Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Resource-Constrained Project Scheduling Problem with Electrical Energy Consumption Optimization

Bc. Jan Macháček

Supervisor: Ing. Vilém Heinz
Field of study: Open Informatics
Subfield: Artificial Intelligence
January 2024

CTU CZECH TECHNICAL UNIVERSITY IN PRAGUE

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Macháček Jan**	Personal ID number:	**483753**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence**		

II. Master’s thesis details

Master’s thesis title in English:

Resource-Constrained Project Scheduling Problem with Electrical Energy Consumption Optimization

Master’s thesis title in Czech:

Resource-Constrained Project Scheduling Problem s optimalizací spotřeby elektrické energie

Guidelines:

The student will focus on extending the well-known Resource-Constrained Project Scheduling Problem (RCPSP) with constraints and criteria related to (electrical) energy [1]. The problem definition is inspired by the existing literature on energy optimization [2] and real-world production. The following conditions will be considered:
- Energy price in different time intervals using Time of Use (ToU) pricing [3] and its maximum contracted energy using Critical Peak Pricing (CPP).
- Varying consumption based on the task mode execution called Multi-mode RCPSP (e.g.,faster machine operation results in higher energy consumption) [4] will be considered.
- The energy price will be regarded as both a hard constraint and an optimization criterion (weighted linear combination optimization) [5].

The work will build on the current state-of-the-art in the field of RCPSP. The student will:
- Formally define the problem and create new datasets based on the existing instances of well-known benchmark sets for MM-RCPSP.
- Use Constraint Programming to formulate the problem.
- Use standard metaheuristics like genetic algorithm to provide an alternative approach and scalability for larger instances.
- Evaluate if a hybrid algorithm combining the two aforementioned methods can further improve provided results (warm-starting…), and if so, construct it.
- Consider the use of machine learning for solution initiation or subproblem prediction.
- Evaluate the effectiveness of all approaches and their applicability in different real-world scenario sizes.

Bibliography / sources:

[1] Humyun Fuad Rahman, Ripon K. Chakrabortty, Sondoss Elsawah, Michael J. Ryan,Energy-efficient project scheduling with supplier selection in
manufacturing projects,Expert Systems with Applications,Volume 193,2022,116446, ISSN 0957-4174,
https://doi.org/10.1016/j.eswa.2021.116446. (https://www.sciencedirect.com/science/article/pii/S0957417421017310)
[2] Hironori Okubo, Toshiyuki Miyamoto, Satoshi Yoshida, Kazuyuki Mori, Shoichi Kitamura, Yoshio Izui,Project scheduling under partially renewable resources and resource consumption during setup operations,Computers & Industrial Engineering, Volume 83, 2015,Pages 91-99, ISSN 0360-8352,
https://doi.org/10.1016/j.cie.2015.02.006.(https://www.sciencedirect.com/science/article/pii/S0360835215000637)
[3] Baigang Du, Tian Tan, Jun Guo, Yibing Li, Shunsheng Guo, Energy-cost-aware resource-constrained project scheduling for complex product system with activity splitting and recombining, Expert Systems with Applications, Volume 173, 2021,114754, ISSN 0957-4174,
https://doi.org/10.1016/j.eswa.2021.114754. (https://www.sciencedirect.com/science/article/pii/S0957417421001950)
[4] T. Samukawa and H. Suwa, “An Optimization of Energy-Efficiency in Machining Manufacturing Systems Based on a Framework of Multi-Mode RCPSP,” Int. J. Automation Technol., Vol.10 No.6, pp. 985-992, 2016 DOI: 10.20965/ijat.2016.p0985
[5] He, L., Zhang, Y. Bi-objective Optimization of RCPSP under Time-of-use Electricity Tariffs. KSCE J Civ Eng 26, 4971–4983 (2022).
https://doi.org/10.1007/s12205-022-0095-4 DOI: https://doi.org/10.1007/s12205-022-0095-4

Name and workplace of master's thesis supervisor:

Ing. Vilém Heinz   Department of Control Engineering FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **11.08.2023**   Deadline for master's thesis submission: **09.01.2024**

Assignment valid until: **16.02.2025**

Ing. Vilém Heinz Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to thank my supervisor Ing. Vilém Heinz for guiding this thesis and for his helpful comments.

Next, I would like to thank CIIRC CTU for the opportunity to use their server for the computations used in this thesis.

I would also like to thank everyone who supported me during my studies.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, January 7, 2024

. . . .

Abstract

In this thesis, we study the Resource-Constrained Project Scheduling Problem (RCPSP) with the constraints and criteria related to (electrical) energy consumption. We extend the Multi-Mode RCPSP (MMRCPSP) variant so that the different energy consumption of activities based on their mode, maximum contracted energy limits, and hourly Time-of-Use electricity prices are considered. We formulate and study two similar problems, where the total energy cost is either restricted by a hard budget limit or combined with the project makespan in the weighted linear combination criterion. Constraint Programming (CP) models and a Genetic Algorithm (GA) extending an existing good-performing GA for RCPSP to our problems are proposed and implemented to solve these problems. Furthermore, for extra tuning of the activity mode selection in the GA, we propose a machine learning model. We also construct a hybrid approach, in which the GA warm-starts the CP model. For the evaluation of the proposed methods, new datasets are created based on the existing MMRCPSP benchmark datasets PSPLIB and MMLIB. In the experiments, we analyze the optimality of the exact approaches, compare different initial generation creation methods for the GA, study the benefits of the use of the machine learning model in the GA, and evaluate all proposed approaches on instance sets of different sizes.

Keywords: Resource-Constrained Project Scheduling Problem, Multi-Mode RCPSP, Constraint Programming, Genetic Algorithm, Electrical Energy Consumption Optimization

Supervisor: Ing. Vilém Heinz

Abstrakt

V této práci studujeme Resource-Constrained Project Scheduling Problem (RCPSP) s omezeními a kritérii spojenými se spotřebou (elektrické) energie. Rozšiřujeme Multi-Mode RCPSP (MMRCPSP) variantu tak, že je uvažována různá spotřeba energie aktivit na základě jejich módu, limity maximální nasmlouvané energie a hodinová Time-of-Use cena elektřiny. Formulujeme a studujeme dva podobné problémy, kde celková spotřeba energie je buď omezena tvrdým limitem na rozpočet nebo zkombinována s makespanem projektu ve váženém lineárním kritériu. Pro řešení těchto problémů jsou navrženy a implementovány Constraint Programming (CP) modely a genetický algoritmus (GA), který rozšiřuje existující dobře performující GA pro RCPSP na naše problémy. Dále navrhujeme machine learning model pro další zlepšení výběru módu aktivity v GA a konstruujeme hybridní přístup, ve kterém GA warm-startuje CP model. Pro vyhodnocení navržených metod jsou vytvořeny nové datasety vycházející z existujících MMRCPSP benchmark datasetů PSPLIB a MMLIB. V experimentech analyzujeme optimalitu exaktních přístupů, porovnáváme různé metody vytváření počáteční generace pro GA, studujeme přínos použití machine learning modelu v GA a vyhodnocujeme všechny navržené přístupy na sadách instancí o různé velikosti.

Klíčová slova: Resource-Constrained Project Scheduling Problem, Multi-Mode RCPSP, Programování s omezujícími podmínkami, Genetický algoritmus, Optimalizace spotřeby elektrické energie

Překlad názvu: Resource-Constrained Project Scheduling Problem s optimalizací spotřeby elektrické energie

Contents

1 Introduction **1**
1.1 Resource-Constrained Project Scheduling Problem 1
1.2 Multi-Mode Resource-Constrained Project Scheduling Problem 2
1.3 Thesis Outline 3

2 Related Work **5**
2.1 Exact Methods 5
2.2 Metaheuristic Methods 6
2.3 Energy in RCPSP 7

3 Problem Statement **9**
3.1 Problem with Hard Energy Budget Constraint 9
3.2 Problem with Total Energy Cost in Objective 10

4 Exact Approach **13**
4.1 Constraint Programming 13
4.2 CP Model for Problem with Hard Energy Budget Constraint 14
4.3 CP Model for Problem with Total Energy Cost in Objective 16
4.4 Warm-Started CP Model 16

5 Metaheuristic Approach **17**
5.1 Genetic Algorithm 17
5.1.1 Chromosome 18
5.1.2 Neighborhood Search Operator 19
5.1.3 Initial Generation 21
5.1.4 Parent Selection 22
5.1.5 Crossover Operator 22
5.1.6 Mutation Operator 23
5.1.7 Next Generation Selection . . . 23
5.1.8 Stop Condition 23
5.1.9 Parameters 23
5.2 Mode Selection Model 24

6 Dataset **27**
6.1 MMRCPSP Datasets 27
6.2 Our Dataset 27
6.2.1 Project Horizon 28
6.2.2 Energy Consumption of Activity 28
6.2.3 Electricity Price 29
6.2.4 Energy Budget Limit 29
6.2.5 Energy Consumption Limits . 30
6.2.6 Created Instance Sets 30

7 Evaluation **33**
7.1 Configuration of Experiments . . 33
7.1.1 Scenarios of Problem with Total Energy Cost in Objective 34
7.2 Experiments 35
7.2.1 Optimality of CP Models . . . 35
7.2.2 Effect of Initial Generation in GA 37
7.2.3 Effect of Mode Selection Model in GA 39
7.2.4 Comparison of Proposed Approaches 41
7.3 Summary of Experiments 44

8 Conclusion **45**
8.1 Future Work 46

Bibliography **47**

A List of Abbreviations **51**

B List of Notations **53**

C Measured Data in Further Experiments **55**
C.1 Effect of Initial Generation in GA 55
C.2 Comparison of Proposed Approaches 56

Figures

6.1 Electricity price distribution during a single day . . . 29

7.1 Comparison of initial generation creation methods on problems with the hard energy budget constraint 37
7.2 Comparison of initial generation creation methods on problems with total energy cost in the objective . 38
7.3 Comparison of GAs with the trained mode selection model and different model probabilities . . . 40
7.4 Comparison of GAs without/with the trained mode selection model . 41
7.5 Comparison of all approaches in time on problems with the hard energy budget constraint . . . 42

C.1 Comparison of fallback methods of initial generation creation on problems with hard energy budget constraint . . . 55
C.2 Comparison of all approaches in time on problems with the total energy cost in the objective in the $scenario_{C_{\max}}$ scenario . . . 56
C.3 Comparison of all approaches in time on problems with the total energy cost in the objective in the $scenario_{equal}$ scenario . . . 57
C.4 Comparison of all approaches in time on problems with the total energy cost in the objective in the $scenario_{energy}$ scenario . . . 58

Tables

5.1 GA parameter settings . . . 24

6.1 Domains of used parameters in the problem instance generation . . . 31
6.2 Characteristics of the used dataset 31

7.1 Scenarios of the problem with total energy cost in the objective . . . 34
7.2 Optimality of found solutions in problems with the hard energy budget constraint . . . 36
7.3 Optimality of found solutions in problems with total energy cost in the objective . . . 36
7.4 Comparison of APDs from the critical path lower bound of all approaches on problems with the hard energy budget constraint . . . 42
7.5 Comparison of APDs from the weighted criterion lower bound of all approaches on problems with total energy cost in the objective in the $scenario_{C_{\max}}$ scenario . . . 43
7.6 Comparison of APDs from the weighted criterion lower bound of all approaches on problems with total energy cost in the objective in the $scenario_{equal}$ scenario . . . 43
7.7 Comparison of APDs from the weighted criterion lower bound of all approaches on problems with total energy cost in the objective in the $scenario_{energy}$ scenario . . . 43

Chapter 1
Introduction

With the movement towards cleaner production, the question of energy consumption is gaining more and more interest nowadays, especially from manufacturing companies. In connection with the stricter emission goals, many industrial companies have been attempting to reduce their energy consumption together with CO_2 emissions in the last few years. In addition, the level of attention devoted to energy consumption has increased even more with the recent noticeable increases and volatility in energy prices.

In a factory, there are typically a lot of different machines, tasks, and operations that need to be efficiently scheduled. These tasks are often constrained by various relations between them, and the available resources (e.g., machines or workers) are limited. Moreover, energy is consumed during the processing of individual tasks, and the electricity price can vary during the day. Hence, the energy consumption within the whole manufacturing process must be taken into consideration, too.

Many scheduling problems in the industry can be modeled by the well-known Resource-Constrained Project Scheduling Problem (RCPSP). However, energy consumption is usually not considered in this problem. Therefore, the goal of this thesis is to extend the RCPSP with relevant constraints and criteria related to electrical energy[1] to better reflect the real-world production requirements. Besides the formulation of the studied problem variant, the goal is also to provide algorithms based on both exact and metaheuristic approaches and evaluate their effectiveness in instance sets of different sizes.

1.1 Resource-Constrained Project Scheduling Problem

Resource-Constrained Project Scheduling Problem (RCPSP) is a standard project scheduling problem that has been widely investigated for more than 50 years. Despite the age of the problem, both it and a lot of its extensions and variants [1], [2] are still heavily studied due to its practical application.

[1]In this thesis, we will focus on electrical energy as a major energy source in industry. Thus, by the term energy, we mean electrical energy in the rest of this thesis unless otherwise stated.

In RCPSP, we have K types of resources and there are available R_k resources of each type $k \in \{1, \ldots, K\}$ in each time unit t. Resources can be, for example, machines, equipment, or workers. Given these resources, the goal is to schedule a finite set of n activities (also called tasks or operations) $A = \{a_1, \ldots, a_n\}$. Each activity $a_i \in A$ has specified its duration d_i and the number of resources needed for its processing. Particularly, each activity a_i requires r_{ik} units of resource k for each resource type $k \in \{1, \ldots, K\}$ during each time unit t in which it is processed. Moreover, precedence constraints that restrict the order of processing of some activities are present in this problem. Preemption is not allowed, and thus, when an activity a_i starts being processed at time s_i, its processing cannot be stopped, and it is finished at time $s_i + d_i$. All parameters of the problem are assumed to be non-negative integers.

Specifically, the goal of RCPSP is to find a schedule $S = (s_1, \ldots, s_n)$, where s_i is the start time of processing of activity a_i, which satisfies both precedence and resource constraints and minimizes the project makespan $C_{\max}$ that is the finish time of the last processed activity (i.e., $C_{\max} = \max_{a_i \in A} s_i + d_i$).

Precedence constraints are defined by a precedence relation $\prec\ \subseteq A \times A$. For each pair of activities (a_i, a_j), such that $a_i \prec a_j$, activity a_i has to be processed earlier than activity a_j meaning activity a_j can start after the activity a_i was processed (i.e., $s_j \geq s_i + d_i$).

Resource constraints, expressed in Eq. (1.1), ensure that there is enough amount of each type of resource k in each time unit $t \in \{0, \ldots, T\}$, where a project horizon T is an upper bound of the project makespan.

$$\sum_{\substack{a_i \in A \\ s_i \leq t < s_i + d_i}} r_{ik} \leq R_k, \quad \forall k \in \{1, \ldots, K\}, \forall t \in \{0, \ldots, T\} \tag{1.1}$$

1.2 Multi-Mode Resource-Constrained Project Scheduling Problem

An extension of the standard RCPSP that considers that activities can be processed in one of several modes is called *Multi-Mode Resource-Constrained Project Scheduling Problem* (MMRCPSP). It allows modeling alternative ways of processing a concrete activity (e.g., at different processing speeds or with different resource requirements).

In MMRCPSP, each activity a_i has a finite set of modes M_i in which it can be executed. The mode influences the activity duration and the number of resources the activity needs. Therefore, for each activity a_i and each its processing mode $m \in M_i$, a duration d_{im} and resource requirement r_{imk} for each resource k are given.

Two[2] main categories of resources are distinguished in MMRCPSP – renewable and non-renewable [3]. Renewable resources, which are usually repre-

[2]There is also a third category called doubly constrained, but it does not have to be considered explicitly because it can be replaced by renewable and non-renewable resources.

sented by machines, equipment, or human resources, restrict the availability of resources in each time unit. Such resources allocated for the processing of an activity are released immediately after the activity is processed. On the other hand, non-renewable resources, which are usually represented by cost (money), limit their consumption over the whole project.

Besides finding a list of start times, the goal of MMRCPSP is also to select exactly one mode for each activity such that the resulting schedule (formed by both start times and assigned modes) satisfies both resource and precedence constraints and minimizes the project makespan $C_{\max}$.

1.3 Thesis Outline

In this thesis, we formulate and study two new problem variants that extend the Multi-Mode RCPSP with energy-related constraints and criteria with respect to real-world production. Specifically, the following aspects of energy are considered on top of MMRCPSP.

- The consumption of the activity depends on the mode in which it is processed (e.g., the faster execution of an activity requires more resources and results in higher energy consumption).
- Electricity prices vary during the day, and a concrete price is given for each hour by Time-of-Use (TOU) tariffs.
- Energy consumption in each 15-minute time interval is constrained by the maximum contracted energy limit.
- Two different views on how to regard the total energy cost are studied in two similar problems. The project's total energy cost is either constrained by a hard energy budget limit or is part of the optimization criterion.

The rest of this thesis is organized as follows. Chapter 2 summarizes the related literature on methods how RCPSP and MMRCPSP are solved and relevant problems that consider energy. In Chapter 3, we extend Multi-Mode RCPSP with energy-related constraints and criteria and formulate two new problems – *Problem with Hard Energy Budget Constraint* (in Section 3.1) and *Problem with Total Energy Cost in Objective* (in Section 3.2). Chapter 4 presents the Constraint Programming (CP) paradigm and provides CP models for both problems representing the exact approach to the problems. A meta-heuristic approach is proposed in Chapter 5, where we extend and adapt an existing well-performing genetic algorithm for RCPSP to our problems. At the end of this chapter, we also propose a simple machine learning model that selects the most appropriate activity mode based on activity and sub-problem characteristics. In Chapter 6, we create new problem instances of various sizes based on existing MMRCPSP benchmark datasets. All implemented algorithms are experimentally evaluated in Chapter 7. Finally, Chapter 8 summarizes this thesis and outlines possible future work.

Chapter 2
Related Work

With respect to the practical relevance and the time RCPSP and MMRCPSP have been studied, a lot of methods were proposed to solve these problems. Since RCPSP and hence also MMRCPSP (since an RCPSP instance is implicitly also an instance of MMRCPSP with single mode for each activity) are strongly $\mathcal{NP}$-hard [4], there have been developed both exact and metaheuristic approaches.

In this chapter, we will describe the exact methods in Section 2.1 and the metaheuristic ones in Section 2.2. In Section 2.3, we will then focus on related problem variants that consider energy.

2.1 Exact Methods

Exact approaches aim to find the optimal solution or prove that the problem instance is not feasible. Moreover, lower/upper bounds on the solution quality are usually provided during the search. Several different approaches have been developed for RCPSP and MMRCPSP.

As a combinatorial optimization problem, RCPSP can be formulated using *(Mixed-)Integer Linear Programming (MILP)* and then solved by a MILP solver. Many different MILP formulations have been proposed in the literature. According to how time and resource-sharing characteristics are modeled, they can be divided into three categories – time-indexed, sequencing, and event-based formulations [5].

Time-indexed MILP formulations (e.g., in [6] and [7] for RCPSP or in [8] for MMRCPSP) use binary variables to indicate a state of activity at a certain time. Particularly, a binary variable x_{it} is used for each activity a_i and time unit t, and it is equal to 1 if and only if activity a_i starts at time t. For MMRCPSP, binary variables also contain information about the execution mode of activity (i.e., variable x_{imt} is equal to 1 if and only if activity a_i is executed in mode m and starts at time t).

Sequencing MILP formulations (for instance, in [9]) involve two kinds of variables. Integer variables correspond to the start times of individual activities, and binary variables specify the relative order of activities. Specifically, for each pair of activities (a_i, a_j), there is a binary variable x_{ij} that is equal to 1 if activity a_i should be completed before the activity a_j can start (i.e., $s_i + d_i \leq s_j$).

In the event-based MILP formulation, the time horizon is decomposed into a set of events. An event corresponds to the time when an activity starts or ends. Two different event-based formulations for RCPSP were proposed in [10], in which variables are indexed by events (not by time). For instance, binary variables indicating whether an activity either starts/ends or is in progress at a particular event were used.

Besides MILP, many *enumeration methods* have been proposed, especially for the MMRCPSP variant. These techniques are based on the branch-and-bound method and the idea of enumerating partial schedules [3]. For instance, an enumeration scheme based on the concepts of mode and extension alternatives was proposed in [11].

Furthermore, *Constraint Programming (CP)* approach has become popular in the last few years. This is mainly because of the recent significant progress in the development of more efficient CP solvers for scheduling (e.g., *IBM ILOG CP Optimizer* [12]). In addition, compared to the linear algebraic constraints in the MILP modeling, CP also offers more complex constraints such as logic constraints, scheduling constraints, global constraints, and others [13].

2.2 Metaheuristic Methods

Finding the optimal schedule in a reasonable time is not usually possible for problem instances of a larger scale. In such cases, the heuristic and metaheuristic approaches come into play. There have been developed a lot of metaheuristic techniques for RCPSP and its variants based on various approaches in the literature [3], [14], [15], [16].

Firstly, several so-called single-pass heuristics based on different priority rules, determining a certain precedence-feasible order of activities, have been proposed in the RCPSP literature [15]. These heuristics can be further improved by applying some improvement techniques, such as serial/parallel *Schedule Generation Schemes (SGS)* [17], forward/backward scheduling (i.e., scheduling in the reversed direction), or justification schemes (shifting activities in a schedule). These heuristic methods can provide good solutions, but the optimality gaps are still not satisfactory [15]. This fact motivated the further development of more complex metaheuristic methods.

Very popular and successful metaheuristics in both RCPSP and MMRCPSP are *Genetic Algorithms (GAs)* that are inspired by the evolution principle of the survival of the fittest individuals. In GA, the set of candidate solutions (schedules) forms a generation. In each iteration, operators (the most common are *crossover* and *mutation*), which emulate genetic evolution, are applied to these individuals to create new child schedules. At the end of each iteration, individuals are selected for the next generation depending on the fitness function (the better the fitness value is, the higher the probability of being selected). GAs were proposed, for example, in [18], where the authors used two separate populations and extended the serial SGS by a mode improvement procedure for the multi-mode variant, in [19], where a local search strategy was incorporated to the GA operators, or in [20], where two new variants of crossover were developed.

However, many promising metaheuristics based on other techniques have been developed as well. For instance, a metaheuristic using the *Simulated Annealing* approach was presented in [21], a method based on *Ant Colony Optimization* was proposed in [22], *Particle Swarm Optimization* approach was used in [23], or *Tabu Search* algorithm was presented in [24].

Finally, with the huge advancements in the field of *machine learning* in recent years, some attempts to use machine learning techniques also for combinatorial problems such as RCPSP have been proposed. Nevertheless, due to the complexity of RCPSP, they usually focus only on a particular subproblem. Generally, the methods learn from multiple project characteristics (e.g., network complexity, resource strength, or resource factor) on small instances and are trained to perform well on the bigger ones. For instance, a decision tree approach was constructed in [25] to classify and detect the best-performing priority rule, a multilayer feed-forward neural network was trained in [26] to select an appropriate priority rule or a prediction model was learned to rank different configurations of branch-and-bound procedures in [27]. A more advanced approach was proposed in [28], where a graph neural network was trained to learn high-dimensional embeddings from characteristics of individual activities, and a deep reinforcement learning was used to learn the scheduling policy.

Last but not least, many *hybrid algorithms* that combine multiple of these techniques have been proposed. The hybridization may be either integrative, where the local search methods are combined with population-based metaheuristics, or collaborative, where possibly different pure metaheuristics, executed sequentially or in parallel, exchange information about the search process [16].

2.3 Energy in RCPSP

Scheduling problems are usually closely related to the industrial environment. Since industrial machines typically consume a lot of energy, it is natural to consider energy consumption in scheduling problems.

Furthermore, the price of electricity usually is not constant but rather varies during the day. There exist several different pricing policies that are applied in production planning [29], [30]. One of the most frequent pricing schemes are *Time-of-Use (TOU)* tariffs, where electricity prices vary in different parts of the day. Usually, the day is divided into off-peak, mid-peak, and on-peak periods with constant electricity prices that are known in advance. *Real-Time Pricing (RTP)* is similar to the TOU strategy, but it is more dynamic because the electricity prices typically differ and change in hourly intervals. Another policy is called *Critical Peak Pricing (CPP)*, where high consumption during a concrete time interval is heavily penalized.

Given a large number of RCPSP studies, only a few of them investigated adding some energy-related constraints and objectives to their problems. In the rest of this section, we will describe some of them.

In [31], the authors assumed partially renewable resources (a generalization of both renewable and non-renewable resources) that allowed them to independently model power restriction during peak hours and contract demand constraints. Contract demand limits the average energy consumption over a certain period of time. Power restriction constraints limit the use of electric power for 1-hour time intervals during peak hours to some percentage of the contracted value. In addition, multiple modes, setup operations, and energy consumed during them were considered in their problem.

A bi-objective model for RCPSP under TOU electricity tariffs was proposed in [32]. Besides electricity costs, labor costs during various shifts were considered as well. Electricity consumption during different hours and labor costs were combined in one objective called total project cost. The second objective was the project makespan. Moreover, the impact of the machine on/off strategy (considering transitions between machine states) was analyzed.

Energy cost minimization for a different scheduling problem of unrelated parallel machines, which does not contain precedence constraints, was studied in [33]. Both electricity consumption and demand charges were involved in the total electricity cost objective in their model. Electricity cost was determined by the RTP policy.

Energy consumption was also considered in [34], where multiple suppliers were assumed as a part of the so-called green project indicator criterion. Besides energy consumption, this criterion combines noise pollution and the safety of workers. The second objective in the proposed bi-objective problem was the minimization of the project makespan.

A bi-objective model was established in [35] for solving an RCPSP variant for a complex product system under the static TOU electricity tariffs. The optimized objectives were project delay and energy consumption cost.

To sum up, energy can be involved either in the problem constraints (e.g., contract demand and power restriction during peak hours in [31]) or in the objective function – the minimization of total electricity cost in [33], total project cost in [32], green project indicator in [34], or energy consumption cost in [35]. The formulated problems are either single-objective or bi-objective, and the standard scheduling objectives, such as project makespan or project delay, are also optimized.

Although these problems are based on some real-world situations, each of them contains only some important aspects, while other important ones are not considered in the problem. Specifically, similarly to [32] and compared to [31], we suppose that the problem should consider the non-constant electricity price during the day. Moreover, similarly to [31], contract demand should be limited over a certain time interval. Furthermore, we believe the total energy cost can be involved either as a constraint or as a part of the objective function. Therefore, in Chapter 3, we will formulate two new problems that take these aspects into account.

Chapter 3

Problem Statement

An important aspect of the problem where energy consumption is considered is definitely its price, which can vary over time. We suppose energy costs can be handled in two possible ways (i.e., to consider it either in constraints or in the objective). Hence, in this chapter, we will formulate two similar problems that extend the Multi-Mode RCPSP formulation, and we will study these two problems in the rest of the thesis.

On top of the standard MMRCPSP formulation, both problems consider that energy is consumed while processing individual activities, the electricity price varies over time, and the consumed energy is restricted in regular time intervals. And as indicated, they differ in the way the energy costs are regarded. The total energy expenses of the project are either limited by a hard constraint in the form of an energy budget (Section 3.1) or involved in the optimization criterion (Section 3.2).

3.1 Problem with Hard Energy Budget Constraint

The first problem assumes there is a hard energy budget on the total energy price that cannot be exceeded. The problem is formulated as follows.

Same as in the MMRCPSP definition, we are given a finite set of activities A that needs to be processed. Each activity a_i has a finite set of processing modes M_i. The duration d_{im} of the processing of an activity a_i and the number of resources r_{imk} of each resource type k the activity needs are given for each processing mode $m \in M_i$. Precedence constraints between the activities and resource constraints are defined as well.

Additionally, processing of each activity a_i in mode m consumes c_{im} units of electric energy in each time unit t during which it is processed (i.e., the energy consumed by an activity is assumed to be the same in each time unit). The consumption c_{im} is the total energy consumption of all resources used to process the activity in a single time unit. All resources in this problem are renewable (i.e., R_k units of resource k are available at each time unit t). The cost of electricity is modeled with Time-of-Use (TOU) electricity tariffs, where the electricity prices differ in different parts of the day. The tariffs are modeled by a function $p(t)$ that assigns an electricity price to each time unit $t \in \{0, \dots, T\}$.

Besides the precedence and resource constraints, the following two energy-related constraints are added. Both of them share the term $c(S,t)$, expressed in Eq. (3.1), which denotes the total amount of consumed energy by schedule S at time t (i.e., the sum is over all activities that are processed at time t). We recall that a schedule is formed by a certain start time s_i and mode $m \in M_i$ for each activity a_i.

$$c(S,t) = \sum_{\substack{a_i \in A \\ s_i \le t < s_i + d_{im}}} c_{im} \tag{3.1}$$

The first added constraint aims to avoid high penalties for high electricity consumption that are common in CPP scenarios. Therefore, it restricts energy consumption during regular time intervals so as not to overstep the contracted amount. Particularly, in a schedule S, the total amount of consumed energy during each 15-minute[1] time interval $q \in Q$ cannot exceed the given energy consumption limit C_q. Thus, the inequality, expressed in Eq. (3.2), has to be satisfied for each quarter $q \in Q$.

$$\sum_{t \in q} c(S,t) \le C_q, \quad \forall q \in Q \tag{3.2}$$

The second constraint, expressed in Eq. (3.4), reflects TOU prices. It limits the schedule S such that the total price of consumed energy over the whole project $energyCost(S)$, expressed in Eq. (3.3), cannot exceed the given energy budget B.

$$energyCost(S) = \sum_{t=0}^{T} c(S,t)p(t) \tag{3.3}$$

$$energyCost(S) \le B \tag{3.4}$$

Similarly to MMRCPSP, the goal of this problem is to non-preemptively schedule all activities (i.e., to find a schedule S containing a start time s_i and a processing mode $m \in M_i$ for each activity a_i) such that all constraints are satisfied and the project makespan $C_{\max}$ is minimized.

3.2 Problem with Total Energy Cost in Objective

In the problem with a hard energy budget constraint, defined in Section 3.1, the project's duration is optimized, and the total energy expenses of the project are regarded only in the hard energy budget constraint. Nevertheless, in practice, the total energy cost of the project is sometimes also an important metric when the energy budget is not so strict.

Hence, we define a slightly different problem. The only difference compared to the previous one is that it does not consider the energy budget

[1]We are using 15-minute intervals because it is a common practice in the industry.

constraint (Eq. (3.4)), and instead, it involves the project's total energy cost directly in the optimization criterion.

Specifically, the goal of this problem is to find a schedule S, which satisfies all the constraints and minimizes the weighted linear objective expressed in Eq. (3.5).

$$weightedObjective(S) = \alpha\, C_{\max} + \beta\, energyCost(S) \tag{3.5}$$

This criterion combines two aspects – the project's duration $C_{\max}$ and the project's total energy cost $energyCost(S)$. These aspects are weighted by parameters α and β that specify the trade-off between the project makespan and its energy cost. The parameters have to be set by the user before the optimization. For instance, they can be set by the managers to determine which aspect is more important in their business case.

Chapter 4

Exact Approach

Although exact methods can usually find the optimal solution in a reasonable time only for smaller problem instances, they can still be useful even for larger ones. For such instances, they can provide at least feasible solutions and, in some cases, even solutions of good quality.

From possible exact techniques, we selected a Constraint Programming (CP) approach for solving our problems. It is a not-so-explored approach that has become quite popular in scheduling in the last few years, especially due to effective problem formulation.

In Section 4.1, we will briefly present the CP paradigm together with expressions and constraints employed in the proposed models. After that, the CP models for both our problems will be proposed in Sections 4.2 and 4.3. And in Section 4.4, we will describe the warm-starting technique that can improve the efficiency of the CP models.

4.1 Constraint Programming

Constraint Programming (CP) is a powerful paradigm for modeling and solving combinatorial problems. In this paradigm, the problem is modeled in a declarative way by defining sets of variables, domains, and constraints. It aims to assign a value from its domain to each variable such that all constraints on the variables are satisfied. There is also an objective function that evaluates the current assignment of values to all variables. Hence, the goal of the CP solver is to find a feasible solution with the optimal objective value.

In this thesis, we will use the *IBM ILOG CP Optimizer* [12] because it is a state-of-the-art solver. Besides the speed, its advantage is also a wide range of global constraints and interval variables. In the rest of this section, we will focus on those expressions and constraints used in our CP models and their notation.

The essential element of the CP model for scheduling problems is an *interval variable*, which usually represents some activity in a schedule. It is a decision variable whose domain is a set of all possible intervals. Interval variable x is denoted by an expression `interval`(x) and it is characterized by its start time `startOf`(x), finish time `endOf`(x), and the size of the interval `sizeOf`(x) representing the start time, finish time and the duration of an activity.

Moreover, an interval variable can be *optional*, which is an important feature for many use cases (e.g., in our case, they will allow us to model alternative processing modes of activity easily). The optional interval variable can be either present or absent in the solution. A present optional interval variable has assigned a specific interval, whereas absent optional interval variables are (roughly speaking) not considered by any constraint in the model. An optional interval variable y is denoted as `optionalInterval`(y).

However, it is important to note that until a solution is found, it may not be known whether an optional interval variable will be present or not [12]. The presence of an optional interval variable y in the solution is expressed by the logical expression `presenceOf`(y), which is evaluated to 1 if the interval variable is present in the model or 0 if the variable is absent.

A group of possible optional interval variables can be connected to a single interval variable, for example, using the `alternative`$(x, \{y_1, \dots, y_l\})$ constraint. It models an exclusive alternative for the interval variable x between a set of optional interval variables $\{y_1, \dots, y_l\}$. Particularly, exactly one of the optional variables y_i is present in the solution, and the interval of variable x is the same as the interval of variable y_i.

Furthermore, a lot of constraints for specifying the order of activities, sequences of activities, overlapping of activities, or temporal constraints can be modeled. In our models, we will use the constraint `endBeforeStart`(x_i, x_j) to define the precedence constraints, which restricts that the start time `startOf`(x_j) of interval x_j has to be greater than or equal to the finish time `endOf`(x_i) of interval x_i.

Last but not least, cumulative functions are provided for modeling resource consumption. Specifically, we will employ the `pulse`(x, z) expression that specifies that z units of a particular resource are used during the interval x.

4.2 CP Model for Problem with Hard Energy Budget Constraint

In this section, we will propose a CP model for the problem with hard energy budget constraint, defined in Section 3.1. Since we need to express the energy-related constraints that are time-dependent, we have to explicitly incorporate time units into the model and use the so-called time-indexed model. Therefore, for each activity, we will distinguish the optional interval variables not only with the processing mode but also with the different start times.

In order to reduce the number of optional interval variables, a set of possible start times of each activity a_i is limited by a lower bound s_i^{LB} and an upper bound s_{im}^{UB}. Both bounds are determined from the directed acyclic graph of precedent activities. In particular, the lower bound is the length of the longest path to a given activity from the first activity, where the length of each activity a_j is the minimum duration over all possible modes of activity $\min_{m \in M_j} d_{jm}$. The upper bound is computed similarly from the last activity and the subtraction from the project horizon. In addition, its value can differ for individual mode m and depends on the activity duration in the given mode.

The proposed CP model is formulated as follows.

$$\text{Minimize} \quad \max_{\forall x_i} \texttt{endOf}(x_i) \tag{4.1}$$

Subject to

$$\texttt{interval}(x_i) \qquad \forall a_i \in A \tag{4.2}$$

$$\texttt{optionalInterval}(y_{ims}) \qquad \forall a_i \in A, m \in M_i, s \in \{s_i^{LB}, \dots, s_{im}^{UB}\} \tag{4.3}$$

$$\texttt{sizeOf}(y_{ims}) = d_{im} \qquad \forall y_{ims} \tag{4.4}$$

$$\texttt{startOf}(y_{ims}) = s \qquad \forall y_{ims} \tag{4.5}$$

$$\texttt{alternative}(x_i, \bigcup_{\substack{m \in M_i \\ s \in \{s_i^{LB}, \dots, s_{im}^{UB}\}}} \{y_{ims}\}) \qquad \forall x_i \tag{4.6}$$

$$\texttt{endBeforeStart}(x_i, x_j) \qquad \forall a_i, a_j \in A, a_i \prec a_j \tag{4.7}$$

$$\sum_{\forall y_{ims}} \texttt{pulse}(y_{ims}, r_{imk}) \le R_k \qquad \forall k \in \{1, \dots, K\} \tag{4.8}$$

$$\sum_{\forall y_{ims}} \texttt{presenceOf}(y_{ims}) e_q^{cons}(a_i, m, s) \le C_q \qquad \forall q \in Q \tag{4.9}$$

$$\sum_{\forall y_{ims}} \texttt{presenceOf}(y_{ims}) e^{cost}(a_i, m, s) \le B \tag{4.10}$$

For each activity $a_i \in A$, a corresponding interval variable x_i is constructed in Eq. (4.2). Moreover, an optional interval variable y_{ims} is created for each activity a_i, processing mode m, and possible start time s in Eq. (4.3). Each optional interval variable has a fixed duration and start time that are specified in constraints in Eqs. (4.4) and (4.5), respectively. All these interval variables of each activity are connected together in Eq. (4.6).

Precedence constraints are specified in Eq. (4.7), and resources are constrained in Eq. (4.8). The energy-related constraints, which are added on top of the MMRCPSP, expressing energy consumption limits and the hard energy budget limit are described in Eqs. (4.9) and (4.10). Note that both the energy

$$e_q^{cons}(a_i, m, s) = \sum_{t \in [s, s+d_{im}] \cap q} c_{im} \tag{4.11}$$

consumed by an activity a_i in processing mode m with start time s in a 15-minute time interval q and the energy cost

$$e^{cost}(a_i, m, s) = \sum_{t=s}^{s+d_{im}} c_{im} p(t) \tag{4.12}$$

of such activity are constants that can be precomputed in advance. Finally, the objective function in Eq. (4.1) minimizes the project makespan (i.e., the finish time of the last activity).

4.3 CP Model for Problem with Total Energy Cost in Objective

The CP model for the problem with total energy cost in the objective, introduced in Section 3.2, is very similar to the CP model for the problem with hard energy budget constraint in Section 4.2. The only two differences are that the hard energy budget constraint, expressed in Eq. (4.10), is not in this model, and the model optimizes the weighted objective function, defined in Eq. (3.5). In particular, it minimizes the CP expression in Eq. (4.13) instead of Eq. (4.1).

$$\text{Minimize} \quad \alpha \max_{\forall x_i} \texttt{endOf}(x_i) + \beta \sum_{\forall y_{ims}} \texttt{presenceOf}(y_{ims}) e^{cost}(a_i, m, s) \tag{4.13}$$

4.4 Warm-Started CP Model

It usually takes a lot of time until a feasible solution is found by the CP solver. Therefore, providing a *warm-start* solution (i.e., a feasible solution) to the CP solver often helps to improve its efficiency.

Specifically, we will use a schedule found by a genetic algorithm (described in Section 5.1) in a small number of iterations as a warm-start schedule to the CP solver.

Furthermore, since proposed CP models are time-indexed and contain many optional interval variables, it is reasonable to attempt to reduce them when we have a warm-start schedule. In case of a problem with the hard energy budget constraint, where the project makespan is minimized, the number of variables in the model can be reduced (and hence, we implemented this improvement) by decreasing the project horizon to the makespan of the warm-start schedule. Unfortunately, this improvement cannot be generally made in case of a problem with the total energy cost in the objective due to the more complex objective function.

Chapter 5

Metaheuristic Approach

Exact approaches are not very scalable and often become unusable for larger instances. Thus, we studied the current state-of-the-art metaheuristic methods, an alternative approach that can find feasible solutions for such instances. We chose an existing Genetic Algorithm (GA) for RCPSP, adjusted it to our problems, and implemented it.

The implemented GA for our problems will be described in Section 5.1. Besides that, in the attempt to further improve the GA performance, a machine learning model to select activity mode during the rescheduling of activity in the sub-problem in the GA will be proposed in Section 5.2.

5.1 Genetic Algorithm

The selected genetic algorithm, which we decided to extend and adapt to our problems, is a *Genetic Algorithm with Neighborhood Search (GANS)* [19]. In GANS, the neighborhood (local) search (NS) is incorporated into a GA framework, and the authors showed that it improves the efficiency of searching the solution space while keeping the randomness of the GA approach. Even though it is not the newest algorithm, according to the data reported by the authors in [16], it is still one of the best metaheuristic algorithms for RCPSP. That is why we decided to use it as a baseline for our extension to both of our problems.

We will describe the implemented GA in general for both problems at the same time since the differences between our two problems are relatively small.

The proposed genetic algorithm follows the common GA structure shown in Algorithm 1. Given a problem P, it uses GA operators and tries to find an individual (in our case, a schedule denoted $bestSchedule$) with the lowest objective value among all found feasible solutions during the search until the termination condition, denoted `stopCondition`, is satisfied. The algorithm consists of several important parts, which we will describe in the rest of this section. Most of them are very similar to those in GANS since we adapted this algorithm to our problems. However, we will describe all the parts for the completeness and clarity of our implementation. The algorithm also has several parameters, which will be summarized in Section 5.1.9.

Algorithm 1: GA structure

```
input  : Problem $P$
output : Schedule $bestSchedule$
1  $G \leftarrow$ createInitGeneration($P$)
2  while not stopCondition() do
3      $parentChromosomes \leftarrow$ selectParentPairs($G$)
4      $childChromosomes \leftarrow \{\}$
5      foreach $(C_f, C_m) \in parentChromosomes$ do
6          $child1, child2 \leftarrow$ crossover($C_f$, $C_m$)
7          $childChromosomes \leftarrow childChromosomes \cup$
             {mutation($child1$),mutation($child2$)}
8      end
9      $G \leftarrow$ selectNextGeneration($childChromosomes$)
10 end
11 return $bestSchedule$
```

5.1.1 Chromosome

The fundamental element of the proposed GA is a *chromosome*, which represents a single feasible solution in a solution space and the information of how the neighborhood search (NS) operator (Section 5.1.2) should operate. In our case, a chromosome is a tuple consisting of these four genes:

1. *A core activity* (a_{core}) – An activity that is used to select activities that will be rescheduled in the NS operator.

2. *NS operator improvement* (I_{obj}) – An improvement in the objective value obtained by applying the NS operator with this core activity (i.e., the contribution of the NS operator on the current chromosome).

3. *A schedule* (S) – A current feasible solution defined by start time s_i and processing mode $m \in M_i$ for each activity $a_i \in A$.

4. *An activities order* (A^o) – An order of all activities A which satisfies the precedence constraints of the problem and respects the order the individual activities were put into the schedule.

Compared to the chromosome representation in the GANS algorithm, we do not have a binary variable determining whether a forward or a backward serial *Schedule Generation Scheme (SGS)* method [17], in which each activity is gradually scheduled at the earliest (latest) precedence- and resource-feasible start time, will be used while solving the rescheduling sub-problem. It is because we use only forward search due to the added energy-related constraints in our problems. On the other hand, the activities order A^o is stored to be able to easily perform a left shift on the rescheduled schedule in the NS operator (Section 5.1.2).

5.1.2 Neighborhood Search Operator

Similarly to the GANS algorithm, the key part of the proposed GA is the *Neighborhood Search (NS)* operator. It is based on a local search method (Local Search with Sub-problem Exact Resolution) introduced in [36], which is integrated into the GA operators. Particularly, the NS operator is employed in the GA as a part of the *crossover* and *mutation* operators (Sections 5.1.5 and 5.1.6).

Specifically, the NS operator attempts to improve a feasible schedule by rescheduling some activities in a sub-problem when the start times, modes, and resource allocation of the other activities are fixed. It consists of three essential steps, which we will describe separately before describing the whole NS operator.

1. *Block selection method* – It splits the set of activities A according to the core activity a_{core} of the chromosome into a set of activities A_r containing N_r activities, which are in the block around the core activity (including a_{core}) and will be rescheduled, and the other activities A_f, which will be fixed in the rescheduling sub-problem. The block of activities is formed by the activities that overlap with or are close to the given core activity. For the pseudocode, we refer to [19].

2. *Rescheduling sub-problem* – When the activities are split, the sub-problem is defined by fixing the start times, modes, and resource allocations of the fixed activities A_f. Its goal is to reschedule the rest of the activities (A_r). Similarly to the GANS algorithm, the sub-problem is solved by rescheduling the activities A_r using a forward serial SGS. A random order of activities A_r^o that respects the precedence constraints is constructed for activities in A_r. In addition, a random processing mode is assigned for each activity $a_i \in A_r$ before it is scheduled. Given the order of activities A_r^o, the new schedule S_r is constructed by scheduling the activities A_r one by one (respecting their order, modes, current resource allocations, and precedence constraints) at the first possible start time where the project capacities (resource capacities, energy consumption limits, and eventually energy budget) are not exceeded. In case some activity cannot be scheduled (e.g., due to precedence or capacity constraints), an *empty* schedule is returned.

3. *Left shift* – In the left shift, activities A_{before}^o, scheduled before the block A_r^o, and rescheduled activities in the block A_r^o stay fixed, and activities A_{after}^o, scheduled after the block A_r^o, are rescheduled. Similarly to the Rescheduling sub-problem (second step), a serial SGS is used to schedule the activities A_{after}^o. This time, their modes and order stay the same as in the input chromosome C_{input}.

The pseudocode of the NS operator used in our GA is shown in Algorithm 2, in which these three essential steps are highlighted. It takes the problem P and an input chromosome C_{input} as an input and returns a chromosome C_{output}.

Algorithm 2: Neighborhood Search Operator

```
input      : Chromosome $C_{input}$, Problem $P$
output     : Chromosome $C_{output}$
parameters : Maximum number of iterations $I$,
             Number of activities to be rescheduled $N_r$,
             GA stopping limit $\lambda$
1  for $i \leftarrow 1$ to $I$ do
2      $A_r$, $A_f \leftarrow$ splitActivitiesUsingBlockMethod($C_{input}$, $N_r$)
3      $S_{input} \leftarrow C_{input}.S$
4      $S_r$, $A^o_r \leftarrow$ solveReschedulingSubproblem($S_{input}$, $A_r$, $A_f$)
5      if $S_r$ is empty then
6          continue
7      end
8      $A^o_{before}$, $A^o_{after} \leftarrow$
        splitFixedActivitiesAndPreserveOrder($A_f$, $C_{input}$)
9      if $|A^o_{after}| = 0$ then
10         $S_{result} \leftarrow S_r$
11     else
12         $S_{result} \leftarrow$ applyLeftShift($S_r$, $A^o_{after}$)
13         if $S_{result}$ is empty then
14             $S_{result} \leftarrow S_r$
15         end
16     end
17     $I_{obj} \leftarrow objective(S_{input})$ - $objective(S_{result})$
18     if $I_{obj} > 0$ then
19         $A^o_{result} \leftarrow$ constructActivitiesOrder($A^o_{before}$, $A^o_r$, $A^o_{after}$)
20         return chromosome($C_{input}.a_{core}$, $I_{obj}$, $S_{result}$, $A^o_{result}$)
21     end
22 end
23 return chromosome($C_{input}.a_{core}$, 0, $C_{input}.S$, $C_{input}.A^o$)
```

The NS operator runs until a given limit on the number of iterations I is reached, until a schedule with a better objective value is found, or until the stopping limit λ of the GA is reached. If no schedule with a better objective value is found in I iterations, on the Line 23, the NS operator returns a chromosome with the same values as the input chromosome except for the objective improvement gene, which is set to 0.

In each iteration of the NS operator, a new schedule with a better objective value is attempted to be constructed in the following way. At first, on the Line 2, all activities are split into two disjoint sets A_r and A_f using a *block selection method* according to a core activity of the input chromosome C_{input}. These two sets of activities define a sub-problem where activities from A_f will stay fixed, whereas activities from A_r will be rescheduled. Then, the resulting *rescheduling sub-problem* is solved on the Line 4 and a schedule after rescheduling S_r and the order A_r^o, in which the activities in the block (A_r) were rescheduled, are returned. If no feasible schedule was found during the rescheduling, the NS operator continues with the next iteration on the Line 1.

Otherwise, the rescheduling resulted in a feasible schedule S_r, and the fixed activities A_f are split into two lists A_{before}^o and A_{after}^o according to their start time in the input schedule S_{input} and the start time of the core activity in the input chromosome C_{input} on the Line 8. The order of activities from the input chromosome C_{input} is preserved in both lists. Then, the *left shift* is applied to the schedule after rescheduling S_r for the fixed activities A_{after}^o, which are after the rescheduling block, and creates the result schedule S_{result} on the Line 12. When there is no activity after the rescheduling block, the left shift is not applied, and the schedule after rescheduling S_r is assigned as the result schedule S_{result} on the Line 10. Similarly, the schedule after rescheduling S_r is assigned as the result schedule S_{result} if the left shift fails to create a feasible schedule.

Finally, on the Line 17, the I_{obj} is computed from the objective value of the input schedule S_{input} and the result schedule S_{result}. If there is an improvement in the objective value, a new chromosome is created such that a core activity of the input schedule is preserved, the NS operator improvement gene is set to I_{obj}, and the result schedule S_{result} is used. Moreover, all activities are ordered with respect to the precedence constraints to be able to easily perform a left shift in the next generations of the genetic algorithm. Otherwise, if there is no improvement in the objective value, the algorithm continues with the next iteration on the Line 1.

5.1.3 Initial Generation

Unfortunately, the process of how the initial generation is created in GANS is not mentioned in [19]. Nevertheless, we suppose it is also an important part of the GA. Therefore, we implemented and will evaluate (in Section 7.2.2) four different ways of how to create the initial generation. Each starts with determining a random order of activities that respects the precedence constraints.

1. *SGS method* – It assigns a random mode to each activity and solves the rescheduling sub-problem, in which all activities are scheduled with respect to the given random order in the same way as it is solved in the NS operator (Section 5.1.2).

   Unfortunately, due to capacity constraints in the problem, this greedy method does not have to create a feasible schedule. Hence, it may need to be repeated with a new precedence-feasible random order of activities until a feasible schedule is created. Each attempt is, of course, counted in the result statistics.

2. *Sequential method* – It creates a sequential schedule where activities are placed one right after the other (each activity starts at the time when the

previous activity finished) in the given random order. Each activity is scheduled in the mode with the lowest energy consumption that satisfies the most strict 15-minute energy consumption limit.

It relies on the fact that such a sequential schedule will be feasible. However, in our dataset (see Section 6.2), any order of the sequentially scheduled activities in the least consuming mode will lead to a feasible schedule, and hence, this method always creates a feasible schedule.

3. *Fallback method* – It tries to create a schedule using the *SGS* method. When no feasible schedule is created in the given number of attempts, the *sequential* method is used as a fallback. The number of attempts is a parameter of this method. Hence, for clarity, we will also denote this method as the *fallback-x* method, where x corresponds to the number of attempts.

4. *Half method* – It creates half of the initial generation by the *SGS* method, and the second half of the generation is created by the *sequential* method. This time, each schedule is attempted only once, and hence, the number of chromosomes in the initial generation can be lower in the case of this method. Nevertheless, the number of chromosomes in the generation will be repaired in the first iteration of the GA.

Based on the way the schedules are created, the makespan of schedules created by the *SGS* method should be noticeably lower compared to schedules created by the *sequential* method. On the other hand, there is no guarantee that the initial generation will be constructed in a small number of schedules in the case of the *SGS* method. Therefore, we proposed the other two methods that combine these two methods.

5.1.4 Parent Selection

Parent selection works in the same way as in the GANS algorithm. N_p pairs of parent chromosomes are selected from each generation G. Father chromosomes are selected using a roulette wheel selection method with respect to the objective values of the chromosomes in the current generation. Mother chromosomes are selected likewise, but the improvement I_{obj} achieved by the NS operator is used instead of the objective value.

5.1.5 Crossover Operator

Given two parent chromosomes C_m and C_f, the crossover operator produces two child chromosomes $(C_m.a_{core}, C_m.I_{obj}, C_f.S, C_f.A^o)$ and $(C_f.a_{core}, C_f.I_{obj}, C_m.S, C_m.A^o)$. The NS operator is then applied to these child chromosomes to find and possibly update a better schedule and the order of activities. In any case, the value of the I_{obj} gene is updated. It is either set to the objective value improvement obtained by applying the NS operator or to 0 if there is no improvement.

5.1.6 Mutation Operator

After the crossover operator, a mutation operator is applied to the child chromosomes. It adds randomness to the GA and aims to avoid getting stuck in a local optimum solution. Specifically, with a small probability p_{mut}, a new core activity is randomly chosen and set to the chromosome. In such a case, the NS operator is applied to the new chromosome with this core activity.

Compared to the GANS algorithm, we do not mutate the search direction gene because we use only a forward search direction in the NS operator.

5.1.7 Next Generation Selection

At the end of each iteration of the GA, the next generation of M chromosomes is selected from the child chromosomes. Since some child chromosomes can be duplicated in the generation, we transform them into a set of candidate chromosomes $\mathcal{C}_{candidate}$ (i.e., unique chromosomes without duplicities). The first N_t chromosomes with the lowest objective value are selected directly to the next generation. The rest $M - N_t$ chromosomes are chosen from $\mathcal{C}_{candidate}$ using a roulette wheel selection with respect to the objective value of the chromosome. This roulette wheel selection is the same as the father chromosome selection in Section 5.1.4.

5.1.8 Stop Condition

The proposed GA stops when a given number of schedules λ is produced during the search. New schedules are produced during the initial generation creation and the application of the NS operator (in the crossover and eventually mutation operators). In an iteration of the NS operator, schedules are produced while solving the rescheduling sub-problem and during the left shift. Hence, one application of the NS operator to a chromosome can produce up to $2I$ schedules.

Compared to the GANS algorithm, we do not use a condition based on the number of successive generations without an improvement as a stopping criterion.

Note that the termination condition is checked at any step where a new schedule is created in the GA. It means the algorithm does not have to finish the whole iteration and stops immediately when the given number of schedules is produced.

5.1.9 Parameters

As indicated earlier, the GA has several parameters that have to be specified. In the experiments, we used the same parameter values as the GANS algorithm. For clarity, the parameters of the implemented GA and their values are summarized in Table 5.1.

Notation	Meaning	Value
N_r	The number of activities to be rescheduled in each sub-problem (it depends on the number of activities n in the problem)	$0.2n$
I	The maximum number of iterations in one application of the NS operator	10
M	The number of chromosomes in each generation	40
N_p	The number of parent pairs in each generation	20
p_{mut}	The probability of applying the mutation operator	0.02
N_t	The number of child chromosomes that are directly selected into the next generation	8
λ	The maximum number of schedules to be produced	50000

Table 5.1: The parameter settings of the implemented GA used in the experiments. Most parameters have the same notation and values as the parameters of the GANS algorithm in [19].

5.2 Mode Selection Model

The crucial part of the implemented GA is solving the rescheduling sub-problem in the NS operator. In the sub-problem, modes of activities that are rescheduled are assigned at random. This randomization creates diverse individuals in a generation, which is an important aspect of the GA. However, such sub-problem solutions often lead to infeasible schedules. Thus, we decided to try to create a machine learning model that would predict the mode of the activity based on its characteristics. In this section, we will describe how we trained the model and how it selects the most suitable mode for an activity in the rescheduling sub-problem.

Specifically, we assume there are some characteristics of sub-problems shared within different-sized instances. Hence, we used the optimal solutions of small instances (found by an exact approach) to train the model to learn these characteristics and to learn to select more suitable modes on the larger instances.

Firstly, we start with the creation of the dataset for the model. From an optimal schedule of an instance, several samples are created. In particular, we create several different sub-problems from an instance by randomly selecting different core activities. Having a sub-problem, all activities in the sub-problem are gradually scheduled one by one (using the SGS, exactly the same way it is done in the NS operator in the GA), and a sample is created for each activity and its mode in the optimal schedule. A data sample consists of a pair of input and target feature vectors, which will be described in the following paragraphs.

Since the number of modes can differ and the characteristics of a mode do not have to correspond to the mode identifier, we trained the model for the regression task instead of classification. In particular, we defined a three-dimensional feature vector that characterizes a certain mode of activity. To select a mode for an activity, this vector is computed for each possible mode of the activity, and the mode corresponding to the vector with the lowest Euclidean distance from the predicted target vector is selected.

The individual target features of activity a_i and mode m are:

1. The duration of activity a_i in mode m divided by the maximum duration over all the modes of activity – i.e., $\frac{d_{im}}{\max_{m' \in M_i} d_{im'}}$
2. Energy consumed by activity a_i in mode m divided by the maximum energy consumption over all the activity modes – i.e., $\frac{c_{im} d_{im}}{\max_{m' \in M_i} c_{im'} d_{im'}}$
3. The maximum ratio of resources required by activity a_i in mode m and resource capacity over all the resource types – i.e., $\max_{k \in \{1,\dots,K\}} \frac{r_{imk}}{R_k}$

Similarly, we defined the input feature vector that characterizes the sub-problem and the activity. We tried several different features (some of them worked better, some of them worked worse). In the final model that is evaluated in Section 7.2.3, the input vector consisting of the following 8 features[1] was used. Nevertheless, we do not claim this is the best possible configuration of features, and on the contrary, we believe better features exist.

1. The maximum minus the minimum duration of activity a_i over all its modes – i.e., $\max_{m \in M_i} d_{im} - \min_{m \in M_i} d_{im}$
2. The size of the window in which activity a_i can be scheduled (determined by its earliest start time (EST_i) and latest finish time (LFT_i)) divided by the minimum window size of yet unscheduled activities A^u in the sub-problem – i.e., $\frac{LFT_i - EST_i}{\min_{a_j \in A^u} LFT_j - \max_{a_j \in A^u} EST_j}$
3. The number of direct successors of activity a_i in the sub-problem divided by the number of yet unscheduled activities A^u in the sub-problem – i.e., $\frac{|succs(a_i)|}{|A^u|}$
4. The maximum minus the minimum of the maximum resource consumption ratio of activity a_i ($ratio_k = \max_{m \in M_i} \frac{r_{imk}}{R_k}$) over all the resource types – i.e., $\max_{k \in \{1,\dots,K\}} ratio_k - \min_{k \in \{1,\dots,K\}} ratio_k$
5. The maximum energy consumption of activity a_i in a single quarter over all its modes (where d^q_{im} denotes the duration of activity a_i in mode m in a single quarter) divided by the average available quarter energy consumption capacity in the sub-problem (denoted $\overline{C^{available}_q}$) – i.e., $\frac{\max_{m \in M_i} c_{im} d^q_{im}}{\overline{C^{available}_q}}$

[1]Note that depending on sub-problem characteristics in a sample, the denominator of some features can be 0 or negative. In such a case, the feature is set to 0. For clarity, we omit this in their description.

6. The maximum energy cost of activity a_i (i.e., the maximum electricity price p^{max} in the sub-problem times its maximum consumption over all its modes) divided by the available energy budget (denoted $B^{available}$) in the sub-problem – i.e., $\frac{p^{max} \max_{m \in M_i} c_{im} d_{im}}{B^{available}}$

7. The maximum ratio of the average consumption of resource k by all yet unscheduled activities A^u and their modes in the sub-problem (denoted $\overline{r_k}$) and the resource capacity over all the resource types – i.e., $\max_{k \in \{1,\dots,K\}} \frac{\overline{r_k}}{R_k}$

8. The maximum energy cost of all yet unscheduled activities A^u (i.e., the maximum electricity price p^{max} in the sub-problem times the sum of their maximal consumption) divided by the available energy budget (denoted $B^{available}$) in the sub-problem – i.e., $\frac{p^{max} \sum_{a_j \in A^u} \max_{m \in M_j} c_{jm} d_{jm}}{B^{available}}$

Our multi-output regression model is a simple neural network, implemented in PyTorch framework[2], consisting of 3 linear layers interleaved with ReLU layers. The input size of the network is 8, the output size is 3, and the size of hidden layers is 32. The network was trained on the training set (80 % of the dataset) using an Adam optimizer[3] with a learning rate of 0.001, batch size of 32, and mean square error loss[4] function for 200 epochs.

The resulting model is the model with the lowest loss on the validation set (10 % of the dataset) during the training. The model weights were stored in a file, and this model is used in the evaluation in Section 7.2.3.

Nevertheless, to preserve some randomness that is vital for the NS operation in GA to fulfill its purpose, this model is used only with a certain probability p_{model}, which is a parameter of this model. Otherwise, the original mode assignment procedure (i.e., a random activity mode is assigned) is used in the GA.

[2] `https://pytorch.org/`

[3] `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`

[4] `https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html`

Chapter 6
Dataset

Since we defined new problems, we cannot easily use some existing datasets, but we need to create new instances of our problems. We will use two standard benchmark datasets for MMRCPSP and extend them to create instances of our problems. More details about these datasets and how we transform them into new ones are provided in this chapter.

6.1 MMRCPSP Datasets

In the case of MMRCPSP, there are three standard benchmark datasets (*PSPLIB*, *Boctor*, and *MMLIB*), which are used in most of the research work.

PSPLIB [37] is a well-known project scheduling library that contains many datasets for RCPSP and also for several of its extensions. It has been a standard benchmark dataset in project scheduling research for decades. For MMRCPSP, there are seven sets of instances differing in the number of activities from 10 to 30 activities.

The Boctor dataset [38] is a less popular dataset consisting of 2 subsets of instances with 50 or 100 activities and a different number of renewable resources and modes. Each subset contains only 120 instances, which is significantly less than 640 instances in PSPLIB sets. Moreover, non-renewable resources are not considered in these instances.

Nevertheless, some shortcomings were observed in both these older datasets and motivated the introduction of a newer benchmark dataset MMLIB [14] with larger instances. It contains two sets of instances, MMLIB50 and MMLIB100, with 50 and 100 activities, respectively. These instances have similar properties to PSPLIB instances. In addition, there is a special set MMLIB+ that contains instances with a higher number of resources and modes.

6.2 Our Dataset

As we have indicated, to create instances of our problems, we extended instances from the aforementioned standard MMRCPSP datasets. Namely, we used instances from PSPLIB and MMLIB datasets because they are more popular and contain a lot of instances of different scales.

Compared to MMRCPSP instances, instances of our problems need to be extended with energy consumption of activities per time unit, electricity price, energy consumption limits for each 15-minute window, and an energy budget in case of the problem with the hard energy budget constraint. This section describes how we decided to determine these values and what instance sets we created for our experiments.

6.2.1 Project Horizon

Firstly, we start with the definition of the project horizon. In MMRCPSP, the project horizon is the length of a sequential schedule, where activities are placed one after the other, and the mode with the longest duration is selected for each activity. In our problem, there are energy consumption limits for 15-minute windows that need to be taken into account, too. Nevertheless, we assume and will set these consumption limits so that each activity can be scheduled in any 15-minute time window. But perhaps only one activity will be able to be scheduled at a single time unit. This assumption ensures that even the two activities with the highest consumption can be placed directly one after the other at any time unit without exceeding the consumption limits. Hence, the project horizon T in our problem, expressed in Eq. (6.1), is the same as in MMRCPSP (i.e., the sum of maximum duration of each activity).

$$T = \sum_{a_i \in A} \max_{m \in M_i} d_{im} \tag{6.1}$$

6.2.2 Energy Consumption of Activity

The energy consumption of activity a_i in a particular processing mode m is derived from its resource[1] requirements r_{imk} of each resource type k and the duration d_{im}. The consumption of activity is the weighted sum of resource requirements where each resource type k has a different weight w_k. To obtain energy consumption in a single time unit, expressed in Eq. (6.2), which is needed in our problem, this consumption is divided by the duration. Note that only the integer part of the number is used to work with integer values. For simplicity, the consumption is the same in each time unit t. Therefore, the result energy consumption of an activity is the product of the duration d_{im} and energy consumption in a single time unit c_{im}.

$$c_{im} = \left\lfloor \frac{\sum_k r_{imk} w_k}{d_{im}} \right\rfloor \tag{6.2}$$

[1]MMRCPSP instances contain information about both renewable and nonrenewable resource requirements and capacities. Since only renewable ones are considered in our problems, the information about nonrenewable resources is not considered in creating new instances.

This choice allows us to have different energy consumption in individual modes. Moreover, it corresponds to real situations because modes that require fewer resources have lower energy consumption. And if we are interested in low energy consumption, it can be better to process the activity in a longer but more economical mode.

6.2.3 Electricity Price

As we have mentioned, we assume the electricity price is given by TOU tariffs in individual hours. Particularly, we defined an electricity price distribution during a single day (Fig. 6.1) that includes high on-peak and low off-peak prices and some transitional hours with prices in between. The price distribution is cyclic, so in case the project schedule is longer than one day, it repeats from the start for subsequent hours. In order to ensure all instances do not start at the same hour, we introduced an offset parameter h_0 (a random integer from interval $[0, 23]$) for each instance. The resulting electricity price in a problem instance is then shifted by this offset, which allows us to model that projects can start at any hour.

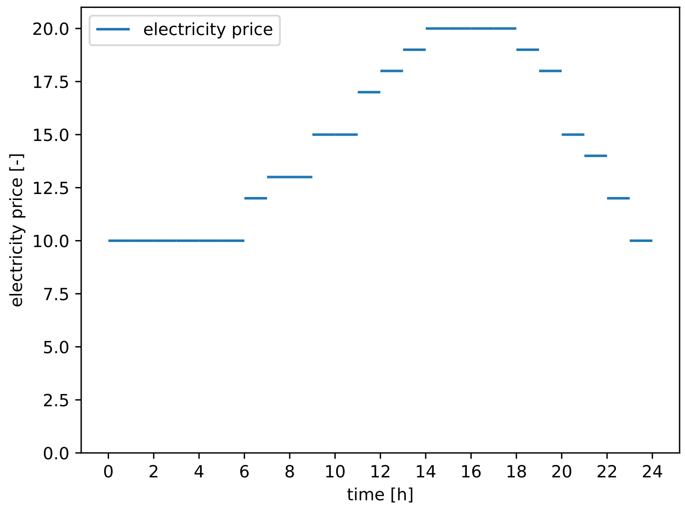


Figure 6.1: Electricity price distribution during a single day.

6.2.4 Energy Budget Limit

Regarding energy budget and energy consumption limits, we used a similar approach to the resource-strength coefficient, which was used in [37] to determine resource capacities. This coefficient allowed the authors to specify how strong the resource constraints are. Therefore, we also defined the lower and upper bounds of these limits and used coefficients to regulate the strength of the constraints.

In the case of energy budget limit, we defined the lower/upper thresholds B^{min}/B^{max} as the maximum electricity price over the whole project horizon $p^{max} = \max_{t \in \{0,\dots,T\}} p(t)$ times the total minimum/maximum energy consumption of all activities (Eqs. (6.3) and (6.4)). Maximum electricity price is used in both cases to ensure the precedence relations between activities would not cause the infeasibility of the project. The result energy budget limit is then computed from these two bounds and an energy budget coefficient $\alpha_B \in [0,1]$ according to Eq. (6.5).

$$B^{min} = p^{max} \sum_{a_i \in A} \min_{m \in M_i} c_{im} d_{im} \tag{6.3}$$

$$B^{max} = p^{max} \sum_{a_i \in A} \max_{m \in M_i} c_{im} d_{im} \tag{6.4}$$

$$B = \left\lfloor (1 - \alpha_B) B^{min} + \alpha_B B^{max} \right\rfloor \tag{6.5}$$

6.2.5 Energy Consumption Limits

We likewise define the lower/upper bounds for energy consumption limits. Before we specify them, we introduce a parameter u^q, which specifies the number of time units in a single 15-minute time window. The lower bound C^{min}, expressed in Eq. (6.6), is the biggest minimum consumption in a single 15-minute time window out of all activities where the duration of an activity is assumed to be the whole 15-minute time window. It guarantees that an arbitrary activity can be scheduled (at least in the most economical mode) in any 15-minute time window and that any two activities can be scheduled sequentially right behind each other. The upper bound C^{max}, expressed in Eq. (6.7), is the sum of maximum consumption in a 15-minute time window over all the activities. In other words, C^{max} is a value of limit that would allow all of the activities to be executed in the same time window. These bounds are the same for all 15-minute time windows. Similarly to the energy budget limit, the energy consumption limit C_q for each time window $q \in Q$ is constructed from these bounds and the energy consumption coefficient $\alpha_C \in [0,1]$ according to Eq. (6.8).

$$C^{min} = \max_{a_i \in A} \min_{m \in M_i} c_{im} u^q \tag{6.6}$$

$$C^{max} = \sum_{a_i \in A} \max_{m \in M_i} c_{im} \min(d_{im}, u^q) \tag{6.7}$$

$$C_q = \left\lfloor (1 - \alpha_C) C^{min} + \alpha_C C^{max} \right\rfloor, \quad \forall q \in Q \tag{6.8}$$

6.2.6 Created Instance Sets

Finally, when we have defined all the parameters of problem instances, we can generate new instances of our problems. Specifically, as indicated earlier, instances of our problem were created from j10 and j30 instances from the PSPLIB [37] dataset and MMLIB50 and MMLIB100 instances from the MMLIB [14] dataset.

The parameters that were used in the generation of a new dataset were set in the following way. Both the energy budget coefficient α_B and energy consumption coefficient α_C were set randomly to one value from the set $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ for each instance. These coefficient values ensure we have instances with both less restrictive and more restrictive limits. As mentioned earlier, the hour offset parameter h_0 is a random integer from interval $[0, 23]$. To model projects of larger size that are longer than a few hours, for each selected standard set of instances, we created two different instance sets differing in the number of time units in a 15-minute time window (parameter u^q). The set with $u^q = 15$ assumes that a single time unit corresponds to 1 minute, whereas the other set with $u^q = 1$ assumes that a single time unit corresponds to 15 minutes. For clarity, the domains of individual instance parameters are summarized in Table 6.1.

The characteristics of the created datasets, which will be used for evaluation in Chapter 7, are shown in Table 6.2.

parameter	**domain**
α_B	$\{0.1, 0.2, 0.3, 0.4, 0.5\}$
α_C	$\{0.1, 0.2, 0.3, 0.4, 0.5\}$
h_0	$[0, 23]$
u^q	$\{1, 15\}$

Table 6.1: The domains of individual problem instance parameters that were used in the generation of a new dataset.

name	**# instances**	$\|A\|$	$\|M\|$	K	u^q
j10_1min	536	10	3	2	15
j10_15min	536	10	3	2	1
j30_1min	640	30	3	2	15
j30_15min	640	30	3	2	1
MMLIB50_1min	540	50	3	2	15
MMLIB50_15min	540	50	3	2	1
MMLIB100_1min	540	100	3	2	15
MMLIB100_15min	540	100	3	2	1

Table 6.2: The characteristics of individual problem instances in the used datasets. Columns $|A|$, $|M|$, K, and u^q show the number of non-dummy activities, modes, type of resources, and the number of units in a 15-minute window in the dataset, respectively.

Chapter 7

Evaluation

In this chapter, we will experimentally evaluate all the proposed approaches on the aforementioned datasets of different sizes. Firstly, the configuration of experiments will be described in Section 7.1. The measured data will be analyzed in Section 7.2, and the results of the experiments will be summarized in Section 7.3.

7.1 Configuration of Experiments

All the proposed approaches were implemented in Python 3.10. Python was chosen because it is a very popular programming language with many libraries (e.g., for data analysis or machine learning) that enable fast implementation. To solve the CP models, the CP solver *IBM ILOG CP Optimizer* 22.1.0[1] was used.

All the experiments were run on the *optim* server operated by the *Czech Institute of Informatics, Robotics and Cybernetics at the Czech Technical University in Prague*[2]. Specifically, one core of Intel(R) Xeon(R) Silver 4110 CPU @ 2.10 GHz processor was used.

In terms of the stopping criteria, as indicated earlier, we use the number of created schedules as a stopping criterion for the genetic algorithm. Compared to the time limit, the number of created scheduled during the search is often the preferred stopping criterion in the literature (e.g., in [16], [19] or [14]) because it is not dependent on the selection of the programming language and the hardware on which the experiments are run. Namely, we set the limit (parameter λ) to 50000 produced schedules in the experiments. For the CP models, the time limit was set to 300 seconds. In the case of warm-started CP, the time limit given to the CP solver is decreased by the time needed to find the warm-start schedule t_{ws} (i.e., the limit is $300 - t_{ws}$ seconds) to provide a fair comparison of the CP and warm-started CP.

In terms of the performance of algorithms, the usual metric in RCPSP is the *Average Percent Deviation (APD)* from a lower bound. Hence, in the case of the problem with the hard energy budget constraint, where the project makespan is minimized, we will observe the APD from the critical

[1]`https://www.ibm.com/docs/en/icos/22.1.0?topic=cp-optimizer`
[2]`https://www.ciirc.cvut.cz/en/`

path lower bound (denoted $criticalPath_{LB}$). This lower bound is the length of the longest path from the first activity to the last activity in the precedence graph, where the length of each activity a_i is the minimum duration over all possible modes of activity $\min_{m \in M_i} d_{im}$. In the case of the problem with total energy cost in the objective, we created a lower bound of the objective value (denoted $weightedCriterion_{LB}$) based on the values of parameters α and β used in the experiments (see Section 7.1.1), and we will compare the methods with respect to the APD from this lower bound.

7.1.1 Scenarios of Problem with Total Energy Cost in Objective

In the problem with total energy cost in the objective, defined in Section 3.2, the criterion contains parameters α (the weight of the project makespan criterion), and β (the weight of the total energy cost criterion), which have to be specified in advance before the optimization. In practice, these parameters would be most likely set by the managers based on their concrete case. For instance, the weighted criterion can correspond to the overall price of the project, where the cost of one hour of production would be involved in the parameter α.

Nevertheless, in the experiments, we are using benchmark datasets with artificial values. Therefore, we defined three scenarios with different weights of the individual criteria to be able to investigate the effects of different weights on the proposed approaches. In addition, the individual criteria are divided by the corresponding lower bounds $criticalPath_{LB}$ and

$$energyCost_{LB} = p^{min} \sum_{a_i \in A} \min_{m \in M_i} c_{im} d_{im}, \tag{7.1}$$

where $p^{min} = \min_{t \in \{0,\dots,T\}} p(t)$ is the minimum electricity price over the whole project horizon, so that both criteria have values of a similar scale. Specifically, the scenarios represent the cases where the weight of the project makespan criterion is notably higher ($scenario_{C_{\max}}$), the weights of both criteria are equal ($scenario_{equal}$), and the weight of the total energy cost is notably higher ($scenario_{energy}$). The concrete values of parameters α and β used in the experiments in respective scenarios are summarized in Table 7.1.

scenario	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$
$scenario_{C_{\max}}$	$\frac{3}{criticalPath_{LB}}$	$\frac{1}{energyCost_{LB}}$
$scenario_{equal}$	$\frac{1}{criticalPath_{LB}}$	$\frac{1}{energyCost_{LB}}$
$scenario_{energy}$	$\frac{1}{criticalPath_{LB}}$	$\frac{3}{energyCost_{LB}}$

Table 7.1: Scenarios of the problem with total energy cost in the objective with the concrete values of parameters α and β used in the experiments.

Since the weight parameters in all scenarios contain lower bounds of both involved criteria, we can derive a lower bound of the weighted criterion (denoted $weightedCriterion_{LB}$) and use it in the experiments. This lower bound is computed as $\alpha\ criticalPath_{LB} + \beta\ energyCost_{LB}$. In particular, the values of $weightedCriterion_{LB}$ are equal to 4 in the case of $scenario_{C_{\max}}$ and $scenario_{energy}$ scenarios and equal to 2 in the case of $scenario_{equal}$ scenario.

7.2 Experiments

After introducing the configuration of the experiments, we can evaluate them. In Section 7.2.1, we will start with the optimality of solutions found by the exact approaches. Different methods of initial generation creation and the use of the proposed mode selection model in the GA will be analyzed in Sections 7.2.2 and 7.2.3, respectively. After that, all proposed approaches will be compared in Section 7.2.4.

7.2.1 Optimality of CP Models

In the case of exact approaches, we compare the performance of the CP and warm-started CP models with respect to the optimality of the found solutions in a given time limit.

Before the comparison, the warm-started CP needs a warm-start schedule. As indicated earlier, the schedule with the lowest objective value found by a GA is given to the warm-started CP model. Particularly, the stopping criterion to obtain the warm-start schedule was set to 1000 generated schedules, and a *"half"* method of initial generation creation (Section 5.1.3) was used to generate the initial generation of the GA. We chose this method because it seemed to be the most efficient one in the early phase of the GA in the majority of problems (see results in Section 7.2.2).

In the experiments, we ran CP and warm-started CP models for 300 seconds on all problem instances in a dataset. The percentage of instances in which an optimal solution was found is shown in Table 7.2 for problems with the hard energy budget constraint and in Table 7.3 for problems with total energy cost in the objective.

In the case of the problems with the hard energy budget constraint (Table 7.2), the results show that for the datasets with 10 activities (j10_1min and j10_15min), optimal solutions were found in all instances. Moreover, the average time to solve an instance was around 2 seconds. For the datasets with larger instances, there can be seen the gain achieved by warm-starting the CP model. In all these datasets, warm-started CP found considerably more optimal solutions. The biggest improvement was obtained for the dataset MMLIB100_15min, where the warm-started CP found 69.44 % of optimal instances compared to 18.52 % found by the CP model without warm-starting. This substantial improvement is largely caused by the huge reduction in the number of variables in the models (on average, 232627 variables in case of CP is reduced to 7875 variables in case of warm-started CP).

In terms of feasibility, except for 4 instances in the MMLIB100_15min dataset (less than 1 % of instances), in which CP model did not find any

dataset name	CP	ws-CP
j10_1min	100.00	100.00
j10_15min	100.00	100.00
j30_1min	69.38	76.41
j30_15min	83.44	88.44
MMLIB50_1min	37.96	50.00
MMLIB50_15min	67.41	78.70
MMLIB100_1min	6.67	33.15
MMLIB100_15min	18.52	69.44

Table 7.2: The optimality of found solutions (in %, rounded to 2 decimal places) by CP and warm-started CP (ws-CP) models in 300 seconds in case of problems with the hard energy budget constraint.

feasible solution, feasible schedules were found in all instances in the given time limit.

Furthermore, generally speaking, a higher percentage of optimal solutions were found in instances with 15-minute time granularity. We suppose this result is caused mainly by the 15-minute energy consumption limits. In the case of 15-minute time granularity, this limit is covered by a single time unit, whereas 15 time units are involved in the limit in the case of 1-minute time granularity. In addition, as expected, the larger the instance is, the lower the percentage of found optimal schedules is.

In the case of the problem with total energy cost in the objective, the experiments were run in all three scenarios but only in datasets with a 1-minute time granularity to have a reasonable number of experiments. We selected these datasets because they seem to be slightly more difficult compared to datasets with 15-minute time granularity. The results (Table 7.3) show that all instances with 10 activities were solved optimally, and the models are more efficient on smaller instances.

	$scenario_{C_{\max}}$		$scenario_{equal}$		$scenario_{energy}$	
dataset name	**CP**	**ws-CP**	**CP**	**ws-CP**	**CP**	**ws-CP**
j10_1min	100.00	100.00	100.00	100.00	100.00	100.00
j30_1min	67.50	67.50	63.75	64.22	45.62	47.66
MMLIB50_1min	26.11	24.81	8.33	7.59	9.44	10.93
MMLIB100_1min	0.00	0.00	0.00	0.00	0.00	0.00

Table 7.3: The optimality of found solutions (in %, rounded to 2 decimal places) by CP and warm-started CP (ws-CP) models in 300 seconds in case of problems with total energy cost in the objective for different scenarios.

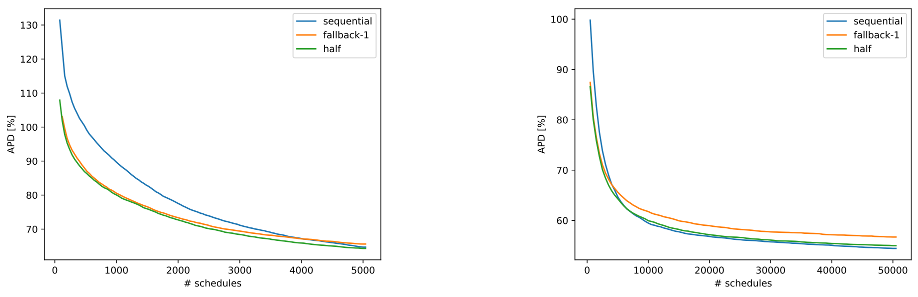
However, the effect of warm-starting is not as significant as in the case of problems with the hard energy budget constraint. This is mainly because the number of variables in the model is not reduced when a warm-start solution is given due to a different optimization criterion. In some scenarios of MMLIB50_1min datasets, the percentage of found optimal solutions was even slightly higher for the CP without warm-starting. Nevertheless, it is also important to mention that CP without warm-starting did not find any feasible solution in the given time limit in roughly half of the instances of the MMLIB100_1min dataset in all three scenarios, which was not the case for the warm-started model, which provided a feasible solution in all instances.

Furthermore, there is a noticeable difference between individual scenarios. A higher percentage of optimal solutions was found in the $scenario_{C_{\max}}$ scenario, where the makespan has a higher weight, and vice-versa, a lower percentage of the optimal solution was found in the other two scenarios, where the weight of the makespan is lower. We suppose this is because the CP solver works better with this standard scheduling criterion.

7.2.2 Effect of Initial Generation in GA

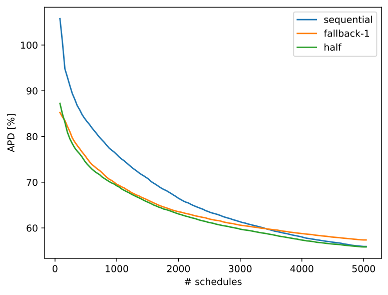
In Section 5.1.3, we proposed and described 4 different methods of how to create an initial generation in our GA. In this section, we will evaluate their effect on the performance. As indicated earlier, we will use the APD from the lower bound to evaluate the performance of the algorithm.

How the APD evolves in the number of generated schedules is shown in Fig. 7.1 for problems with the hard energy budget constraint and in Fig. 7.2 for different scenarios of problems with total energy cost in the objective. Figures show the performance of GA on the MMLIB50_1min dataset. The measured data are shown with the different number of schedules in two sub-figures to be able to observe the GA's performance both in the first iterations of GA (5000 schedules) and during the whole search (50000 schedules). For the clarity of figures, only three initial generation creation methods – *"sequential"*, *"fallback-1"*, and *"half"* are shown in the figures. The *"SGS"* method is excluded from the comparison since it did not even find any feasible schedule in 50000 generated schedules in almost 10 % of instances. Other variants of a fallback method (*"fallback-10"* and *"fallback-100"*) are also omitted since they perform similar or worse than the *"fallback-1"* method (Fig. C.1).

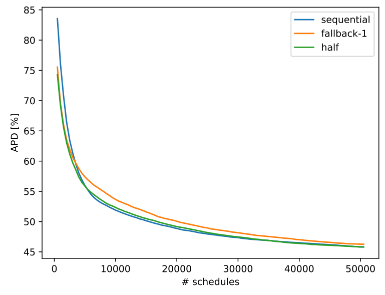


(a) : 5000 schedules **(b) :** 50000 schedules

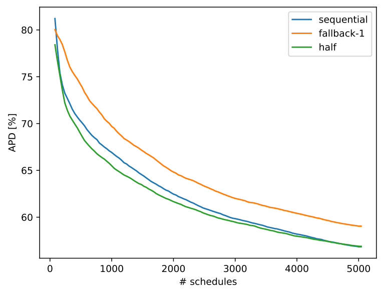
Figure 7.1: APD from the $criticalPath_{LB}$ (in %) in the number of generated schedules for different methods of initial generation creation on MMLIB50_1min instances of the problem with the hard energy budget constraint.



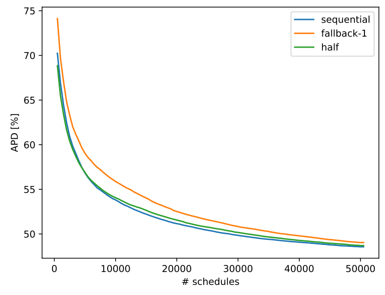
(a) : $scenario_{C_{\max}}$ – 5000 schedules



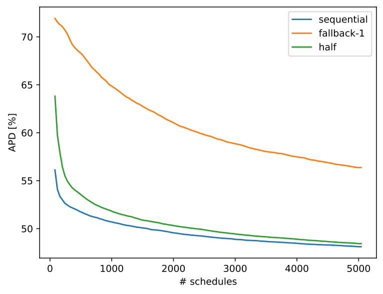
(b) : $scenario_{C_{\max}}$ – 50000 schedules



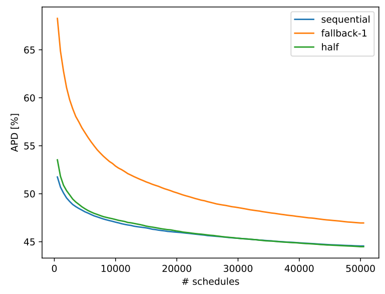
(c) : $scenario_{equal}$ – 5000 schedules



(d) : $scenario_{equal}$ – 50000 schedules



(e) : $scenario_{energy}$ – 5000 schedules



(f) : $scenario_{energy}$ – 50000 schedules

Figure 7.2: APD from the $weightedCriterion_{LB}$ (in %) in the number of generated schedules for different methods of initial generation creation on MMLIB50_1min instances of the problem with total energy cost in the objective.

The measured data in the case of both problems (Figs. 7.1 and 7.2) show that the highest improvements in the objective are achieved in the early phase of the GA. Moreover, the APD of the *“sequential”* method is notably higher compared to the other methods in the early phase. This is definitely caused by the way how the *“sequential”* method constructs the initial schedules. Nevertheless, in the 50000 produced schedules, the *“sequential”* method reaches similar or slightly lower APD compared to the *“half”* method, and both these methods have significantly lower APD compared to the *“fallback-1”* method.

Furthermore, Fig. 7.2 indicates that the lower the weight of the makespan criterion is, the better the performance of the *“sequential”* method is in the early phase. In the case of $scenario_{energy}$, it even dominates over the other two methods.

To sum up, the best performance in the 50000 generated schedules was achieved by the *"sequential"* method of initial generation creation in all experiments. Slightly worse results were obtained by the *"half"* method. Finally, a *"fallback"* method had the worst performance. Nevertheless, the *"sequential"* method does not dominate other methods during the whole search. Specifically, in the lower number of schedules, solutions of higher quality can be obtained by the *"half"* method. That is why we used this method to find a warm-start schedule for the warm-started CP in the other experiments.

7.2.3 Effect of Mode Selection Model in GA

In Section 5.2, we proposed a mode selection model that selects a mode of an activity based on the activity and sub-problem characteristics. This model aims to improve mode assignments in the sub-problem in the NS operator and thus also to improve the GA. Recall that in our base GA implementation, modes are assigned randomly. As mentioned earlier, randomness is a vital factor of the GA, and hence, we believe some randomness should also be preserved when a mode selection model is used. Therefore, in this section, we will compare the performance of the base GA without a model with the performance of the GAs, where the trained mode selection model is used with a certain probability.

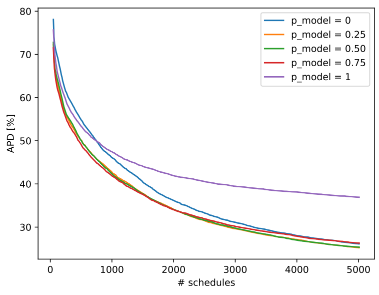
Since the proposed mode selection model is rather a prototype, the model was trained only for the problem with the hard energy budget constraint. Therefore, all experiments presented in this section were performed for this problem. In particular, The model was trained on the optimal solutions of instances in the j10_15min dataset, which were found by warm-started CP. The details about this model, including the hyperparameters and the training phase, were described in Section 5.2.

In the experiments, we run a GA with the trained mode selection model with different model probabilities $p_{model} \in \{0, 0.25, 0.5, 0.75, 1\}$. The model probability p_{model} is a probability with which the mode assignment is done by the model (i.e., the mode of activity is assigned by the model with probability p_{model} and at random with probability $1 - p_{model}$). Note that the case when $p_{model} = 0$ is the base variant without the mode selection model, where all modes are assigned randomly.

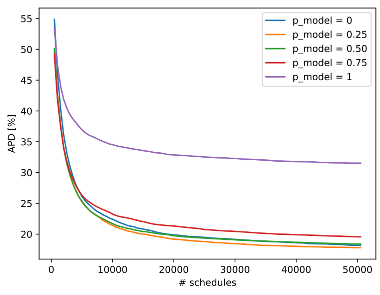
The results obtained by the GA with different model probabilities on datasets with larger instances MMLIB50_15min and MMLIB100_15min are shown in Fig. 7.3. Same to previous GA comparisons, the figure shows the measured APD from the $criticalPath_{LB}$ in the number of produced schedules. The *"half"* method of initial generation creation was used in the case of all variants since we observed the mode selection model improves the GA mainly in the early phase, where this method outperforms the other ones (see Section 7.2.2).

The measured data support our conjecture that randomness is a crucial part of the GA and when there is no randomness in the mode assignment

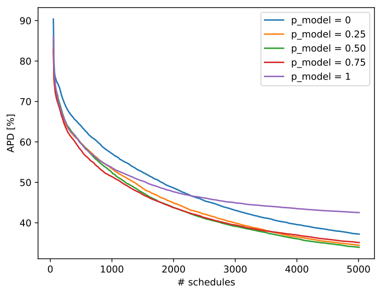
($p_{model} = 1$), with the increasing number of produced schedules, the APD stays significantly higher compared to all other variants, where the randomness is present. In other variants that combine the trained model with random mode assignment, it can be seen that the mode selection model can improve the GA performance in the early phase (their APD is notably lower than the APD of the GA without the model ($p_{model} = 0$)).



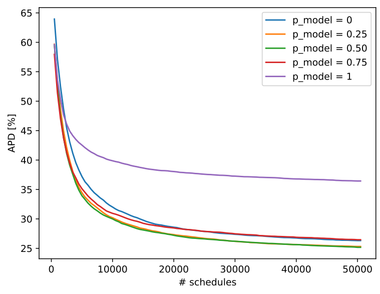
(a) : MMLIB50_15min, 5000 schedules



(b) : MMLIB50_15min, 50000 schedules



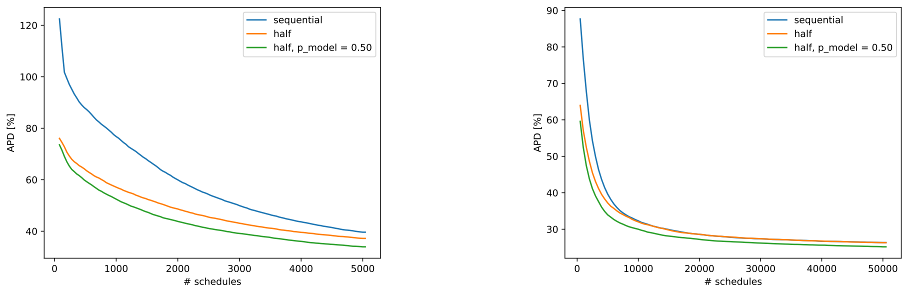
(c) : MMLIB100_15min, 5000 schedules



(d) : MMLIB100_15min, 50000 schedules

Figure 7.3: APD from the $criticalPath_{LB}$ (in %) in the number of generated schedules achieved by GA with the trained mode selection model and different model probabilities p_{model} on MMLIB50_15min and MMLIB100_15min datasets of the problem with the hard energy budget constraint.

With a gradual number of produced schedules, the difference between the GA without and with the model decreases. However, there are variants of model probabilities (p_{model} is 0.25 or 0.5) where the APD is lower also in 50000 number of generated schedules. To better see the effect of the trained mode selection model, in Fig. 7.4, we visualized the comparison of the base GA with the *"sequential"* method of initial generation creation, the base GA with the *"half"* method, and the GA with the *"half"* method and the best variant with mode selection model ($p_{model} = 0.5$) on the MMLIB100_15min dataset. It can be seen that the GA with the *"half"* method and the mode selection model outperformed both GAs without the mode selection model. For instance, the difference between the GA with *"half"* method and random mode assignment and the GA with *"half"* method and the mode selection model is about 4.5 % of APD in the 1000 schedules, and about 1 % of APD in the 50000 schedules.



(a) : 5000 schedules **(b) :** 50000 schedules

Figure 7.4: APD from the $criticalPath_{LB}$ (in %) in the number of generated schedules achieved by GAs without a mode selection model and the GA with the trained mode selection model on MMLIB100_15min dataset of the problem with the hard energy budget constraint.

7.2.4 Comparison of Proposed Approaches

To compare all three proposed approaches (CP, ws-CP, and GA), we visualized how the APD from the lower bound evolves in time[3]. Figure 7.5 shows this comparison on different scaled datasets of the problem with the hard energy budget constraint. Nonetheless, one should read these figures carefully because it is necessary to keep in mind that the genetic algorithm is implemented in Python[4], whereas the used CP optimizer is implemented in C++, and it is optimized for scheduling tasks.

Furthermore, the result APD from the $criticalPath_{LB}$ found by individual approaches in either 300 seconds or 50000 generated schedules on all datasets of the problem with the hard energy budget constraint are summarized in Table 7.4.

Except for the case of small-sized instances (Fig. 7.5a), Fig. 7.5 shows that it takes quite a lot of time (tens or hundreds of seconds) until the CP model finds some solution. After a feasible schedule is found, the model manages to relatively quickly find better solutions. Opposite behavior can be seen in the case of the GA, where the most notable improvements occur primarily in the early phase during the first generations. Therefore, it is reasonable that the

[3]In the figures showing this visualization, the measured APD values at a certain time are visualized only if a feasible solution was found in all instances in the dataset at that time by the respective algorithm. Therefore, the values of the CP model sometimes start later or are not present in the figure. And since the GA uses a different stopping condition, its visualized APD values stop at the time all instances in the dataset were computed.

[4]As mentioned in the introduction of this chapter, we chose Python due to the considerably faster implementation time compared to C++. The speed of the method, hence, reflects a trade-off between the implementation and computation time. Thus, the comparison is from the practical standpoint, where both implementation and computation time play a role.

warm-started CP that combines a relatively good warm-start schedule found by GA with the optimized CP search for scheduling outperformed both of them in the case when the project makespan is minimized.

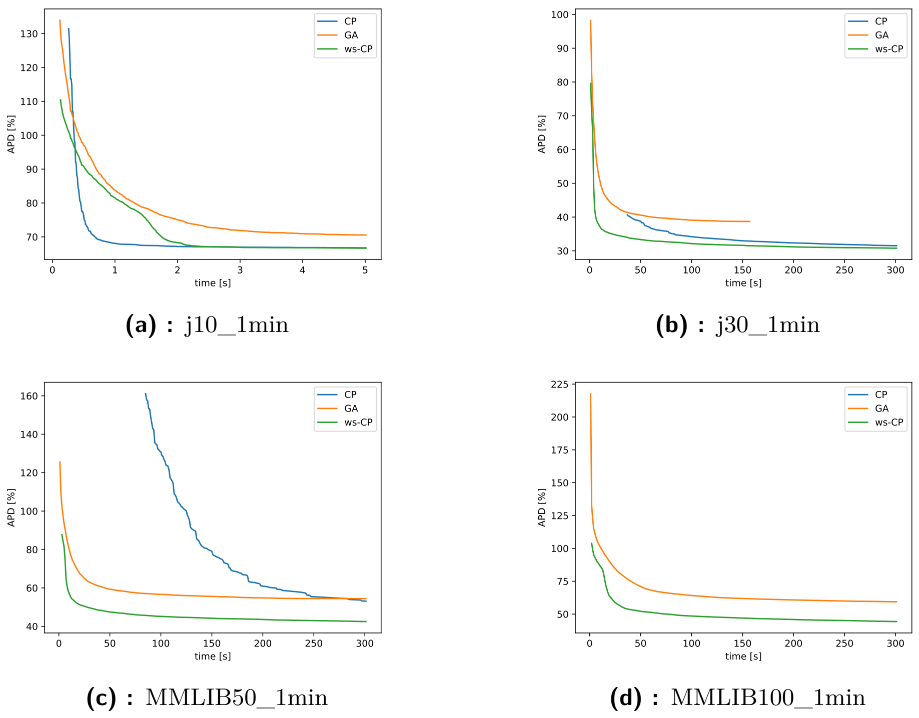


Figure 7.5: APD from the $criticalPath_{LB}$ (in %) in time for different approaches on selected datasets of the problem with the hard energy budget constraint.

dataset name	CP	ws-CP	GA
j10_1min	**66.48**	**66.48**	69.10
j10_15min	**30.49**	**30.49**	31.56
j30_1min	31.50	**30.78**	38.70
j30_15min	8.36	**7.75**	11.84
MMLIB50_1min	53.13	**42.49**	54.45
MMLIB50_15min	13.96	**9.82**	18.25
MMLIB100_1min	598.27	**44.33**	58.10
MMLIB100_15min	240.59	**12.61**	26.34

Table 7.4: APD from the $criticalPath_{LB}$ (in %, rounded to 2 decimal places) for individual algorithms on all datasets of the problem with the hard energy budget constraint. The lowest values for each dataset are bold.

Similar differences between the compared approaches were also observed on selected datasets in the case of the problem with total energy cost in the objective regardless of the scenario (Figs. C.2 to C.4). This time, the efficiency of the GA in the early phase is more significant and it takes more time until the other two approaches perform better. Warm-starting the CP model still improves the CP. However, its benefits on larger instances are not so big, and in the case of the MMLIB50_1min dataset, CP even outperformed warm-started CP in the last (tens of) seconds of the search. On the other hand, the CP without warm-starting results in significantly worse APD in the case of the MMLIB100_1min dataset and, more importantly, did not find any feasible solution in the given time limit in almost half of the instances of the problem with total energy cost in the objective. To provide the concrete values, the result APD from the $weightedCriterion_{LB}$ after either 300 seconds or 50000 generated schedules for individual approaches are summarized in Tables 7.5 to 7.7 for respective scenarios.

dataset name	**CP**	**ws-CP**	**GA**
j10_1min	**46.10**	**46.10**	46.92
j30_1min	27.19	**27.14**	29.46
MMLIB50_1min	**41.59**	43.47	45.82
MMLIB100_1min	142.59	82.30	**59.50**

Table 7.5: APD from the $weightedCriterion_{LB}$ (in %, rounded to 2 decimal places) for individual algorithms on selected datasets of the problem with total energy cost in the objective in the $scenario_{C_{\max}}$ scenario. The lowest values for each dataset are bold.

dataset name	**CP**	**ws-CP**	**GA**
j10_1min	**40.30**	**40.30**	40.88
j30_1min	**32.88**	**32.88**	34.69
MMLIB50_1min	**45.32**	46.74	48.56
MMLIB100_1min	158.34	83.39	**68.44**

Table 7.6: APD from the $weightedCriterion_{LB}$ (in %, rounded to 2 decimal places) for individual algorithms on selected datasets of the problem with total energy cost in the objective in the $scenario_{equal}$ scenario. The lowest values for each dataset are bold.

dataset name	**CP**	**ws-CP**	**GA**
j10_1min	**29.37**	**29.37**	29.66
j30_1min	**31.50**	31.51	33.47
MMLIB50_1min	**41.19**	41.42	44.56
MMLIB100_1min	143.69	73.07	**65.66**

Table 7.7: APD from the $weightedCriterion_{LB}$ (in %, rounded to 2 decimal places) for individual algorithms on selected datasets of the problem with total energy cost in the objective in the $scenario_{energy}$ scenario. The lowest values for each dataset are bold.

7.3 Summary of Experiments

From the measured results, we can conclude that different approaches are more suitable for different problem instances.

Specifically, for the small instances with 10 activities, the CP model is the best option since it finds an optimal solution in the order of a few seconds. Finding a warm-start schedule is counterproductive in this case. For the larger instances, the number of variables in the CP model increases, and its efficiency notably decreases. In such instances, the warm-started CP model starts to benefit from the provided warm-start schedule from the GA. It performs better than the CP model since it takes some time (tens of seconds in case of instances with 30 activities and notably more in case of larger ones) until a feasible schedule is found by the CP.

As the results in Section 7.2.4 indicate, the genetic algorithm is suitable mainly for middle and large-scale instances where the CP solver takes some time to find a feasible solution. Specifically, in Section 7.2.2, we saw that even when the initial generation is created in a naive way, like placing activities one by one in the case of the *"sequential"* method, GA is able to significantly improve the initial solutions in the first few generations.

However, the improvements in the solution quality in the later generations of the GA are not that big. Therefore, it can be reasonable to stop the GA after a few generations and try to improve the schedule by the CP solver (ws-CP), especially in the case of the problem with the hard energy budget constraint, in which a standard scheduling criterium ($C_{\max}$) is minimized.

In terms of the mode selection model in the GA, in Section 7.2.3, we saw that even though the trained model is a relatively simple neural network, the results showed that with the appropriate value of model probability p_{model} (e.g., 0.25 or 0.5 in our experiments), the GA with the mode selection model outperform the base GA with random mode assignment. Hence, it can improve the proposed GA, and it makes sense to further investigate the incorporation of more complex machine learning models to the GA.

All in all, from the results of both of our problems, we conclude that the problem becomes harder when the objective function becomes more complex, such as the weighted criterion in the case of the problem with total energy cost in the objective. Particularly, as we saw in Section 7.2.1, the percentage of optimal solutions found by the CP models was notably lower compared to the problem with hard energy budget constraint, even though the number of variables in both CP models was the same.

Chapter 8

Conclusion

In this thesis, we studied a Resource-Constrained Project Scheduling Problem (RCPSP) together with electrical energy consumption. The goal was to extend the standard RCPSP formulation with the constraints and criteria related to electrical energy with respect to existing literature and real-world production.

Since, to the best of our knowledge, there is not any RCPSP variant in the literature that would consider both hourly electricity prices and restrict energy consumption during regular time intervals, we decided to formulate a new problem variant. In particular, we focused on the Multi-Mode RCPSP (MMRCPSP), which assumes that an activity can be processed in one of the multiple alternative modes, and we extended each processing mode with energy consumption so that it varies depending on mode. In addition, we added the hourly Time-of-Use (TOU) electricity prices and maximum contracted energy limits during regular 15-minute time intervals to the problem. With this new extension of MMRCPSP, we formulated two similar problems differing in how the total energy cost is regarded. In the case of the problem with the hard energy budget constraint, the total energy cost is restricted by a hard limit and the project makespan is minimized. Whereas in the case of the problem with total energy cost in the objective, the hard budget limit is not considered, and the total energy cost is combined with the project makespan in the weighted linear combination criterion.

To solve these problems, we proposed and implemented both exact and metaheuristic approaches. Specifically, we created CP models using the CP solver IBM ILOG CP Optimizer [12]. As a metaheuristic approach, we designed and implemented a genetic algorithm. It extends a genetic algorithm GANS [19], which is one of the best metaheuristic algorithms for the RCPSP, and adapts it to our problems. Moreover, we combined both approaches into a warm-started CP model, where a GA is used for finding the warm-start schedule, and the CP model tries to further improve the warm-start solution. To further improve the implemented GA for the larger instances, we also trained a relatively simple machine learning model to select the most appropriate mode for activity in the GA sub-problem based on the activity and sub-problem characteristics in the optimal solutions of the smaller instances.

For the evaluation of the proposed methods, we created several datasets based on the existing MMRCPSP benchmark datasets PSPLIB [37] and MMLIB [14]. We selected four sets of instances differing in the number of activities in an instance (10, 30, 50, or 100 activities). From each set, we created two new datasets varying in the time unit granularity (a single time unit in an instance corresponds to either 1 minute or 15 minutes). In addition, we extended the instances with energy consumption of activities per time unit, energy consumption limits for each 15-minute window, electricity price, and an energy budget limit.

Besides the comparison of all proposed approaches, in the experiments, we also analyzed the optimality of found solutions by the exact CP models, compared different methods of creating the initial generation for the GA, and evaluated the use of the mode selection machine learning model in the GA.

The results of the experiments showed that the selection of an appropriate method depends on the problem size, criterion, and available time for finding the solution. For the smaller instances with 10 activities, the CP model is the best option that is able to find optimal solutions in a few seconds. On the other hand, for the larger instances, the better option is to use the GA or combine it with the CP (i.e., warm-started CP). Last but not least, the incorporation of a trained mode selection model to the GA can also slightly improve the search on the larger instances.

8.1 Future Work

In the experiments, we showed that the use of a trained mode selection model can improve the performance of the GA. Nevertheless, the proposed mode selection model is rather a prototype since not much time was devoted to the designing and testing of efficient features because this was not in the scope of this thesis. Thus, future work may be in the development of better training data with more sophisticated features or proposing more complex models and evaluating their integration in the GA.

Although the experiments were performed on different-sized datasets that extend the standard benchmark datasets, they are still artificially created and can differ from real-world problems. Hence, it might be interesting to see how all the proposed algorithms would perform on real-world production data. We also believe that implementing the genetic algorithm in a faster programming language might be beneficial for practical usage.

However, all of these suggestions are beyond the scope of this thesis.

Bibliography

[1] S. Hartmann and D. Briskorn, “A survey of variants and extensions of the resource-constrained project scheduling problem,” *European Journal of Operational Research*, vol. 207, no. 1, pp. 1–14, 2010.

[2] S. Hartmann and D. Briskorn, “An updated survey of variants and extensions of the resource-constrained project scheduling problem,” *European Journal of Operational Research*, vol. 297, no. 1, pp. 1–14, 2022.

[3] J. Węglarz, J. Józefowska, M. Mika, and G. Waligóra, “Project scheduling with finite or infinite number of activity processing modes – A survey,” *European Journal of Operational Research*, vol. 208, no. 3, pp. 177–205, 2011.

[4] J. Blazewicz, J. K. Lenstra, and A. R. Kan, “Scheduling subject to resource constraints: Classification and complexity,” *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11–24, 1983.

[5] C. Artigues, O. Koné, P. Lopez, and M. Mongeau, “Mixed-integer linear programming formulations,” in *Handbook on Project Management and Scheduling*, vol. 1 of *International Handbooks on Information Systems*, Springer, 2015.

[6] A. A. B. Pritsker, L. J. Waiters, and P. M. Wolfe, “Multiproject scheduling with limited resources: A zero-one programming approach,” *Management Science*, vol. 16, no. 1, pp. 93–108, 1969.

[7] N. Christofides, R. Alvarez-Valdés, and J. M. Tamarit, “Project scheduling with resource constraints: A branch and bound approach,” *European Journal of Operational Research*, vol. 29, no. 3, pp. 262–273, 1987.

[8] F. B. Talbot, “Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case,” *Management Science*, vol. 28, no. 10, pp. 1197–1210, 1982.

[9] R. Alvarez-Valdés and J. M. Tamarit, “The project scheduling polyhedron: Dimension, facets and lifting theorems,” *European Journal of Operational Research*, vol. 67, no. 2, pp. 204–220, 1993.

[10] O. Koné, C. Artigues, P. Lopez, and M. Mongeau, "Event-based MILP models for resource-constrained project scheduling problems," *Computers & Operations Research*, vol. 38, no. 1, pp. 3–13, 2011.

[11] A. Sprecher, S. Hartmann, and A. Drexl, "An exact algorithm for project scheduling with multiple modes," *Operations-Research-Spektrum*, vol. 19, pp. 195–203, 1997.

[12] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím, "IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG," *Constraints*, vol. 23, pp. 210–250, 2018.

[13] V. Heinz, A. Novák, M. Vlk, and Z. Hanzálek, "Constraint Programming and constructive heuristics for parallel machine scheduling with sequence-dependent setups and common servers," *Computers & Industrial Engineering*, vol. 172, p. 108586, 2022.

[14] V. Van Peteghem and M. Vanhoucke, "An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances," *European Journal of Operational Research*, vol. 235, no. 1, pp. 62–72, 2014.

[15] A. Agarwal, S. Colak, and S. Erenguc, "Metaheuristic methods," in *Handbook on Project Management and Scheduling*, vol. 1 of *International Handbooks on Information Systems*, Springer, 2015.

[16] R. Pellerin, N. Perrier, and F. Berthaut, "A survey of hybrid metaheuristics for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 280, no. 2, pp. 395–416, 2020.

[17] R. Kolisch and S. Hartmann, *Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis*, pp. 147–178. Springer US, 1999.

[18] V. Van Peteghem and M. Vanhoucke, "A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 201, no. 2, pp. 409–418, 2010.

[19] S. Proon and M. Jin, "A genetic algorithm with neighborhood search for the resource-constrained project scheduling problem," *Naval Research Logistics (NRL)*, vol. 58, no. 2, pp. 73–82, 2011.

[20] E. N. Goncharov and V. V. Leonov, "Genetic algorithm for the resource-constrained project scheduling problem," *Automation and Remote Control*, vol. 78, pp. 1101–1114, 2017.

[21] 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*, vol. 149, no. 2, pp. 268–281, 2003.

[22] H. Li and H. Zhang, "Ant colony optimization-based multi-mode scheduling under renewable and nonrenewable resource constraints," *Automation in Construction*, vol. 35, pp. 431–438, 2013.

[23] G. Koulinas, L. Kotsikas, and K. Anagnostopoulos, "A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem," *Information Sciences*, vol. 277, pp. 680–693, 2014.

[24] C. C. Ribeiro, P. Hansen, K. Nonobe, and T. Ibaraki, "Formulation and tabu search algorithm for the resource constrained project scheduling problem," *Essays and Surveys in Metaheuristics*, pp. 557–588, 2002.

[25] W. Guo, M. Vanhoucke, J. Coelho, and J. Luo, "Automatic detection of the best performing priority rule for the resource-constrained project scheduling problem," *Expert Systems with Applications*, vol. 167, p. 114116, 2021.

[26] A. Golab, E. Gooya, A. Falou, and M. Cabon, "A multilayer feed-forward neural network (MLFNN) for the resource-constrained project scheduling problem (RCPSP)," *Decision Science Letters*, vol. 11, no. 4, pp. 407–418, 2022.

[27] W. Guo, M. Vanhoucke, and J. Coelho, "A prediction model for ranking branch-and-bound procedures for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 306, no. 2, pp. 579–595, 2023.

[28] X. Zhao, W. Song, Q. Li, H. Shi, Z. Kang, and C. Zhang, "A deep reinforcement learning approach for resource-constrained project scheduling," in *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1226–1234, IEEE, 2022.

[29] K. Biel and C. H. Glock, "Systematic literature review of decision support models for energy-efficient production planning," *Computers & Industrial Engineering*, vol. 101, pp. 243–259, 2016.

[30] J. Busse, C. Rüther, and J. Rieck, "Energy cost-oriented scheduling with job prioritization," in *2018 5th International Conference on Systems and Informatics (ICSAI)*, pp. 514–520, IEEE, 2018.

[31] H. Okubo, T. Miyamoto, S. Yoshida, K. Mori, S. Kitamura, and Y. Izui, "Project scheduling under partially renewable resources and resource consumption during setup operations," *Computers & Industrial Engineering*, vol. 83, pp. 91–99, 2015.

[32] L. He and Y. Zhang, "Bi-objective optimization of RCPSP under time-of-use electricity tariffs," *KSCE Journal of Civil Engineering*, vol. 26, no. 12, pp. 4971–4983, 2022.

[33] J. B. Abikarram, K. McConky, and R. Proano, “Energy cost minimization for unrelated parallel machine scheduling under real time and demand charge pricing,” *Journal of Cleaner Production*, vol. 208, pp. 232–242, 2019.

[34] H. F. Rahman, R. K. Chakrabortty, S. Elsawah, and M. J. Ryan, “Energy-efficient project scheduling with supplier selection in manufacturing projects,” *Expert Systems with Applications*, vol. 193, p. 116446, 2022.

[35] B. Du, T. Tan, J. Guo, Y. Li, and S. Guo, “Energy-cost-aware resource-constrained project scheduling for complex product system with activity splitting and recombining,” *Expert Systems with Applications*, vol. 173, p. 114754, 2021.

[36] M. Palpant, C. Artigues, and P. Michelon, “LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search,” *Annals of Operations Research*, vol. 131, pp. 237–257, 2004.

[37] R. Kolisch and A. Sprecher, “PSPLIB – A project scheduling problem library,” *European Journal of Operational Research*, vol. 96, no. 1, pp. 205–216, 1996.

[38] F. F. Boctor, “Heuristics for scheduling projects with resource restrictions and several resource-duration modes,” *International Journal of Production Research*, vol. 31, no. 11, pp. 2547–2558, 1993.

Appendix A

List of Abbreviations

APD	Average Percent Deviation
CPP	Critical Peak Pricing
CP	Constraint Programming
GANS	Genetic Algorithm with Neighborhood Search [19]
GA	Genetic Algorithm
MILP	Mixed-Integer Linear Programming
MMRCPSP	Multi-Mode Resource-Constrained Project Scheduling Problem
NS	Neighborhood Search
RCPSP	Resource-Constrained Project Scheduling Problem
RTP	Real-Time Pricing
SGS	Schedule Generation Scheme
TOU	Time-of-Use electricity tariffs

Appendix B
List of Notations

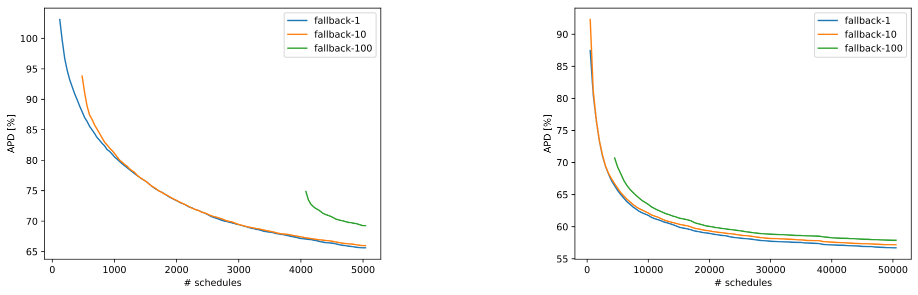
The following is the list of symbols (and their meaning) used in the definition of the problems studied in this thesis.

Symbol	Meaning
A	a finite set of activities
a_i	an activity with index i
M_i	a finite set of processing modes of activity a_i
m	a particular processing mode
d_{im}	a duration of processing of activity a_i in mode m
K	a number of types of resources
R_k	a number of resources of type k, $k \in \{1, \dots, K\}$
r_{imk}	a required number of resources of type k by activity a_i in mode m
T	a project horizon
t	a discrete time unit, $t \in \{0, \dots, T\}$
s_i	a start time of activity a_i
S	a schedule (it consists of start time s_i and processing mode $m \in M_i$ for each activity a_i)
$a_i \prec a_j$	a precedence constraint between activities a_i and a_j
$C_{\max}$	a project makespan
$p(t)$	an electricity price at time t
c_{im}	a energy consumption of activity a_i in mode m in a single time unit
$c(S, t)$	a total amount of consumed energy by schedule S at time t
q	a quarter period (15-minute time window)
Q	a finite set of all quarter periods
C_q	an energy consumption limit in a quarter q
$energyCost(S)$	a total price of consumed energy by a schedule S
B	a budget on the total energy cost of the whole project
α	the weight of the project makespan criterion in the weighted objective criterion
β	the weight of the total energy cost criterion in the weighted objective criterion

Appendix C

Measured Data in Further Experiments

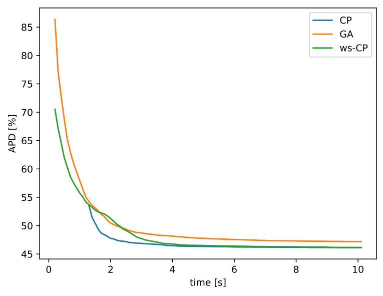
C.1 Effect of Initial Generation in GA



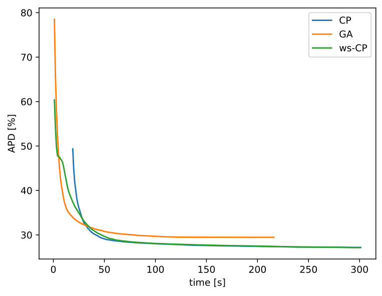
(a) : 5000 schedules **(b) :** 50000 schedules

Figure C.1: APD from the $criticalPath_{LB}$ (in %) in the number of generated schedules for *"fallback"* methods of initial generation creation on MMLIB50_1min instances of the problem with hard energy budget constraint.

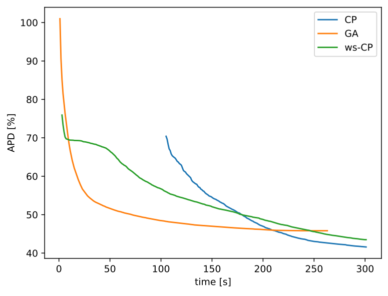
C.2 Comparison of Proposed Approaches



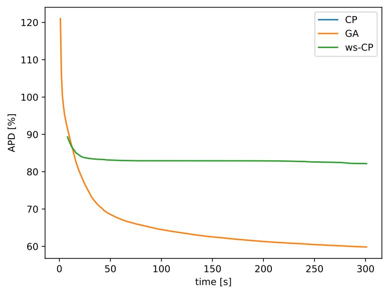
(a) : j10_1min



(b) : j30_1min

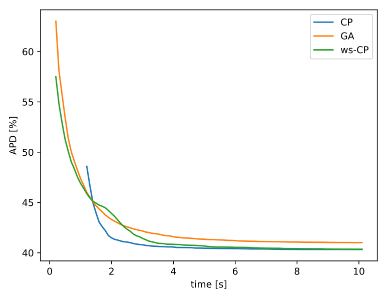


(c) : MMLIB50_1min

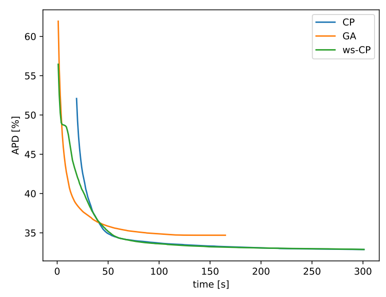


(d) : MMLIB100_1min

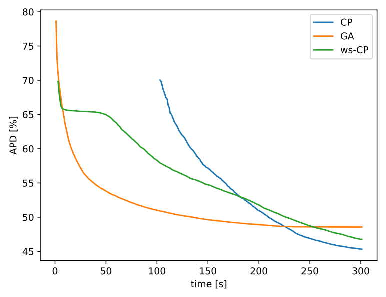
Figure C.2: APD from the $weightedCriterion_{LB}$ (in %) in time for different approaches on selected datasets of the problem with the total energy cost in the objective in the $scenario_{C_{\max}}$ scenario.



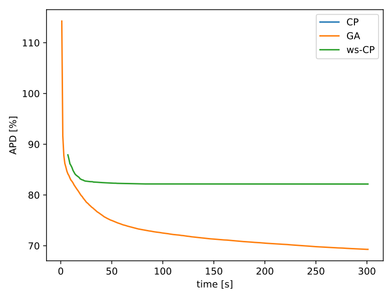
(a) : j10__1min



(b) : j30__1min

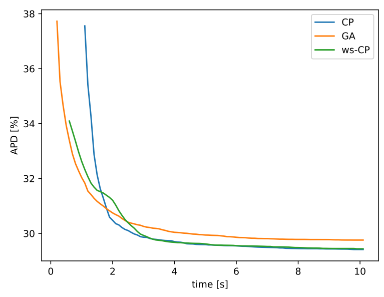


(c) : MMLIB50__1min

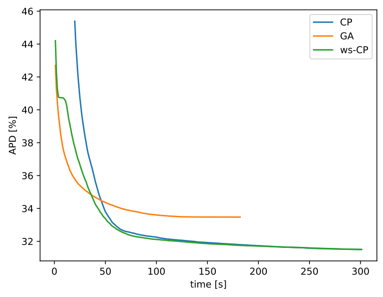


(d) : MMLIB100__1min

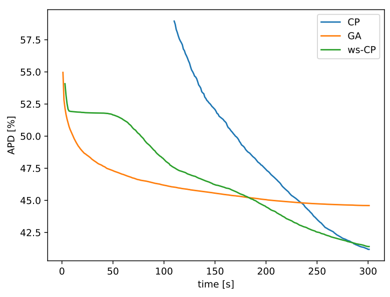
Figure C.3: APD from the $weightedCriterion_{LB}$ (in %) in time for different approaches on selected datasets of the problem with the total energy cost in the objective in the $scenario_{equal}$ scenario.



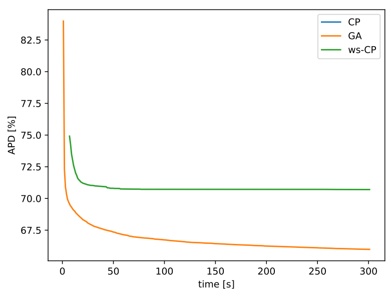
(a) : j10__1min



(b) : j30__1min



(c) : MMLIB50__1min



(d) : MMLIB100__1min

Figure C.4: APD from the $weightedCriterion_{LB}$ (in %) in time for different approaches on selected datasets of the problem with the total energy cost in the objective in the $scenario_{energy}$ scenario.