionary computation*, 3(2):199–230, 1995.

[61] Julian F Miller. Cartesian genetic programming. Em *Cartesian Genetic Programming*, páginas 17–34. Springer, 2011.

[62] Markus F Brameier e Wolfgang Banzhaf. *Linear genetic programming*. Springer Science & Business Media, 2007.

[63] Torsten Hildebrandt e Jürgen Branke. On using surrogates with genetic programming. *Evolutionary computation*, 23(3):343–367, 2015.

[64] Joc Cing Tay e Nhu Binh Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453–473, 2008.

[65] Su Nguyen, Mengjie Zhang, e Kay Chen Tan. Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. *IEEE transactions on cybernetics*, 47(9):2951–2965, 2016.

[66] Kenneth R Baker. Sequencing rules and due-date assignments in a job shop. *Management science*, 30(9):1093–1104, 1984.

[67] Su Nguyen, Yi Mei, Bing Xue, e Mengjie Zhang. A hybrid genetic programming algorithm for automated design of dispatching rules. *Evolutionary computation*, 27(3):467–496, 2019.

[68] John H Blackstone, Don T Phillips, e Gary L Hogg. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *The International Journal of Production Research*, 20(1):27–45, 1982.

[69] Sean Luke. Ecj then and now. Em *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, páginas 1223–1230, 2017.

[70] Marko Durasevic, Domagoj Jakobovic, Marcella Scoczynski Ribeiro Martins, Stjepan Picek, e Markus Wagner. Fitness landscape analysis of dimensionally-aware genetic programming featuring feynman equations. Em *International Conference on Parallel Problem Solving from Nature*, páginas 111–124. Springer, 2020.

[71] Reinhard Haupt. A survey of priority rule-based scheduling. *Operations-Research-Spektrum*, 11(1):3–16, 1989.