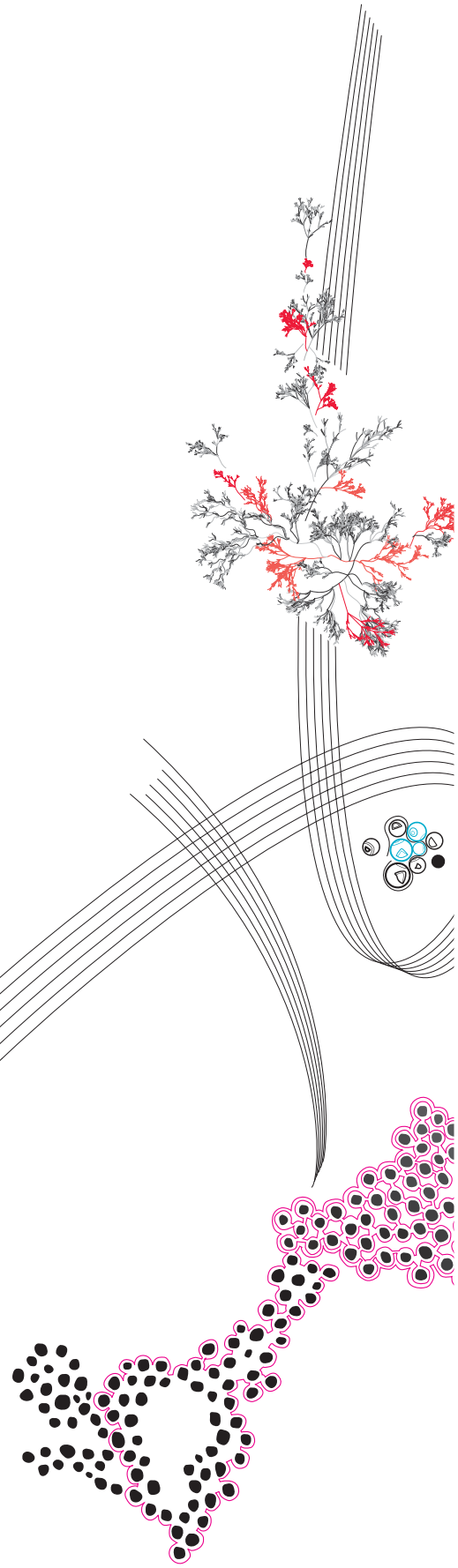


MSc Computer Science
Research Topics

Enhancing Portfolio Management Efficiency through Automated Scheduling: Harnessing Genetic Algorithms

Niels Bos

Supervisor: Dr. Ing. Moritz Hahn (UT) & Gido Cooiman (Fortes)

August, 2024

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

Abstract

Project Portfolio Management (PPM) is crucial for companies with many running projects. It strategically determines project initiation timing and optimally allocates resources, ensuring efficient project execution, maximizing organizational resources, and ultimately achieving strategic objectives. This thesis investigates the optimization of PPM efficiency through Genetic Algorithms (GAs) for the automated scheduling of projects within a portfolio, as crafting comprehensive capacity plans remains labour-intensive. This thesis describes a mapping from the Resource Constrained Project Scheduling Problem (RCPSP) to portfolio scheduling. It extends the RCPSP to accommodate diverse objectives, time-dependent resource capacities, and specific start/end time constraints to form the novel Multi-Objective RCPSP with time-varying resource capacities and demands and set start/-time constraints (MORCPSP/t-SE). To find optimal schedules that meet MORCPSP/t-SE constraints, the Nondominated Sorting Genetic Algorithm II (NSGA2) is proposed for its ability to optimize multiple objectives and offer a range of solutions. Moreover, the approach uses Swarm Particle and Bayesian optimization for hyperparameter optimization. The algorithm is validated against benchmark problems and the results are compared and analysed. Ultimately, this thesis seeks to contribute to an automated way of generating multiple portfolio scenarios simultaneously that provide insight into the effects of possible scheduling decisions.

Keywords: Project Portfolio Scheduling, Genetic Algorithm, NSGA2, Hyperparameter Optimization

Acknowledgements

First and foremost, I would like to thank my daily supervisor Gido Cooiman for the invaluable brainstorming sessions, guidance in the domain of portfolio management and overall insights throughout this journey. A special thanks also to my supervisor at the university, Moritz Hahn, for your guidance and feedback, which were greatly appreciated.

I am also grateful to Shenghui Wang for setting me on the right track at the start and providing valuable feedback at the end. Next, I would like to thank Marieke Huisman for being part of the committee and providing insightful feedback.

To Fortes and all the people working there, a big thank you. The opportunity for the project, the stimulating discussions, the educational AI-related journeys, and the fun times at the office significantly eased the research process.

To my fellow students at 'The Bridge', I would like to extend a huge thank you, for your support, motivation, and fun times that were crucial in keeping me inspired and driven.

Lastly, I want to express my deep gratitude to my family and friends. Your support throughout my academic years has been truly invaluable. Thank you for being part of this adventure.

Contents

0.1 Glossary and Acronyms . . . 4
Acronyms . . . 4

1 Introduction **6**
1.1 Motivation . . . 6
1.2 Research gap . . . 7
1.3 Fortes Change Cloud . . . 7
1.4 Research objectives . . . 7
1.5 Thesis structure . . . 8

2 Background **9**
2.1 Mapping portfolios to projects for automated scheduling . . . 9
2.2 Resource constrained scheduling problems . . . 10
2.3 Genetic algorithm . . . 11
2.3.1 Genetic encoding . . . 12
2.3.2 Crossover . . . 12
2.3.3 Mutation . . . 12
2.3.4 Evaluation and selection . . . 12
2.3.5 Applying genetic algorithms to portfolio scheduling . . . 12
2.3.6 Hyperparameters . . . 13
2.4 Pareto optimization . . . 13
2.5 Nondominated sorting genetic algorithm II . . . 15
2.6 Activity list . . . 16
2.7 Constraint handling . . . 17
2.7.1 Dynamic penalty . . . 18
2.7.2 Constrained dominance principle . . . 18
2.7.3 Violation as extra objective . . . 18
2.8 Hyperparameter optimization . . . 18
2.8.1 Bayesian optimization . . . 19
2.8.2 Particle swarm optimization . . . 19
2.9 Assessment of multi-objective optimizers . . . 19

3 Related work **21**
3.1 Project portfolio selection and scheduling . . . 21
3.2 Project scheduling . . . 22
3.3 Bridging literature and methodology . . . 23

4 Formal problem definition **25**

5 Methodology **28**
5.1 Genetic algorithm approach . . . 28
5.1.1 Non-dominated sorting . . . 28
5.1.2 Crowding distance . . . 29
5.1.3 Individual representation . . . 31
5.1.4 Fitness functions . . . 31
5.1.5 Initial population . . . 31
5.1.6 Crossover . . . 32
5.1.7 Mutation . . . 32
5.1.8 Selection . . . 33
5.1.9 Overview of novelties . . . 33
5.2 Additional constraint handling . . . 33
5.2.1 Start and end time constraints . . . 34
5.2.2 Capacity allocation constraints on project categories . . . 34
5.3 Hyperparameter optimization . . . 36
5.3.1 Pareto front evaluation . . . 36
5.3.2 Individual optimization . . . 36
5.3.3 Hyperparameter optimization . . . 37
5.4 Validation . . . 38
5.4.1 Dataset . . . 38
5.4.2 Benchmark comparison . . . 39
5.4.3 Manual validation . . . 39

6 Results & analysis **41**
6.1 Experimental setup . . . 41
6.2 Constraint handling comparison . . . 41
6.3 Hyperparameter optimization . . . 42
6.3.1 Mutation rate . . . 43
6.3.2 Crossover rate . . . 43
6.3.3 Number of crossover points . . . 43
6.3.4 Bayesian and Particle Swarm Optimization . . . 43
6.4 Benchmark comparison . . . 47
6.5 Manual runs . . . 51
6.5.1 Results of problem instance j30t1_5_4 . . . 52
6.5.2 Results of problem instance j120t1_2_3 . . . 54
6.5.3 Results of problem instance j120t4_5_6 . . . 54
6.5.4 Results of problem instance j120t6_7_8 . . . 54
6.5.5 Summary of findings from manual runs . . . 58

7 Conclusions **59**
7.1 Main contributions . . . 59
7.2 Genetic algorithm for project scheduling . . . 59
7.3 Metrics for assessing quality . . . 61
7.4 Future work . . . 62

References **64**

Appendix **70**

A Schedule visualisations **70**

0.1 Glossary and Acronyms

Acronyms

AHP Analytic Hierarchy Process. 21, 22

AL Activity List. 16, 17, 31, 32, 60

BO Bayesian Optimization. 19, 36–38, 41, 43, 59, 61

DCDP Dynamic Constrained Dominance Principle. 35, 41, 42, 54, 60

FCC Fortes Change Cloud. 7, 26

FTE Full-Time Equivalents. 9, 11

GA Genetic Algorithm. 6–9, 11–13, 16–19, 21–24, 31, 36, 59–61

HGA Hybrid Genetic Algorithm. 22, 23

HPO Hyperparameter Optimization. 18, 19, 36, 37, 43, 47, 59, 61

HV Hypervolume. 19, 20, 35, 36, 41–46, 60, 61

IGD Inverted Generational Distance. 19, 20, 36

MOEA Multi-Objective Evolutionary Algorithms. 17, 21, 22

MORCPSP/t-SE Multi-Objective Resource Constrained Project Scheduling Problem with time-varying resource capacities and demands and start/end time constraints. 25, 28, 31, 33, 59–61

MRCMPSP Multi-Mode Resource-Constrained Multiple Project Scheduling Problem. 10, 23

MRCPSP Multi-Mode Resource Constrained Project Scheduling Problem. 10, 22, 23

NAC No Additional Constraints. 51–58

NSGA-II Nondominated Sorting Genetic Algorithm II. 15, 16, 18, 21–23, 28, 33, 35, 36, 42, 59, 61, 62

PCC Project Category Constrained. 51–55, 57, 58

PCSEC Project Category Start/End time Constrained. 51–55, 57, 58

PPM Project Portfolio Management. 6, 8, 9, 21–24, 59, 62

PPSS Project Portfolio Selection and Scheduling. 21–24

PSO Particle Swarm Optimization. 19, 21–23, 36–38, 41, 43, 47, 59–62

RCMPSP Resource-Constrained Multiple Project Scheduling Problem. 10

RCPSP Resource Constrained Project Scheduling Problem. 10, 11, 16, 22–25, 36, 38, 62

RCPSP/t Resource Constrained Project Scheduling Problem with time-dependent resource capacities and requests. 10, 25, 38, 59, 60

SA Simulated Annealing. 22, 60, 62

SGS Serial Generation Scheme. 31

SPEA Strength Pareto Evolutionary Algorithm. 22

TCTP Time/Cost Trade-off Probem. 10

Chapter 1

Introduction

1.1 Motivation

In the fast-moving and changing economy, Project Portfolio Management (PPM) is essential to maintain a clear overview of running and upcoming projects in an organization. PPM aims to coordinate projects that compete for the same resources while contributing to the same goals, for strategic benefits [50]. The goals of PPM can be distinguished as maximising the portfolio's value, seeking the right balance between projects, ensuring that the portfolio is strategically aligned and ensuring not too many running projects simultaneously for limited resources [50]. PPM has become standard practice in companies' management and has been a highly researched topic in management and product development management research [42]. Moreover, it has been developed in global standards [51], showing PPM's importance and impact in organizations.

The actual practice of PPM in an organization's context is often more difficult than the easy-to-understand frameworks suggest [6][51]. As established by Blichfeldt and Pernille [9], organizations often encounter resource-related challenges in their portfolio planning. These challenges manifest in various forms, such as the simultaneous selection of too many projects without adequate resources or the oversight of small projects consuming resources without proper accounting in PPM frameworks. Related to such resource problems, Engwall and Jebrant [18] identified the resource allocation syndrome. In the researched organizations, the management issues revolved around resources, and how they were allocated and redistributed if projects lagged behind their schedules. They state that, often, resource and schedule estimates are too strict. Lastly, one difficulty was handling changes within projects, roles or responsibilities [17]. Overall, while PPM in theory is a perfect solution for organizations with many running projects simultaneously, it is challenging to use it in practice optimally.

There is an interest in automating project portfolio planning to bridge the gap between theoretical ideals and practical challenges in PPM. Automating the portfolio planning process provides a way to quickly create portfolio plannings that adhere to capacity, time and other predefined constraints. It offers a compelling solution for a systematic and algorithmic approach to portfolio planning. By leveraging automation techniques, organizations can address resource-related issues, optimize project selection, and ensure a balanced allocation of resources in line with project demands much faster.

One promising approach to automate this process is using *Genetic Algorithms* (GAs). GAs are a type of evolutionary algorithm, that mimics the process of natural selection to get to near-optimal solutions. GAs iteratively select, mutate, and recombine solutions to improve them. Their adaptability and efficiency in exploring large search spaces make them

particularly suitable for complex scheduling tasks, such as scheduling in project portfolio management. An important aspect of GAs is tuning *hyperparameters*, which influence factors such as recombination and mutation rates. Properly setting these hyperparameters is crucial for the algorithm's efficiency and quality of solutions.

1.2 Research gap

While automated scheduling has been extensively researched for many years, with a particular focus on project scheduling problems, there remains a notable gap in research regarding the automation of portfolio schedules. The gap lies specifically in the domain of scheduling project portfolios with certain constraints such as set start or end times, precedence relationships and resource constraints, while optimizing the schedule on multiple objectives [13][8][65][1]. Although the fundamentals overlap with project planning, the complexities introduced in a portfolio context justify dedicated research. Additionally, existing research often simplifies scenarios, making them unsuitable for the complexity of real-world situations. Therefore, there is a need to extend research efforts to incorporate real-life cases, ensuring that automated portfolio planning solutions are applicable and effective in practical organizational contexts.

1.3 Fortes Change Cloud

This research is in cooperation with Fortes[2]. Fortes is a Dutch company located in Enschede. Fortes Change Cloud (FCC) is the standard tool for large public organizations like the ministries of the Netherlands and is also used by large enterprises such as Vopak and Friesland Campina. FCC is a software application, in which organizations manage their entire change portfolio and maximize strategic value creation. One of their core features is the multi-aspect capacity planning overview on skills, finance, and objectives, which visualizes projects' required skills and the capacity of skills. Also, it allows companies to move projects around in time to create portfolio planning and identify possible bottlenecks.

Another important aspect of capacity planning is the scenario mode. Scenario mode functions as a playground for portfolio managers to see the effect of certain choices. For instance, it allows projects to be moved in time, automatically updating the required skills in a certain time accordingly. Afterwards, the scenario can be adopted, saved or discarded.

At the moment, the capacity planning and scenario creation is a manual process. Automation is essential to manage growing portfolio complexity and maximize strategic value. Fortes is interested in project portfolio schedule scenario automation and a way to generate portfolio scenarios based on user input to fill this gap.

1.4 Research objectives

The goal of this thesis is to find an efficient method for automating project portfolio scheduling, ensuring adherence to predefined constraints while optimizing multiple objectives. These objectives often include minimizing duration and cost and maximizing resource utilization, which are crucial for achieving strategic alignment. This goal will be achieved through several key steps. First, a comprehensive understanding of current practices in automated scheduling will be developed. Subsequently, a novel algorithm will be

[1] This gap was confirmed through casual conversations at the company Fortes

[2] www.fortesglobal.com

designed for portfolio automation, considering real-world organizational scenarios. Lastly, the algorithm's effectiveness will be validated against benchmark problems and with manual testing and validation. This leads to the following main research question:

How can genetic algorithms be effectively employed to generate diverse project portfolio scenarios that comply with predefined constraints while optimizing multiple objectives?

The main research question will be divided into two subquestions with additional subquestions. These are:

RQ1: How can a genetic algorithm solve the multi-objective Resource-Constrained Project Scheduling Problem with set start/end time constraints and capacity allocation to project categories constraints?

- **RQ1.1:** How can a project portfolio schedule be structured and encoded for use within a genetic algorithm framework?
- **RQ1.2:** How can set start/end time constraints be formalized and included in the GA?
- **RQ1.3:** How can additional capacity allocation to project categories constraints be included in the GA and be solved?
- **RQ1.4:** What existing methods or frameworks are suitable for solving multi-objective scheduling problems, and how can they be adapted or extended to solve the problem at hand?

RQ2: What criteria and performance metrics can be used to assess the quality and effectiveness of the GA and the generated project portfolio scenarios?

- **RQ2.1:** How can optimal values for hyperparameters of the GA be determined?
- **RQ2.2:** How can a generated solution be validated without knowing the actual optimal solution?
- **RQ2.3:** How will validation data with optimal solutions be gathered or created?

1.5 Thesis structure

This thesis is structured as follows. Chapter 2 provides fundamental background knowledge, including methods and concepts. Chapter 3 highlights related works in the domain of automating PPM and project scheduling. Chapter 4 gives a formal definition of the problem and Chapter 5 explains the methods of solving the problem, optimizing the algorithm and validating the algorithm. The results and an analysis thereof are stated in Chapter 6, in the same structure as Chapter 5. Subsequently, a conclusion and future work are provided in Chapter 7.

Chapter 2

Background

This chapter describes the fundamental concepts and techniques utilized throughout the thesis. It begins by examining the critical process of mapping portfolios to projects, which sets the stage for applying established project scheduling methods to the broader domain of portfolio management. Next, the chapter defines various project scheduling problems and their associated terminology, followed by an explanation of GAs, which are employed to solve these predefined problems. Subsequently, the concepts of Pareto optimization are described, often used for multi-objective optimization, along with a GA approach to optimize on multiple objectives. Then, the chapter describes methods to incorporate constraint handling within a GA. Finally, techniques for hyperparameter tuning and ways to assess the outcomes of multi-objective optimizers are discussed.

2.1 Mapping portfolios to projects for automated scheduling

Project scheduling has been a active research topic for many years. To enable the established techniques originally designed for project scheduling, portfolios will be mapped to projects. This mapping indicates the fundamental similarities between portfolios and projects regarding resource-constrained scheduling.

Both projects and portfolios have a certain set of processes, namely activities and projects respectively. Projects within a portfolio can be compared to activities within a project, as both represent discrete units of work. Just as activities, projects must be scheduled within certain precedence and resource constraints. This mapping can be made since PPM does not consider the individual tasks within a project, instead, PPM aims at strategic planning on a larger scale. Lastly, the optimization objectives are typically the same when automating the scheduling process for both projects or portfolios. Often, the duration, costs or resource usage are minimised.

With the mapping established, the techniques and definitions for project scheduling can be extended to the domain of PPM. Section 2.2 will go more in-depth on the variations of project scheduling.

Throughout this thesis, an example portfolio will be used to provide easy-to-understand examples. The portfolio consists of eight projects, named: Alpha, Beta, Gamma, Delta, Epsilon, Zeta, Eta, and Theta. The example organisation managing this portfolio is a company with 6 employees able to work on the projects, equating to a capacity of 6 Full-Time Equivalents (FTE). The company's goal is to complete these projects as quickly as possible while spreading its resources as evenly over time as possible. Each project in the portfolio has varying durations, resource demands and dependencies on other projects, adding complexity to the resource allocation and project scheduling processes. The next

sections will continue with this example.

2.2 Resource constrained scheduling problems

In the past decade, substantial efforts have been invested in automating scheduling, primarily through the application of neural networks, machine learning, and constraint programming [41][8]. Many variations of the scheduling problem are described in the literature [45]. One fundamental variant is the Basic Project Scheduling Problem, in which activities are scheduled with precedence constraints. This scheduling problem encompasses challenges related to optimizing project duration, resource allocation, and estimating project costs, among others [45].

Delving deeper into project scheduling literature, different problem variations have been identified, each catering to specific dimensions of complexity:

- **Resource Constrained Project Scheduling Problem (RCPSP) [35][30][49][67]:**
  - Objective: Schedule activities subject to both precedence and resource constraints.
- **Resource-Constrained Project Scheduling Problem with Multiple Objectives (Multi-Objective RCPSP) [23]:**
  - Extension of RCPSP: Incorporates multiple objectives such as costs and makespan.
- **Resource Constrained Project Scheduling Problem with time-dependent resource capacities and requests (RCPSP/t) [28] [29]**
  - Extension of RCPSP: Incorporates resource capacities and requests varying over time.
- **Multi-Mode Resource Constrained Project Scheduling Problem (MRCPSP) [3][66]:**
  - Extension of RCPSP: Allows activities to be performed in multiple modes, introducing flexibility in execution.
- **Resource-Constrained Multiple Project Scheduling Problem (RCMPSP) [57]:**
  - Extension of RCPSP: Involves scheduling multiple projects on the same resources, navigating shared constraints.
- **Multi-Mode Resource-Constrained Multiple Project Scheduling Problem (MRCMPSP) [60][39][7]:**
  - Combination of MRCPSP and RCMPSP: Addresses scheduling complexities arising from both multi-mode activities and interdependencies across multiple projects.
- **Time/Cost Trade-off Probem (TCTP) [20][39][7]:**
  - Objective: Seek an optimal solution that balances time and cost considerations in scheduling.

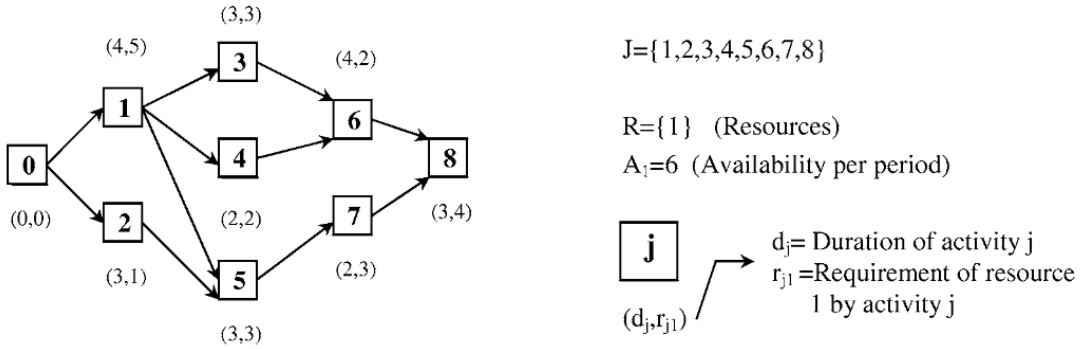


Figure 2.1: Example portfolio visualised by Alcaraz and Maroto [2]

Understanding the terminology associated with these scheduling problems is crucial. Resource constraints impose limitations on project resources, such as machines or personnel. These constraints are dictated by the maximum hours the resources can work. In this thesis, consumable resources such as building materials are not included, since portfolio management generally operates at a higher level, not regarding consumable resources. Precedence constraints establish a sequential order for activities, where the completion of one activity is a prerequisite for starting another. Multi-objective optimization introduces the challenge of optimizing various variables simultaneously, such as business value, cost reduction, or project duration. Multi-mode scheduling accommodates activities associated with different modes, allowing for diverse execution methods or execution by different resources. Lastly, multi-project scheduling deals with the simultaneous scheduling of multiple projects within the same portfolio, necessitating the consideration of shared constraints and interdependencies among projects and their activities.

Continuing with the example portfolio introduced in Section 2.1, the projects are labeled numerically, with Alfa being 1 and Theta being 8. Each project is assigned specific durations, resource demands, and precedence relations, as depicted in Figure 2.1. Project 0 serves as the starting point, with arrows indicating the precedence relations between projects. For example, project 2 must finish before starting project 5, due to some dependency. The tuple associated with each node represents the duration and resource demand, respectively. For example, project 1 demands a 5 FTE for each of its 4 timeslots. These new constraints give the portfolio a structure similar to a RCPSP.

2.3 Genetic algorithm

A GA is a meta-heuristic strategy that is inspired by the process of evolution. In nature, favorable traits for survival are inherited and accumulated over successive generations, leading to the adaptation of species to survive their environment. Similarly, GAs iteratively improve solutions by mimicking survival, genetic *crossover*, and *mutation*. This approach offers an effective and efficient method for solving complex problems in a relatively short time.

The following subsections explain the principles of GAs using the following scenario. Consider the simple problem of finding the maximum value of the mathematical function $f(x) = -x^2 + 14x$ with $0 \leq x < 16$.

2.3.1 Genetic encoding

For the GA, the solutions must be represented as chromosomes with genes. Often, solutions are encoded as binary digits, but can also be encoded as integers or complex structures. Regarding the simple problem mentioned before, a solution, x, can be represented by a 4-bit binary string. The chromosome would look something like **'1100'**, representing the number 12, where a bit represents a gene in the chromosome.

2.3.2 Crossover

To generate new solutions, old solutions are combined, simulating the process of genetic recombination. It involves two parents exchanging genes to create offspring. The goal is to combine favorable traits to generate better solutions. There are multiple methods for crossover, for instance, chromosomes can be swapped, or chromosomes can be sliced and exchanged. Let us consider two parent solutions, **'1100'** and **'1010'** (see Section 2.3.1 for the encoding explanation). The crossover operation is swapping the first half of genes. This results in the offspring: **'1000'** and **'1110'**

2.3.3 Mutation

Mutation introduces small, random changes to an individual's genes, to generate more variation in generations. This helps to explore new regions in the solution space that may not be accessible by genetic crossover alone. In the example, a mutation operator might be flipping a bit in the chromosome. therefore the individual with chromosome **'1110'** could be mutated to **'1111'** by swapping the last bit.

2.3.4 Evaluation and selection

The process of evaluating and selection is a critical component in GAs. When a new generation is formed, the fitness score of an individual is accessed with a fitness function. The fitness function evaluates how well a solution performs concerning the optimization criteria and the objective function of the problem. When the fitness of the individuals in the population is assessed, the best-scoring individuals are selected to form the next generation. Continuing with the example formula, $f(x) = -x^2 + 14x$ with $0 \leq x < 16$, consider the following population [**'0101'**, **'0110'**, **'1110'**].

- Chromosome 1: $f(5) = 45$
- Chromosome 2: $f(6) = 48$
- Chromosome 3: $f(14) = 0$

Chromosomes 1 and 2 have the highest fitness, therefore they will be selected to become part of the new generation.

This iterative process of evaluation, selection, and genetic operations continues across generations, gradually refining the population and moving towards solutions that maximize the objective function.

2.3.5 Applying genetic algorithms to portfolio scheduling

Continuing with the example introduced in the previous sections, a schedule for the portfolio can be represented as an ordered list of projects. The initial population for the GA consists of a set of randomly generated project schedules. For readability, the projects will

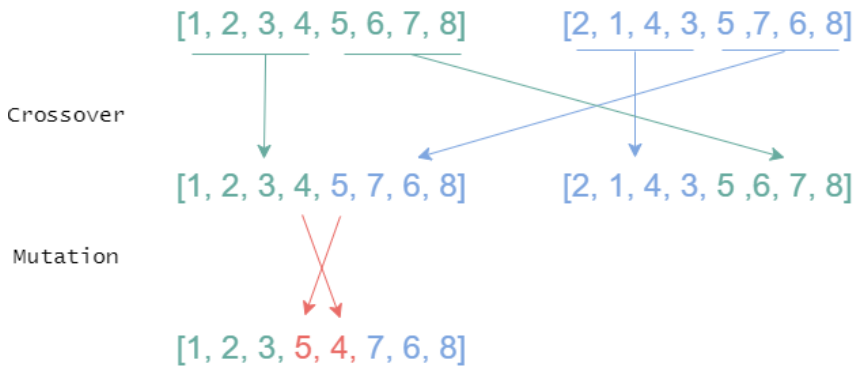


Figure 2.2: Crossover and mutation visualization

be denoted by their numeric labels (see Section 2.2).
Initialization: The initial population is created by generating multiple permutations of the project list. Each permutation represents a potential schedule. For instance, one individual might be $[1, 2, 3, 4, 5, 6, 7, 8]$, while another might be $[2, 1, 4, 3, 5, 7, 6, 8]$.
Crossover: For crossover, the first half of one parent can be combined with the remaining projects of the other. Applying the crossover operator to the individuals from the previous step results in the children: $[1, 2, 3, 4, 5, 7, 6, 8]$ and $[2, 1, 4, 3, 5, 6, 7, 8]$. See Figure 2.2 for more detail.
Mutation: Two randomly chosen subsequent projects can be swapped to introduce diversity in the population. For example, assume the randomly chosen positions are 4 and 5. Applying this mutation operator to the individual $[1, 2, 3, 4, 5, 6, 7, 8]$, the resulting mutated individual would be $[1, 2, 3, 5, 4, 6, 7, 8]$.
Evaluation and selection: A key metric for evaluating the schedules is their total duration, which should be minimized. For example, if schedule $[1, 2, 3, 4, 5, 6, 7, 8]$ has a duration of 14 time units, it would be selected over all other schedules with a longer total duration.

2.3.6 Hyperparameters

Hyperparameters in GAs are settings that influence the algorithm's performance and behavior. These parameters include population size, generation number, mutation rate, and crossover rate.

Each hyperparameter differently influences the algorithm's behavior. For example, the population size determines the number of individuals in each generation. Larger populations increase genetic diversity, which helps explore the solution space more thoroughly and reduces the risk of premature convergence. However, larger populations also require more computational resources. Therefore, proper tuning of the hyperparameters is essential for the effective performance of a GA.

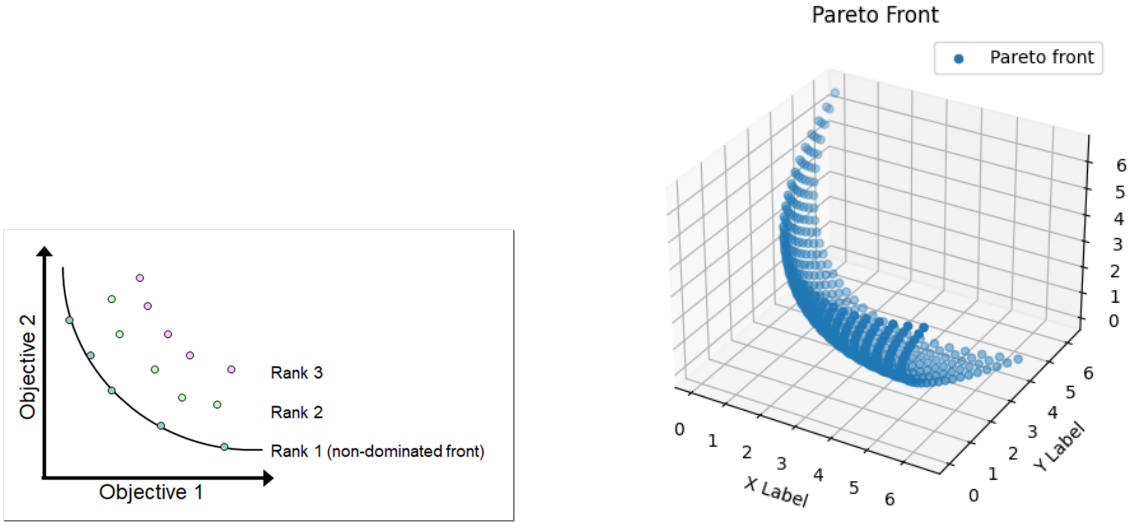
2.4 Pareto optimization

Pareto optimization provides a powerful framework for addressing problems involving multiple conflicting objectives, where improving one objective often worsens the other objectives. The fundamental concept behind Pareto optimization is Pareto efficiency, a situation in which no further improvements to the objectives can be made without sacrificing performance in another objective. This means that a solution is considered Pareto optimal if

it represents the best possible compromise between objectives, with no feasible alternative offering improvement in one of the other objectives.

Central to the concept of Pareto optimization is the notion of dominance. In multi-objective optimization, a solution is said to dominate another if it performs at least as well as the other solution in all objectives and strictly better in at least one objective. Conversely, a solution is considered non-dominated if no other solution in the feasible space dominates it.

The set of all Pareto optimal solutions (non-dominated solutions) is called the Pareto front. Visualized in the objective space, the Pareto front delineates the trade-off between the different objectives, offering decision-makers valuable insights into the problem and optimization process. Typically, in two-objective optimization, the Pareto front is a curve, while in higher dimensions, it forms a surface (see Figure 2.3).



(a) Two objectives [14]

(b) Three objectives [58]

Figure 2.3: Example visualizations of a Pareto front with multiple objectives

Understanding dominance and non-dominated solutions is essential in Pareto optimization as it allows for the identification of solutions that represent the best possible compromises among competing objectives. By considering non-dominated solutions, decision-makers can explore the trade-offs inherent in the problem and make informed decisions that balance conflicting objectives effectively.

To illustrate the notion of Pareto optimization and the principle of dominance, consider the portfolio example introduced in previous sections. The objectives are to minimize the total duration and to minimize the variance in resource allocation. The following schedules are considered:

- Schedule A:
  - Duration: 20
  - Resource variance: 3
- Schedule B:
  - Duration: 18
  - Resource variance: 5

- Schedule C:
  - Duration: 20
  - Resource variance: 6

To begin with, Schedules A and B do not dominate one another, as each has one superior objective. Schedule A dominates Schedule C, as it matches C in duration but is better in resource variance. Schedule B also dominates Schedule C, as it is better in both objectives. Consequently, Schedules A and B are on the Pareto front, since they are non-dominated solutions.

2.5 Nondominated sorting genetic algorithm II

A popular algorithm to solve multi-objective problems is the Nondominated Sorting Genetic Algorithm II (NSGA-II) [15]. NSGA-II provides an efficient way to maintain a diverse population while still converging to optimal solutions. This is achieved through the process of non-dominated sorting, which categorizes solutions into different fronts based on their dominance relations with the other solutions. Solutions in the first front are non-dominated solutions (the Pareto front). The subsequent fronts contain solutions that are dominated by the earlier fronts.

To illustrate this, consider a scenario with two conflicting objectives: minimizing cost and maximizing quality. Consider the solutions: A, B, C, D. If A is the cheapest solution and of higher quality than B and C, it is non-dominated and placed in the first front. B and C, which are dominated by A but not by each other, would be placed in the second front. If D is of higher quality and cost than A, it is not dominating nor dominated by A, resulting in a place in the Pareto front.

In addition to non-dominated sorting, NSGA-II enhances diversity within the population by employing *crowding distance*. Crowding distance is a measure of the density of solutions in the objective space. The algorithm encourages the preservation of solutions in a less densely populated region.

Continuing the cost and quality example, suppose E, F and G are in the same front. If solutions E and F are very close in terms of cost and quality, while solution G is further apart, the crowding distance for G would be larger. NSGA-II would prefer solution H over F and G to maintain diversity in the population.

The main procedure of NSGA-II is visualized in Figure 2.4 and starts with an initial population. The population is sorted based on the nondomination. Each solution is assigned a fitness equal to its nondomination level. Then tournament selection, recombination and mutation are used to generate children. The algorithm utilizes elitism, meaning that it compares the previously found best solutions with the current children. This ensures that the best solutions are not lost during the evolution process and continue to contribute to the next generations. The children and old population are combined and non-dominated sorting is applied. The resulting first fronts are incorporated into the new generation until reaching a point where the addition of the subsequent front would exceed the predefined population size. Subsequently, this succeeding front undergoes sorting based on crowding distance, after which its individuals are incrementally included until the population size criterion is met.

To illustrate, consider the initial population $H_1, I_1, J_2, K_3, L_3, M_3$. The subscript indicates the front to which each individual belongs after non-dominated sorting. Using tournament selection, pairs like (H_1, L_3) and (M_3, K_3) are chosen at random. First, the individual H_1 is chosen over L_3 since it belongs to a higher-ranked front. Second, M_3 and

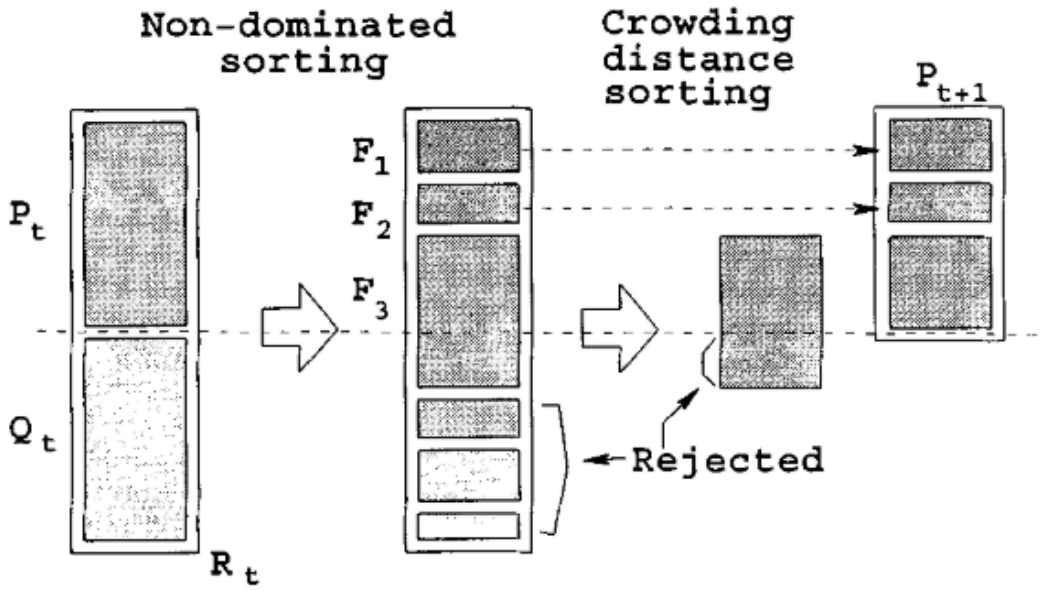


Figure 2.4: Main procedure of NSGA-II [15]

K_3 are in the same front, therefore, the individual with the largest crowding distance will be chosen. For the example, let us assume that K_3 has a larger crowding distance, meaning that K_3 and H_1 proceed for recombination, creating new children. This recombination and mutation process repeats until the desired number of new individuals is generated.

The combined population now consists of the initial population and the new children: $H_1, I_1, J_2, K_3, L_3, M_3, N, O, P, Q, R, S$. Next, the algorithm performs non-dominated sorting on this combined population. Assume the new sorting results in some children moving to higher fronts due to better performance. For instance, if R and S are now in the first front, the updated fronts might be: $H_1, I_1, R_1, S_1, J_2, P_2, Q_2, K_3, L_3, M_3, N_3, O_3$. Suppose we need to select 6 individuals for the next generation. We start by selecting all individuals in the highest fronts until we reach or exceed the population limit. Adding the individuals from the first front H_1, I_1, R_1, S_1, provides four individuals. Moving to the second front J_2, P_2, Q_2 and adding them would exceed the population limit. Therefore, crowding distance is used to select the most diverse individuals from the second front. Suppose J_2 and P_2 have larger crowding distances than Q_2, so they are selected. This forms the new population $H_1, I_1, R_1, S_1, J_2, P_2$ for the next iterations.

2.6 Activity list

Every GA approach needs a representation of an individual, to be able to apply crossover and mutation operators. For RCPSP-like problems, the most common approach is a precedence *Activity List (AL)* [35] or a derived version of it. The AL representation was determined to perform the best to solve RCPSP [25]. In this representation, the solution is encoded as a precedence feasible list of the activities. In this list, each activity can appear in any position after its predecessors. A schedule is constructed by scheduling the activities, one by one, in the order of the given list, so when an activity is scheduled, it is automatically scheduled after its predecessor (forward scheduling). To visualise this approach, Alcaraz and Maroto [2] used the following example, depicted in Figures 2.1 and 2.5.

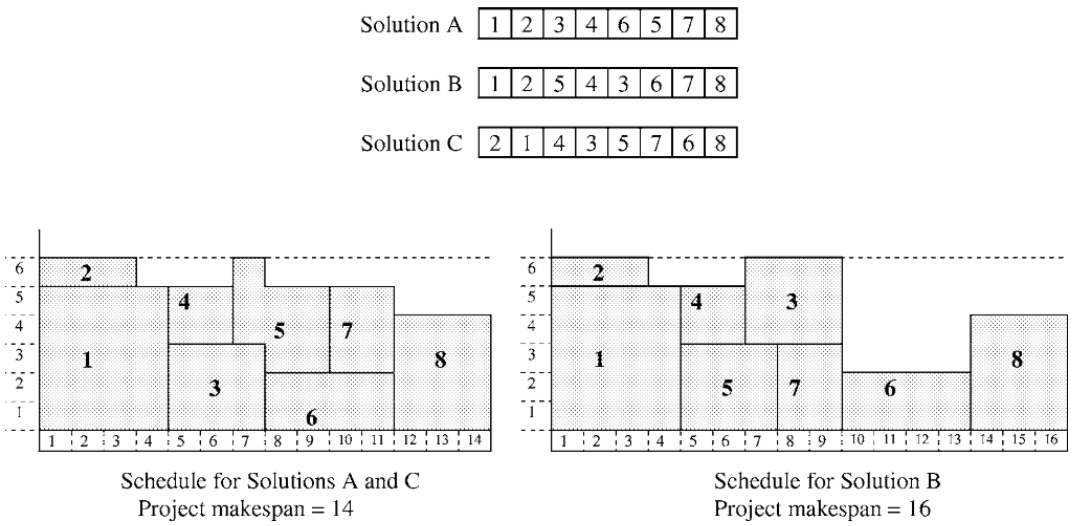


Figure 2.5: Individuals and related schedules for the example project visualised by Alcaraz and Maroto [2]

Figure 2.1 shows an example project, in which the cells are activities and the arrows show the precedence. Figure 2.5 first shows three possible solutions in their AL representation and underneath shows the schedules associated with the solutions.

2.7 Constraint handling

Handling constraints in GAs is crucial to ensure that solutions meet the required criteria. Several techniques have been developed to handle constrained optimization problems effectively. Petridis et al. [47] introduced multiple methods for constraint handling. One of these methods is restricting the search space to only contain feasible solutions, similar to how a AL representation always adheres to precedence constraints. Other methods are removing infeasible solutions from the population, adding a penalty to the fitness function and repeating infeasible solutions.

For multi-objective evolutionary algorithms (MOEAs), [48] categorizes constraint handling methods into the following four groups:

1. Penalty function
   A penalty is provided to the objectives whenever a constraint is not met.

2. Separation of constraints and objectives
   A separate value for constraint violations is used in the selection process.

3. Hybrid method
   A combined technique of the other methods.

4. Retaining the infeasible solutions in the population
   Adding the constraints violation value as an extra objective to optimize on

The next subsections describe the constraint handling methods used in this research. The methods are based on the techniques described earlier and techniques described in [33] as they were shown to be consistent and effective.

2.7.1 Dynamic penalty

The dynamic penalty method involves penalizing solutions that violate constraints in the objective functions, thus guiding the search toward feasible regions of the solution space. The penalties are dynamically adjusted based on the degree of constraint violation and the generation. Larger generation numbers lead to a larger penalty. This allows more exploration in the early generations since the penalty is still small, but convergence towards feasible regions in later generations.

One of the main advantages of this method is the great balance between exploration and exploitation by the dynamically adjusting penalties. However, finding the right penalty function to achieve this balance can be challenging. The penalty function must permit marginally infeasible solutions, while effectively discouraging entirely infeasible solutions.

2.7.2 Constrained dominance principle

The constrained dominance principle extends the concept of dominance in traditional NSGA-II to account for constraints. A solution dominates another solution if it not only exhibits better objective values but also satisfies constraints more effectively. By prioritizing solutions that adhere to constraints, this principle ensures that feasible solutions are favored during the selection process, ultimately leading to the generation of more practical and viable schedules.

This technique is simple to implement in existing NSGA-II implementations. Due to its simplicity, it lacks exploration in its early stages, since information within infeasible solutions is not regarded.

2.7.3 Violation as extra objective

Incorporating violation as an additional objective entails treating constraint violations as explicit objectives to be minimized alongside primary objectives. By formulating violation metrics as auxiliary objectives, the algorithm seeks to simultaneously optimize both primary objectives and constraint satisfaction. This approach facilitates the exploration of the trade-offs between conflicting objectives and constraint adherence, resulting in a diverse set of solutions that strike an optimal balance between performance and feasibility.

Retaining information from infeasible solutions throughout the evolutionary process brings both benefits and drawbacks. One advantage is the increased exploration by preserving information. However, searching infeasible search space can lead to redundant exploration, diverting resources that could have been used more effectively in searching feasible space.

2.8 Hyperparameter optimization

Hyperparameters, such as population size, mutation rate, and crossover probability, govern the behavior and performance of GAs. Optimizing these parameters is crucial as they each influence the algorithm's convergence speed, solution quality, and precision differently. In particular, selecting appropriate hyperparameters ensures efficient exploration and exploitation of the solution space, leading to improved outcomes in optimization tasks.

Hyperparameter Optimization (HPO) for GAs can be challenging due to the stochastic nature of GAs. Guzman et al. [24] propose a *heteroscedastic* Bayesian optimization for HPO in a stochastic system. Heteroscedasticity is the behaviour where the variance

between points varies in different regions. Their research states that their suggested framework for tuning stochastic models outperformed homoscedastic Bayesian optimization. A Particle Swarm Optimization (PSO) approach was taken by Altarabichi et al. [4]. In their research, HPO was used to determine how much randomness was optimal in a Deep Neural Network. Other frequently used methods for HPO are Grid Search, Random Search, and GAs.

2.8.1 Bayesian optimization

Bayesian Optimization (BO) is a powerful technique for optimizing black-box functions that are expensive to evaluate and may contain noise and randomness. In the context of HPO for GAs, BO provides a framework to iteratively build a probabilistic model of the objective function based on some initially observed evaluations. With this model, BO selects hyperparameter configurations to explore, balancing exploration of promising regions with exploitation of known high-performing areas. The probabilistic model is updated and new configurations are explored. This enables efficient convergence to optimal or near-optimal hyperparameter settings, even in the presence of noise and limited evaluation budgets.

2.8.2 Particle swarm optimization

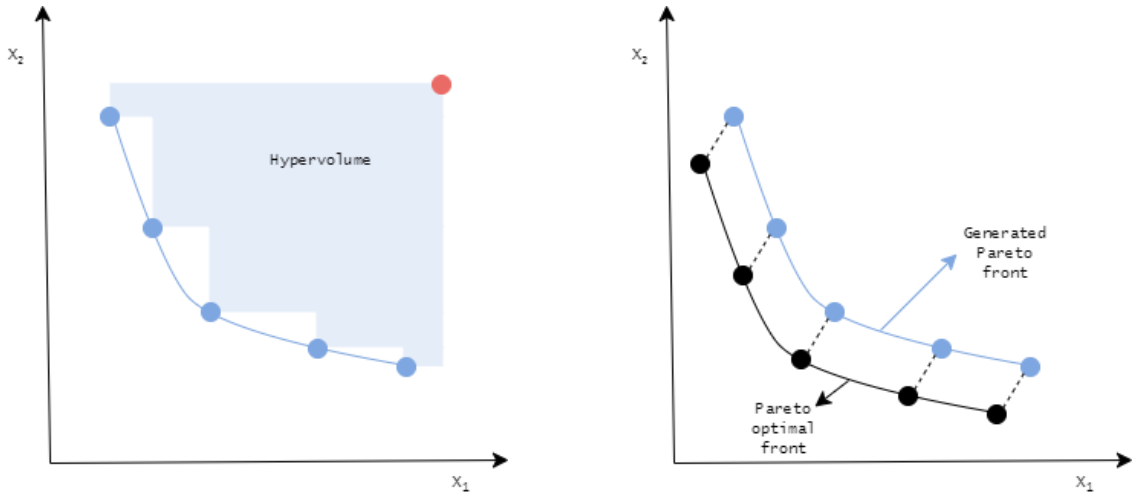
PSO randomly initializes a swarm of particles, where each particle has a position in a d dimensional space for d hyperparameters. The swarm searches the space through movements guided by the current, globally best particle. PSO is a population-based stochastic optimization algorithm inspired by the social behaviour of birds flocking. In PSO, a swarm of particles moves through a d dimensional search space for d hyperparameters, where each particle represents a potential solution characterized by a position and velocity. The particles adjust their positions based on their own experience and the best-performing particle in the swarm. By iteratively updating the positions of particles based on their performance and that of the swarm, PSO can efficiently explore the hyperparameter space and converge to promising regions by gradually lowering the velocity per generation.

2.9 Assessment of multi-objective optimizers

When comparing Pareto fronts, determining which one is superior is not always straightforward. Two primary metrics measure the performance of a Pareto front. The first is a *Hypervolume (HV) indicator*, which quantifies the volume of the objective space enclosed by the front concerning a reference point (see Figure 2.6a). This indicator provides a complete measure of the solution space covered by the front, capturing both the spread and convergence of solutions.

To elaborate on the HV indicator, consider a reference point established in the multi-dimensional objective space. Then, for each point in the Pareto front, the volume from that point to the reference point is computed. The union of these volumes represents the HV, offering a comprehensive view of the effectiveness of the Pareto front in encompassing the desired objective space.

Another metric utilized for comparison is the Inverted Generational Distance (IGD) (see Figure 2.6b). Unlike HV, which focuses on coverage within the Pareto front, the IGD assesses the convergence of the front towards known optimal solutions. It quantifies the average distance from points on the Pareto front to the nearest optimal solution. A lower IGD value indicates better convergence towards the true Pareto front, aiding in the evaluation of the front's effectiveness in exploring the solution space.



(a) HV (b) IGD

Figure 2.6: Example visualizations of Pareto front assessment techniques

Chapter 3

Related work

PPM is a critical aspect of strategic decision-making in organizations, making it a well-researched topic. PPM includes project selection, scheduling and resource allocation to achieve optimal performance under various constraints and uncertainties. This chapter provides insight into related research within these topics, highlighting contributions and techniques. First, an overview of related works on portfolio selection and scheduling automation is presented. Subsequently, research on project scheduling is discussed. Although project scheduling techniques can be applied to portfolio scheduling due to their analogous nature (as explained in Section 2.1, a clear distinction is made between the related works on project portfolio selection and scheduling and project scheduling to highlight the unique aspects and contributions of each field.

3.1 Project portfolio selection and scheduling

Project Portfolio Selection and Scheduling (PPSS) is the activity of defining the best set of projects to pursue the strategic objectives and finding the optimal time to start the projects. To begin with, focusing solely on project selection, [56] provides a literature overview of techniques applied for project selection considering project interdependencies. They categorised the found approaches into five categories: multi-criteria, linear programming, non-linear programming, metaheuristics and other approaches. Relevant to this research are the multicriteria and metaheuristic approaches, as this study integrates aspects of both. Wu et al. [61] integrated a fuzzy Analytic Hierarchy Process (AHP) technique and the NSGA-II to create optimal project selection in portfolios with uncertainty, interdependencies and multiple strategic goals. Similarly, Gomede and de Barros [22] used NSGA-II for optimization and AHP for post-optimization for project selection with multi-criteria. Abassi et al. [1] also implemented the NSGA-II to maximize total outcome, minimize total risk, and maximize strategic advantages. Moreover, they compared it to a multi-objective PSO and found that NSGA-II outperformed the former. To continue on GA approaches, Yu et al. [62] modelled multi-criteria project selection as a 0-1 nonlinear integer programming model and used a GA-based optimization technique for solving. Recently, Hemici and Zouache [32] proposed a new MOEA for portfolio selection. Their approach was based on multi-population to explore the search space greatly and find an optimal trade-off between risk and return. Similarly, Zhou et al. [64] introduced a new MOEA where optimization is conducted in distinct regions of the objective space to mitigate premature convergence. By dividing the objective space into multiple subregions and independently optimizing each, they aimed to maintain population diversity and prevent premature convergence in the evolutionary process. Additionally, they addressed the issue of constraints leading to

infeasible solutions by implementing a greedy repair strategy.

Another metaheuristic approach was taken by Carazo et al. [11], using a Scatter Search procedure for multi-objective PPSS. In their model, they assumed strong interdependence between projects and considered multiple goals and constraints. A Strength Pareto Evolutionary Algorithm (SPEA) was used by [16] for finance-based project scheduling. SPEA is an algorithm that evolves a set of solutions to find the best trade-offs between multiple conflicting objectives by prioritizing non-dominated solutions and maintaining an archive of the best-found solutions to guide the search process. Another type of evolutionary algorithm was used by [34]. They developed a differential evolution metaheuristic algorithm to solve multi-skilled scheduling problems. The multi-skilled scheduling problem was also handled by [12], focusing on IT project portfolios. They proposed a Pareto ant colony optimization algorithm and found it more efficient than NSGA-II. Zhang et al. [63] incorporated fuzzy numbers into the model for the PPSS problem to account for the uncertainty. A MOEA using a Gaussian Process was used to solve the problem. Ghodioosi et al. [21] handled uncertainty in portfolio scheduling using three steps. They used a neural network to estimate missing or uncertain parameter values (parameters such as project duration, cost, demands, etc), a shuffle frog leap algorithm (a combination between a GA and PSO) for optimization and K-means algorithm to cluster the candidate projects into a portfolio.

To conclude, various approaches have been taken to address the challenges of PPM. The most popular methods include multi-criteria decision-making frameworks, such as the fuzzy AHP, and evolutionary algorithms like NSGA-II and GAs. Moreover, advanced metaheuristic techniques such as Scatter Search, SPEA, and differential evolution have been widely utilized.

3.2 Project scheduling

Section 3.1 provided examples of literature in which the PPSS problem is addressed. This section will examine the RCPSP variants, highlighting the evolutionary algorithms employed to solve these challenges.

One of the first explorations of GA for solving project scheduling problems was done by Hartmann [25]. He showed how a permutation-based GA can be used for the RCPSP and that it outperformed other GAs and other approaches at the time. The permutation-based approach was compared to a priority value-based and a priority rule-based approach. This is in line with the state-of-the-art research on heuristics for RCPSP [31]. However, they also mention the successful implementation of *Simulated Annealing (SA)* by Bouleimen and Lecocq [10]. SA is a probabilistic optimization technique that mimics the annealing process by iteratively accepting new solutions while the temperature is decreasing, allowing worse solutions to be accepted at high temperatures but not at low temperatures. The latter approach was for the MRCPSP. For this problem, also multiple approaches have been taken. To begin with, Özdamar [46] proposes a *Hybrid Genetic Algorithm (HGA)* based on a priority rule encoding. HGA implies that some form of additional optimization is added to the GA to make it more efficient. The HGA proposed by Özdamar incorporates a certain scheduling expertise into the GA. This improved the outcome but made the calculation process slower. Another HGA for solving MRCPSP was proposed by Hartmann [26], in which local search methods were introduced to improve the schedule related to an individual. Alcaraz [3] proposes a pure GA that produces similar results to previous heuristics but does slightly worse compared to the GA with local search methods of Hartmann. In an updated research on the state of the art on RCPSP by Hartmann and Kolisch [36], they mention multiple different approaches that surpassed the former benchmark approaches

that were mentioned. Valls [54] proposed an HGA that introduced a local improvement operator, a new way to combine parents and a two-phase strategy. Hartmann [27] proposes a heuristic called self-adapting GA. This heuristic contains an additional gene to represent one of two decoding procedures for computing a schedule for an individual. This way, the algorithm learns which of the two procedures works best, therefore also optimizing the algorithm itself. Again, the state-of-the-art research shows that the most popular strategies use GAs.

MRCPSP is a more difficult version of RCPSP, where activities can be performed in multiple modes, meaning the activities can be executed using different resources or methods. Lova [40] proposed a new heuristic approach to solve the problem. It is a HGA that uses a local search method to improve the solutions provided by the GA. To improve the project schedules, i.e., reduce the project completion time, a multimode backwards-forward or forward-backwards method was applied, depending on the additional gene. This approach resulted in drastic reductions in project duration time.

Continuing with MRCPSP, Ghoddousi [20] combined this problem with the discrete time-cost trade-off problem and the resource allocation and resource levelling problem. To solve this, they proposed a multi-objective-based GA that selects the best combination of starting time and execution mode to optimize the time and cost.

Zoraghi [66] extends the MRCPSP with material ordering. In their research, they compare three HGAs, from which the PSO-GA performs best. This algorithm uses particle swarm optimisation to optimise the solution provided by the GA.

Building on RCPSP, Wauters [60] introduced MRCMPSP which has high practical value but is also more complex and difficult to solve. In their research, various methods were proposed with four approaches that stood out. Asta et al. [7] applied a combination of Monte-Carlo Tree Search and hyper-heuristics, Geiger [19] used an iterated variable neighbourhood search, Toffolo [53] applied integer programming and Artigues applied mixed integer programming and large neighbourhood search.

Kuhn [39] applied GA with a simulation-based optimisation tool on MRCMPSP and investigated the influence of parameters of the algorithm to figure out which affected the result the most. Their results show that generation size and population size have the most impact on the outcome.

In summary, evolutionary algorithms, particularly GAs, have emerged as pivotal tools in addressing various challenges within project scheduling. Starting with Hartmann's [25] pioneering work on permutation-based GAs for RCPSP, subsequent research has consistently integrated hybrid models and advanced optimization techniques to enhance efficiency and solution quality. Similarly, for multi-mode and multi-objective variants of the problem, HGA implementations have been successful.

3.3 Bridging literature and methodology

The insights from the preceding sections together form the methodology in this thesis. The discussion on PPSS shows that multi-objective optimization is crucial in effective PPM automation. Balancing multiple objectives such as cost, risk, and strategic alignment is one of the most important tasks in PPM. The NSGA-II is applied in many methodologies as it is particularly effective in addressing such problems. However, a notable gap in current research is the handling of various constraints such as dynamic capacities. Therefore, existing approaches limit their applicability in real-world scenarios. Some of these additional constraints were handled by methods described in Section 3.2. This section provides an overview of techniques for solving RCPSP-like problems. These problems integrated

constraints missing in PPSS literature.

This thesis develops a comprehensive methodology for optimizing PPM by synthesising the knowledge from both sections. The integration of multi-objective optimization principles and advanced GA techniques forms the core of this approach. Specifically, the research extends the RCPSP framework to address gaps in PPM literature, making the model more realistic and applicable to real-world scenarios.

Chapter 4

Formal problem definition

In practical applications, certain tasks need to be completed within specific time frames to accommodate contractual agreements or regulations, necessitating precise start or end times. Additionally, employing a multi-objective approach allows decision-makers to explore trade-offs between conflicting objectives, such as duration and costs, enabling them to make informed decisions that balance these objectives. Furthermore, varying resource capacities and requests over time provide a more realistic representation of scheduling scenarios. For example, in a software project, developers are needed at the start to write code. At the same time, testers are required towards the end to perform testing, leading to time-varying resource demands throughout the project.

To address these challenges, this research introduces a new variant of the RCPSP: the multi-objective RCPSP/t with additional start/end time constraints (MORCPSP/t-SE). This variant extends the traditional RCPSP by incorporating multiple objectives to optimize, time-dependent resource capacities and requests, and additional start/end time constraints. Tasks are thus not only subject to resource limitations but can also be constrained by specific start or end times, adding an extra layer of complexity to the scheduling problem. By addressing these additional constraints, MORCPSP/t-SE aims to provide a more realistic representation of scheduling scenarios encountered in real-world applications.

Formalisation of the MORCPSP/t-SE

The MORCPSP/t-SE can be defined as follows. There are $J-2$ activities to be scheduled and activities 1 and J are virtual activities corresponding to the start and end, respectively. The activities are constrained by precedence, resource or set start/end time constraints.

- **Precedence constraints**: Precedence constraints force activity j not to be started before all its immediate predecessors have finished.
- **Resource constraints**: Each activity requires varying resources for each of the timeslots in its duration. These required resources are drained from capacities that vary over time and are shared by all activities.
- **Start/end time constraints**: Each activity can have a set start or end time constraint.
  - Set start time: The activity must start in a specific time slot.
  - Set end Time: The activity must finish before a specified time slot.

Consider a project that consists of the activities (jobs) labelled ($j = 1, 2, ..., J$). The precedence relations are given by the set P_j, indicating that activity j can not be started until the activities in P_j have finished.

Each activity has a duration of d_j time units. In each of the time units, an activity j requires l_{jrt} units of resource r. In timeslot t resource r has the limited capacity L_{rt} and the sum of resource requests of activities scheduled in t can not exceed L_{rt}. Moreover, it is assumed that preemption of activities is not allowed, meaning that once an activity starts, it must be finished and cannot be paused. This decision was made because FCC also does not allow preemption.

For the additional set start/end time constraints, two sets S and E are defined for the start and end time constraints, respectively. The sets contain the tuples (j, t), implying that activity j should start or end at timeslot t. S is a constraint set if:

$$S \subseteq \{(j, t) \mid j \in J, t \in T\} \tag{4.1}$$

and

$$\forall j.\ \exists! t.\ (j, t) \in S \tag{4.2}$$

This means that the S only contains existing activities and each activity can appear once in the set. Next to that, the binary variables indicating whether an activity starts or ends in the specified time slot are defined as follows:

$$s_{jt} = \begin{cases} 1, & \text{if } (j, t) \in S \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

To ensure compliance with all constraints defined in the set S:

$$s_{jt} = 1, \forall (j, t) \in S \tag{4.4}$$

Due to the nature of the problem where preemption is not allowed, the end times variables e_{jt} can be derived from the start time variables s_{jt}.

$$\forall j.\ e_{j,t+d_j} = s_{jt} \tag{4.5}$$

To ensure compliance with all constraints defined in the set E:

$$\forall (j, t) \in E. \sum_{t'=t}^{T} e_{jt'} = 0 \tag{4.6}$$

Multi-objective implies that multiple objectives are considered while optimizing. It can be formulated as follows:

$$\textit{Minimize} \quad f(X) = \{f_1(X), f_2(X), \ldots, f_m(X)\} \tag{4.7}$$

$$\text{Subject to:} \quad X \in D \quad \text{where} \quad D \quad \text{represents all feasible solutions.} \tag{4.8}$$

where:

- $f(X)$ is a vector of the objective functions.
- $f_n(X)$ is the n-th objective function to be minimized.

- There are m objectives.
- X is a feasible schedule that adheres to the constraints listed above.
- D is the set of all feasible solutions, encompassing all constraints.

Initially, minimizing makespan and resource utilization smoothness will be considered as objectives. Makespan is the eventual duration of the schedule and resource utilization smoothness is the average deviation from the average resource usage. Eventually, other objectives like costs or expected business value can also be considered.

Chapter 5

Methodology

5.1 Genetic algorithm approach

To find solutions for the MORCPSP/t-SE, introduced in Chapter 4, the NSGA-II [15] will be used. As explained in Section 2.5, NSGA-II is an efficient evolutionary algorithm based on Pareto fronts and domination. It uses elitism and crowding distance operators to keep its best solutions and diversity in the population. It was chosen for its proven ability to effectively balance multiple conflicting objectives, its robustness in handling a wide range of scheduling constraints, and its demonstrated performance in delivering high-quality solutions across various domains [55]. Additionally, its solid theoretical foundation in the literature regarding scheduling problems provides confidence in its applicability [20][59][44].

5.1.1 Non-dominated sorting

Non-dominated sorting serves as the foundational mechanism within NSGA-II. It entails assigning a rank to each individual in the population based on its non-domination level. The sorting algorithm operates as follows: initially, two characteristics are computed for each individual, the domination count and the set of dominated solutions. The domination count denotes the number of solutions that dominate the individual, the set of dominated solutions contains all solutions that the individual dominates. Individuals with a domination count of zero are considered non-dominated and form the first front, denoted as F_1. To determine subsequent fronts, the domination count of individuals in the dominated set of each member in F_1 is decremented. Those whose domination count reaches zero are added to F_2, and this process iterates for subsequent fronts until all individuals have been sorted accordingly. An overview of this process can be seen in algorithm 1. This algorithm differs slightly from the proposed algorithm by [15], as duplicate dominance checks are prevented. This is achieved by recognizing that the second loop (line 4) does not need to loop over all individuals in the population. Specifically, once an individual i is compared to individual j, there is no need to compare j to i again, as the dominance relationship is symmetric and has already been established.

To illustrate the steps of non-dominated sorting, consider a simplified example with a population of four schedules, each evaluated based on two objectives: total duration and resource variance. The schedules for this example are detailed in Table 5.1.

First, the dominance relations among the schedules are calculated, as summarized in Table 5.2. Next, the Pareto front is constructed, which consists of all non-dominated solutions. In this example, Schedules B and C together form F_1.

The second front, F_2, includes solutions that are dominated only by individuals from

F_1. Here, Schedule A is included in F_2 because it is dominated solely by Schedule B from F_1.

Finally, the third front, F_3, comprises solutions that are dominated by individuals from both F_1 and F_2. In this case, Schedule D is in F_3, as it is dominated by Schedules A and B. An overview of the fronts is given in Table 5.3

Schedule	Duration	Resource Variance
A	20	3
B	18	3
C	22	2
D	21	6

Table 5.1: Schedules with their respective durations and resource variances

Comparison	Dominates	Reason
A vs. B	B dominates A	B has a shorter duration
A vs. C	Neither	No dominance
A vs. D	A dominates D	A has shorter duration and lower variance
B vs. C	Neither	No dominance
B vs. D	B dominates D	B has shorter duration and lower variance
C vs. D	Neither	No dominance

Table 5.2: Dominance calculation between schedules

Front	Schedules	Dominance relation
F_1	B, C	Non-dominated
F_2	A	Dominated by B
F_3	D	Dominated by B and A

Table 5.3: Non-dominated fronts

5.1.2 Crowding distance

The crowding distance serves as a crucial metric for assessing solution density within a Pareto front relative to a specific solution. It quantifies how tightly packed solutions are around a given solution within the front. The crowding distance assigned to an individual is computed as the sum of its crowding distances across all objectives. Mathematically, this can be expressed as:

$$CD_i = \sum_{o=1}^{O} cd_{i,o} \tag{5.1}$$

Here, i denotes the individual, O represents the total number of objectives, and cd_{io} signifies the crowding distance calculation for a particular objective o.

The computation of crowding distance for each objective involves several steps. Initially, solutions are sorted based on their objective function values. Boundary solutions, i.e. those with the smallest and largest values, are assigned an infinite distance. This ensures the preservation of diversity within the population and encourages exploration across all

Algorithm 1: Fast Non-dominated Sort

Input : Population
Output: Fronts with non-dominated individuals

```
fronts ← [[]]
i ← 0
foreach individual A in population do
    foreach other individual B in population[i:] do
        Compare A and B to determine dominance
        Update dominance counts and lists of A and B
    end
    i ← i + 1
    If A has no dominators, assign rank 0 and add to fronts[0]
end
i ← 0
while size of fronts[i] > 0 do
    temp ← []
    foreach individual A in fronts[i] do
        foreach other individual B in dominated set of A do
            decrement domination count of B
            If B domination count is 0, assign rank i+ 1 and add to temp
        end
    end
    i ← i + 1
    Add temp to fronts
end
return fronts
```

objective dimensions. Furthermore, the boundary solutions' values establish a scale for normalizing crowding distances across multiple objectives.

For non-boundary solutions within a front, the crowding distance is determined by calculating the difference between the objective function values of adjacent solutions in the sorted front and dividing it by the established scale. This process can be represented by the formula:

$$cd_{i,o} = \frac{|f_o(x_{i+1}) - f_o(x_{i-1})|}{f_o^{max} - f_o^{min}} \tag{5.2}$$

Here, $f_o(x_{i+1})$ and $f_o(x_{i-1})$ are the objective function values of the adjacent solutions to i and f_o^{max} and f_o^{min} represent the maximum and minimum values of the objective functions across the entire front.

5.1.3 Individual representation

For the GA, the individual is represented as an Activity List (AL). An AL [35] is a precedence feasible permutation of the activities, implying that for each activity their preceding activities are positioned somewhere earlier in the list. To formalize, given the AL $\lambda = (j_1, j_2, j_3, ..., j_J)$, if $i = j_h$, it implies that in position $p(i) = h$. A schedule $S = (s_1, s_2, ..., s_J)$, where s_j is the start time for activity j, can be generated using a *Serial Generation Scheme (SGS)*, see Section 5.1.3.

Schedule generation

Using SGS, a schedule S(λ) is created from AL λ by taking the activities one by one, in the order of the list, and scheduling them at the earliest time, still adhering to the precedence and resource constraints. The pseudocode in 2 shows how a schedule can be generated from an AL. The algorithm loops over all activities in the AL and tries to schedule it as early as possible. First, it finds the start time, which is the latest end time of the predecessors. Starting at the start time, it checks if the resource constraints are not violated for each timestep in the duration of the activity. If the constraints are not violated, the activity is scheduled, otherwise, the start time is incremented and the process repeats. This ensures that the earliest possible start time for each activity is found, considering the AL.

5.1.4 Fitness functions

The MORCPSP/t-SE is a multi-objective problem, therefore, the fitness function is comprised of multiple functions. To begin with, one of the objectives is makespan. The makespan of an individual is determined by the sum of the duration of the generated schedule and the constraint penalty. An individual receives a penalty when the set start or end time constraint is violated. The penalty is determined by how many timeslots the activity is misplaced, further away leads to a higher penalty.

The second objective that will initially be considered is the resource utilization smoothness score. This score will be calculated by taking the standard deviation of the resource utilization per timeslot for each different resource and taking the average of the calculated standard deviations.

5.1.5 Initial population

The initial population is created by repeating the following steps, starting with an empty AL. Through an iterative process, activities are randomly chosen from the remaining set

Algorithm 2: Schedule Activities

Input : Activity list
Output: Scheduled activities

```
1  schedule ← empty list
2  foreach activity A in the activity list do
3      starttime ← max(predecessor activities endtime)
4      timeslot ← starttime
5      while timeslot − starttime < duration do
6          if demand > resources in timeslot then
7              timeslot ← starttime
8              starttime ← starttime + 1
9          end
10         timeslot ← timeslot + 1
11     end
12     schedule ← schedule + starttime
13 end
14 return schedule
```

and added to the sequence. Before integration, the algorithm verifies that the selected activities can be feasibly scheduled next by ensuring they adhere to precedence constraints. This is done by checking if all preceding activities of the randomly chosen activity are already selected, ensuring they are listed before. This iterative refinement continues until a valid AL is formed. This is repeated until the satisfied population size is reached, fostering diversity within the initial population and reducing the search space by eliminating invalid schedules.

5.1.6 Crossover

For crossover, *n-point crossover* will be applied. N-point crossover is a technique in which n crossover points are chosen randomly along the length of the parent chromosomes. For this, two individuals will be selected as parents, a mother M and a father F. Then n random integers $q_1, q_2, ..., q_n$ with $1 \leq q_1 \leq q_2 \leq ... \leq q_n \leq J$ are taken. Two new individuals, daughter D and son S, are produced from the parents. First, D is created by taking all activities from M with position $i = 1, ..., q_1$ and filling the positions $i = q_1, ..., q_2$ with the activities in F. When adding the activities from F, duplicate activities are not added. This process is repeated for all q, ending by filling positions $i = q_n, ..., J$ with activities in F. This method ensures that every activity appears in the list exactly once and precedence constraints are met (proven by Hartmann [25]). The son is created by taking the first activities from F instead of M and performing the same steps.

5.1.7 Mutation

Two mutation operators were initially considered. The first mutation operator, proposed by Hartmann [25], involves exchanging activities j_i and j_{i+1} for an individual λ at each position $i = 1, ..., J-1$, with the probability of $p_{mutation}$, provided that the new AL satisfies the precedence constraints. The second operator is a right-shift mutation, where for each position $i = 1, ..., J-1$, the activity j_i is moved randomly to a position between i and J with the probability of $p_{mutation}$ also if the new AL satisfies the precedence constraints.

Ultimately, only the right-shift mutation operator will be applied. It introduces more significant changes to the schedule, thus adding more diversity to the population. In contrast, swapping the positions of consecutive activities often resulted in minimal changes to the individual's *phenotype*, that is to say, visible changes in the generated schedule.

5.1.8 Selection

To create a new generation, first, 2-tournament selection will be used to find two parents. In this technique, two different randomly chosen individuals compete for reproduction. The individual with the lower rank is selected as the winner. If there is a tie in rank, the individual with the higher crowding distance is chosen. The two chosen parents produce two children with crossover and mutation operators. This process is repeated until the desired population size is reached. This approach is chosen over truncation selection (selecting the fittest individuals) to maintain a certain diversity within the population and prevent premature conversion to local optima.

5.1.9 Overview of novelties

This thesis introduces several novel contributions to the application of the NSGA-II algorithm for solving the MORCPSP/t-SE. These novelties include:

- **Optimized Non-Dominated Sorting Algorithm**: Modification of the traditional non-dominated sorting process to prevent duplicate dominance checks, which reduces computational complexity and enhances efficiency.
- **Tailored Mutation Operators**: Introduction of a right-shift mutation operator that induces more significant changes to the schedule, thereby enhancing solution diversity.
- **Constraint Penalty in Fitness Function**: Incorporation of a constraint penalty in the fitness function to ensure strict adherence to start and end time constraints.
- **Choice of N-Point Crossover**: Application of the n-point crossover method, that allows adaptability and optimization in the amount of crossover points.
- **Minimise duplicate individuals**: After crossover and mutation, the individual is checked for duplication within the population. If a duplicate is found, the individual undergoes an additional mutation.

5.2 Additional constraint handling

To improve the applicability of the scheduling algorithm to real-world scenarios, constraint handling mechanisms must be incorporated. As previously stated, two primary additional constraints are regarded. These are the temporal constraints dictating the start and end times of projects, and the allocation of a specified percentage of capacity towards certain project categories within the initial time horizon. Note that the regular capacity and precedence constraints are not included here, as they are adhered to by the design of the algorithm. Both constraints have a different handling mechanism as will be explained in the following sections.

5.2.1 Start and end time constraints

To adhere to the start and end time constraints, a dynamic penalty (see Section 2.7.1) will be integrated into the objectives. The size of the penalty is determined by the extent of the constraint violation. This is measured by calculating the number of timeslots between the scheduled timeslot of the project and the timeslot specified by the constraint. If the scheduled timeslot is too late or too early relative to the set start time or set end time, respectively, the difference is considered a violation.

The difference in timeslots is converted to the violation using the principle of the triangular number series. Specifically for the makespan objective, with n representing the difference in timeslots, the violation penalty is calculated as:

$$Violation = \frac{n(n+1)}{2} \tag{5.3}$$

It is crucial to apply a penalty to other objectives as well, to prevent an infeasible solution from achieving high scores in these objectives and thus appearing in the Pareto front. While the makespan uses the triangular number series for penalty calculation, other objectives cannot and must employ different scoring metrics. To address this, the makespan constraint penalty is converted into a percentage of the makespan objective. This percentage is then applied as a penalty to the other objectives.

By incorporating dynamic penalties, the algorithm can navigate instances where strict adherence to start/end time constraints may yield no feasible solutions. This approach facilitates the discovery of optimal solutions while minimizing constraint violations. For instance, when faced with a start time constraint conflicting with project predecessors' scheduling constraints, the algorithm can explore alternatives effectively.

5.2.2 Capacity allocation constraints on project categories

This constraint entails that a certain amount of capacity is allocated for projects within a certain category in a certain time horizon. For instance, projects can be categorized under a cost reduction category. In such cases, a specific constraint could involve allocating some percentage of the capacity to these projects in the upcoming year. These additional constraints are valuable in aligning the scheduling process with strategic objectives, thereby providing the generated schedules with greater real-world relevance and value.

For this constraint, a violation score is calculated as follows: First, for each project category, the total allocated capacity within the specified time range is calculated. Next, these totals are converted to percentages. With the percentages established, the Euclidean distance between the percentage scores and the expected percentages is computed. The Euclidean distance serves as a measure of dissimilarity between actual and expected percentages. To obtain an assessment of constraint violations across all project categories, the average distance per category is derived. It is calculated by dividing the Euclidean distance by the square root of the number of project categories. From this sequence of calculations, a violation score is computed that represents the average distance per category from the desired percentage distribution.

Since the percentages per project category are more like guidelines than strict constraints, a margin is defined to indicate acceptable deviations. For example, if the margin is set to 10%, and the average distance per category falls within this margin, it is considered compliant and not a violation. This approach ensures that minor deviations from the target percentages, which may be inevitable or even beneficial in practice, do not result in unnecessary penalties.

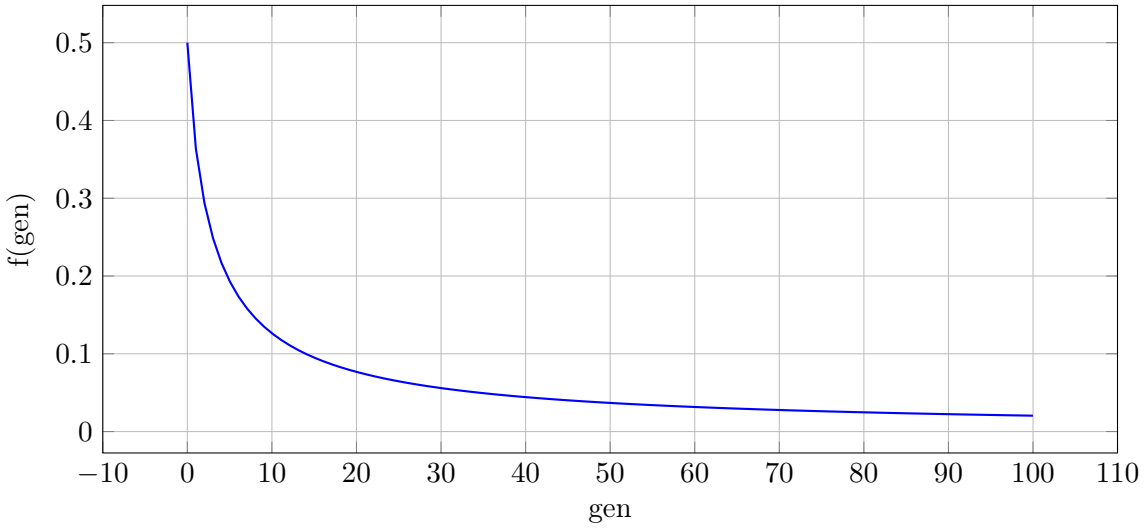


Figure 5.1: Plot of the function 5.4

As explained in Section 2.7, multiple constraint handling techniques can be used in multi-objective optimization problems. For this constraint, two distinct approaches are explored and employed (see Sections 5.2.2 and 5.2.2). The approaches will be compared by running the algorithm on multiple identical problem instances with identical constraints. The problem instances will be executed 100 times for each of the five different margin values. Different margin values will be tested to observe how the different handling methods are affected by the margin. Additionally, the average score of the solution and the standard deviation of the runs are stored. The solution will be evaluated using the HV indicator (see Section 5.3.1). Ultimately, this will show which method provides a better average score and lower variance in the solutions.

Dynamic constrained dominance principle

To apply the Dynamic Constrained Dominance Principle (DCDP), the domination algorithm in the NSGA-II was adapted to account for the constraints. A solution is feasible if the constraint violation falls within the dynamic threshold margin or the overall margin. This threshold logarithmically decreases as the generation number linearly increases, see Figure 5.1 for the threshold values for generation numbers. The threshold function is defined as:

$$f(\text{gen}) = \frac{0.5}{1 + \left(\frac{\text{gen}}{3}\right)^{0.9}} \tag{5.4}$$

This threshold function was chosen since it allows for more exploration in the early generations and forces convergence toward a feasible solution in the later generations. The values were commensurate with percentages and were determined through subjective intuition. In DCDP, feasible solutions dominate infeasible ones. If both solutions are infeasible, the solution with a lower constraint violation dominates. If both solutions are feasible, the regular procedure for dominance is applied (see Section 2.4).

Violation as extra objective

In the approach of incorporating violation as an additional objective, the optimization process extends beyond the original objectives to include the minimization of constraint

violations as an objective. This introduces a trade-off between optimizing the primary objectives and minimizing the constraint violations.

The additional objective value is the constraint violation, calculated as explained in Section 5.2.

5.3 Hyperparameter optimization

The proposed NSGA-II implementation contains several hyperparameters that control the behaviour of the algorithm. These include: the number of generations, population size, mutation rate, crossover rate, and number of crossover points. All hyperparameters influence the convergence rate, solution quality, and efficient exploration and exploitation in the search space.

To fine-tune these parameters, multiple HPO methods are applied. Initially, the individual effects of the parameters are researched by plotting the algorithm's outcome with one adapted hyperparameter against the generations. However, it is important to notice that the parameters might influence each other. For example, altering one parameter might affect the effectiveness of another, thereby impacting the overall performance.

Since the search space of the optimal configuration is too large, HPO techniques are applied. BO and PSO are selected for this problem as they can handle stochastic systems. The NSGA-II implementation is inherently stochastic due to the randomness in its evolution steps.

5.3.1 Pareto front evaluation

Before optimizing parameters, a method to evaluate the quality of a Pareto front must be selected. Section 2.9 discusses two such methods: the HV indicator and the IGD. This research employs the HV indicator as it does not require a ground truth, unlike the IGD method. Given that the scheduling problems addressed in this study lack a ground truth, the HV indicator is more suitable. For the HV indicator, a reference point needs to be provided. The reference point will be obtained by running the algorithm on a specific problem, observing the outputted Pareto front, and choosing a reference point some degree larger than the biggest values in the front. The reference point must be bigger than the values of the Pareto front in the appropriate dimensions to be able to calculate a HV. Throughout the subsequent sections, any mention of the algorithm's output score refers to the HV of the Pareto front produced by the algorithm.

5.3.2 Individual optimization

To find the individual influence of the hyperparameters, different values of a certain parameter are compared, while keeping the other parameters constant. First, an initial guess for good parameter values has been made, see Table 5.4. This initial parameter value is based on frequently used values in similar research [26][59][20]. The chosen values are also in line with research on general GA parameter tuning [43][5] and RCPSP specific parameter tuning [52]. [43] and [5] state that the relation between crossover and mutation rate does not influence fitness a lot, as long as they are not both small. This is logical, as a GA requires genetic alterations to improve solutions and avoid prolonged stagnation.

To prevent calculating every possible parameter value, a strategy of selecting multiple test values for each parameter is opted, as outlined in Table 5.4. These values are strategically chosen, with smaller increments around the initial value and larger increments as the values move further away. Parameters such as generation number and population size

Table 5.4: Initial and tested parameter values

Parameter	Initial value	Tested values
# generations	50	N/A
population size	50	N/A
mutation rate	0.05	[0.01, 0.05, 0.1, 0.2, 0.4, 0.7]
crossover rate	0.8	[0.1, 0.3, 0.5, 0.7, 1]
# crossover points	2	[1, 2, 3, 5, 10]

do not have predefined test values. Due to the intuitive relationship, increasing the values generally leads to improved results but simultaneously increases computation time, the values do not need testing values.

Instead, a different approach will be taken to find the most efficient values for population size and number of generations. Multiple population sizes will be compared with a fixed generation number, set at 200. Each population size will undergo 100 iterations. The average score for the Pareto front will be plotted for each generation, creating a plot with average scores per generation for each of the different population sizes. From this plot, the convergence rates can be extracted, pinpointing the generation number beyond which the algorithm's average performance demonstrates diminishing improvements.

For the remaining hyperparameters, the following procedure will be implemented. Each tested value will be run 100 times while keeping the other parameters fixed at their initial constant values. Running each value 100 times minimizes the impact of randomness in the algorithm. The average scores per generation for each parameter value will be plotted, illustrating how the Pareto front scores evolve over generations for each parameter value. This will help identify the most optimal parameter value, indicated by the highest final average score. In addition to plotting the score per generation, the standard deviation of the scores per generation will also be plotted. This will demonstrate the stability of the results for each specific parameter value. A high deviation is undesirable as it suggests that randomness significantly affects the outcomes, leading to greater uncertainty in the algorithm's performance.

5.3.3 Hyperparameter optimization

The previous section describes the methods to find the individual impact of the hyperparameters. However, the hyperparameters may also interact with each other or be optimal only in specific configurations. To identify these optimal configurations, BO and PSO will be applied. These two HPO techniques are chosen for several reasons. Firstly, both BO and PSO are highly effective in stochastic systems, which is crucial given the randomness in the evolutionary steps of the algorithm. Secondly, they are widely used in HPO, meaning that there are libraries and practical applications to draw from. Thirdly, both methods are great at finding global optima and reducing the risk of getting stuck in local optima due to their effective balance of exploration and exploitation. By applying these methods, their results can be compared to identify possible similarities, providing deeper insights into the optimal hyperparameter configurations.

Before applying the optimization methods, search bounds must be set to prevent the search space from becoming too large. To begin with, the population size and number of generations will be constant throughout the optimization and set to their initial value, as their influence on the outcome is trivial. For the remaining parameters, the bounds will be the initial and final element of the tested values, depicted in Table 5.4. These values

are chosen to give the optimization technique enough coverage for every parameter while still having sufficient direction.

Bayesian optimization

BO will be implemented using the 'gp hedge' acquisition function. An acquisition function in BO determines the next set of hyperparameters to evaluate, by balancing exploration (sampling in uncertain regions) and exploitation (sampling where high values are predicted). The 'gp hedge' method combines the predictions of multiple acquisition functions, leveraging the strength of different functions.

Particle Swarm optimization

PSO itself has some hyperparameters that influence the movement of each particle through the search space. These hyperparameters are cognitive coefficient (set to 0.5), social coefficient (set to 0.3), inertia weight (set to 0.9) and number of particles (set to 10). The cognitive coefficient controls how much a particle's own best position influences its movement, encouraging individual learning. The social coefficient controls how much the global best position influences the particle, promoting social learning from the swarm. The inertia weight influences the particle's momentum, balancing exploration and exploitation by controlling the impact of the previous velocity. The values of these parameters are default and recommended values by the Pyswarms library.

5.4 Validation

The validation process is divided into two parts, both assessing different aspects of the algorithm's capabilities and effectiveness. First, a benchmark comparison will be done to find the algorithm's capability to find near-optimal solutions. Second, manual validation will be done to verify the adherence to the specified constraints.

5.4.1 Dataset

To validate the algorithm's ability to optimize makespan, the datasets created by Hartmann [28] are used. These datasets are derived from the standard RCPSP benchmark set [38], a well-established dataset in the RCPSP research community, available at PSPLIB [37][1]. Hartmann adapted these sets to include timed resource capacities and requests for the RCPSP/t, enhancing the complexity and realism of the test instances.

Hartmann's adaptation involved manipulating parameters that control resource alterations within the problem instances. Specifically, two probabilities, P^R and P^r, determine the likelihood of reducing resource capacities and resource availabilities and requests, respectively. Higher values of these probabilities result in more frequent modifications to the resources. Additionally, two factors, F^R and F^r, represent the magnitude of these reductions, with smaller values indicating stronger reductions.

The probabilities were set to 0.05, 0.1, and 0.2. Importantly, the probabilities for altering capacities and requests were kept equal, meaning $P^R = P^r$. The factors were set to 0 and 0.5, with the same equality applied, i.e., $F^R = F^r$. This parameter configuration led to six distinct instance types for each set size. The dataset included:

[1] www.om-db.wi.tum.de/psplib/main.html

- Six different instances derived from the set with $n = 30$ activities, resulting in $6 \times 480 = 2880$ instances in total.
- Six different instances derived from the set with $n = 120$ activities, resulting in $6 \times 600 = 3600$ instances in total.

These varied instances provided a robust basis for evaluating the algorithm's performance across different levels of complexity and resource constraints.

The dataset is labelled as follows: 'j30t' denotes the set with 30 activities, where 'j' stands for job or activity, '30' represents the number of jobs, and 't' indicates time dependency. Within this set, subsets are labelled as j30t1 or j30t2, where the numbers denote different parameter configurations used during data generation. Further granularity is given by labels such as 'j30t1_2_3', where '1' denotes the parameter configuration ($P^R = 0.05$ and $F^R = 0.5$) and '2' and '3' together identify the problem instance within that subset.

5.4.2 Benchmark comparison

To evaluate the performance of the proposed algorithm, each problem instance in the dataset will be executed to determine if it can match the best-known schedule (obtained by Hartmann[2][29]) and to count the number of attempts required. The process involves running each sub dataset (e.g., j30t1, j30t2, j120t1, etc.) independently to view the influence of the P^R, P^r, F^R, F^r and amount of activities. For each problem instance in a sub dataset, the algorithm's result will be compared to the best-known schedule. If the result is worse, the algorithm will be rerun up to nine additional times. The number of attempts needed to match the best-known makespan will be recorded for each instance. For each problem instance, the algorithm's best result will be compared to the best-known makespan. This can either be a negative number, if the algorithm's makespan was smaller, indicating a better schedule, or a positive number if the best-known makespan was not matched.

This procedure will be repeated for all instances in the dataset. The collected data will include the number of attempts taken or the deviation from the best-known result for each instance, allowing for an assessment of the algorithm's ability to find the optimal solution and its average deviation from the best-known schedule.

Since the algorithm is multi-objective and the dataset evaluates only the makespan, validation will focus solely on this criterion. The algorithm produces a Pareto front with multiple optimal solutions. The solution with the smallest makespan from the Pareto front will be selected for comparison with the benchmark.

5.4.3 Manual validation

Since the benchmark dataset focuses solely on makespan, other aspects of the algorithm, such as start and end time constraints, will be validated manually. To achieve this, two problem instances will be randomly selected from each sub dataset. Additionally, start and end time constraints and capacity allocation for project category constraints will also be chosen at random.

First, all projects in the dataset will be assigned a category randomly, represented by integers 1 to 3. A distribution for these categories will then be selected, for example, [0.5, 0.4, 0.1], along with a time horizon, say 20. This distribution implies that within the first

[2]The results of Hartmann were obtained through direct communication via email.

20 time steps, 50% of the capacity must be allocated to projects within Category 1, 40% to Category 2, and the remaining 10% to Category 3. Beyond these 20 time steps, the distribution of capacity among the category becomes irrelevant.

In addition to the category constraints, start and end time constraints will also be imposed. Specifically, one project will be assigned a start time constraint, and another project will be assigned an end time constraint.

The algorithm will then be run on randomly chosen and created test cases from different subsets. Random selection is used to prevent any bias and instances from different subsets are used to represent different scenarios. The solutions will be manually evaluated to ensure they adhere to the imposed constraints. This evaluation will be conducted by reviewing visualizations of the schedules and capacity distributions. Moreover, these schedules will be compared to the schedules created for the same problem instances without the constraints. This comparison aims to highlight the impact of the constraints and to determine whether the algorithm can still generate near-optimal solutions under the given constraints.

Chapter 6

Results & analysis

6.1 Experimental setup

The proposed algorithm, optimization, validation and parsing scripts were all developed in Python (version 3.12.3) due to its ease of prototyping, library availability, and personal familiarity with the language. While languages like C++ offer faster execution speeds, the complexity would have posed significant challenges. Therefore, Python's flexibility and libraries made it an ideal choice for rapid experimentation.

Several libraries were utilized to optimize or validate the algorithm:

- **pymoo**: This library was used for calculating the HV indicator, providing the quantitative measure of a Pareto front.
- **scikit-optimize**: This library provided the BO functionality.
- **pyswarms**: This library offered the implementation for the PSO.

These libraries were chosen because they are widely used, popular in the Python community, and are well-documented and open source.

The test sets from PSPLIB (see section 5.4.1) were provided in text format. To use the sets, a PSPLIB parser was developed based on the source code provided at GAMS[1]. This parser enabled seamless integration of the benchmark datasets into the algorithm for testing and validation.

6.2 Constraint handling comparison

A comparative analysis was conducted between the DCDP method and an additional objective constraint-handling approach. The algorithm was applied to four problem instances, each running for 100 iterations. The constraint was set at 0.5 for Category 1, 0.3 for Category 2, 0.2 for Category 3, and a time horizon of 20 timesteps. The mean HV and standard deviations were observed for both methods, across five different margin values. Figure 6.1 illustrates the results of this comparison.

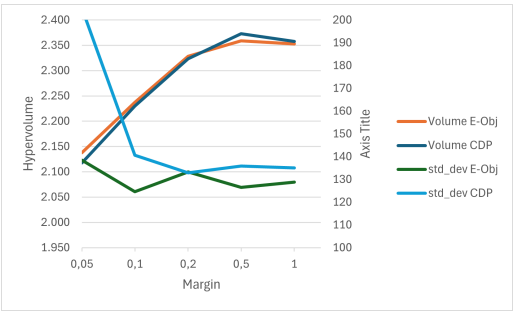
Particularly noteworthy are the values on the left side of the plot, where margins of 0.05 and 0.1 are highlighted since these are the most realistic and applicable scenarios. These results reveal that the DCDP method consistently achieves higher standard deviation scores, especially at a margin value of 0.05. Despite this, the mean HVs of both methods

[1] www.gams.com/latest/gamslib_ml/libhtml/gamslib_rcpsp.html

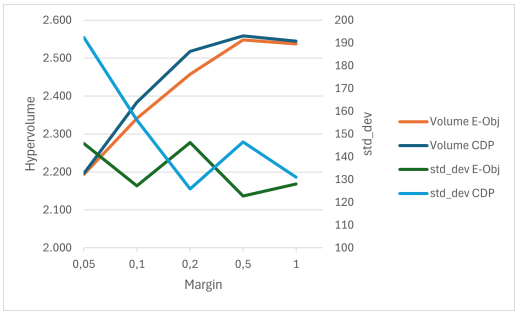
are quite similar, with DCDP showing a slight improvement, suggesting that the quality of the resulting Pareto fronts is comparable.

It is important to note that Figures 6.1c and 6.1d indicate a HV of 0 at a margin of 0.05 for the additional objective constraint handling method. This arises due to the reference point being too small in one or both objectives, causing the objectives to exceed this point. This can also be the cause of the lower standard deviation at a margin of 0.05.

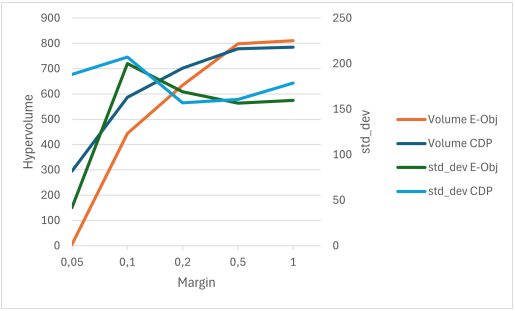
The findings suggest that both methods are effective for constraint handling. The DCDP method exhibits higher standard deviations but yields superior Pareto fronts. This can be attributed to the fact that the DCDP method avoids dilution of the population with infeasible individuals, unlike the additional objective method. This exclusion allows for a more focused optimization of the other objectives.



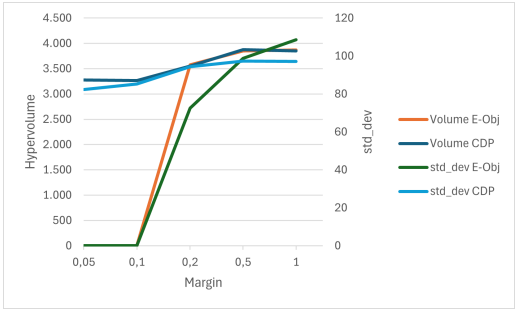
(a) j30t1_10_4



(b) j30t2_11_1



(c) j30t3_15_8



(d) j30t4_20_10

Figure 6.1: Constraint handling method comparison with HV mean and standard deviation (y-axis) and different margin values (x-axis)

6.3 Hyperparameter optimization

Initially, the individual impact of key hyperparameters was assessed to determine the hyperparameter configuration for the NSGA-II algorithm that is the most effective and efficient. This was accomplished by running the algorithm on three randomly selected problem instances from the J120t1 dataset, each time varying one hyperparameter while keeping the others constant and running each configuration 100 times. The primary metrics for evaluation were the HV indicator and the standard deviation of the HV.

The plots for the individual parameter assessments (Figures 6.2, 6.3, 6.4) are organized with the generation number on the x-axis and the HV or standard deviation on the y-axis. Different colours represent the different parameter configurations. These plots allow for a visual comparison of how various configurations perform over successive generations,

highlighting trends and performance stability across the tested hyperparameters.

The following sections detail the results for mutation rate, crossover rate, and number of crossover points, as well as the overall findings from automated HPO techniques.

6.3.1 Mutation rate

The results of the mutation rate optimization are illustrated in Figure 6.2. The HV plots show lower mutation rates, specifically 0.01 and 0.05, yield the highest HV values, indicating superior performance. Conversely, higher mutation rates of 0.2, 0.4, and 0.7 result in significantly lower HV scores after the fifth generation, suggesting suboptimal performance. Next, the standard deviation plots reveal that higher mutation rates generally lead to lower standard deviations, implying more consistent results. However, mutation rates of 0.05 and 0.1 also exhibit relatively low standard deviations, balancing both high performance and stability. Notably, the mutation rate of 0.01 consistently achieves the highest standard deviation. Moreover, Figure 6.2d shows a clear deviation from the others, rising steeply while the rest generally trends downward. This indicates that the results in later generations of the runs with a mutation rate of 0.01 exhibited great variability. However, it is important to note that such an outlier might be coincidental. Upon re-running, this deviation was not replicated, suggesting that the observed difference might be due to random variations.

6.3.2 Crossover rate

The results of the crossover rate optimization are illustrated in Figure 6.3. The HV plots show no distinguishable differences between the different rates at generation. Each of the various rates shows a similar curve and end value with slight variations between the three problem instances. This suggests that the crossover rate has a more subtle impact on the overall performance when measured by HV alone. In contrast, the standard deviation plots show more pronounced differences between the rates. Overall, a rate of 1 scored the highest and rates of 0.5 and 0.7 scored the lowest.

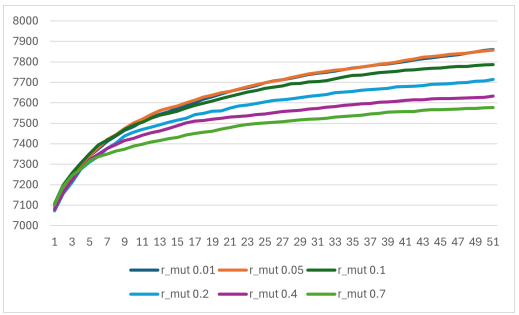
6.3.3 Number of crossover points

The results of the optimization for the number of crossover points are illustrated in Figure 6.4. The HV plots demonstrate a clear pattern: configurations with 1 or 2 crossover points consistently achieve the highest HV values. In contrast, configurations with 10 crossover points consistently result in lower scores. The standard deviation plots reveal similar trends. Configurations with higher numbers of crossover points lead to higher standard deviations, indicating greater variability and less consistent outcomes. Conversely, using 1 or 2 crossover points results in lower standard deviations.

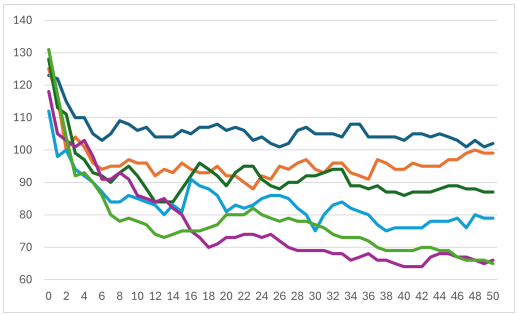
6.3.4 Bayesian and Particle Swarm Optimization

To determine the optimal hyperparameter configuration, HPO techniques were applied to multiple problem instances. Both BO and PSO were executed with a total of 1000 iterations. For PSO, this total was divided into a population of 10 with 100 iterations each, whereas BO had 1000 iterations directly. Each of the optimized problem instances underwent optimization six times in total, three times using each optimization method.

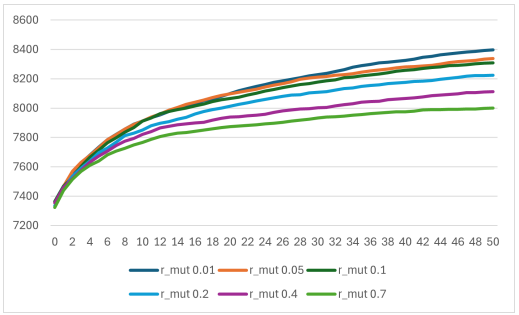
The results are depicted in Figure 6.5. This figure illustrates the optimal configuration obtained from each run. In Figure 6.5a, the differences between the outcomes of the BO (represented by dots) and PSO (represented by crosses) are shown. Moreover, the difference between different problem instances is shown, indicated by a different colour. Figure 6.5b



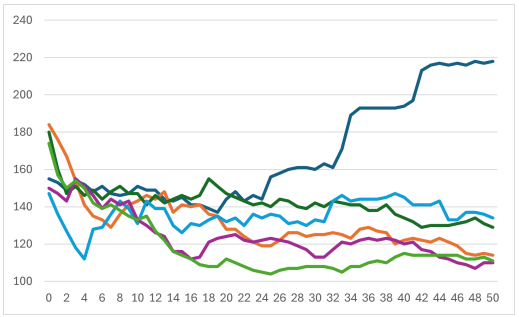
(a) j120t1_1_10



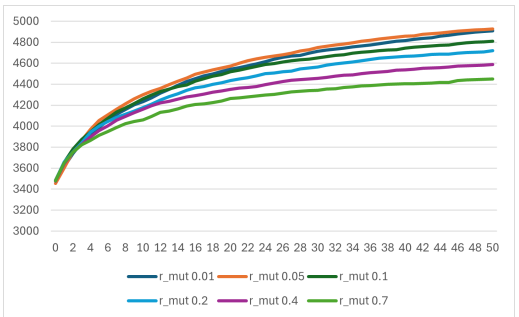
(b) j120t1_1_10



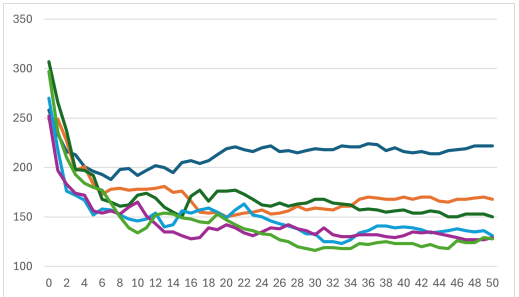
(c) j120t1_2_3



(d) j120t1_2_3

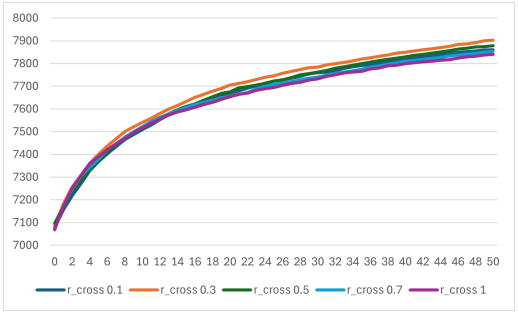


(e) j120t1_5_10

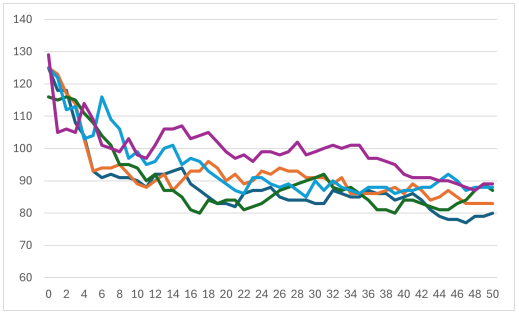


(f) j120t1_5_10

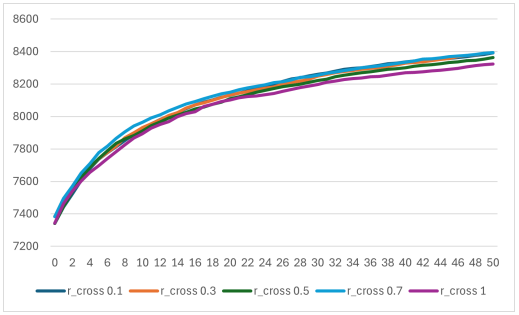
Figure 6.2: Comparison of mutation rate values with HV (left) or the standard deviation (right) on the y-axis and generation number on the x-axis.



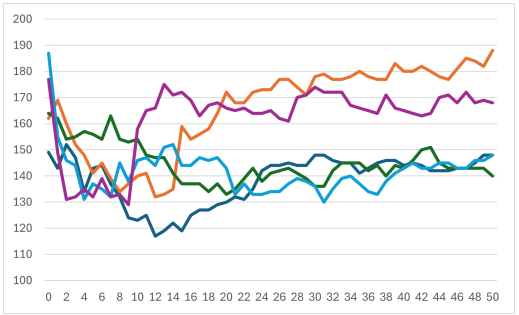
(a) j120t1_1_10



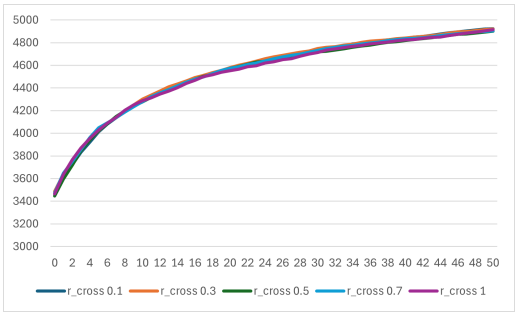
(b) j120t1_1_10



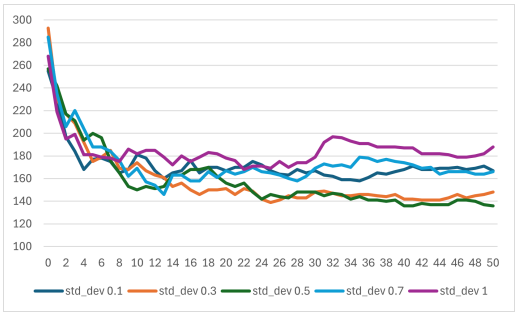
(c) j120t1_2_3



(d) j120t1_2_3

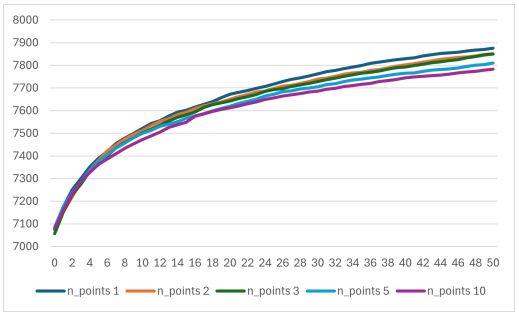


(e) j120t1_5_10

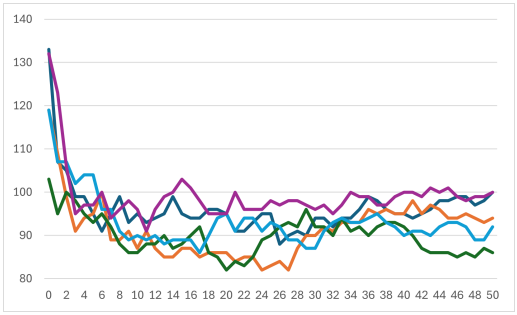


(f) j120t1_5_10

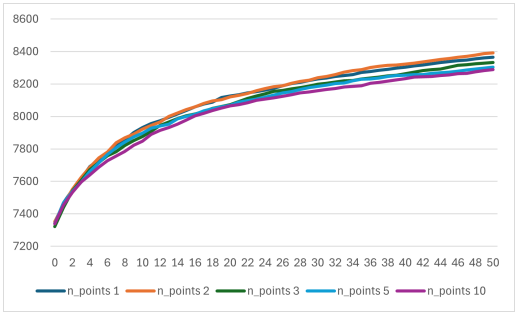
Figure 6.3: Comparison of crossover rate values with HV (left) or the standard deviation (right) on the y-axis and generation number on the x-axis.



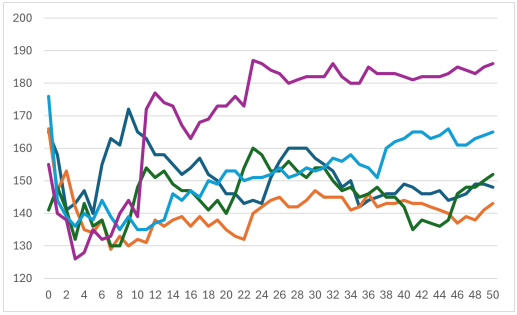
(a) j120t1_1_10



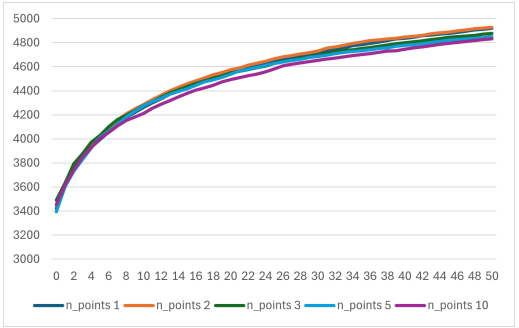
(b) j120t1_1_10



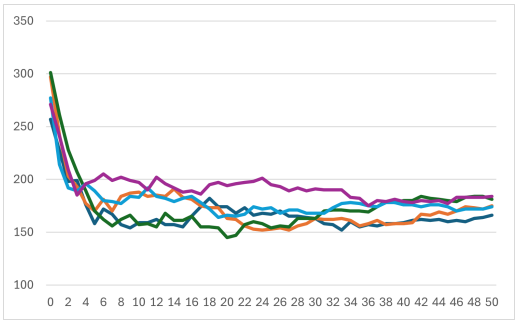
(c) j120t1_2_3



(d) j120t1_2_3



(e) j120t1_5_10



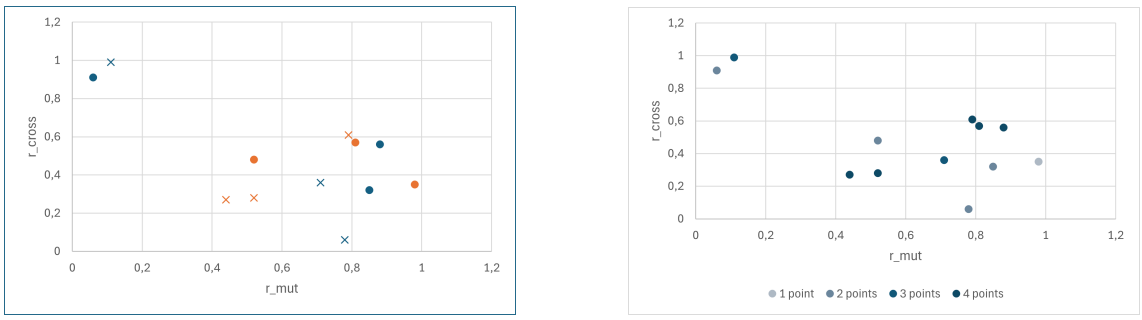
(f) j120t1_5_10

Figure 6.4: Comparison of crossover points values with HV (left) or the standard deviation (right) on the y-axis and generation number on the x-axis.

presents the same distribution of data points, however, the colour represents the number of crossover points included in the optimal configuration.

The results demonstrate high variety in the plots. To begin with, there is considerable diversity in the combinations of crossover and mutation rates, even for identical problem instances. This implies that the HPO methods are still greatly affected by the stochastic nature of the algorithm, with no single optimal combination of these hyperparameters emerging. Notably, the majority of combinations have a mutation rate above 0.5 and a crossover rate between 0.3 and 0.6. Also, there are no combinations in the lower-left quadrant, $[0, 0.4] \times [0, 0.4]$ and only one on the border of the upper-right quadrant, $[0.6, 1] \times [0.6, 1]$. This indicates that at least one hyperparameter must be sufficiently high to ensure population diversity, while both hyperparameters cannot be excessively high to preserve valuable characteristics. This finding is also in line with other research on the effects of the hyperparameters [43][5].

The results in Figure 6.5b indicate a preference towards four crossover points. There is no clear correlation between the combination of crossover and mutation rates and the number of crossover points. However, when examining the plot Figure 6.5a, it is revealed that all but one of the outcomes of PSO have four crossover points. However, this may be influenced by chance, given the small sample size.



(a) Grouped by problem instance and optimization technique

(b) Grouped by amount of crossover points

Figure 6.5: Visualisations HPO results

6.4 Benchmark comparison

To validate the capability of the algorithm to find solutions with an optimal value in an objective, the algorithm was run on all problem instances in the j30t and j120t datasets (see Section 5.4.1 for more detail) and compared with the results from Hartmann [28]. The bar charts in Figures 6.6 and 6.7 illustrate the distribution of attempts required to find a solution with an improved or identical makespan value, compared to Hartmann's result. The x-axis represents the number of tries, while the y-axis shows the number of successful solutions on each attempt. A problem instance was run a maximum of ten times. If no successful solution was found, the run was added to the ten number of tries bucket.

To begin with, the bar charts of the j30t datasets and Table 6.2 clearly show that most of the best solutions are found in the initial try, with only a few problem instances failing to get a successful solution. Next to that, there is a notable difference between the datasets 1, 2 and 3 and 4, 5 and 6. The latter datasets show more unsuccessful runs. These datasets have capacity drops of 50% instead of 100% and the results indicate that the algorithm encounters more difficulty with such configurations.

The bar charts in Figure 6.7 show big differences in distribution compared to the j30t data. Notably, there is a higher number of unsuccessful runs, leading to a larger bar at ten tries. This trend is particularly evident in datasets 4, 5, and 6, which show more problem instances requiring ten tries than those solved in one try. Interestingly, the j120t3 dataset performs best, similar to how the j30t3 dataset performed best.

Next to the number of tries, the difference between the benchmark dataset solution and the generated solution for each problem instance was recorded. Based on this, the average error rate and the average improvement were calculated and presented in Table 6.3. The average error rates are around 6.6% of the total makespan. Moreover, in approximately $\frac{366}{3600} * 100 \approx 10.2\%$ of the problem instances, an improved solution was found, with an average improvement of 5 timesteps. This shows that there is still a lot of variability in solving problem instances with a lot of projects.

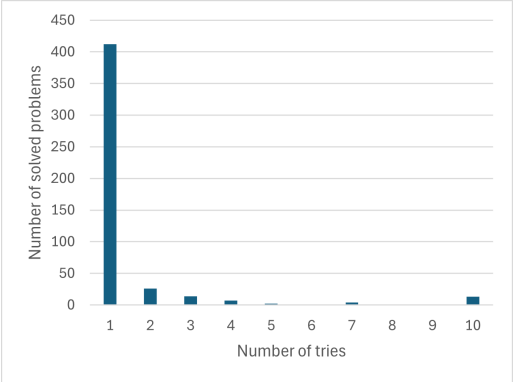
Overall, the proposed algorithm performs slightly worse on the makespan objective compared to the benchmark algorithm. This is most likely because the proposed algorithm considers multiple objectives, which results in a less concentrated focus on optimizing the makespan. Consequently, fewer solutions are specifically tailored to optimize this particular objective, reducing the likelihood of finding the optimal project configuration.

		# Tries									
		1	2	3	4	5	6	7	8	9	10
Dataset	j30t1	392	42	13	2	7	6	1	2	1	14
	j30t2	412	26	14	7	2	1	4	1	0	13
	j30t3	450	15	7	2	1	1	2	0	0	2
	j30t4	357	53	20	5	4	4	1	1	5	30
	j30t5	356	49	11	12	5	1	2	2	5	37
	j30t6	329	58	26	11	9	8	3	5	3	28
	j30t	2296	243	91	39	28	21	13	11	14	124
	j30t (%)	79.7	8.4	3.2	1.4	1	0.7	0.5	0.4	0.5	4.3

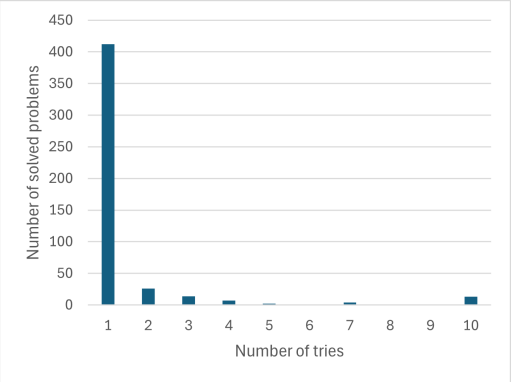
Table 6.1: Distribution of attempts to best solution by number of tries for J30t

		# Tries									
		1	2	3	4	5	6	7	8	9	10
Dataset	j120t1	265	49	28	19	18	12	8	13	9	179
	j120t2	278	61	34	15	12	11	8	6	5	170
	j120t3	373	63	22	10	11	6	9	2	4	100
	j120t4	197	65	43	28	11	16	11	11	14	204
	j120t5	129	59	48	25	21	9	14	18	7	270
	j120t6	118	53	30	22	14	16	14	13	8	312
	j120t	1360	350	205	119	87	70	64	63	47	1235
	j120t (%)	37.8	9.7	5.7	3.3	2.4	1.9	1.8	1.8	1.3	34.3

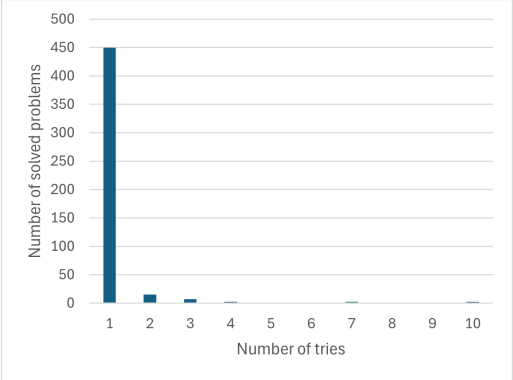
Table 6.2: Distribution of attempts to best solution by number of tries for J120t



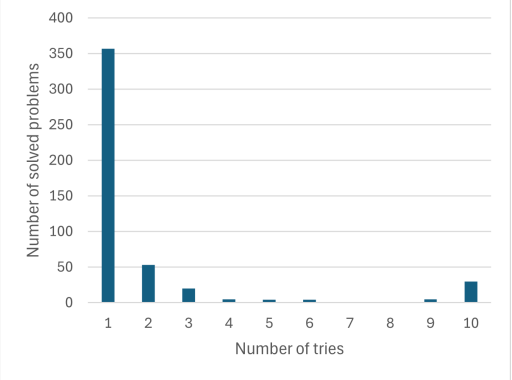
(a) Dataset j30t1



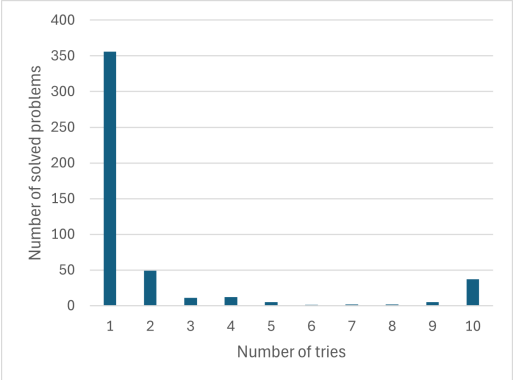
(b) Dataset j30t2



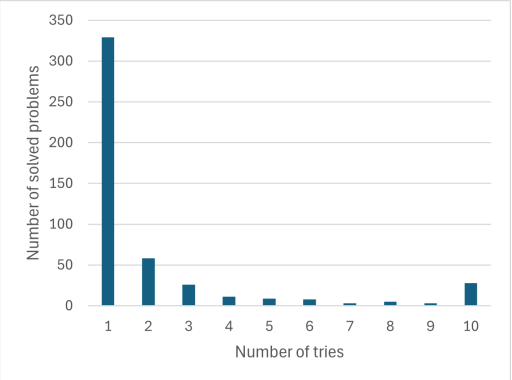
(c) Dataset j30t3



(d) Dataset j30t4

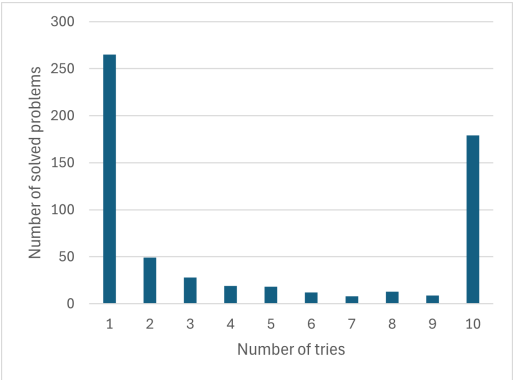


(e) Dataset j30t5

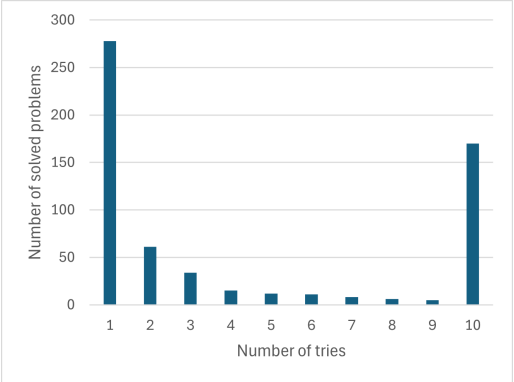


(f) Dataset j30t6

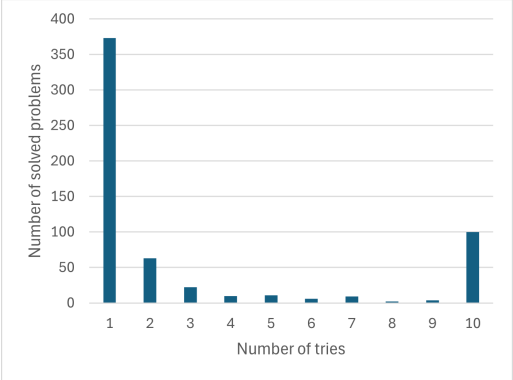
Figure 6.6: Distributions of tries to achieve the best outcome for J30t dataset



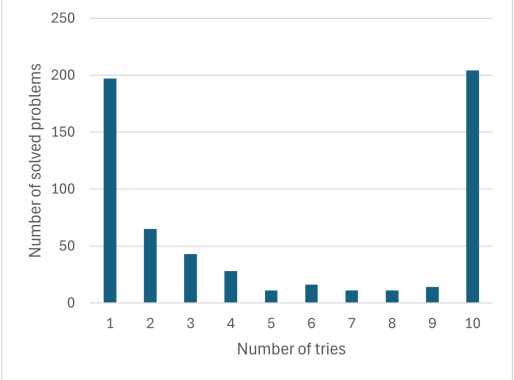
(a) Dataset j120t1



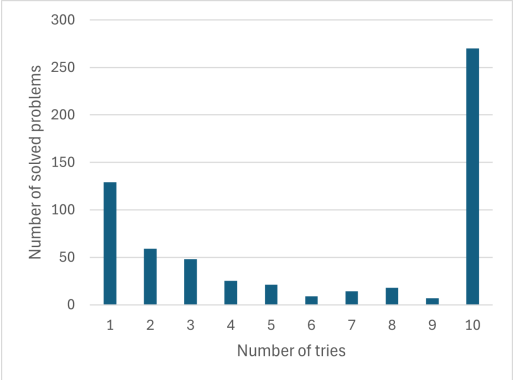
(b) Dataset j120t2



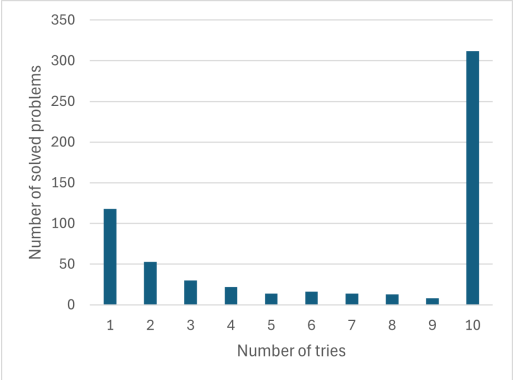
(c) Dataset j120t3



(d) Dataset j120t4



(e) Dataset j120t5



(f) Dataset j120t6

Figure 6.7: Distributions of tries to achieve the best outcome for J120t dataset

Dataset	Total Improved	Avg Improvement	Avg Error rate
j120t1	64	2.66	0.059
j120t2	44	3.95	0.076
j120t3	25	15.52	0.110
j120t4	101	2.23	0.043
j120t5	74	2.59	0.052
j120t6	58	3.03	0.058
j120t	366	5.00	0.066

Table 6.3: Improvement and Error Metrics by Dataset

6.5 Manual runs

The algorithm's capacity to manage additional constraints was assessed using one problem instance from the j30t1 dataset and three problem instances from j120t dataset (specifically j120t1, j120t4 and j120t6). The selected subsets of data were carefully chosen to introduce a variety of challenges for manual validation. A j30t1 problem instance was specifically selected due to its suitability for clear visualization, given its relatively smaller size of 30 projects as opposed to 120. The j120t problem instances were incorporated to demonstrate the algorithm's capability to manage higher complexity levels, also with different levels of varying capacity requests.

The validation process involved a manual review of the generated schedules and their properties. Each problem instance was initially executed with No Additional Constraints (NAC). Subsequently, runs were performed incorporating only the Project Category Constrained (PCC), followed by runs that included both the Project Category Start/End time Constrained (PCSEC). The set start/end time constraints were determined by analyzing the NAC run schedule and selecting constraints that necessitate adjustments in the schedule.

For each execution, Pareto fronts across different generations were plotted. A schedule containing the project's resource requests per resource was also generated, along with a plot depicting the capacity distribution across the project categories. It is important to note that the solution with the lowest makespan in the Pareto front was chosen for the visualization. This solution was chosen because it is the smallest and was most often the easiest to visualize and analyze.

In the Pareto plots, the coloured dots represent solutions from specific generations, with blue indicating the initial population and red indicating the final generation. The Y-axis represents the resource utilization smoothness objective, while the X-axis represents the makespan objective.

The schedule plots visualize project start times, durations, and resource requests per resource. Each coloured block represents a project, with its position and width indicating the timeslots during which the project is active. The height of the block corresponds to the number of resources it uses. The schedule plots include four separate schedules, one for each resource type. Each schedule also features a black line that marks the maximum resource capacity for that resource across the timeslots. At certain points in the schedule, the capacity drops to zero or 50%, as illustrated by the black line.

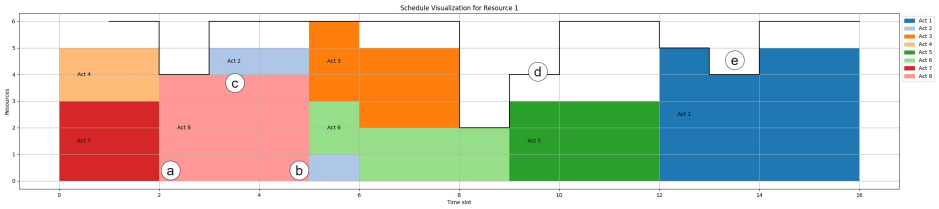


Figure 6.8: Example schedule plot

Figure 6.8 provides an example schedule. In this case, there is a single resource that varies over time. There are 8 activities scheduled, each with a duration and capacity demand. Activity 1 (the dark blue block) has a varying capacity demand.

(a) Start of the block denotes the starting timeslot, 2, of activity 8 (the pink block).

(b) End of the block denotes the ending timeslot, 5, of activity 8.

(c) The height of the block denotes the capacity demand of activity 8.

(d) The black line visualizes the capacity that varies over time.

(e) Activity 1 (the dark blue block) shows varying capacity demands in its second timeslot.

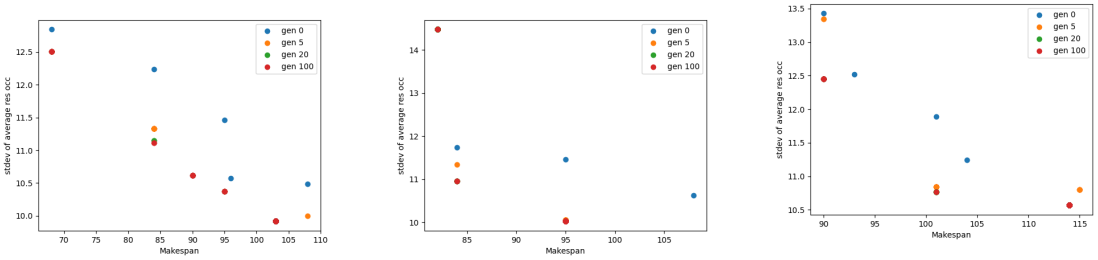
The capacity distribution plots illustrate the proportion of capacity allocated to projects within a specific category over a certain number of timesteps, alongside the target allocation. The solid lines indicate the actual allocation, while the dashed lines of the same colour represent the target allocation. The proportion is calculated as the total utilized capacity divided by the allocated capacity for the projects in each category. In the test cases, three categories were created, and projects were randomly distributed among these categories. The target capacity distribution is: Category 1 - 50%, Category 2 - 20%, and Category 3 - 30%. Additionally, a time horizon of 20 timesteps was used.

6.5.1 Results of problem instance j30t1_5_4

Figures 6.9, 6.10, 6.11 present the outcomes of the manual validation for problem instance j30t1_5_4. Starting with Figure 6.9, the plots illustrate the evolution of Pareto fronts across generations. The NAC run emerges as the top performer in the final Pareto fronts, followed by the PCC run. Although the PCC run yields comparable results for the extreme solutions (those closest to the axes), it consistently performs worse on the other objective. The PCSEC run exhibits the poorest performance across all objectives, also showing significantly higher makespans.

The set start/end time constraints for the PCSEC run were set to: Project 4 starts at timeslot 3 and Project 3 ends before timeslot 20. Examining Figure 6.10c, the schedule adapts to the set start/end time constraints. In the upper schedule, Project 4 (the light orange box) does not start in the initial timeslots as observed in the NAC run, but instead, is delayed. Similarly, Project 3 (the dark orange box) is scheduled at the earliest possible time due to the constraint requiring its completion by timeslot 20. However, the schedule indicates it finishes at timeslot 23. This discrepancy arises because there was insufficient capacity to execute the project in earlier timeslots, leading to penalties applied to the objectives in the PCSEC run. This also accounts for the higher objective values observed in the Pareto front for the PCSEC run.

Figure 6.11 shows that the NAC run allocates approximately 55% of the utilized capacity to the Category 3 projects and 25% on Category 1, which is far from the targets. The PCC and PCSEC runs are closer to the targets showing the impact of the constraint.

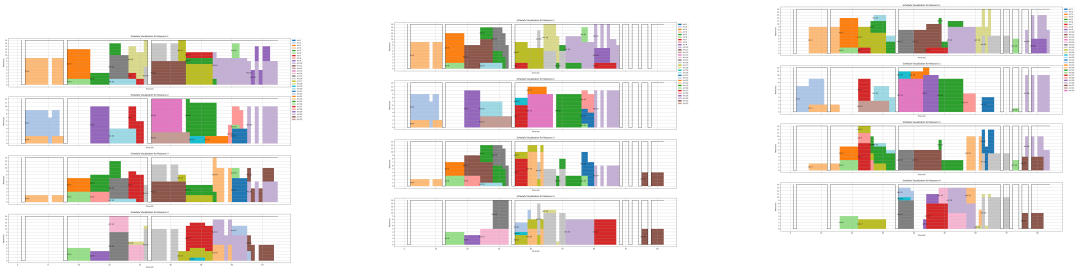


(a) No additional constraints

(b) Project category constrained

(c) Category and set start/end time constrained

Figure 6.9: Pareto fronts for problem instance j30t1_5_4

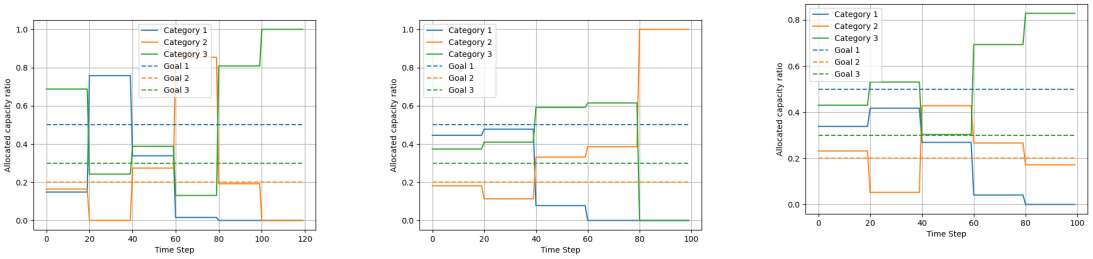


(a) No additional constraints (enlarged A.1)

(b) Project category constrained (enlarged A.2)

(c) Category and set start/end time constrained (enlarged A.6)

Figure 6.10: Schedule with capacity usage for problem instance j30t1_5_4



(a) No additional constraints

(b) Project category constrained

(c) Category and set start/end time constrained

Figure 6.11: Capacity distribution on project category for problem instance j30t1_5_4

6.5.2 Results of problem instance j120t1_2_3

Figures 6.12, 6.13 and 6.14 illustrate the results of the manual validation for problem instance j120t1_2_3. The Pareto fronts shown in Figure 6.12 demonstrate more significant improvement over generations and contain a greater number of solutions compared to the j30t1_5_4 results. The increased complexity of the problem instance likely provides more options and project configurations, leading to richer solution sets. Consistent with the j30t1_5_4 outcomes, the NAC run exhibits the best Pareto fronts, followed by the PCC and PCSEC runs.

The start and end time constraints for the PCSEC run were defined as follows: Project 4 begins at timeslot 3, and Project 37 must finish before timeslot 40. Figure 6.13 displays the schedules for each run. In the light orange box on the left of the schedule for the second resource (second from the top), Project 4 can be observed. In the NAC and PCC runs, Project 4 starts in the initial timeslot, whereas in the PCSEC run, it shifts to the third timeslot to meet the constraint. Additionally, Project 37, represented by the yellow-green block, starts around timeslot 40 in the first resource schedule for both the NAC and PCC runs but is moved to start around timeslot 28 in the PCSEC run, thus satisfying the constraint, leading to no additional penalties as seen in the Pareto front.

Lastly, the capacity plots in Figure 6.14 show a clear difference in the NAC run and the PCC and PCSEC runs. The latter two runs show smaller deviations from the target distribution, indicating the influence of the constraint.

6.5.3 Results of problem instance j120t4_5_6

Figures 6.15, 6.16, and 6.17 illustrate the outcomes of the manual validation for problem instance j120t4_5_6.

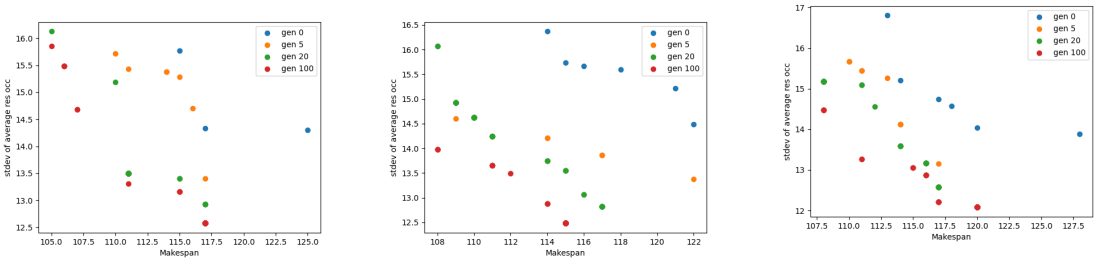
The Pareto fronts depicted in Figure 6.15 reveal distinct differences among the three runs. The NAC run significantly outperformed the others on the makespan objective. Notably, the PCC run features two solutions in the top left corner of generations 0 and 5. Under normal circumstances, the solution from generation 5 would be included in the Pareto front, as it is not dominated by any solutions in subsequent generations. However, due to the DCDP constraint handling method, these solutions were permissible in early generations but failed to pass when the dynamic threshold became more stringent. The PCSEC run's Pareto front contains only two solutions, which is relatively low, likely due to the stringent constraints preventing other solutions from meeting the required criteria.

Figure 6.16 displays the generated schedules. This problem instance, being from the j120t4 dataset, features a varying capacity that drops to 50%, as illustrated by the black bar only descending to the 50% mark in the schedules. The start and end time constraints for this instance were set such that Project 4 begins at timeslot 3, and Project 15 must end before timeslot 20. In the NAC run in the schedule for the first resource, Project 15 (the dark grey box) starts at timeslot 29 in the first resource schedule, and Project 4 (the light orange box) starts in the initial timeslot. In the PCSEC run, Project 4 is shifted to the right, and Project 15 now ends at timeslot 15, thereby fully adhering to the constraints.

Lastly, the capacity distribution plots in Figure 6.17 show a marked difference between the NAC run and the other two runs, with the latter exhibiting distributions more closely aligned with the target.

6.5.4 Results of problem instance j120t6_7_8

Figures 6.18, 6.19, and 6.20 present the outcomes of the manual validation for problem instance j120t6_7_8.

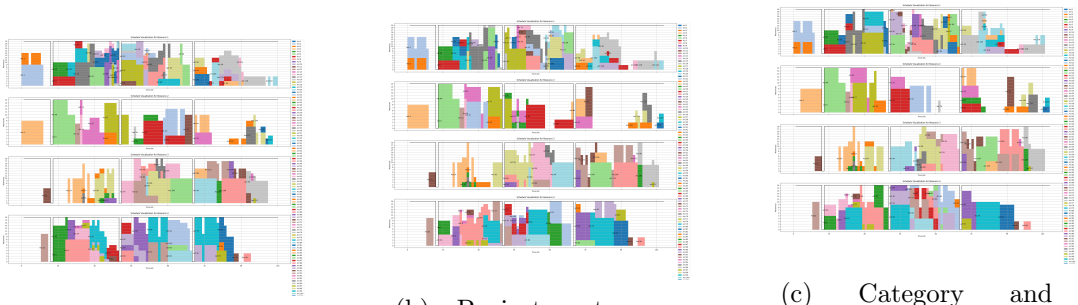


(a) No additional constraints

(b) Project category constrained

(c) Category and set start/end time constrained

Figure 6.12: Pareto fronts for problem instance j120t1_2_3

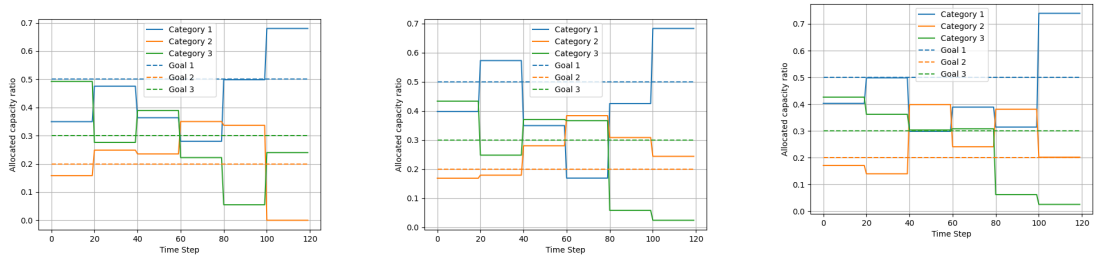


(a) No additional constraints (enlarged A.4)

(b) Project category constrained (enlarged A.5)

(c) Category and set start/end time constrained (enlarged A.6)

Figure 6.13: Schedule with capacity usage for problem instance j120t1_2_3



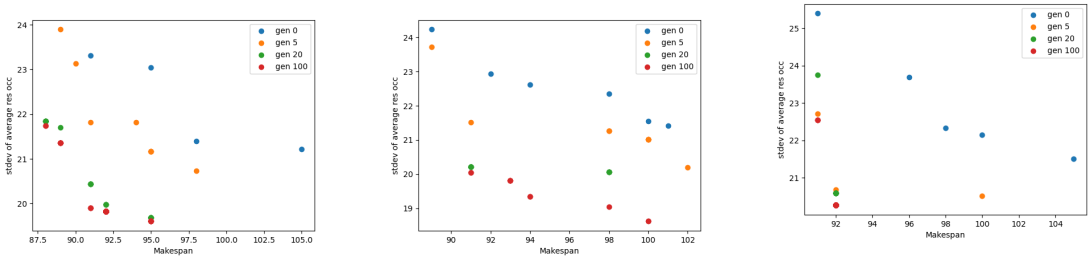
(a) No additional constraints

(b) Project category constrained

(c) Category and set start/end time constrained

Figure 6.14: Capacity distribution on project category for problem instance j120t1_2_3

The Pareto fronts shown in Figure 6.18 again show differences among the three runs. However, this time the NAC run's Pareto front does not scores better on objectives compared to the PCC run. Nevertheless, the NAC run's Pareto front does contain more options. The PCSEC run's Pareto front is also interesting, showing significantly worse scores on the makespan objective. Next to that, some solutions from generations 0 and 5 do not get dominated by solutions from generation 20, similar to the PCC run of problem instance

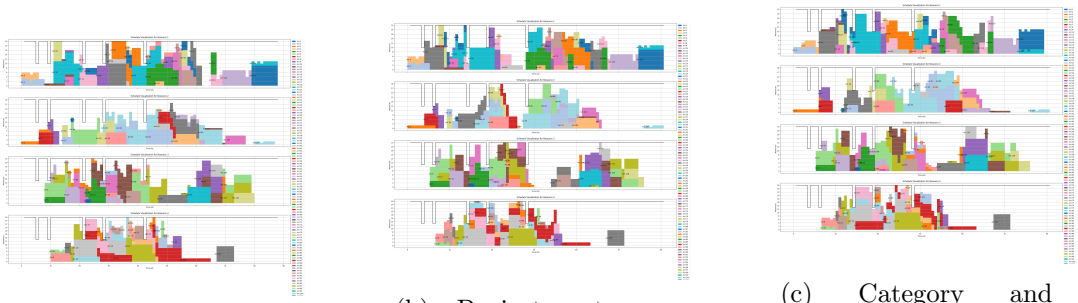


(a) No additional constraints

(b) Project category constrained

(c) Category and set start/end time constrained

Figure 6.15: Pareto fronts for problem instance j120t4_5_6

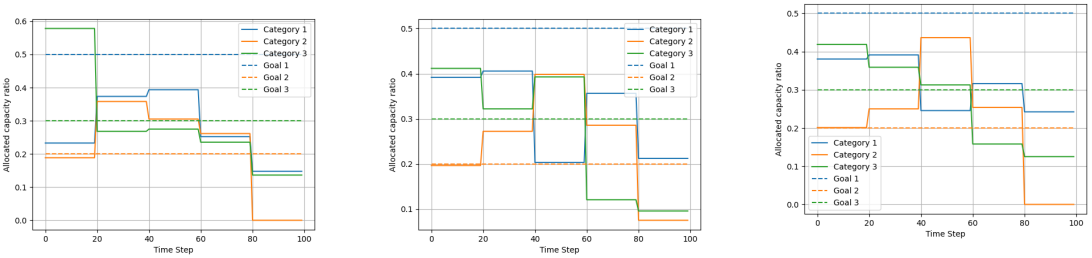


(a) No additional constraints (enlarged A.7)

(b) Project category constrained (enlarged A.8)

(c) Category and set start/end time constrained (enlarged A.9)

Figure 6.16: Schedule with capacity usage for problem instance j120t4_5_6



(a) No additional constraints

(b) Project category constrained

(c) Category and set start/end time constrained

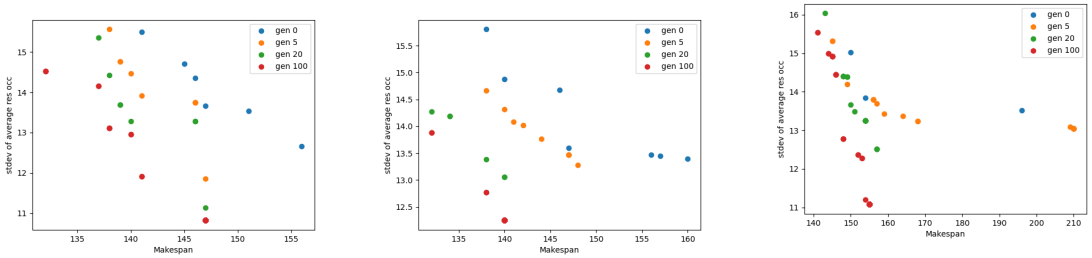
Figure 6.17: Capacity distribution on project category for problem instance j120t4_5_6

j120t4_5_6. Lastly, the run has some outliers on the makespan objective. This could be caused by a violation of the start/end time constraint.

Figure 6.19 illustrates the generated schedules. In this instance, which comes from the j120t6 dataset, variation in capacity requests occurs much more often. The start and end time constraints for this instance are that Project 4 begins at timeslot 3, and Project 19 must conclude before timeslot 20. In the NAC run, Project 19, depicted in light blue,

starts at timeslot 16 in the first resource schedule, while Project 4, shown in light orange, starts in the initial timeslot. In the PCSEC run, Project 4 is shifted to the right, starting at timeslot 3 as required, and Project 19 ends at timeslot 20, thereby fully complying with the constraints.

Lastly, the capacity distribution plots in Figure 6.20 again show a difference between the NAC run and the other two runs. The PCC and PCSEC runs show distributions that are closer to the targets.

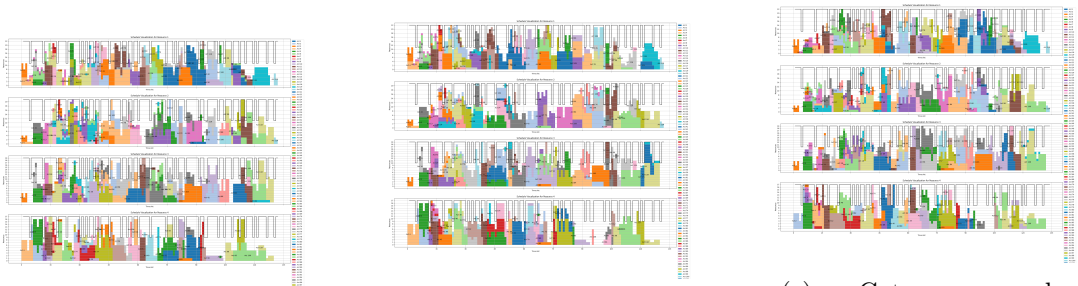


(a) No additional constraints

(b) Project category constrained

(c) Category and set start/end time constrained

Figure 6.18: Pareto fronts for problem instance j120t6_7_8

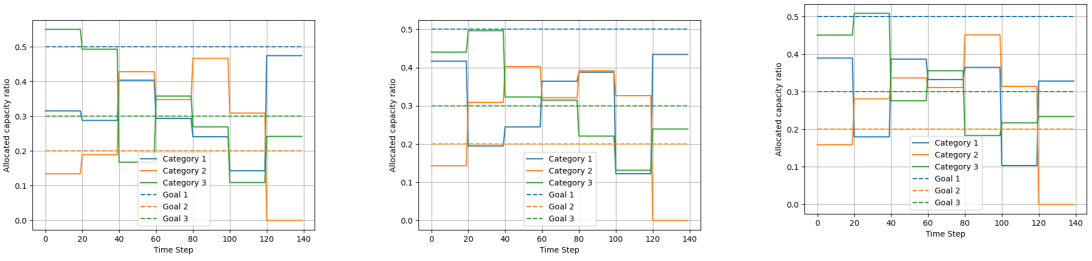


(a) No additional constraints (enlarged A.10)

(b) Project category constrained (enlarged A.11)

(c) Category and set start/end time constrained (enlarged A.12)

Figure 6.19: Schedule with capacity usage for problem instance j120t6_7_8



(a) No additional constraints

(b) Project category constrained

(c) Category and set start/end time constrained

Figure 6.20: Capacity distribution on project category for problem instance j120t6_7_8

6.5.5 Summary of findings from manual runs

The manual runs across different problem instances reveal several consistent findings regarding the algorithm's performance under varying constraints. The primary patterns observed are as follows:

Pareto front analysis

Across all problem instances, the NAC runs consistently exhibited the best performance in the Pareto fronts, particularly in the makespan objective. The NAC runs frequently populated the Pareto front with a higher number of solutions and demonstrated lower overall objective values compared to the PCC and PCSEC runs. While the PCC runs did not perform as well overall as the NAC runs, they achieved similar extreme objective values (those closest to the axes). The PCSEC runs generally exhibited the poorest performance across both objectives, with higher makespans and less favourable resource utilization smoothness. The additional start/end time constraints appear to introduce significant challenges, resulting in fewer solutions in the final Pareto front and higher objective values.

Schedule analysis

The schedule plots for each problem instance reveal a consistent pattern where the NAC run deviated from the similar PCC and PCSEC runs, and the introduced constraints were adhered to as much as possible. This was especially evident in the PCSEC runs, which adapted the schedules to meet start/end time constraints, often leading to delays in project start times. In cases of limited capacity, extensions beyond desired end times appeared, incurring penalties that were reflected in the higher objective values observed in the Pareto fronts.

Capacity distribution analysis

The capacity distribution plots indicate that the PCC and PCSEC runs consistently achieved distributions closer to the target allocations compared to the NAC runs. This alignment underscores the effectiveness of incorporating constraints to manage capacity allocations more effectively across project categories. The PCC and PCSEC runs showed only minor differences in their capacity distributions, highlighting the small influence of the start/end time constraints on the project category constraints.

Chapter 7

Conclusions

7.1 Main contributions

This thesis makes several key contributions to the field of PPM and optimization using GAs.

- Firstly, the thesis demonstrated that project scheduling techniques can be effectively applied to portfolio scheduling. This was achieved by adapting the RCPSP/t to create the new MORCPSP/t-SE to fit portfolio requirements.
- Moreover, a formalization of the MORCPSP/t-SE is provided.
- In addition, a method to consider strategic goals within the problem framework was integrated, employing multiple constraint-handling techniques to address these goals and constraints.
- To solve the newly defined problem, the existing NSGA-II was adapted, incorporating new crossover and mutation operators, a duplicate solution mitigator, and an improved main loop.
- Next, the thesis addressed the limited research on HPO for GAs in scheduling problems by employing two techniques for HPO: BO and PSO.
- Ultimately, this research provides a validated method for organizations to find an optimized schedule for their complex portfolios with many constraints and strategic objectives.

7.2 Genetic algorithm for project scheduling

RQ1: How can a genetic algorithm solve the multi-objective Resource-Constrained Project Scheduling Problem with set start/end time constraints and capacity allocation to project categories constraints?

RQ1.1: How can a project portfolio schedule be structured and encoded for use within a genetic algorithm framework?

To adapt established techniques originally designed for project scheduling to fit project portfolio scheduling, portfolios were analogously mapped to projects (Section 2.1). This involved aligning projects with portfolios based on shared constraints and structures.

In the proposed GA, a project portfolio schedule is represented as a precedence AL (Section 5.1.3). This method is widely used in project scheduling problems due to its effectiveness and ease of use. The AL representation proved effective, facilitating a straightforward mechanism for mutation and crossover. Due to the design of the AL, projects could be swapped for mutation and the crossover operation between two projects always produces valid children that respect the precedence constraints. This ensured that the genetic operations maintained the feasibility of the solutions throughout the evolutionary process.

RQ1.2: How can set start/end time constraints be formalized and included in the GA?

Set start/end time constraints were formalized as two sets containing tuples that define a project and its start or end timeslot. Together with a multi-objective approach and the RCPSP/t defined by Hartmann [28] a new RCPSP variant was created and named the multi-objective RCPSP/t with additional start/end time constraints (MORCPSP/t-SE). The set start/end time constraints were handled by the dynamic penalty constraint handling method. The penalty worsened the objectives of a solution that violated the constraints. A penalty was chosen to still allow infeasible solutions in the population since the constraints could lead to no possible feasible schedules. Figure 6.10c shows such an example where perfect adherence to the constraints was not possible due to capacity and precedence constraints. Figure 6.16c illustrates an example of a feasible schedule with set start/end time constraints.

RQ1.3: How can additional capacity allocation to project categories constraints be included in the GA and be solved?

Capacity allocation to project categories constraints specifies the proportion of total capacity to be allocated to various project categories within a defined time horizon. To manage these constraints, two techniques were employed: DCDP and treating violations as an additional objective. The violation penalty was determined by calculating the Euclidean distance between the actual capacity distribution percentages and the target percentages. If this distance exceeded a predefined threshold, a violation penalty was applied based on the specific handling method used. The results from the example instances, illustrated in the figures in Section 6.2, indicate minimal differences between the two handling methods. The average HV of the runs for both techniques was nearly identical. However, the DCDP method exhibited a higher standard deviation. Next to that, the manual runs in Section 6.5 clearly showed the effectiveness of the DCDP constraint handling method.

RQ1.4: What existing methods or frameworks are suitable for solving multi-objective scheduling problems, and how can they be adapted or extended to solve the problem at hand?

In portfolio management, developing a schedule that optimizes multiple objectives is critical for achieving strategic goals. Over the years, various multi-objective optimization techniques have been applied, including PSO, SA, and GAs. A common element among multi-objective optimization methods is the Pareto front, which identifies multiple optimal solutions across different objectives.

This research applied the NSGA-II algorithm to tackle the MORCPSP/t-SE. The results demonstrated that NSGA-II could achieve comparable performance to GAs specifically designed for single-objective optimization. To address the additional constraints, the NSGA-II algorithm was enhanced with constraint-handling techniques, caching mechanisms, and tailored crossover and mutation strategies specific to scheduling problems. These adaptations enabled NSGA-II to manage the complexities of multi-objective optimization in project scheduling effectively.

7.3 Metrics for assessing quality

RQ2: What criteria and performance metrics can be used to assess the quality and effectiveness of the GA and the generated project portfolio scenarios?

RQ2.1: How can optimal values for hyperparameters of the GA be determined?

In this research, two methods for determining optimal hyperparameter values for the algorithm were employed. First, an assessment of the individual impact of the parameters was conducted by running the algorithm on selected problem instances, varying one hyperparameter at a time while keeping others constant. The HV indicator and standard deviation thereof were used as primary metrics for evaluation.

The results, illustrated in Figures 6.2, 6.3 and 6.4, indicate that a mutation rate of 0.05 or 0.1 performs best, crossover rates had little effect on the HV mean but rates of 0.5 and 0.7 scored the lowest standard deviation and two crossover points had the best outcomes on both HV and standard deviation.

Second, to find an optimal configuration of hyperparameters with parameters that influence each other, two HPO methods, BO and PSO, were applied. These methods were utilized to handle the stochastic nature of the algorithm. Figure 6.5 shows the results of the HPO runs of both methods. The results indicate that there is not a single optimal configuration and that the optimization is heavily influenced by the randomness in the algorithm.

RQ2.2: How can a generated solution be validated without knowing the actual optimal solution?

Validating solutions to NP-hard problems is inherently challenging due to the vast number of possible solutions and the computational expense of evaluating each one. This research addressed this challenge by validating the proposed algorithm against a benchmark dataset. Each problem instance in the dataset was executed up to 10 times, or until the benchmark solution was either found or improved upon.

The results demonstrated that for the j30t dataset, the proposed algorithm found a solution with the same or an improved objective value, compared to the benchmark solution, on the first attempt for 80% of the problem instances (see Table 6.2). Detailed results are illustrated in Figure 6.6. Conversely, the j120t problem instances proved to be significantly more challenging as shown in the plots in Figure 6.7. While the algorithm did not always find the benchmark solution for these harder instances, the average error rate was a mere 0.066. This indicates that the solutions generated by the algorithm were, on average, very close to the benchmark. Furthermore, in some instances, the algorithm even managed to find improved solutions.

In addition to the benchmark validation, manual validation was conducted to assess the

algorithm's performance under multiple objectives and constraints. The figures in Section 6.5 present Pareto fronts, schedules with capacity usage, and capacity distribution across project categories. During these manual runs, the algorithm consistently adhered to the predefined constraints wherever possible and minimized violations where adherence was not feasible. This was validated by a thorough analysis of various visualizations of the generated schedules.

Overall, the combination of benchmark comparisons and manual validation provides robust evidence of the algorithm's efficacy in generating high-quality solutions, even in the absence of a known optimal solution.

RQ2.3: How will validation data with optimal solutions be gathered or created?

The validation data for the algorithm was gathered using problem instances provided by Hartmann [28], which are adaptations of the standard RCPSP benchmark set. These instances included timed resource capacities and requests and provided a robust basis for evaluating the algorithm's performance across various levels of complexity and resource constraints. This dataset allowed for both benchmark comparisons and manual validation, ensuring a comprehensive assessment of the algorithm's ability to find near-optimal solutions and adhere to additional constraints.

7.4 Future work

This thesis provides a step towards optimizing PPM using the NSGA-II. Moving forward, several areas can be expanded or explored to enhance the findings of this thesis. The first area for future work involves performing case studies for the proposed algorithm. While the current study provides multiple methods for validation, the methods were conducted with project scheduling data. Portfolio scheduling might contain different challenges that were not regarded. For example, quantifying the performance of portfolio planning is significantly more complex due to the need to balance multiple objectives and constraints. Additionally, factors such as strategic alignment and change management, add layers of complexity that are not typically addressed in project-level scheduling. Therefore, including case studies from diverse industries with varying project characteristics would enhance the proposed approach's generalizability and validate its effectiveness across different domains.

To further address the limitations of the use of the project scheduling dataset, future research should enhance it or create a new validation dataset to incorporate additional metrics tailored to PPM. Such metrics could include costs, expected revenue, risk, etc., to create more realistic objectives and constraints. Moreover, it can facilitate comparisons with other approaches. Conducting comparative analysis with other evolutionary techniques like PSO and SA would offer insights into the strengths and weaknesses of each approach.

Before comparing the algorithm to other approaches, the algorithm should be written in a language like C or C++. This enhancement would facilitate comparative studies against existing benchmarks and alternative optimization techniques. Such comparisons are important when evaluating performance in speed and scalability, which are essential factors for practical application.

Finally, to improve the proposed algorithm, problem-specific crossover and mutation operations that enhance the solution quality should be developed. Such customized op-

erators could leverage domain-specific knowledge to improve the search process of the algorithm. Similarly, the algorithm could be enhanced with a local search function to improve the solutions.

Bibliography

[1] Darya Abbasi, Maryam Ashrafi, and Seyed Hassan Ghodsypour. A multi objective-bsc model for new product development project portfolio selection. Expert Systems with Applications, 162:113757, 2020. URL: https://www.sciencedirect.com/science/article/pii/S0957417420305819, doi:10.1016/j.eswa.2020.113757.

[2] J. Alcaraz and C. Maroto. A Robust Genetic Algorithm for Resource Allocation in Project Scheduling. ERS Monograph, 2016(9781849840682):16–34, 2001. doi:10.1183/2312508X.10004715.

[3] J. Alcaraz, C. Maroto, and R. Ruiz. Solving the Multi-Mode Resource-Constrained Project Scheduling Problem with genetic algorithms. Journal of the Operational Research Society, 54(6):614–626, 2003. doi:10.1057/palgrave.jors.2601563.

[4] Mohammed Ghaith Altarabichi, Sławomir Nowaczyk, Sepideh Pashami, Peyman Sheikholharam Mashhadi, and Julia Handl. Rolling the dice for better deep learning performance: A study of randomness techniques in deep neural networks. Information Sciences, 667:120500, 2024. URL: https://www.sciencedirect.com/science/article/pii/S0020025524004134, doi:10.1016/j.ins.2024.120500.

[5] Maria Angelova and Tania Pencheva. Tuning genetic algorithm parameters to improve convergence time. International Journal of Chemical Engineering, 2011, 01 2011. doi:10.1155/2011/646917.

[6] Norm P Archer and Fereidoun Ghasemzadeh. An integrated framework for project portfolio selection. International Journal of Project Management, 17(4):207–216, 1999.

[7] Shahriar Asta, Daniel Karapetyan, Ahmed Kheiri, Ender Özcan, and Andrew J. Parkes. Combining Monte-Carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. Information Sciences, 373:476–498, 2016. URL: http://dx.doi.org/10.1016/j.ins.2016.09.010, arXiv:1511.04387, doi:10.1016/j.ins.2016.09.010.

[8] Zied Bahroun, Moayad Tanash, Rami As'ad, and Mohamad Alnajar. Artificial Intelligence Applications in Project Scheduling: a Systematic Review, Bibliometric Analysis, and Prospects for Future Research. Management Systems in Production Engineering, 31(2):144–161, 2023. doi:10.2478/mspe-2023-0017.

[9] Bodil Stilling Blichfeldt and Pernille Eskerod. Project portfolio management - There's more to it than what management enacts. International Journal of Project Management, 26(4):357–365, 2008. doi:10.1016/j.ijproman.2007.06.004.

[10] K. Bouleimen and H. Lecocq. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version.

European Journal of Operational Research, 149(2):268–281, 2003. doi:10.1016/S0377-2217(02)00761-0.

[11] Ana F. Carazo, Trinidad Gómez, Julián Molina, Alfredo G. Hernández-Díaz, Flor M. Guerrero, and Rafael Caballero. Solving a comprehensive model for multiobjective project portfolio selection. Computers Operations Research, 37(4):630–639, 2010. URL: https://www.sciencedirect.com/science/article/pii/S0305054809001646, doi:10.1016/j.cor.2009.06.012.

[12] Rong Chen, Changyong Liang, Dongxiao Gu, and Huimin Zhao. A competence-time-quality scheduling model of multi-skilled staff for it project portfolio. Computers Industrial Engineering, 139:106183, 2020. URL: https://www.sciencedirect.com/science/article/pii/S0360835219306527, doi:10.1016/j.cie.2019.106183.

[13] Navee Chiadamrong and Pisacha Suthamanondh. Fuzzy multi-objective chance-constrained portfolio optimization under uncertainty considering investment return, investment risk, and sustainability. International Journal of Knowledge and Systems Science, 13:1–39, 2022. doi:10.4018/IJKSS.302660.

[14] Karim Chichakly. Pareto Front visualisation, 1 2018. URL: https://blog.iseesystems.com/modeling-tips/multiobjective-optimization/.

[15] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2):182–197, 2002. doi:10.1109/4235.996017.

[16] Ashraf Elazouni and Mohammad Abido. Multiobjective evolutionary finance-based scheduling: Individual projects within a portfolio. Automation in Construction, 20(7):755–766, 2011.

[17] Suvi Elonen and Karlos A. Artto. Problems in managing internal development projects in multi-project environments. International Journal of Project Management, 21(6):395–402, 2003. doi:10.1016/S0263-7863(02)00097-2.

[18] Mats Engwall and Anna Jerbrant. The resource allocation syndrome: The prime challenge of multi-project management? International Journal of Project Management, 21(6):403–409, 2003. URL: http://dx.doi.org/10.1016/S0263-7863(02)00113-8, doi:10.1016/S0263-7863(02)00113-8.

[19] Martin Josef Geiger. Iterated Variable Neighborhood Search for the resource constrained multi-mode multi-project scheduling problem. 2013. URL: http://arxiv.org/abs/1310.0602, arXiv:1310.0602.

[20] Parviz Ghoddousi, Ehsan Eshtehardian, Shirin Jooybanpour, and Ashtad Javanmardi. Multi-mode resource-constrained discrete time-cost-resource optimization in project scheduling using non-dominated sorting genetic algorithm. Automation in Construction, 30:216–227, 2013. URL: http://dx.doi.org/10.1016/j.autcon.2012.11.014, doi:10.1016/j.autcon.2012.11.014.

[21] MR Ghodoosi, Ramin Maftahi, and Vahidreza Yousefi. Proposing a hybrid approach to predict, schedule and select the most robust project portfolio under uncertainty. European Online Journal of Natural and Social Sciences, 5(4):pp–1099, 2016.

[22] Everton Gomede and Rodolfo Miranda de Barros. A multicriteria approach to project portfolio selection: Using multiobjective optimization and analytic hierarchy process. In 2014 9th Iberian Conference on Information Systems and Technologies (CISTI), pages 1–7, 2014. doi:10.1109/CISTI.2014.6876928.

[23] Helton Cristiano Gomes, Francisco de Assis das Neves, and Marcone Jamilson Freitas Souza. Multi-objective metaheuristic algorithms for the resource-constrained project scheduling problem with precedence relations. Computers Operations Research, 44:92–104, 2014. URL: https://www.sciencedirect.com/science/article/pii/S0305054813003158, doi:10.1016/j.cor.2013.11.002.

[24] Rel Guzman, Rafael Oliveira, and Fabio Ramos. Heteroscedastic bayesian optimisation for stochastic model predictive control. IEEE Robotics and Automation Letters, 6(1):56–63, 2021. doi:10.1109/LRA.2020.3028830.

[25] Sönke Hartmann. A competitive genetic algorithm for resource-constrained project scheduling problem. 2005 International Conference on Machine Learning and Cybernetics, ICMLC 2005, pages 2945–2949, 1998. doi:10.1109/icmlc.2005.1527446.

[26] Sönke Hartmann. Project Scheduling with Multiple Modes: A Genetic Algorithm. Annals of Operations Research, 102(1-4):111–135, 2001. doi:10.1023/A:1010902015091.

[27] Sönke Hartmann. A self-adapting genetic algorithm for project scheduling under resource constraints. Naval Research Logistics, 49(5):433–448, 2002. doi:10.1002/nav.10029.

[28] Sönke Hartmann. Project scheduling with resource capacities and requests varying with time: A case study. Flexible Services and Manufacturing Journal, 25(1-2):74–93, 2013. doi:10.1007/s10696-012-9141-8.

[29] Sönke Hartmann. Time-Varying Resource Requirements and Capacities, pages 163–176. Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-05443-8_8.

[30] Sönke Hartmann and Dirk Briskorn. An updated survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research, 297(1):1–14, 2022. doi:10.1016/j.ejor.2021.05.004.

[31] Sönke Hartmann and Rainer Kolisch. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. European Journal of Operational Research, 127(2):394–407, 2000. doi:10.1016/S0377-2217(99)00485-3.

[32] Meriem Hemici and Djaafar Zouache. A multi-population evolutionary algorithm for multi-objective constrained portfolio optimization problem. Artificial Intelligence Review, 56:1–42, 09 2023. doi:10.1007/s10462-023-10604-2.

[33] Jared G. Hobbie, Amir H. Gandomi, and Iman Rahimi. A comparison of constraint handling techniques on nsga-ii. Archives of Computational Methods in Engineering, 28:3475–3490, 2021. doi:10.1007/s11831-020-09525-y.

[34] Hamed Kazemipoor, Reza Tavakkoli-Moghaddam, Parisa Shahnazari-Shahrezaei, and Amir Azaron. A differential evolution algorithm to solve multi-skilled project portfolio scheduling problems. The International Journal of Advanced Manufacturing Technology, 64, 02 2012. doi:10.1007/s00170-012-4045-z.

[35] Rainer Kolisch and Sönke Hartmann. Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis, pages 147–178. Springer US, Boston, MA, 1999. doi:10.1007/978-1-4615-5533-9_7.

[36] Rainer Kolisch and Sönke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research, 174(1):23–37, 2006. doi:10.1016/j.ejor.2005.01.065.

[37] Rainer Kolisch and Arno Sprecher. Psplib - a project scheduling problem library. Jan 1997. doi:10.1016/s0377-2217(96)00170-1.

[38] Rainer Kolisch, Arno Sprecher, and Andreas Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science, 41(10):1693–1703, 1995. URL: http://www.jstor.org/stable/2632747.

[39] Mathias Kühn, Taiba Zahid, Michael Völker, Zhugen Zhou, and Oliver Rose. Investigation of genetic operators and priority heuristics for simulation based optimization of Multi-Mode Resource Constrained Multi-Project Scheduling Problems (MM-RCMPSP). Proceedings - 30th European Conference on Modelling and Simulation, ECMS 2016, 2(Cd):481–487, 2016. doi:10.7148/2016-0481.

[40] Antonio Lova, Pilar Tormos, Mariamar Cervantes, and Federico Barber. An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. International Journal of Production Economics, 117(2):302–316, 2009. doi:10.1016/j.ijpe.2008.11.002.

[41] Augusto Hayashida Marchinares and Igor Aguilar-Alonso. Project portfolio management studies based on machine learning and critical success factors. Proceedings of 2020 IEEE International Conference on Progress in Informatics and Computing, PIC 2020, pages 369–374, 2020. doi:10.1109/PIC50277.2020.9350787.

[42] Miia Martinsuo. Project portfolio management in practice and in context. International Journal of Project Management, 31(6):794–803, 2013. URL: http://dx.doi.org/10.1016/j.ijproman.2012.10.013, doi:10.1016/j.ijproman.2012.10.013.

[43] Mohsen Mosayebi and Manbir Sodhi. Tuning genetic algorithm parameters using design of experiments. pages 1937–1944, 07 2020. doi:10.1145/3377929.3398136.

[44] Paweł B. Myszkowski, Maciej Laszczyk, and Joanna Lichodij. Efficient selection operators in nsga-ii for solving bi-objective multi-skill resource-constrained project scheduling problem. In 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), pages 83–86, 2017. doi:10.15439/2017F317.

[45] Nestor Raul Ortiz-Pimiento and Francisco Javier Diaz-Serna. The project scheduling problem with non-deterministic activities duration: A literature review. Journal of Industrial Engineering and Management, 11(1):116–134, 2018. doi:10.3926/jiem.2492.

[46] Linet Özdamar. A genetic algorithm approach to a general category project scheduling problem. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 29(1):44–59, 1999. doi:10.1109/5326.740669.

[47] Vassilios Petridis, Spyros Kazarlis, and Anastasios Bakirtzis. Varying fitness functions in genetic algorithm constrained optimization: The cutting stock and unit commitment problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 28(5):629–640, 1998. doi:10.1109/3477.718514.

[48] Iman Rahimi, Amir H. Gandomi, Fang Chen, and Efrén Mezura-Montes. A review on constraint handling techniques for population-based algorithms: from single-objective to multi-objective optimization, 4 2023. doi:10.1007/s11831-022-09859-9.

[49] Ewa Ratajczak-Ropel. Resource-Constrained Project Scheduling, pages 33–67. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-62893-6_4.

[50] Scott J Edgett Robert G. Cooper and Elko J Kleinschmidt. Portfolio Management in New Product Development: Lessons from the Leaders—I. Research-Technology Management, 40(5):16–28, 1997. doi:10.1080/08956308.1997.11671152.

[51] D. W. Ross and P. E. Shaltry. The new pmi standard for portfolio management. In PMI® Global Congress 2006—EMEA, Madrid, Spain, 2006. Project Management Institute.

[52] Xingke Tian and Shengrui Yuan. Genetic algorithm parameters tuning for resource-constrained project scheduling problem. AIP conference proceedings, 1 2018. doi:10.1063/1.5033723.

[53] Túlio A.M. Toffolo, Haroldo G. Santos, Marco A.M. Carvalho, and Janniele A. Soares. An integer programming approach to the multimode resource-constrained multiproject scheduling problem. Journal of Scheduling, 19(3):295–307, 2016. URL: http://dx.doi.org/10.1007/s10951-015-0422-4, doi:10.1007/s10951-015-0422-4.

[54] Vicente Valls, Francisco Ballestín, and Sacramento Quintanilla. A hybrid genetic algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research, 185(2):495–508, 2008. doi:10.1016/j.ejor.2006.12.033.

[55] Shanu Verma, Millie Pant, and Vaclav Snasel. A comprehensive review on nsga-ii for multi-objective combinatorial optimization problems. IEEE Access, 9:57757–57791, 2021. doi:10.1109/ACCESS.2021.3070634.

[56] Gustavo Barbi Vieira, Hévilla Souza Oliveira, Jônatas Araújo de Almeida, and Mischel Carmen Neyra Belderrain. Project portfolio selection considering interdependencies: A review of terminology and approaches. Project Leadership and Society, 5:100115, 2024. URL: https://www.sciencedirect.com/science/article/pii/S2666721523000364, doi:10.1016/j.plas.2023.100115.

[57] Félix Villafáñez, David Poza, Adolfo López-Paredes, Javier Pajares, and Ricardo del Olmo. A generic heuristic for multi-project scheduling problems with global and local resource constraints (RCMPSP). Soft Comput., 23(10):3465–3479, January 2018.

[58] Stefan Volkwein. Solving a Convex Multiobjective Optimization Problem, 2021. URL: https://www.mathematik.uni-konstanz.de/en/volkwein/python/oppy/Notebooks_Web/multOpt_Parabolas.html.

[59] Hong Wang, Dan Lin, and Min-Qiang Li. A competitive genetic algorithm for resource-constrained project scheduling problem. In 2005 International Conference on Machine Learning and Cybernetics, volume 5, pages 2945–2949 Vol. 5, 2005. doi:10.1109/ICMLC.2005.1527446.

[60] Tony Wauters, Joris Kinable, Pieter Smet, Wim Vancroonenburg, Greet Vanden Berghe, and Jannes Verstichel. The Multi-Mode Resource-Constrained Multi-Project Scheduling Problem: The MISTA 2013 challenge. Journal of Scheduling, 19(3):271–283, 2014. URL: http://dx.doi.org/10.1007/s10951-014-0402-0, doi:10.1007/s10951-014-0402-0.

[61] Yunna Wu, Chuanbo Xu, Yiming Ke, Xinying Li, and Lingwenying Li. Portfolio selection of distributed energy generation projects considering uncertainty and project interaction under different enterprise strategic scenarios. Applied Energy, 236:444–464, 2019. URL: https://www.sciencedirect.com/science/article/pii/S0306261918318282, doi:10.1016/j.apenergy.2018.12.009.

[62] Lean Yu, Shouyang Wang, Fenghua Wen, and Kin Keung Lai. Genetic algorithm-based multi-criteria project portfolio selection. Annals of operations research, 197:71–86, 2012.

[63] Xiaoxiong Zhang, Keith W Hipel, and Yuejin Tan. Project portfolio selection and scheduling under a fuzzy environment. Memetic Computing, 11:391–406, 2019.

[64] Yuan Zhou, Hai-Lin Liu, Wenqin Chen, and Jingqian Li. A novel multi-objective evolutionary algorithm solving portfolio problem. Journal of Software, 9, 01 2014. doi:10.4304/jsw.9.1.222-229.

[65] Samaneh Zolfaghari and Seyed Meysam Mousavi. A novel mathematical programming model for multi-mode project portfolio selection and scheduling with flexible resources and due dates under interval-valued fuzzy random uncertainty. Expert Systems with Applications, 182:115207, 2021. doi:10.1016/j.eswa.2021.115207.

[66] Nima Zoraghi, Aria Shahsavar, Babak Abbasi, and Vincent Van Peteghem. Multi-mode resource-constrained project scheduling problem with material ordering under bonus–penalty policies. Top, 25(1):49–79, 2017. doi:10.1007/s11750-016-0415-2.

[67] Önder Halis Bettemir and Rifat Sonmez. Hybrid genetic algorithm with simulated annealing for resource-constrained project scheduling. Journal of Management in Engineering, 31(5):04014082, 2015. URL: https://ascelibrary.org/doi/abs/10.1061/%28ASCE%29ME.1943-5479.0000323,

Appendix A

Schedule visualisations

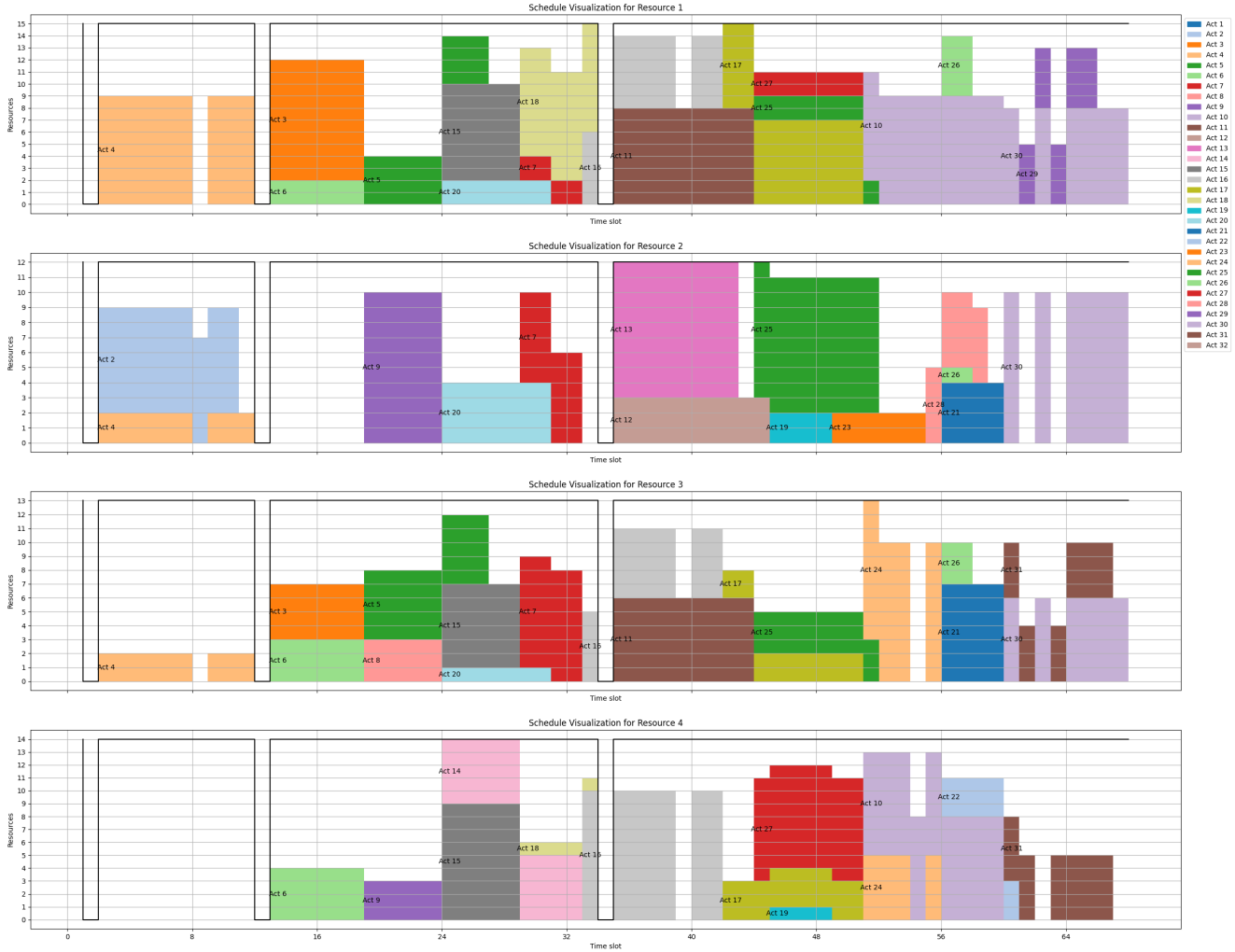


Figure A.1: Schedule visualization for problem instance j30t1_5_4, no constrained

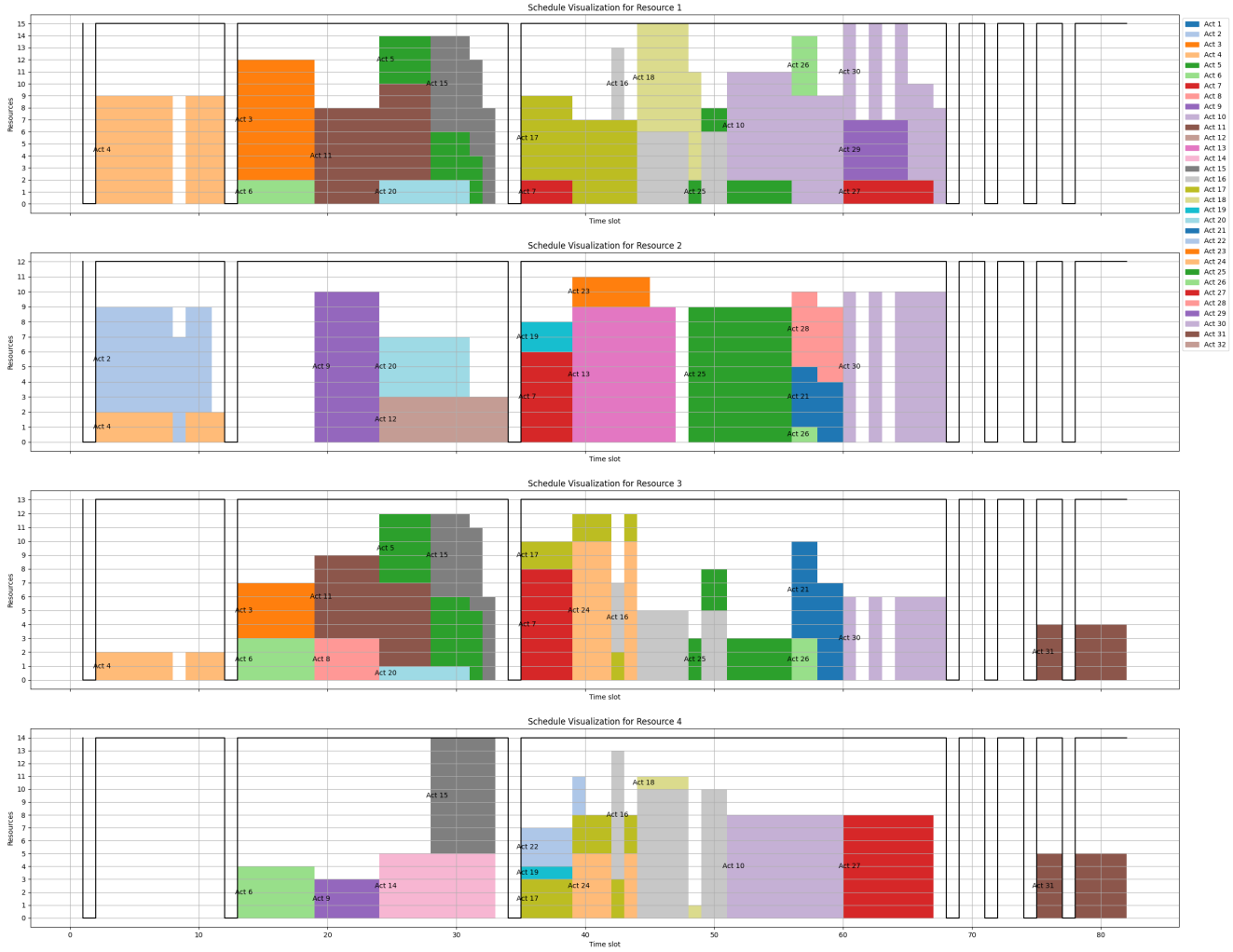


Figure A.2: Schedule visualization for problem instance j30t1_5_4, project category constrained

Figure A.3: Schedule visualization for problem instance j30t1_5_4, start/end time constrained

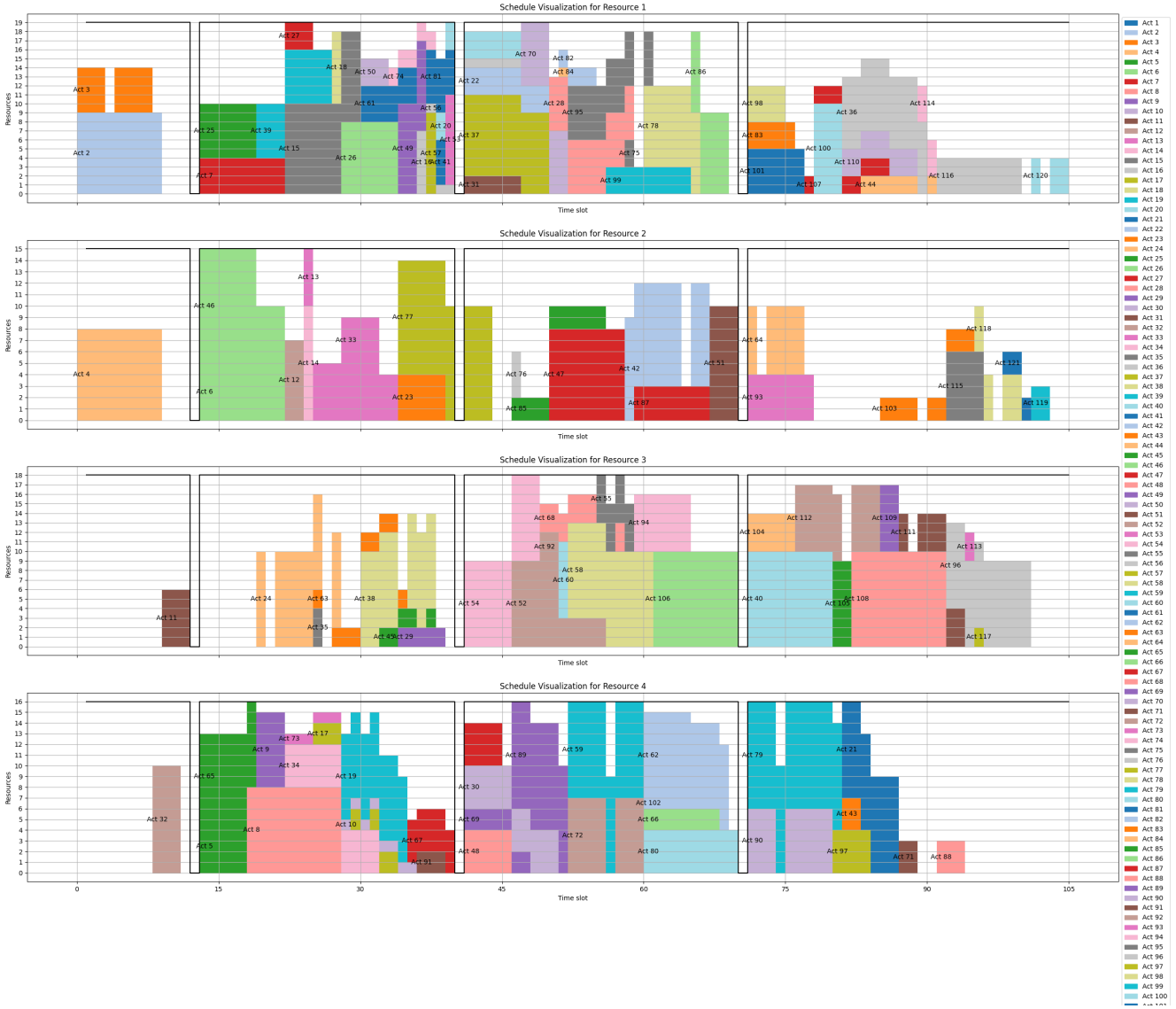


Figure A.4: Schedule visualization for problem instance j120t1_2_3, no constrained

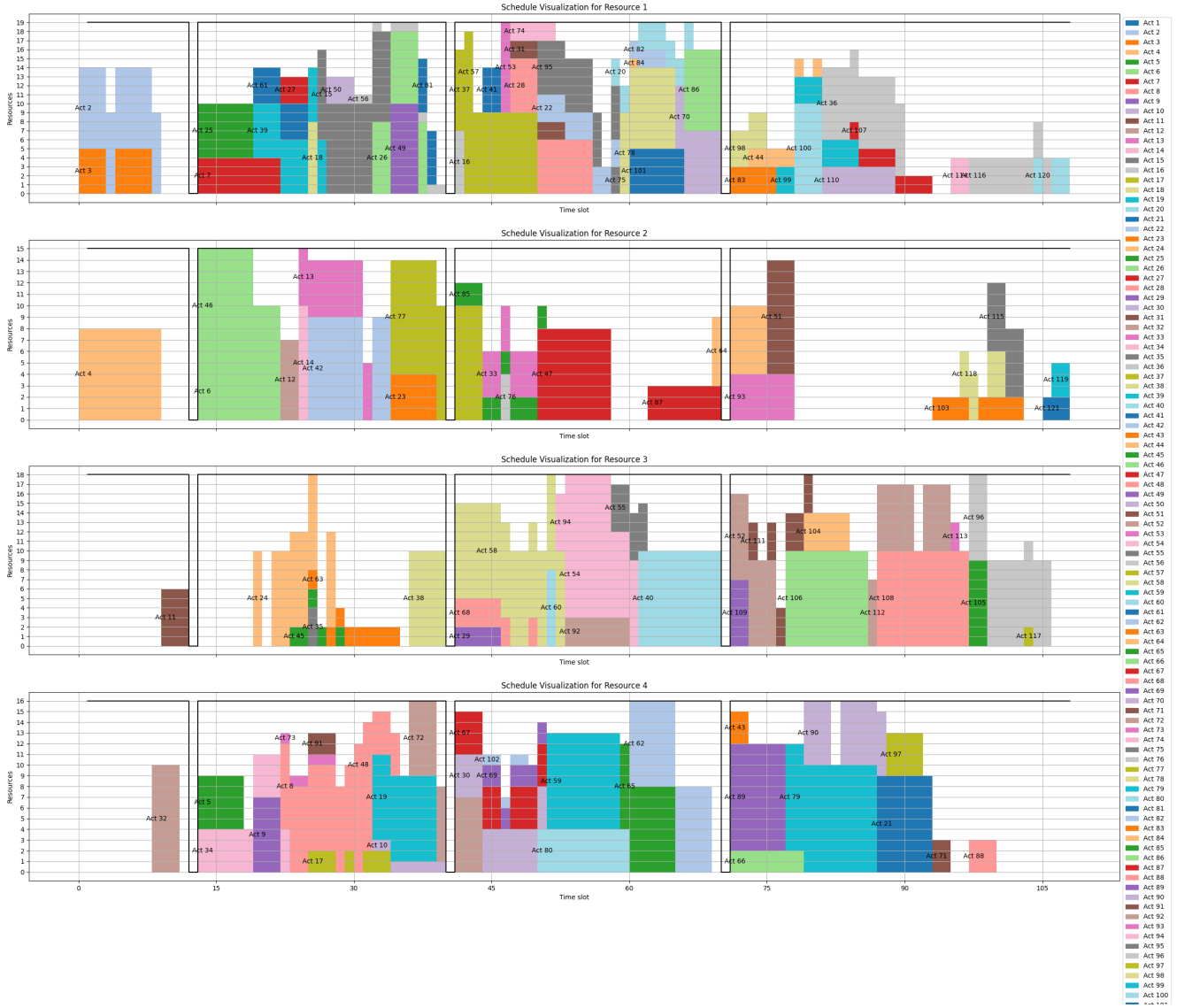


Figure A.5: Schedule visualization for problem instance j120t1_2_3, project category constrained

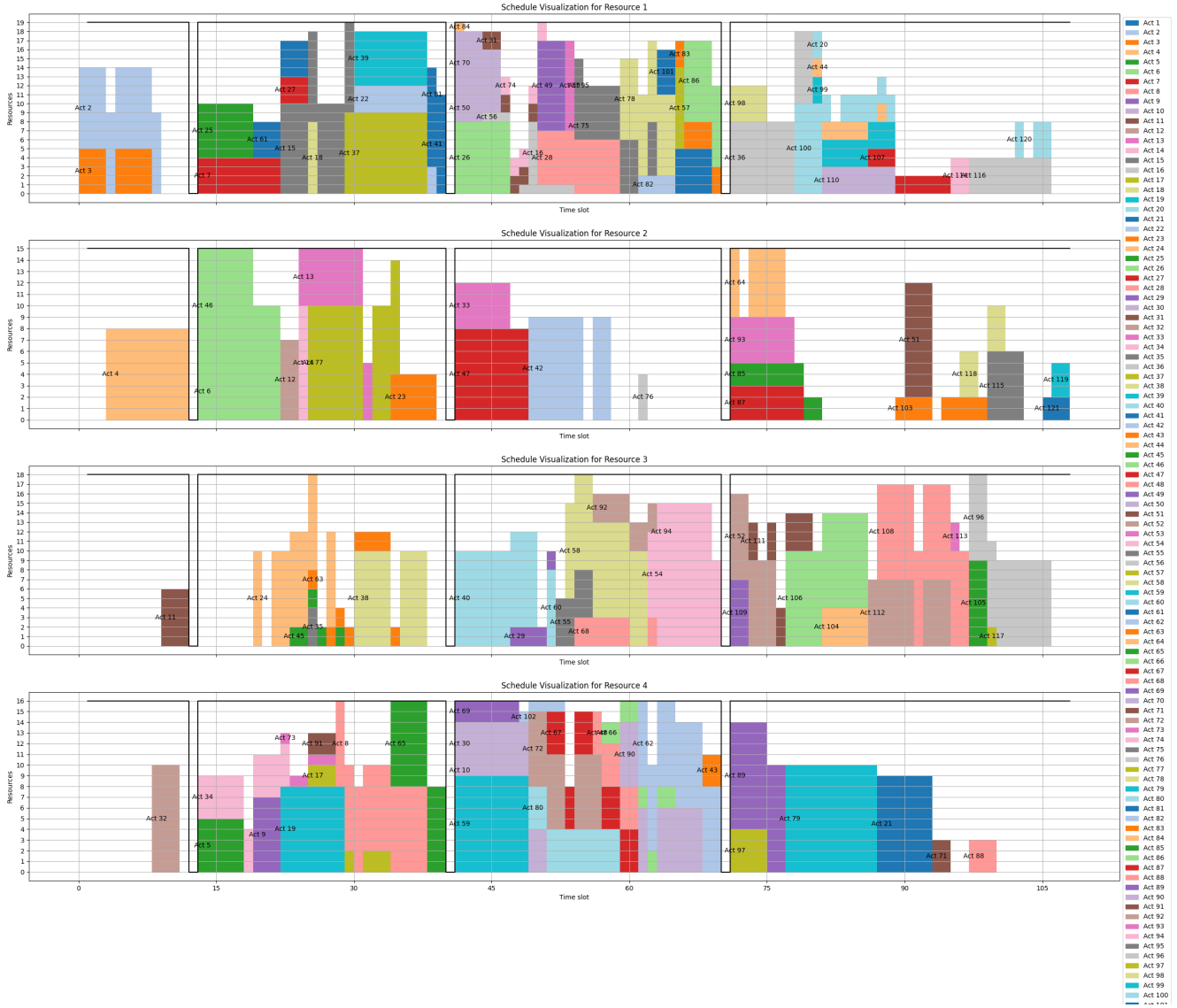


Figure A.6: Schedule visualization for problem instance j120t1_2_3, start/end time constrained

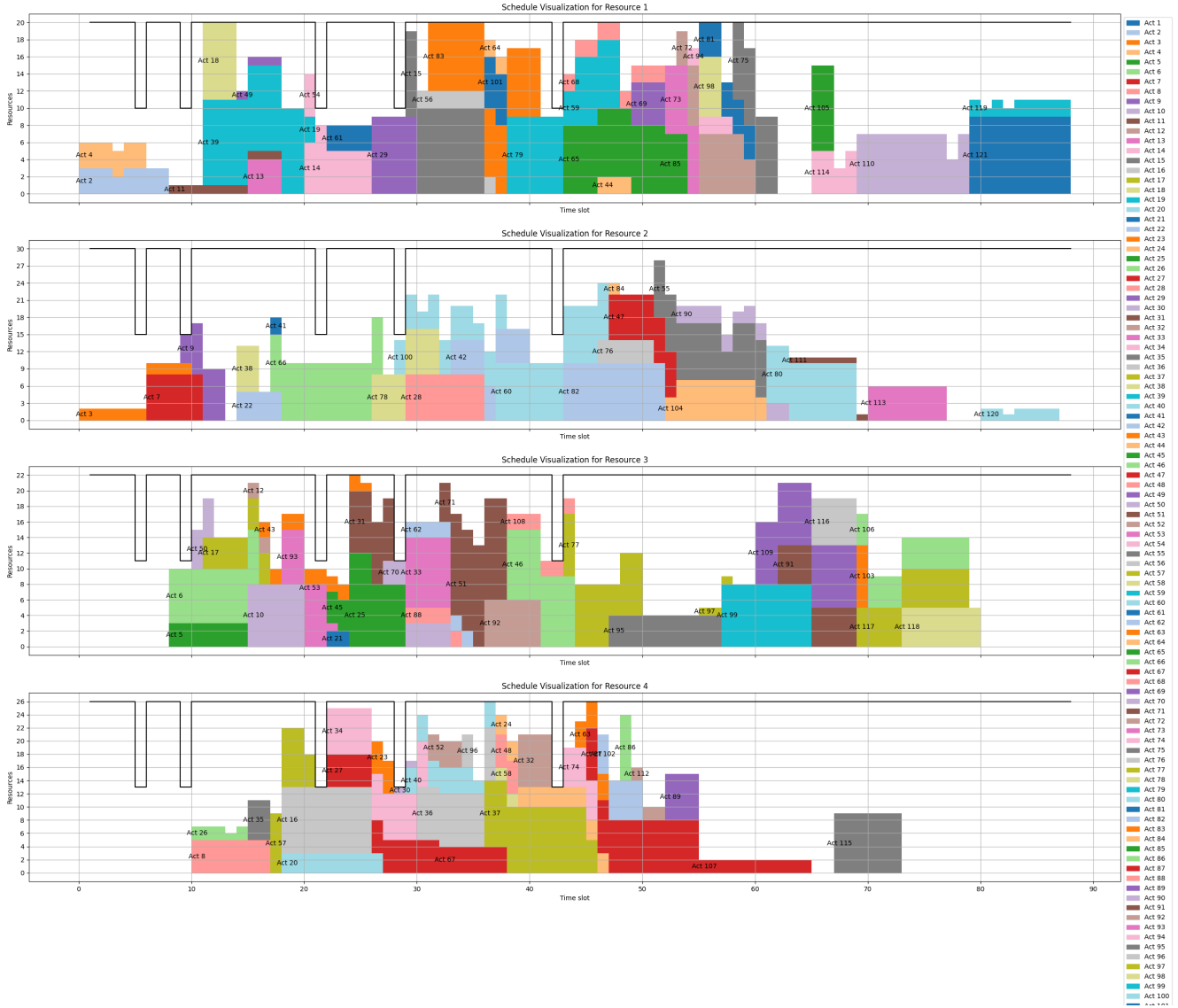


Figure A.7: Schedule visualization for problem instance j120t4_5_6, no constrained

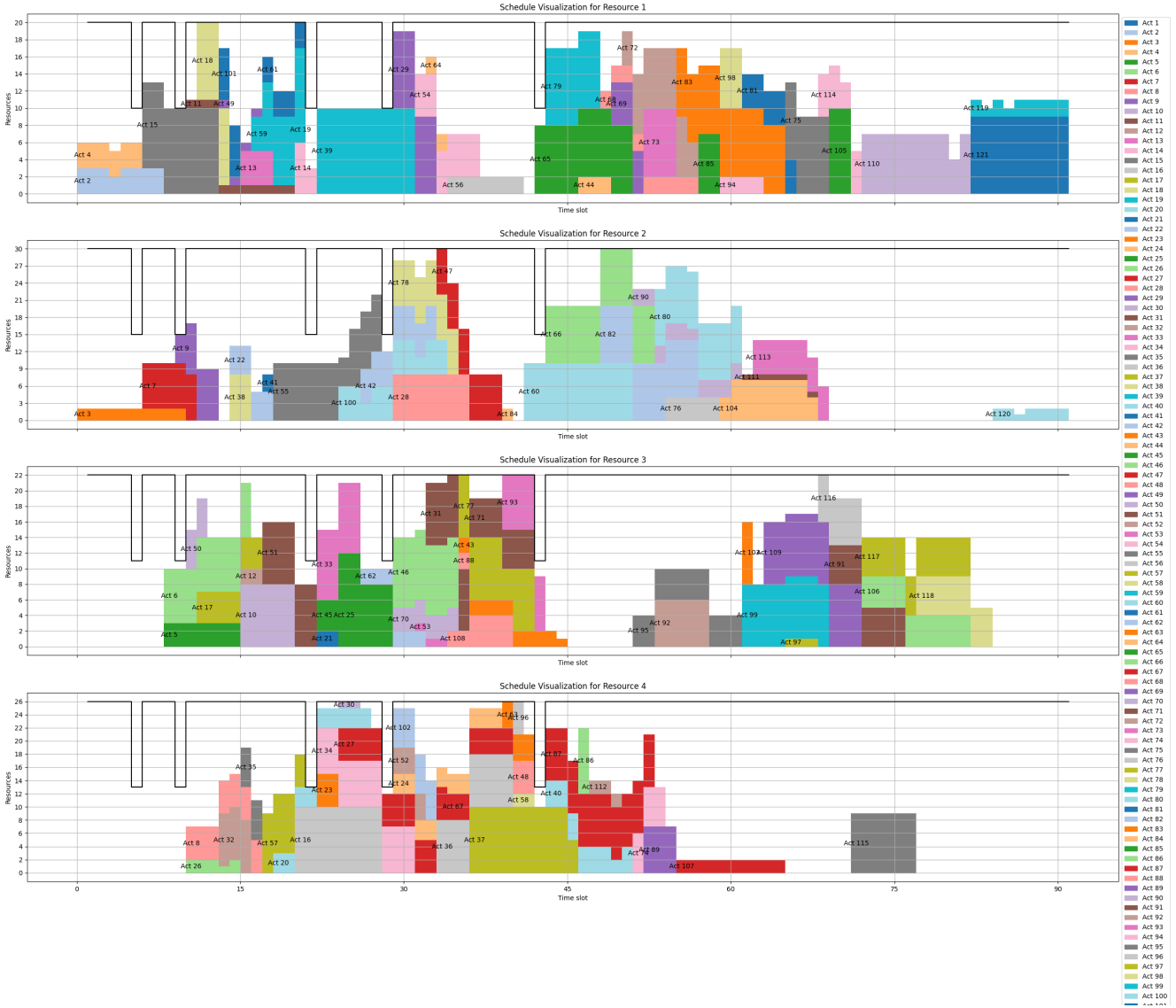


Figure A.8: Schedule visualization for problem instance j120t4_5_6, project category constrained

Figure A.9: Schedule visualization for problem instance j120t4_5_6, start/end time constrained

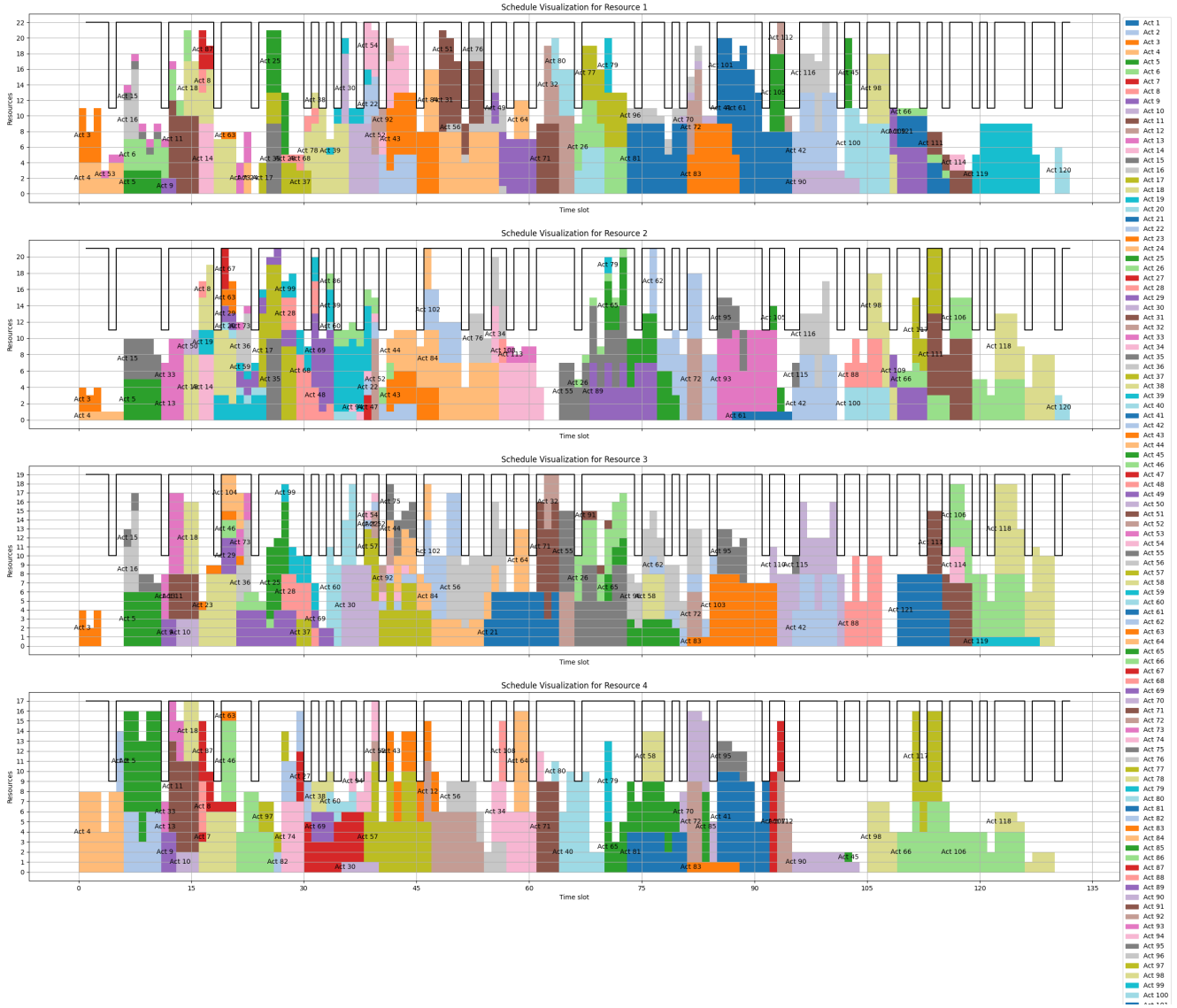


Figure A.10: Schedule visualization for problem instance j120t6_7_8, no constrained

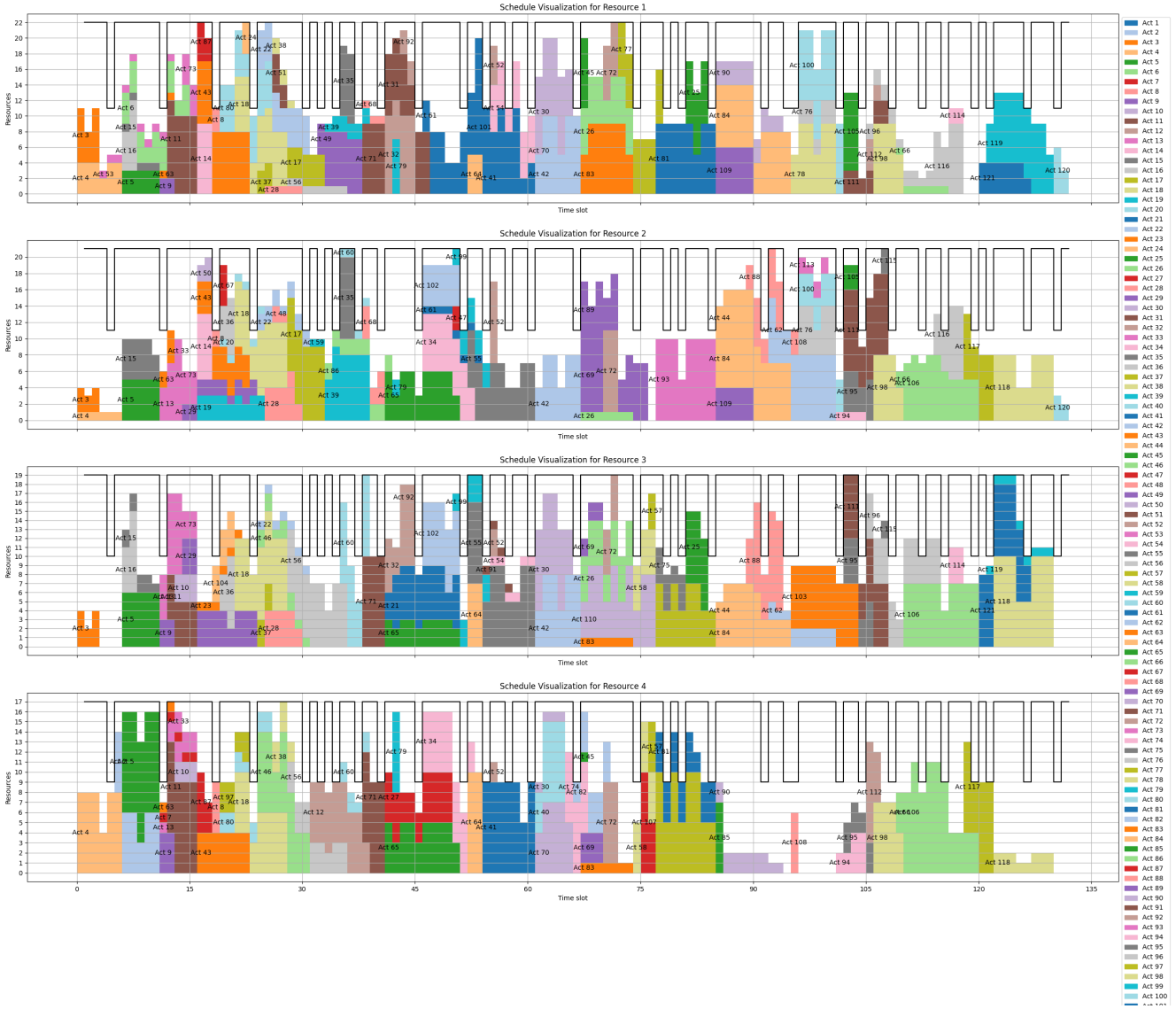


Figure A.11: Schedule visualization for problem instance j120t6_7_8, project category constrained

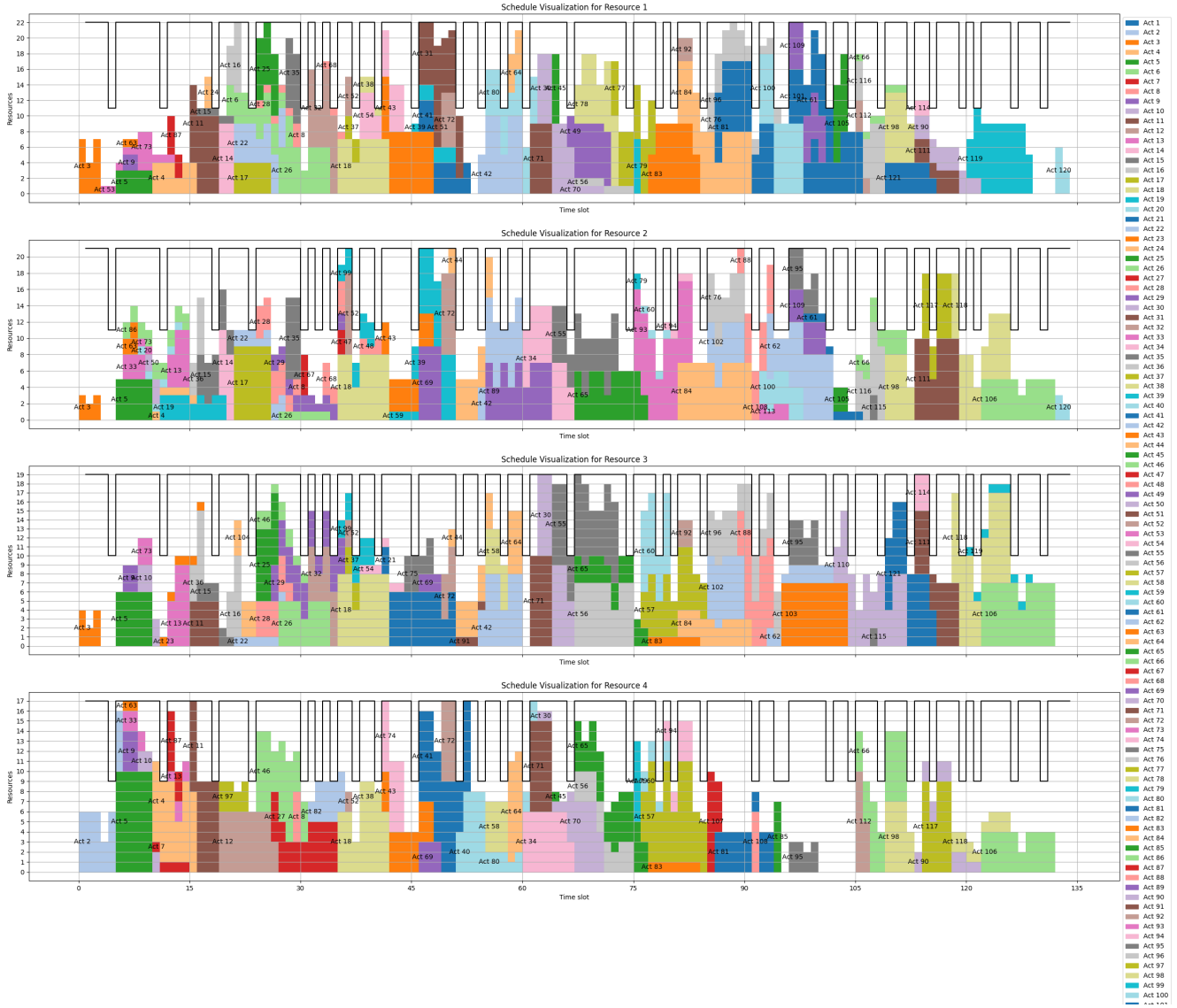


Figure A.12: Schedule visualization for problem instance j120t6_7_8, start/end time constrained