Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Scheduling of energy-demanding operations on multiple machines with respect to energy consumption limits

Kiryl Kalodkin

Supervisor: Ing. István Módos
Field of study: Open Informatics
Subfield: Software Engineering
May 2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Kalodkin**	Jméno: **Kiryl**	Osobní číslo: **464630**
Fakulta/ústav:	**Fakulta elektrotechnická**		
Zadávající katedra/ústav:	**Katedra počítačů**		
Studijní program:	**Otevřená informatika**		
Studijní obor:	**Softwarové inženýrství**		

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Rozvrhování energeticky náročných operací na více strojích s ohledem na energetické limity

Název diplomové práce anglicky:

Scheduling of energy-demanding operations on multiple machines with respect to energy consumption limits

Pokyny pro vypracování:

The goal of this thesis is to propose an algorithm for scheduling energy-demanding operations on multiple machines with consideration to the energy consumption limits. The following tasks should be done:
1. Study the problem of job-shop scheduling and review the existing works.
2. Extend the job shop scheduling problem with energy consumption limits.
3. Choose an appropriate approach for solving the scheduling problem.
4. Design and implement an algorithm for the scheduling problem.
5. Extend the classical job-shop benchmark instances with energy consumption limits and test the implemented algorithm on these instances."

Seznam doporučené literatury:

[1] Jacek Błażewicz, Wolfgang Domschke, Erwin Pesch. The job shop scheduling problem: Conventional and new solution techniques, In European Journal of Operational Research, Volume 93, Issue 1, 1996, Pages 1-33, ISSN 0377-2217
[2] Lennart Merkert, Iiro Harjunkoski, Alf Isaksson, Simo Säynevirta, Antti Saarela, Guido Sand. Scheduling and energy ? Industrial challenges and opportunities, In Computers & Chemical Engineering, Volume 72, 2015, Pages 183-198, ISSN 0098-1354
[3] István Módos, Přemysl Šůcha, Zdeněk Hanzálek. Algorithms for robust production scheduling with energy consumption limits, In Computers & Industrial Engineering, Volume 112, 2017, Pages 391-408, ISSN 0360-8352

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. István Módos, katedra řídicí techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **03.01.2018** Termín odevzdání diplomové práce: ____________ /

Platnost zadání diplomové práce: **30.09.2019**

Ing. István Módos podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

Acknowledgements

I would like to thank my supervisor István Módos for consultation and support. Also I would like to thank CIIRC for working place and Faculty of Electrical Engineering for granted computing power for experiments.

Declaration

Abstract

This thesis deals with the Job Shop scheduling with energy limits, which is an NP-hard problem. This problem is important for manufacturing and energy provider because overconsumption leads to instability of the energy system.

Two exact and four heuristic methods are considered in this work. The proposed heuristic methods extend existing approaches. For implementations of exact and some heuristic methods, IBM CP Optimizer was used, whereas the best method, which is provided by this work, doesn't need any external library. For comparison of methods, an instance generator was used, which is proposed by this work. The instance generator uses the existing benchmark instances and extends them with energy limits. The generated instances were used in experiments to verify effectiveness and correctness of the designed methods.

The best heuristic algorithm solves instances, which have size 50 × 15, where 50 is number of jobs and 15 is number of machines.

Keywords: Scheduling, NP-hard problem, Job Shop, Constraint programming, Taboo Search, Global search, Path Relinking, Energy limits

Supervisor: Ing. István Módos
Department of Control Engineering

Abstrakt

Tato práce se zabývá Job Shop rozvrhováním s ohledem na energetické limity, což je NP-těžký problém. Tento problém je důležitý pro výrobní společnosti a dodavatele elektrické energie protože nadměrná spotřeba elektrické energie vede na nestability energetického systému.

V práce jsou představený dvě exaktní a čtyři heuristické metody. Navržený heuristické metody používají rozšiřují existující přístupy. Pro implementaci exaktních a některých heuristických metod byl použitý IBM CP Optimizer, ale nejlepší metoda, která je představená v teto práci, nepotřebuje žádné externí knihovny. Pro porovnání metod v práci je představen generátor instancí. Generátor instancí používá existující benchmarkové instance a rozšiřuje je o energetické limity. Generované instance byly použitý v experimentech pro verifikaci efektivity a správnosti vyvinutých metod.

Nejlepší heuristická metoda řeší instance rozměrem 50 × 15, kde 50 je počet jobů 15 je počet strojů.

Klíčová slova: Rozvrhování, NP-těžký problem, Job Shop, Programování s omezujícími podmínkami, Taboo Search, Globální prohledavání, Path Relinking, Energetické limity

Překlad názvu: Rozvrhování energeticky náročných operací na více strojích s ohledem na energetické limity

Contents

Figures

Tables

Chapter 1

Introduction

This work deals with Job Shop scheduling problem (JSSP) with energy limits.

Production is not the trivial process which needs schedule many operations on many machines at the same time because this process need manage algorithms. For example, the wood factory produces paper and carton. For carton wood must cut on cutting machine, boiling in acid, processing on carton machine, washing and drying. For paper, wood must cut on cutting machine, whiten and processing on the paper machine.

Given the economic importance of scheduling exist more works, which is centered on obtaining the optimal solution. Effective scheduling can improve output and reduce production cost. The JSSP is strongly NP-hard.

This problem is important for manufacturing because they want to produce as much as possible products, but also they must fulfill energetic limits contracted with the energy provider. These limits are important for the power system because if consumers consume more energy, then power system have, power system loses stability and can be blackouts. Moreover, the wires which connect consumers to power system have maximum power capacity. Overloading of the wires shortness their lifespan. In the contract is written how much energy plant can consume for a metering interval. If the plant consumes more than this limit, the plant must pay a penalty for the power system. In the Czech Republic, the metering interval is 15 minutes.

Figures 1.1 and 1.2 represent benchmark instance schedule for JSSP and

JSSP with energy limits.

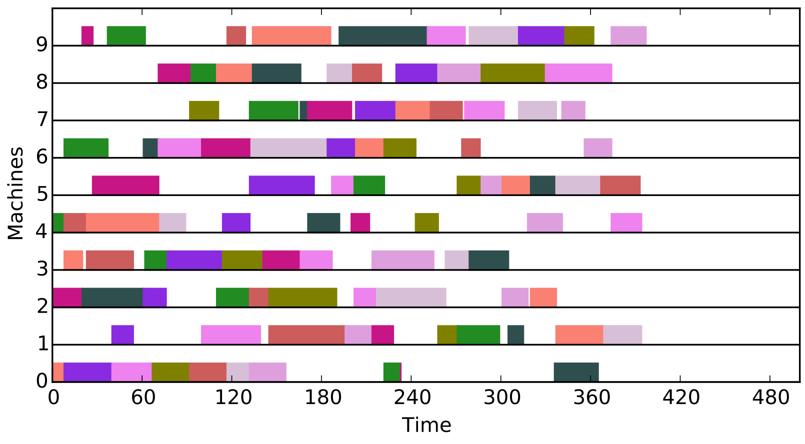

Figure 1.1: JSSP without energy limits

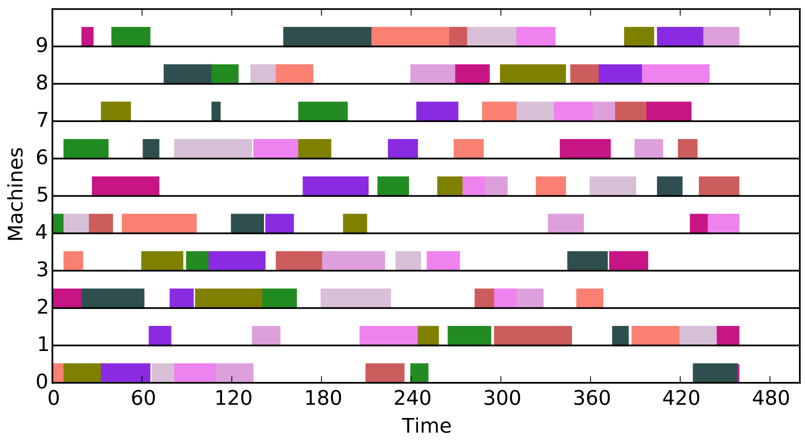

Figure 1.2: JSSP with energy limits

1.1 Related works

For solving JSSP many algorithms exist, however, most of these algorithms don't consider energy limits.

Article [BJS92] describes Branch and Bound approach. The principle of this way is the enumeration and estimating of feasible solutions. The solutions are represented as a tree. For all nodes without leafs algorithm calculates successor by fix some operation ordering on the machines. After each successor is handled by the same way. The examination of a branch stops if this branch represents only one solution or algorithm proves that this branch doesn't contain optimal solution. This approach detects not the optimal solution as early as possible but in the worst case algorithm enumerates all solutions. This approach solves the solution size 10x10 in 19 minutes. For larger solutions algorithm need a good initial solution, which it can get from the heuristic algorithm.

For solving this problem is used Taboo search way, which is described in articles [ZLGR06] and [MRX01]. This way is based iterative moves between neighborhoods until the stopping criterion is satisfied. For taboo some neighborhood is used Taboo List. This list stores solutions, which was visited recently. The article [ZLGR06] describes effective moves for finding the neighborhood. Taboo search and neighborhood structure, which are described in [ZLGR06] is used in this work because Taboo search algorithm quickly converges to optimal solution and solves large instances.

Article [BV98] describes local search with shifting bottleneck for solving JSSP. This method is based on the Guided Local Search, which uses neighborhood trees to escape from local optimum. Also, this article describes the method for estimate schedule makespan. This algorithm solves large instances, for some of them algorithm improves the best-known solution.

Genetic Algorithm approach for JSSP is described in the article [LC10]. This is a metaheuristic inspired by the natural selection process. Genetic Algorithm belongs to the class of evolutionary algorithms and is used for generating high-quality solutions by copying bio-inspired operations such as mutation, crossover, and selection. In the article design chromosome structure, crossover and mutation. The crossover is based on generating new solution on base exist solutions. The mutation changes existing solution for generating a new one.

Article [PLC14] describes Taboo Search/Path Relinking algorithm, which incorporates a Taboo search procedure from [ZLGR06]. The algorithm works very fast and solves instances which have big size. For Taillard's instances this size is 30x20, but for Demirkol instances this size is 50x20. Taboo Search/Path Relinking algorithm is used in this work.

Also exist another approaches. For example article [WLZW97] describes optimization approach for solving JSSP. This approach is based on Lagrangian relaxation. In [BM14] is used the parallel algorithm for solving JSSP. This approach used opportunities, which provide modern multi-core computers.

All of these articles don't consider energy limits, which is very important for enterprises. Few works exist which consider scheduling problems with the energy limits. In [MvH17] is described the algorithm, which solves scheduling problem on one machine, but JSSP is considered more than one machine. This work has used the algorithm, which calculates earliest operation start time with considering the energy limits.

Article [MA16] is described minimizing energy consumption and makespan in two-machine Flow Shop scheduling problem. In the Flow Shop scheduling problem, each work must be processed on the set of machines in a certain order. This article compares constructive heuristics and multi-constructive genetic algorithm.

1.2 Contribution of the thesis

The contribution of this work is a problem statement, exact and heuristic algorithms for solving JSSP with energy limits and experiments, which compare exact and heuristic approaches.

The exact algorithm needs a lot of memory, which a common customer computer does not have. On the other hand, our proposed heuristic algorithm can find solutions for instances, which have size 20×30 and larger and doesn't rely on any external commercial solver and find solutions, which have the similar makespan as the one found by the exact algorithms within the time limit.

Also we prove, that the problem of makespan calculation with fixed ordering of operations on the machine with considering the energy limits is NP-hard.

1.3 Thesis overview

In problem statement chapter the problem will be formalized and the basic concepts will be defined.

In algorithm chapter will be described exact and heuristic algorithms for solving JSSP with energy limits. Exact section 4.1 will describe Optional Variables approach and Overlap approach. For implementation, these algorithms will use IBM CP Optimizer framework. Heuristic section 4.2 will describe Taboo search and Path Relinking algorithms and their modification, which consider energy limits.

In experiment chapter will be shown performance and computation result of algorithms for different instances, the proposed algorithms will be compared and will be select the best way.

Chapter 2

Problem statement

2.1 Job Shop without energy limits

In this subsection JSSP without energy limits is outlined. In JSSP we have a set $M = \{1..m\}$ of m machines and a set $J = \{1..n\}$ of n jobs. Each job $j \in J$ consists of n_j ordered operations $O_{j,1}..O_{j,n_j}$. Let o_k be an operation, and $O = \{0, 1..o, o+1\}$ denotes the set of all operations which must be scheduled, where operations 0 and $o+1$ are dummy operations, that represent initial and final operations respectively. Each operation $o_k \in O$ must be processed on one dedicated machine $h \in M$ and has fixed processing time $p_k \in \mathbb{R}_{\geq 0}$, operations 0 and $o+1$ have no processing time. The processing time is in an interval, during which machine h processes operation o_k. The set of the operations, which are to be processed on machine h is denoted as L_h.

A machine can process at most one operation at the same time and, once operation starts processing on the machine, it must complete processing on the same machine without preemption, i.e., an operation starts and end once.

Let $predj_k$ be the job predecessor of the operation o_k; the first operation 0 has no predecessor. For each first operation in the jobs, the job predecessor is operation 0. The job predecessor is an operation which immediately precedes o_k in the same job. The operations are interrelated by two kinds of constraints. First, the operation o_k must be scheduled on the certain machine h. Second, the operation o_k must be scheduled after its predecessor $predj_k$ is completed.

Let s_k be a start time of operation o_k. The schedule is an assignment of an operation to a machine in time. It has fixed operation ordering on machines. The machine predecessor $predm_k \in L_h$ is an operation which immediately precedes operation $o_k \in L_h$ on machine $h \in M$. For the first operations on machines, the machine predecessor is operation 0.

The goal of JSSP is to find s_k which minimize the maximum completion time of the operations, i.e., makespan:

$$\min \max_{o_k \in O}(s_k + p_k) \tag{2.1a}$$

subject to:

$$s_k \geq 0, o_k \in O \tag{2.1b}$$
$$s_k \geq s_{predj_k} + p_{predj_k}, k = 1...o+1 \tag{2.1c}$$
$$s_i - s_j \geq p_i \quad \text{or} \quad s_j - s_i \geq p_j, o_i \in L_h, o_j \in L_h, o_i \neq o_j, h \in M \tag{2.1d}$$

The equation 2.1a is an objective function which minimizes makespan. Constraint 2.1b requires that the start time of all the operations not be negative. Equation 2.1c sets job precedence among operations of the same job. Constraint 2.1d prohibits overlaps between operations on the same machine.

2.1.1 Disjunctive graph

To illustrate schedules and makespan computation the JSSP is typically represented by disjunctive graph $G = (O, A, \bigcup_{h=1}^{m} A_h)$ [ZLGR06]. In this graph, O is the set of vertices which correspond to the operation set, A is a set of conjunctive arcs, which connect consecutive operations of the same job $A = \{(k, k+1) \mid k = 1, 2....h_j - 1, j \in J\}$, $\bigcup_{h=1}^{m} A_h$ is a set of disjunctive arcs connecting operations of the same machine. The length of $(i, j) \in A$ is p_i. The length of $(i, j) \in \bigcup_{h=1}^{m} A_h$ is processing time p_i.

The graph G can be split into sub-graph $D = (O, A)$ and m sub-cliques $G_h = (O_h, A_h)$. For each graph G_h we define a selection $S_h \in A_h$ so that each disjunctive arc is replaced by conjunctive arc so that S_h must be total ordering, which contains all operations from G_h. This information allows defining the feasible solution S as a tuple $(O, A, \bigcup_{h=1}^{m} S_h)$.

The critical path is the longest path from dummy operation 0 to dummy operation $o+1$ in the graph. The length of this path is equal to makespan of the feasible solution. Operations which belong to this path are called critical operations. A sequence of critical operations that are processed on the same machine one after another is called critical block.

For example, consider three jobs and three machines problem is that given in table 2.1. Conjunctive-disjunctive graph for this example is shown in figure 2.1. The feasible solution is represented in figure 2.2. The number, which associated with an operation is its start time in the solution. The critical path is highlighted by a fat line, and its length is 170.

Job	(Machine, Processing Time)		
Job 1	(1, 50)	(2, 30)	(3, 60)
Job 2	(3, 40)	(1, 10)	(2, 20)
Job 3	(2, 20)	(3, 20)	(1, 10)

Table 2.1: An example without energy limits

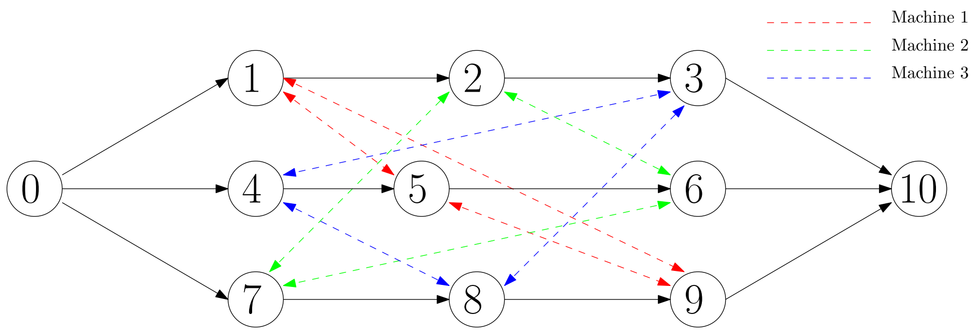

Figure 2.1: Conjunctive-disjunctive graph of the example

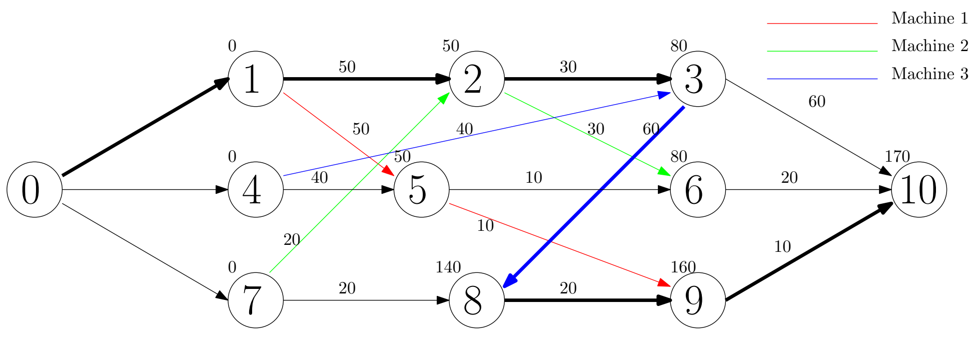

Figure 2.2: The solution of the example

2.2 Job Shop with energy limits

In this section, we extend JSSP with the additional constraint that takes energy limits into account.

In addition to JSSP, each machine h is consuming power $P_k \in \mathbb{R}_{>0}$ when h is processing operation o_k. The total consumed energy of o_k is then computed as $P_k \cdot p_k$.

The operations have to be scheduled within a scheduling horizon $H \in \mathbb{Z}_{>0}$. The scheduling horizon is divided into a set of metering intervals $\Omega = \{1..\frac{H}{D}\}$ where $D \in \mathbb{Z}_{>0}$ is a metering interval length and H is divisible by D. For all metering intervals $\omega_e \in \Omega$ we define energy limit E_{max}, which represents the upper bound of energy consumption, i.e., the energy limit. Moreover, the intersection length between metering interval ω_e and operation o_k is denoted as $Overlap(\omega_e, o_k)$.

The energy limits require that for all metering intervals $\omega_e \in \Omega$, the energy consumption of the operations in the metering interval is less or equal then the energy limit E_{max}, i.e:

$$\sum_{o_k \in O1} Overlap(\omega_e, o_k) \cdot P_k \leq E_{max}, \omega_e \in \Omega \tag{2.2}$$

The overlap depends on start time s_k of the operation o_k and its processing time p_k.

A formulation for JSSP with energy limits is the extension of the JSSP problem presented in section 2.2 with the constraints for energy limits 2.3e.

$$\min \max_{o_k \in O}(s_k + p_k) \tag{2.3a}$$

subject to:

$$s_k \geq 0, k \in O \tag{2.3b}$$

$$s_k \geq s_{predj_k} + p_{predj_k}, k = 1...o+1 \tag{2.3c}$$

$$s_i - s_j \geq p_i \quad \text{or} \quad s_j - s_i \geq p_j, (o_i, o_j) \in L_h, h \in M \tag{2.3d}$$

$$\sum_{o_k \in O} Overlap(\omega_e, o_k) \cdot P_k \leq E_{max}, \omega_e \in \Omega \tag{2.3e}$$

2.3 Example

We extend the example 2.1 with power consumption of the operations and energy limits. The solution of example is in table 2.2. The figures 2.3 and 2.4 represent solutions with the same ordering. The tuples, which are associated with the operations on Gantt diagram represent (a job, operation number in the job). Operations, which belong to the same job have the same color.

Notice, that in some cases we need to shift operation to decrease the overlap with some metering intervals to decrease energy consumption in them. For example operation (2,3) was shifted. The solution for the instance with energy limits has larger makespan than the solution without them because additional constraints decrease space of solutions.

Job	(Machine, Processing Time, Power)		
Job 1	(1, 50, 3)	(2, 30, 5)	(3, 60, 5)
Job 2	(3, 40, 2)	(1, 10, 6)	(2, 20, 5)
Job 3	(2, 20, 3)	(3, 20, 7)	(1, 10, 6)

Table 2.2: An extended example

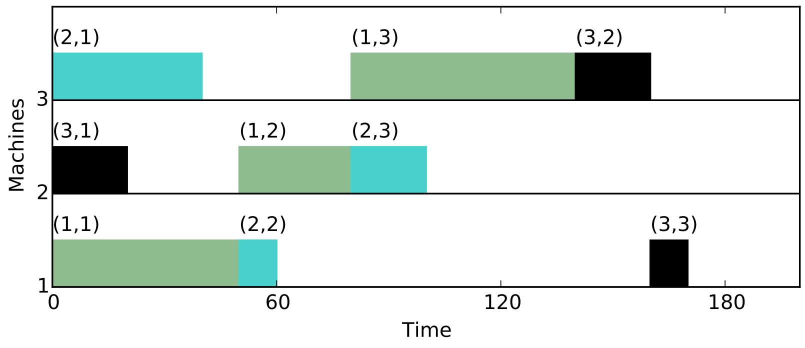

Figure 2.3: Gantt chart of the solution for example without energy limits

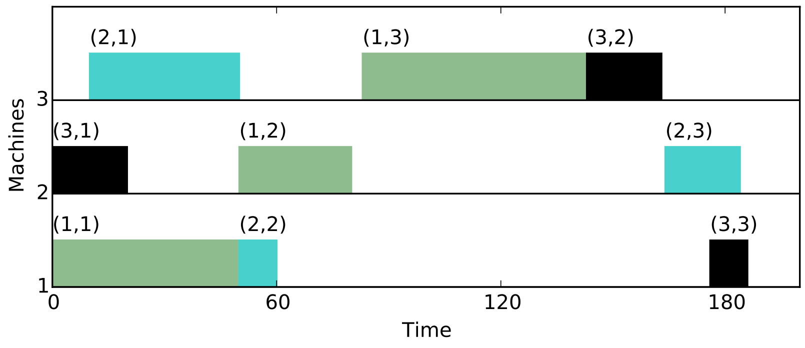

Figure 2.4: Gantt chart of the solution for example with energy limits

Chapter 3

Theoretical background

This section describes the theoretical background to the algorithms, which is used for the solving JSSP with energy limits. The exact methods are built on the Constraint programming, whereas it heuristic methods use the Taboo search.

3.1 Constraint programming

The Constraint programming is the form of declarative programming in which relations between variables are stated in the form of constraints.

Formally, let $X = \{x_1, x_2 ... x_n\}$ be a finite set of variables, $D = \{D_1, D_2 ... D_n\}$ be a finite set of domains of variables, $C = \{C_1, C_2 ... C_m\}$ is finite set of constraints. The domain D_i is the set of all possible values of x_i. The constraint C_i is a statement, which consists of the subset $X'_i \subseteq X$ and relation R_i on X'_i. $f(X)$ is the objective function. The solution is the complete assignment for all variables x_i from their domains such that all constraints are satisfied and $f(X)$ is optimal.

In scheduling metering intervals and operations can be represented as interval variables, for which *start*, *end*, and *length* are defined. The variable can be optional, which means that we can decide not to consider it in the solution. A present optional variable is variable, which is considering in the solution.

Examples of constraints, which are used in the Constraint programming from [IBM]:

1. $EndBeforeStart(a, b)$ constraint requires that if the optional variables a and b are present in the solution then $end(a) \leq start(b)$.

2. $StartBeforeEnd(a, b)$ constraint requires that if the optional variables a and b are present in the solution then $start(a) \leq end(b)$.

3. $EndMax(a) \leq A$ constraint requires that if the optional interval variable a are present in the solution then $end(a) \leq A$

4. $StartMin(a) \geq A$ constraint requires that if the optional interval variable a is present in the solution then $start(a) \geq A$

5. $Span(a, \{b_1, b_2...b_n\})$ constraint requires that interval variable a must spans over all present interval variables b_i if a presents in the solution, i.e., formally $(start(a) \leq start(b_i)) \wedge (end(a) \geq end(b_i)), \forall b_i$

6. $NoOverlap(a_1, a_2...a_n)$ constraint requires that all present interval variables from the set $\{a_1, a_2...a_n\}$ are pairwise non-overlapping.

3.2 Local Search

Local Search is a heuristic method, which is used for solving hard optimization problems. This algorithm can be formulated as finding a solution which corresponds to the best criterion function. In each iteration, Local Search generates set of neighbors around a current solution, which is called seed, selects from the neighbors the best one according to the objective function, which becomes a seed for the next iteration. For generating the neighbors around the seed, Local Search uses the Neighborhood structure, which is a rule, that perturbs the seed to generate candidates solutions.

The algorithm doesn't remember the old solutions, which leads to the problem: Local Search can't escape from a local optimum.

3.3 Taboo Search

Taboo Search algorithm from [GL97] is the well-known heuristic for the global optimization. This algorithm has been applied to many combinatorial problems. Basically, this is the Local Search, which can move between local optimums.

The previous solutions store in the taboo set. When Taboo Search finds a new solution, it checks if this solution is not in the taboo set. If the taboo set contains the solution Taboo Search selects another solution from the neighborhoods. This approach helps to escape from local optimum which is in contrast to Local Search, because Taboo Search is not allowed to return to solutions, which where visited before (within the capacity of taboo list).

3.4 Relinking procedure

In some cases, the Taboo Search can't leave a local optimum and get to the global optimum. Relinking procedure generates new solutions from the high-quality solutions. For two solutions S_i and S_g, Relinking procedure generates so called $PathSet$, which is a set of the solutions. The first solution S_i is called initial, the second S_g is called Guiding. To create this set the Relinking procedure do perturbation in Initial solution, so those perturbations leads to the Guiding solution. After each perturbation new solution is added to the $PathSet$.

After the solutions on the path are generated, each of the solutions is tried to be improved by slightly Taboo search with small iteration count, and the algorithm selects the solution, which has the best criterion. At the end, the best solution will be improved by strong Taboo search with large iteration count as far as possible and will be returned by Relinking procedure.

As can be seen from figure 3.1, Taboo Search without the Relinking procedure can't get the solution S_{global} from solutions S_i and S_g because it can't escape these local optimal solutions in define iteration count. However if Taboo Search obtains solutions S_3 or S_4 it can convergence to global optimal solution S_{global}.

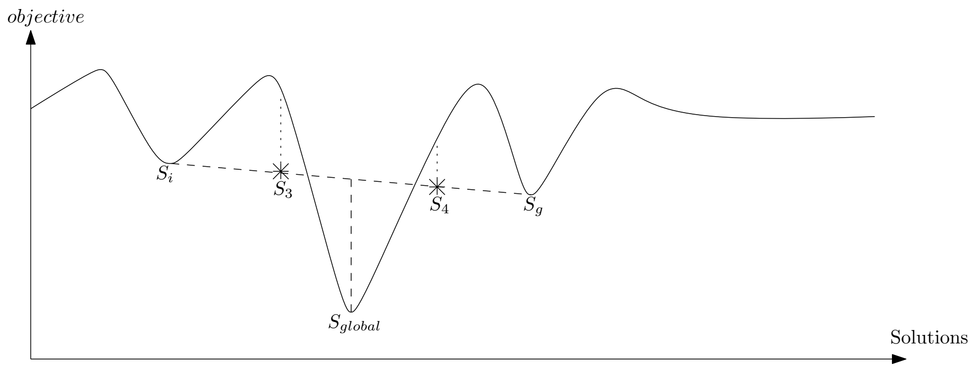

Figure 3.1: Heuristic principle

Chapter 4

Algorithms for JSSP with energy limits

This chapter explains algorithms, which are used to solve the scheduling problem described in Chapter 2. The first part explains the exact methods, which always find the optimal solution, but they need a lot of time to find. Therefore, exact methods can typically solve only small instances.

The the second part describes the heuristic methods, which find suboptimal solutions, but don't need a lot of time and usually found a solution that is close to the optimal one. Therefore, heuristic solutions are used in practice for solving large instances.

4.1 Exact methods

For the exact methods, Constraint programming is used because the problem has complex constraints, which are difficult to describe by ILP.

JSSP with energy limits was formulated in two approaches. The first approach is based on the optional variables, whereas the second one is based on computing the overlap between metering intervals and operations.

4.1.1 Optional Variables

In this method, each operation is represented by two ways: as an interval variable, which length is processing time of the operation, and as a set of optional interval variables, which represent the overlap between the operation and metering intervals $\omega_e \in \Omega$.

For ensuring consistency between two representations, we introduce the following constraints. The first one: the sum of lengths of optional interval variables from the set of optional variables, which corresponds the operation, and its length must be equal. The second one: interval variable, which represents the operation, must start with the first present optional interval variable from the set of optional variables, which belongs it, and must end with the last one. The first constraint requires that the sum of parts of the operation in each metering interval must be equals to the processing time of the entire operation. The second constraint ensures that the operation overlaps all its appearances in metering intervals from the set. These constraints don't allow case, which is shown on the figure 4.1, because $(start(t_{ie}) \geq start(t_i)) \wedge (end(t_{ie}) \leq end(t_i))$, interval can't overlap each other and $\sum_{\omega_e \in \Omega}(t_{ie}) = t_i$.

Energy consumption for each metering interval is computed as a sum of all variables, which represent overlap operations with this metering interval, multiplied by powers of appropriate operations.

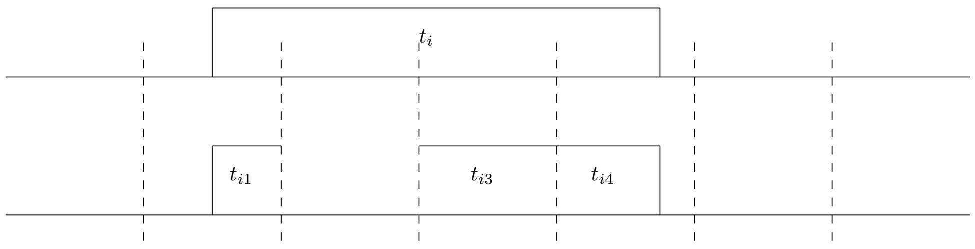

Figure 4.1: JSSP without energy limits

Formal representation

Each operation o_k is represented by two ways. The first way is an interval variable t_k. The length of t_k is the processing time of the operation o_k, i.e., $length(t_k) = p_k$. The second way is a set of optional interval variables $opt_k = \{t_{k1}, t_{k2}, , ...t_{k|\Omega|}\}, o_k \in O$, which represents overlap between operation o_k and metering intervals $\omega_e \in \Omega$.

Constraints:

$$length(t_{ke}) \leq D, t_{ke} \in opt_k, o_k \in O \tag{4.1}$$

$$EndMax(t_k) \leqslant H, o_k \in O \tag{4.2}$$

$$StartMin(t_{ke}) \geq D \cdot (e-1), o_k \in O, \omega_e \in \Omega \tag{4.3}$$

$$EndMax(t_{ke}) \leq D \cdot e, o_k \in O, \omega_e \in \Omega \tag{4.4}$$

$$\sum_{\omega_e \in \Omega} t_{ke} = t_k, o_k \in O \tag{4.5}$$

$$Span(t_k, \{t_{k0}, t_{k1}, , ...t_{|\Omega|}\}), o_k \in O \tag{4.6}$$

$$end(t_{k'}) \leq start(t_k), o_{k'} = predj_k, k = 1...o+1 \tag{4.7}$$

$$NoOverlap(t_k, t_{k'}), o_k \in L_h, o_{k'} \in L_h, h \in M \tag{4.8}$$

$$\sum_{o_k \in O} (t_{ke} \cdot P_k) \leq E_{max}, \omega_e \in \Omega \tag{4.9}$$

$$end(t_k) \leq C_{max}, o_k \in O \tag{4.10}$$

Objective function:

$$min(C_{max}) \tag{4.11}$$

Equation 4.2 ensures that each operation must be processed during horizon. Optional interval variable t_{ke} must start after the beginning of the metering interval ω_e and must finish before ending the metering interval ω_e, equations 4.3 and 4.4. Sum of lengths of optional interval variables which corresponds t_k and length of t_k must be equal, equation 4.5. Interval variable t_k must starts with appearance the first optional interval variable which belongs to it and must end with the last one, equation 4.6. Operations, which correspond to a same job must create a chain, equation 4.7. Operations, which are processed a same machine can't overlap each other, equation 4.8. Energy consumption in each metering interval $\omega_e \in \Omega$ must be less or equal to E_{max}, equation 4.9. Makespan is equal to the maximum completion time, equation 4.10.

4.1.2 Overlap

In this method operations and metering intervals are represented by interval variables. In contrast to the previous method all operations have common variables, which represent metering intervals, and are represented only one way. It connected with the CP Optimizer that has function $Overlap(t_i, t_j)$, which calculates an intersection between two interval variables t_i and t_j. The length of the interval variable, which corresponds to the operation, is processing time. The length of interval variable, which corresponds to a metering interval $\omega_e \in \Omega$, is D.

Energy consumption for a metering interval is sum intersection operations with this metering interval multiplied by powers the correspond operations. Constraints, which are responsible for the operations sequence in jobs and on machines are same as in the previous method.

Formal representation

Let each operation o_k be represented by interval variable t_k, where the length of t_k is processing time. Each metering interval $\omega_e \in \Omega$ has length D and is represented by interval variable ω_e.

Constrains:

$$start(\omega_{e+1}) = end(\omega_e), \omega_e \in \Omega \setminus \left\{\frac{H}{D}\right\} \tag{4.12}$$

$$start(\omega_1) = 0 \tag{4.13}$$

$$end(t_{k'}) \leq start(t_k), o_{k'} = predj_k, k = 1 ... o + 1 \tag{4.14}$$

$$NoOverlap(t_k, t_{k'}), o_k \in L_h, o_{k'} \in L_h, h \in M \tag{4.15}$$

$$\sum_{o_k \in O} (Overlap(t_k, \omega_e) \cdot P_k) \leq E_{max}, \omega_e \in \Omega \tag{4.16}$$

$$end(t_k) \leq C_{max}, o_k \in O \tag{4.17}$$

Objective function:

$$min(C_{max}) \tag{4.18}$$

Metering intervals start one after one. The first metering interval starts at time $t = 0$, equations 4.12,4.13. Operations, which correspond to a same job must create a chain, equation 4.14. Operations, which are processing on a same machine can't overlap each other, equation 4.15. Energy consumption in each metering interval must be less or equal to E_{max}, equation 4.16. Makespan is equal to the maximum completion time, equation 4.17.

4.2 Heuristic methods

The heart of the algorithm is the Taboo Search/Path Relinking algorithm from [PLC14]. This algorithm combines Taboo Search algorithm and Relinking procedure.

At the start, the algorithm generates the population, which is a set of the random feasible solutions. After the solutions in the population will be optimized with the Taboo Search with small iteration count. In each iteration, the algorithm samples two random solutions from the population and applies to them the Relinking procedure, which returns two new possibly high-quality solutions S^{p+1} and S^{p+2}. The main algorithm adds these solutions to the population. At the end of the iteration, the worst solutions will be deleted from the population to keep it constant size.

Taboo Search/Path Relinking algorithm is described by Algorithm 1. The $SlightTabooSearch(S_i)$ function is described in section 4.2.2.

4.2.1 Population initialization

Procedure $PopulationInitialization$ generates a population of the random feasible solutions. Solutions in the population don't duplicate each other.

Procedure $Repair(S)$, which is used to convert a random solution to a feasible solution is described in [dejW17]. The procedure repairs job precedences between operations. The procedure uses a set of unscheduled

Algorithm 1 Taboo search/path relinking algorithm

```
1: function TabooSearchPathRelinking($J$, $M$)
2:     $P = \{S_1, S_2, ...S_p\} := PopulationInitialization()$
3:     for $i = 1, ..., p$ do
4:         $S_i := SlightTabooSearch(S_i)$
5:     end for
6:     $S_* := argmin(f(S_i)|i = 1, ...p)$
7:     while (Time Limit is not reached) do
8:         Randomly select $S_1, S_2$ from $P$
9:         $S_{p+1} := PathRelinking(S_1, S_2)$
10:        $S_{p+2} := PathRelinking(S_2, S_1)$
11:        if $S_{p+1}$ (or $S_{p+2}$) is better than $S_*$ then
12:            $S_* := S_{p+1}$ (or $S_{p+2}$)
13:        end if
14:        if $S_{p+1} \notin P$ then
15:            $P := P \bigcup S_{p+1}$
16:            Identify a worst solution $S_w \in P$
17:            $P := P \setminus S_w$
18:        end if
19:        if $S_{p+2} \notin P$ then
20:            $P := P \bigcup S_{p+2}$
21:            Identify a worst solution $S_w \in P$
22:            $P := P \setminus S_w$
23:        end if
24:    end while
        return $S_*$
25: end function
```

operations that were not added to the feasible solution but were removed from the infeasible solution. In each iteration, the procedure selects an operation from the unscheduled set and operations from the infeasible solution, for which the machine predecessor was removed from the infeasible solution, and checks if adding the operation to the feasible solution will not be violated job precedences in the feasible solution. If exists any operation, which satisfies this demand, the procedure adds this operation to the feasible solution and removes it from infeasible solution or unscheduled set. If no operation from the infeasible solution or the unscheduled set was added, the procedure moves all operations from infeasible solution to unscheduled set, for which machine predecessors were removed, and will begin a new iteration.

4.2.2 Taboo search procedure

The Taboo Search procedure is described in [ZLGR06]. The Taboo Search procedure gets the initial solution from the algorithm 1 and the algorithm 5.

In each iteration, the Taboo Search generates new neighbors around the seed using a neighborhood structure and selects the best neighbor. If the best neighbor is better than the best-known solution, then this neighbor will become the best-known solution with respect to the makespan and new seed. Otherwise, the new seed will be the best neighbor which is not in the taboo list. The new seed will be added to the taboo list. At the end of the iteration, the algorithm removes the oldest solution from the taboo list. Figure 4.2 shows the flowchart of the Taboo Search procedure. Section 4.2.4 describes how to calculate the makespan for the fixed order of the operations on the machines with respect to energy limits.

The Taboo Search stops after iterating for the given number of iteration. In addition, if the solution not been improved after the maximum number of the disimproving iterations the procedure will stop. For the functions $SlightTabooSearch(S_i)$ and $StrongTabooSearch(S_i)$ these criteria are described in table 4.1. The smallest length of the taboo list is $L = 10 + n/m$. If $n \leq 2 \cdot m$ then the taboo set length is $L_{ts} = 1.4 \cdot L$ otherwise $L_{ts} = 1.5 \cdot L$.

Iteration count for slight Taboo search	50
Iteration count for strong Taboo search	1250
The maximum number of the disimproving iterations	300

Table 4.1: Taboo Search parameters

4.2.3 Neighborhood structure

A neighborhood structure describes a mechanism for effective generating new solutions around current solution. This structure is very important because it influences on effective of the Taboo Search procedure. A neighborhood structure must prevent generation of unnecessary or infeasible solutions if it is possible.

To generate better solutions operations on the critical path need be swapped because the critical path represents makespan. If operations that don't belong to the critical path are swapped, makespan can't be decreased. One of the

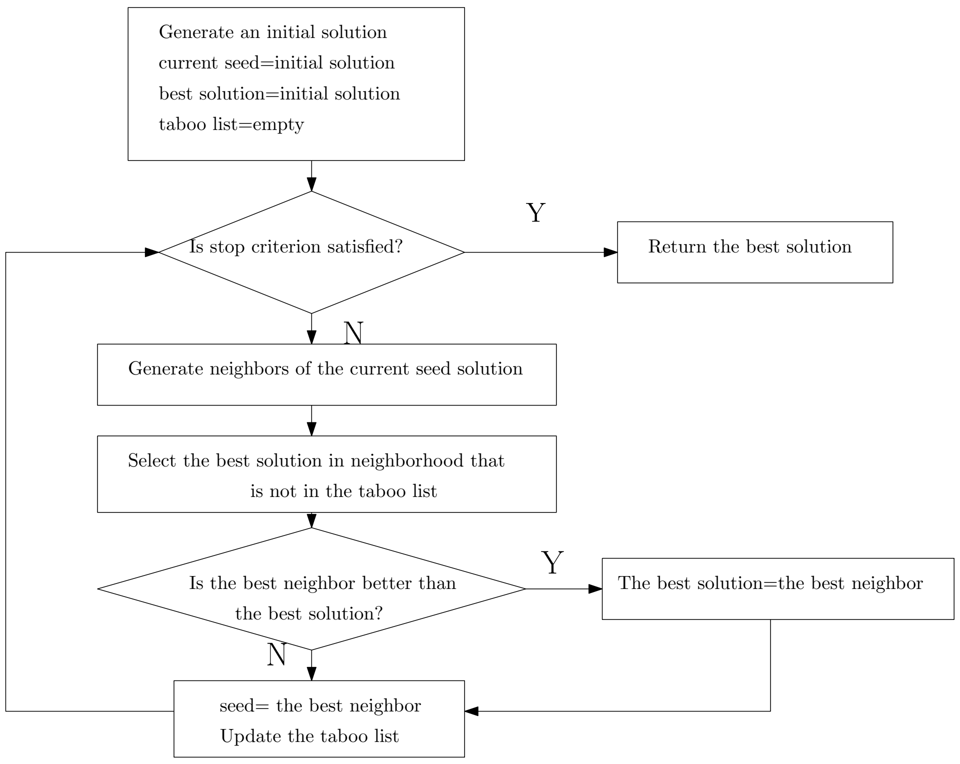
Generate an initial solution
current seed=initial solution
best solution=initial solution
taboo list=empty
Is stop criterion satisfied?
Y
Return the best solution
N
Generate neighbors of the current seed solution
Select the best solution in neighborhood that is not in the taboo list
Is the best neighbor better than the best solution?
Y
The best solution=the best neighbor
N
seed= the best neighbor
Update the taboo list

Figure 4.2: Taboo search algorithm

neighborhood structure that swaps operation on the critical path is N5 from [ZLGR06]. This structure swaps the operations, which belong to the critical path, therefore, has good efficiency.

At the start, the algorithm finds a critical path in the current solution. After this, the algorithm selects a critical block, which is a maximal sequence of adjacent critical operations that are processed on the same machine. In the selected block algorithm swaps operations as described in figure 4.3.

For define the critical path is used the algorithm 2. A critical block is chosen randomly.

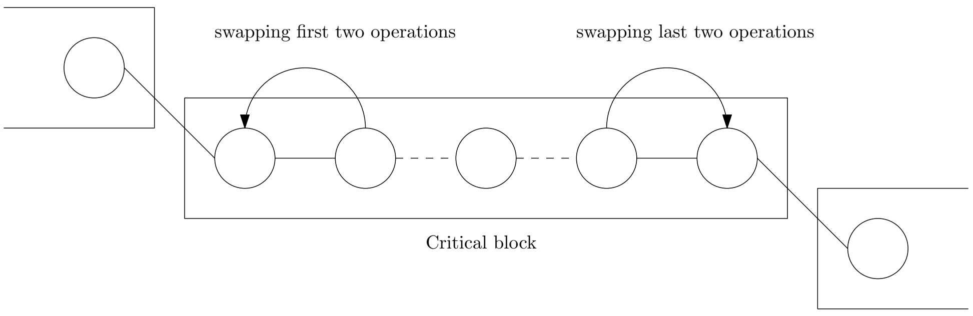

Figure 4.3: N5 neighborhood structure

Critical path calculation

The JSSP with energy limits can also be represented as a disjunctive graph, but for this problem, the sum of processing time of the operations, which belong to the critical path, isn't equal to makespan, because the length of arcs not only depends on processing time. To satisfying the energy constraints, start time of some operations may increase.

The algorithm starts work from the dummy operation $o + 1$ and adds to the critical path the closest previous operation until it gets operation 0. The start time of the operations must be known.

Figure 4.4 shows the critical path in the conjunctive-disjunctive graph with energy limits. As can be seen for the edge $(2, 3)$ the start time of operation o_3 a is not equal to $max(s_{predj_3} + p_{predj_3}, s_{predm_3} + p_{predm_3})$ as in the example without energy limits from table 2.1. In this case, s_3 was increased to satisfy energy demand. The start time of the operation o_9 also was increased, which also connected with energy constraints.

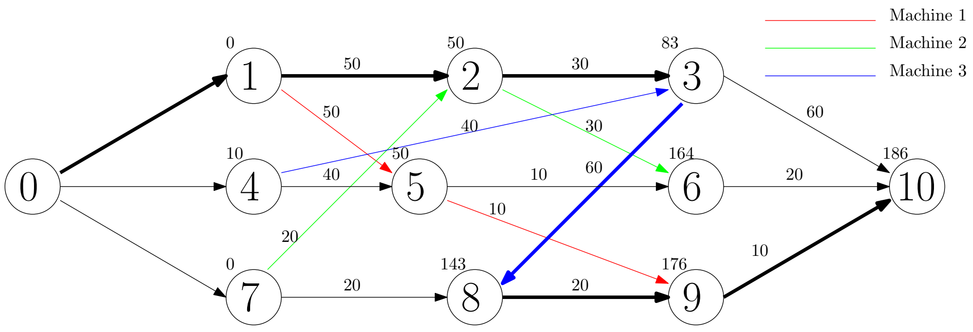

Figure 4.4: Example with energy limits

Algorithm 2 Finding critical path for solution with energy limits

```
1: function CriticalPath($O$)
2:     $CP := \emptyset$
3:     $CP := CP \bigcup (o+1)$
4:     $o_k :=$ any operation $o_k \in O$ such that $s_k + p_k = s_{o+1}$
5:     $CP := CP \bigcup o_k$
6:     while $o_k \neq 0$ do
7:         if $s_k - (s_{predj_k} + p_{predj_k}) < s_k - (s_{predm_k} + p_{predm_k})$ then
8:             $o_k = predj_k$
9:         else
10:            $o_k = predm_k$
11:        end if
12:        $CP := CP \bigcup o_k$
13:    end while
        return $CP$
14: end function
```

4.2.4 Makespan calculation with energy limits for fixed order

Problem complexity

For JSSP without energy limits the makespan calculation with fixed ordering ot the operations on machines is a polynomial problem, whereas for JSSP with energy limits isn't. To prove this statement, we use the theorem that the problem U is strongly NP-hard if some strongly NP-hard problem V polynomialy reduces on U.

Theorem 1. The makespan calculation with energy limits and fixed ordering on machines is strongly NP-hard.

Proof. To prove this theorem, we perform a polynomial reduction from 3-Partition decision problem to the Makespan calculation with energy limits for fixed order problem (MCELFOP).

Let A be a set of $3 \cdot l$ integers $a_1, a_2, \ldots .a_{3 \cdot l}$. Let B be a positive number such that:

$$\forall i \in 1, 2 \ldots 3 \cdot l : \frac{B}{4} < a_i < \frac{B}{2} \tag{4.19a}$$

$$\sum_{a_i \in A} a_i = l \cdot B \tag{4.19b}$$

The 3-Partition decision problem is to determinate if A can be partition into l disjunction subsets A_i, which consist of 3 elements and sum of these elements for all subsets is B, i.e., $\sum_{a_i \in A_j} a_i = B, j \in \{1, 2, \ldots l\}$.

Now we describe how to represent 3-Partition decision problem as MCELFOP. Let we have $3 \cdot l$ machines. For each number a_i we create job j_i which consists of one operation, having processing time of 1 and power consumption of a_i and this job must be processed on the machine h_i. Number of jobs and machines is $3 \cdot l$. Let $D = 1$ and for even metering intervals $E^{even}_{max} = B$ and for odd metering intervals $E^{odd}_{max} = 0$. The number of metering intervals is $2 \cdot D \cdot l$.

The answer of 3-Partition decision problem is YES-instance if and only if for the scheduling instance above exists a schedule for which the makespan is $2 \cdot D \cdot l$.

3-Partition decision problem ⇒ MCELFOP. For each subset A_i from YES-instances of 3-Partition decision problem, select the metering interval with number $2 \cdot i$ and for the operations which correspond to numbers from A_i, set start time to $2 \cdot D \cdot i - 1$. Sum of energy consumption in all even metering intervals is B, since $\sum_{a_i \in A} a_i = B$, for odd metering intervals is 0. Maximum completion time for the operation from the last subset is $2 \cdot D \cdot l$.

3-Partition decision problem ⇐ MCELFOP. Notice, that that operations can't overlap with odd metering intervals, because $E^{odd}_{max} = 0$. In the proof we use notation O_{ω_e} to denote the set of the operations, that are contained in metering interval ω_e.

Now we show, that every odd metering interval contains exactly 3 operations and we will prove this by contradiction. Assume, that there exists a metering interval ω_e such that $|O_{\omega_e}| \geq 4$. Then:

$$\sum_{o_j \in O_{\omega_e}} P_j \overset{4.19a}{>} |O_{\omega_e}| \cdot \frac{B}{4} \overset{|O_{\omega_e}| \geq 4}{\geq} 4 \cdot \frac{B}{4} = B \tag{4.20a}$$

However, equation 4.20a is a contradiction with feasibility of the schedule with respect to the energy limits. Therefore, $|O_{\omega_e}| < 4$ for all metering intervals.

Suppose, that some even metering interval ω_e has $|O_{\omega_e}| \leq 2$, then:

$$|J| = |O_{\omega_e}| + \sum_{\omega \in \Omega \setminus \omega_e} |O_\omega| \leq 2 + 3 \cdot (l-1) = 3 \cdot l - 1 < 3 \cdot l = |J| \tag{4.21a}$$

$$|J| < |J| \tag{4.21b}$$

Equation 4.21b is contradiction, therefore each even metering interval must have 3 operation.

For each metering interval $\omega_{2 \cdot i}$ from YES-instance from MCELFOP create subset $A_i = \{P_j : o_j \in O_{\omega_{2 \cdot i}}\}$. For all subsets A_i sum of elements is B and number of A_i subsets is l. Figure 4.5 illustrates the reduction. □

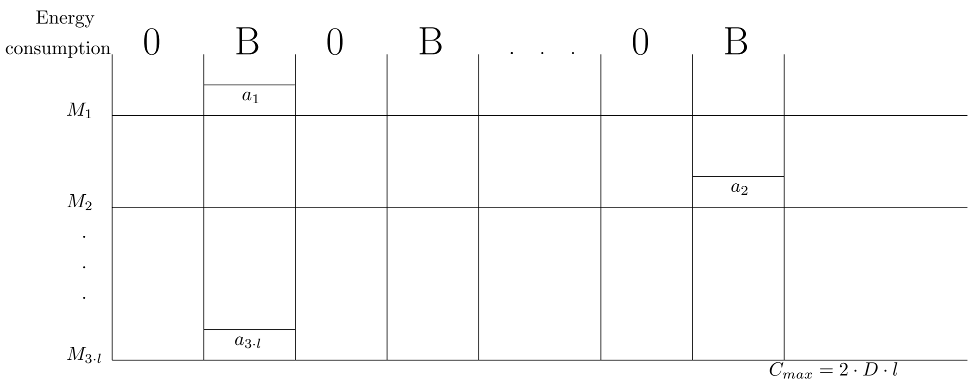

Figure 4.5: 3 Partition problem reduction

This work provides exact and heuristic approaches for solving MCELFOP because this is NP-hard problem. The approaches are described below.

Exact makespan calculation with energy limits for fixed order

The exact way uses the modification of the exact method from section 4.1.2. The following constraint sets fixed ordering on machines and is added to the Overlap method:

$$end(t_{k'}) \leq start(t_k), o_{k'} = predm_k, o_k \in O, o'_k \in O \tag{4.22}$$

Heuristic makespan calculation with energy limits for fixed order

Algorithm 3 schedules the operations from the first metering interval. In each iteration, heuristic creates a set of the operations, for which job and machine predecessors were scheduled. From this set, the algorithm selects the operation, which can be scheduled in the current metering interval and has a maximum rank, where rank represents operation priority (will be describe later). Algorithm schedules selected operation as early as possible. If doesn't exist any operation, which can be scheduled in the current metering interval, the algorithm selects the next one. The process stops when all operations will be scheduled.

Algorithm 3 describes the heuristic way for calculating schedule with considering energy limits. The heuristic starts from the metering interval $\omega_1 \in \Omega$. For the current metering interval $\omega_e \in \Omega$, the algorithm creates subset $O' \subset O$, which consists of the operations, for which job and machine predecessors were scheduled. For all of the operation $o'_k \in O'$ heuristic tries to schedule o'_k (this is computed by the algorithm 4, which is described below) in the current metering interval ω_e. Also for all operations $o'_k \in O'$ algorithm calculates the rank of the operation o_k. For scheduling, heuristic selects an operation $o'_k \in O'$, which can be scheduled in the current metering interval ω_e and has the maximum rank. If no operation exists, which has these specified properties, the algorithm selects the next metering interval ω_{e+1}. This heuristic works while there exists any operation, which was not scheduled.

The rank of an operation $o_k \in j$, $o_k \in L_h$ can be calculated two ways. In the first way, the rank is maximum between a sum of processing time of the operations which are processed on machine h after the operation o_k and a sum of processing time of the operations which are belong to job j and processed after the operation o_k. This way is called the Remaining Work rank. In the second way, the rank is the longest path between operations o_k and $o+1$ in the disjunctive graph from the section 2.1.1. This way is called the Longest Path rank.

The algorithm 4 calculates the minimum start time s'_k of the operation $o'_k \in O'$. At the start, it sets s'_k in maximum between the end of machine and job predecessors. After this algorithm finds the maximum overlap between operation $o'_k \in O'$ and the current metering interval $\omega_e \in \Omega$. By this information, heuristic finds the final start time s'_k. After schedule function, $CheckEnengy$ controls the energy constraints on the remaining metering intervals with which operation o'_k overlaps. The result $s'_k = \propto$ means, that operation o'_k can't be scheduled in the current metering interval ω_e and algorithm will try to schedule this operation on the next metering intervals.

After selecting the new scheduled operation $o'_k \in O'$, heuristic sets it start time s'_k, recalculates energy for all metering intervals with which o'_k overlaps and adds it to set R, which consists of the scheduled operations. The algorithm stops when all operations from the set O will be scheduled.

The algorithm was inspired from [MvH17].

4.2.5 Relinking procedure

The Relinking procedure [PLC14] is used to helps the Taboo search to escape from local optimum.

For two randomly selected solutions S_i and S_g from the population, the Relinking procedure generates a $PathSet$ which connects them. At the start, the algorithm creates a set $NCS(S_i, S_g)$, which consists of all operations having different positions on machines in solutions S_i and S_g. To create $PathSet$, the Relinking procedure selects random operation $o_k \in NCS(S_i, S_g)$ and swaps it with an operation $o_{k'} \in NCS(S_i, S_g)$ so that after swapping in solution S_i operation o_k will be on the same position in both solutions. The new solution is added to the $PathSet$. Function $Dis(S_i, S_g)$ returns number of the operations which have different places in S_i and S_g. After creating $PathSet$ algorithm improves all solution with Taboo search from it.

However, is not efficient to improve all solutions from the $PathSet$, because close solutions differing in a few number of swaps will lead to the same local optimum. Instead, it is better to add solutions to the $PathSet$ only after

Algorithm 3 Heuristic Makespan calculation for fixed order

```
1: function HeuristicMakespanCalculationForTheFixedOrder($E_{max}$,$O$,$P$)
2:     $E := [E^1_{max}, E^2_{max}, ...E^{H/D}_{max}]$
3:     $\omega_e := 1$
4:     $R := \varnothing$
5:     while $R \neq O \backslash \{o+1\}$ do
6:         repeat
7:             $rank := 0$
8:             $o_k := \varnothing$
9:             $s_k := \varnothing$
10:            construct $O'$
11:            for $o'_k \in O'$ do
12:                $s'_k := CalculateStartTime(o'_k, P_k, E, \omega_e, D)$
13:                $rank' := CalcOperationRank(o'_k, s_k)$
14:                if $(s'_k \neq \propto)$ AND $(rank' > rank)$ then
15:                    $rank := rank'$
16:                    $o_k := o'_k$
17:                    $s_k := s'_k$
18:                end if
19:            end for
20:            if $o_k \neq \varnothing$ then
21:                $SetOperationStartTime(o_k, s_k)$
22:                $RecalculateEnergy(o_k, s_k, E)$
23:                $R = R \bigcup o_k$
24:            end if
25:        until $(E_e = 0)$ OR $(o_k = \varnothing)$
26:        $\omega_e := \omega_e + 1$
27:    end while
28:    $s_{o+1} = max_{o_k \in O}(s_k + p_k)$
           return $max_{o_k \in O}(s_k + p_k)$
29: end function
```

performing some number of swaps. Therefore, we construct the *PathSet* as follows: between the first added solution to the *PathSet* and S_i must be α swaps, between solutions in the *PathSet* must be β swaps, between the last added solution in the *PathSet* and the S_g must be at least α swaps. The solutions, which were added to *PathSet*, call Candidate solutions, the remain solutions call Intermediate solutions. Figure 4.6 illustrates this approach. Parameters α and β is given in table 4.2. The Relinking procedure is described by algorithm 5.

Algorithm 4 Calculate start time

```
 1: function CalculateStartTime($o_k$, $P_k$ $E$, $\omega_e$, $D$)
 2:     $s_k =: max(s_{predj_k} + p_{predj_k}, s_{predm_k} + p_{predm_k})$
 3:     $overlap := min(p_k, D, \frac{E_e}{P_k})$
 4:     if $(s_k \leq \omega_e \cdot D)$AND$(overlap > 0)$ then
 5:         if $overlap = p_k$ then
 6:             $s_{overlap} := \omega_e \cdot D - D$
 7:         else
 8:             $s_{overlap} := \omega_e \cdot D - overlap$
 9:         end if
10:         $s_k = max(s_k, s_{overlap})$
11:         if $CheckEnergy(o_k, s_k, E) = FALSE$ then
12:             $s_k :=\propto$
13:         end if
14:     else
15:         $s_k :=\propto$
16:     end if
         return $s_k$
17: end function
```

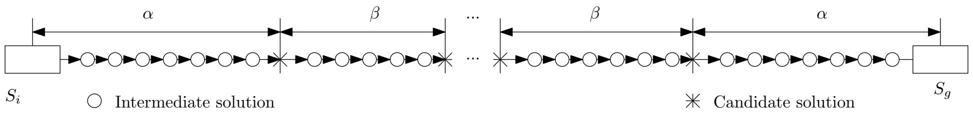

Figure 4.6: Relinking procedure

α	$\frac{(\lvert NCS(S_i,S_g)\rvert)}{5}$
β	$max(\frac{\lvert NCS(S_i,S_g)\rvert}{10}, 2)$

Table 4.2: Parameters of the Relinking procedure

Algorithm 5 Relinking procedure

```
1: function PathRelinking($S_i$, $S_g$)
2:     Generate the $NCS(S_i, S_g)$ set
3:     $PathSet := \emptyset$
4:     for $k = 1...\alpha$ do
5:         Randomly select an operation $o_k \in NCS(S_i, S_g)$
6:         Swap the operation $o_k$ and an operation $o_{k'}$ in $S_i$ so that position of $o_k$ in $S_i$ is the same as in $S_g$
7:         $NCS(S_i, S_g) := NCS(S_i, S_g) \setminus o_k$
8:     end for
9:     $PathSet := PathSet \bigcup S_i$
10:    while $Dis(S_i, S_g) > \alpha$ do
11:        for $k = 1...\beta$ do
12:            Randomly select an operation $o_k \in NCS(S_i, S_g)$
13:            Swap the operation $o_k$ and an operation $o_{k'}$ in $S_i$ so that position of $o_k$ in $S_i$ is the same as in $S_g$
14:            $NCS(S_i, S_g) := NCS(S_i, S_g) \setminus o_k$
15:        end for
16:        $PathSet := PathSet \bigcup S_i$
17:    end while
18:    for $S_j \in PathSet$ do
19:        if $S_j$ is infeasible then
20:            $Repair(S_j)$
21:        end if
22:        $S_j := SlightTabooSearch(S_j)$
23:    end for
24:    $S_r := argmin\{f(S_j), S_j \in PathSet\}$
25:    $S_r := StrongTabooSearch(S_r)$
         return $S_r$
26: end function
```

Chapter 5

Experiments

This chapter compares exact methods from section 4.1 and heuristic methods from section 4.2. The first part of the chapter describes how the test instances were generated and the second part compares exact methods on small instances. At the last part the best exact method is compared against the heuristic methods with the following modifications:

1. for Makespan calculation with energy limits for fixed order is used the heuristic method from section 4.2.4
   a. method uses the Remaining Work ranking function from section 4.2.4. Method is noted HM-RW.
   b. method uses the Longest Path ranking function from section 4.2.4. Method is noted HM-LP.
2. for Makespan calculation with energy limits for fixed order is used the exact method from section 4.2.4:
   a. for horizon calculation is used the heuristic method with the Remaining Work ranking function from section 4.2.4. Method is noted EM-RW.
   b. for horizon calculation is used the heuristic method with the Longest Path ranking function from section 4.2.4. Method is noted EM-LP.

Exact methods notification:

1. the Optional Variables method from section 4.1.1 is noted OPTIONAL
2. the Overlap method from section 4.1.2 is noted OVERLAP

The algorithms were written in C++, and the experiments were running on PC with Intel Xeon CPU E5-2620 v2 2.10 GHz and 64 GB RAM under the Gentoo operating system.

5.1 Generating test instances

To generate the test instances, we extended the standard benchmark instances from literature with energy limits. The following benchmarks were considered:

1. TA instances due to Taillard [vH]
2. SWV instances due to Storer [vH]
3. LA instances due to Lawrence [vH]
4. FT instances due to Fisher and Thompson [vH]
5. ORB instances due to Applegate and Cook [vH]

However, the energy limit was set to be same for all instances.

To generate the power consumption, we use two parameters α and β, which represent the lower bound and the upper bound of power consumption. The power consumption of the operations are then sampled from continuous uniform distribution, equation 5.1. Constraint 5.2 ensures that any operation for a metering interval doesn't consume more energy, than energy limit, equation 5.2.

$$U\left[\frac{\alpha \cdot E_{max}}{m \cdot D}, \frac{2 \cdot E_{max}}{m \cdot D}\right], \alpha \leqslant \beta, \alpha = 1, 1.2 ... 2 \tag{5.1}$$

$$min(p_i, D) \cdot P_i \leq E_{max} \tag{5.2}$$

For each instance were generated 5 random instances with same the α and β. After each instance according to its number is added to the separate sets, which call Generation.

5.2 Experiments with the exact methods

This section compares two exact algorithms described in section 4.1. To implement the methods, was chosen IBM CP Optimizer. The experiment was carried out in the following instances: FT 06 of 6×6 and ORB 07 of 10×10.

Tables 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10, 5.11, 5.12 represent experiment results for instances, which were generated from FT06 and ORB07.

If the cell in the *Objective* column is highlighted in green, it means that method found the optimal solution and proved its optimality during the time limit of 3600 s. The red color indicates that the method didn't prove, that found solution is the optimal one.

Tables 5.1 and 5.2 show how horizon influences on the number of variables and constraints for two methods. As can be seen, the OPTIONAL method needs more variables and constraints than OVERLAP method, which has an impact on the efficiency of the methods.

Method	H = 100		H 200	
	Variables	Constraints	Variables	Constraints
OPTIONAL	258	114	510	121
OVERLAP	50	43	57	50

Table 5.1: Horizon influence for FT 06

Method	H = 100		H = 200	
	Variables	Constraints	Variables	Constraints
OPTIONAL	710	306	1410	313
OVERLAP	118	107	125	114

Table 5.2: Horizon influence for ORB 07

α	OPTIONAL		OVERLAP	
	Time [s]	Objective	Time [s]	Objective
1	0.808	55	0.149	55
1.2	0.822	57	0.682	57
1.4	2.57	58	1.539	58
1.6	0.739	59	2.86	59
1.8	0.76	62	1.455	62
2	9.903	65	0.534	65

Table 5.3: Instance FT 06 Generation 1

α	OPTIONAL		OVERLAP	
	Time [s]	Objective	Time [s]	Objective
1	0.744	56	0.633	56
1.2	1.177	56	2.805	56
1.4	1.434	59	0.647	59
1.6	0.817	59	3.024	59
1.8	0.757	63	0.35	63
2	0.756	65	0.719	65

Table 5.4: Instance FT 06 Generation 2

α	OPTIONAL		OVERLAP	
	Time [s]	Objective	Time [s]	Objective
1	0.841	56	0.45	56
1.2	0.821	56	0.451	56
1.4	0.777	58	1.487	58
1.6	0.758	59	3.509	59
1.8	0.789	63	0.266	63
2	0.757	65	0.712	65

Table 5.5: Instance FT 06 Generation 3

α	OPTIONAL		OVERLAP	
	Time [s]	Objective	Time [s]	Objective
1	0.838	55	0.263	55
1.2	0.748	57	0.657	57
1.4	1.335	58	0.692	58
1.6	0.736	59	3.477	59
1.8	27.25	62	2.976	62
2	0.773	65	0.711	65

Table 5.6: Instance FT 06 Generation 4

α	OPTIONAL		OVERLAP	
	Time [s]	Objective	Time [s]	Objective
1	0.852	55	0.16	55
1.2	0.847	56	0.214	56
1.4	1.737	59	1.706	59
1.6	1.142	59	4.168	59
1.8	0.782	62	1.477	62
2	0.763	65	0.707	65

Table 5.7: Instance FT 06 Generation 5

α	OPTIONAL		OVERLAP	
	Time [s]	Objective	Time [s]	Objective
1	220.8	401	31.94	401
1.2	205.9	401	39.66	401
1.4	82.7	418	342.4	414
1.6	2319	436	2222	435
1.8	546	457	960.2	456
2	49.3	485	7.08	485

Table 5.8: Instance ORB 07 Generation 1

α	OPTIONAL		OVERLAP	
	Time [s]	Objective	Time [s]	Objective
1	100.9	403	26.39	403
1.2	3036	403	45.01	403
1.4	3443	419	257.5	415
1.6	35.51	435	1667	432
1.8	182.5	458	215.2	456
2	50.75	485	7.17	485

Table 5.9: Instance ORB 07 Generation 2

α	OPTIONAL		OVERLAP	
	Time [s]	Objective	Time [s]	Objective
1	622	403	9.377	403
1.2	69.44	409	172.5	407
1.4	2485	412	2040	411
1.6	1107	444	265.9	442
1.8	62.87	459	31.96	458
2	48.6	485	7.08	485

Table 5.10: Instance ORB 07 Generation 3

α	OPTIONAL		OVERLAP	
	Time [s]	Objective	Time [s]	Objective
1	68.19	401	32.41	401
1.2	480.2	402	10.45	402
1.4	246.4	419	320.2	416
1.6	60.78	439	89.41	438
1.8	464.1	358	271.3	458
2	49.8	485	7.228	485

Table 5.11: Instance ORB 07 Generation 4

α	OPTIONAL		OVERLAP	
	Time [s]	Objective	Time [s]	Objective
1	287.6	401	31.68	401
1.2	2735	405	63.9	404
1.4	246.7	420	19.94	418
1.6	3270	437	637.2	437
1.8	64.83	462	15.63	461
2	48.79	485	7.166	485

Table 5.12: Instance ORB 07 Generation 5

5.3 Analysis of the experiments with the exact methods

Tables 5.1 and 5.2 show that the OPTIONAL method it slower because it has more number of variables and constraints than the OVERLAP method. Therefore, in the most case, the OVERLAP method is faster.

The experiments show that for small instances the OPTIONAL and OVERLAP methods work similarly, but for the bigger instances, the OPTIONAL method is slower than the OVERLAP method in 75% of cases.

5.4 Experiments with the heuristic methods

This section compares the OVERLAP method against the heuristic methods. Because on the large and medium instances the OVERLAP method can't find the optimal solution in the reasonable time the OVERLAP method is considered as the heuristic method.

Exact makespan calculation for fixed order was implemented with the IBM CP Optimizer. Horizon for the OVERLAP method was calculated as average makespan of 100 random generated solutions. The horizon for this solution was calculated by Remaining Work ranking function from section 4.2.4.

For experiments the next instances were selected:

1. the small instances
   a. FT 06, size 6×6, the horizon for the Overlap method is 100
2. the medium instances
   a. LA 01 and LA 05, size 10×5, the horizon for the Overlap method is 1600
   b. SWV 06 and SWV 10, size 20×15, the horizon for the Overlap method is 8000
   c. LA 31 and LA 35, size 30×10, the horizon for the Overlap method is 7500
3. the large instances
   a. TA 41 and TA 49, size 30×20, the horizon for the Overlap method is 15000
   b. TA 51 and TA 57, size 50×15, the horizon for the Overlap method is 17500

For all instances, 5 instances with the same α and β were generated. All methods have the time limit of 600 s.

The OVERLAP method was able to prove optimality within the time limit only for some instances generated from FT 06; for all other instances were not proven optimal.

The table 5.13 shows the average objective and standard deviation for the small and medium instances. The table 5.14 shows average time, when the best solution was found, and standard deviation for the small and medium instances.

For the large instances, the Overlap method in the most case didn't find any solution. The tables 5.15 and 5.16 show the average objective value and average time when the best solution was found. Table 5.17 shows how many solutions the Overlap method found during the time limit. Table 5.18 shows results for the large instances for which the Overlap method found a solution.

In tables 5.13, 5.14, 5.15, 5.16 the average objective and standard deviation were calculated for all Generations, α and β.

The figures 5.1, 5.2, 5.3, 5.4, 5.5, 5.6 illustrate the convergence to the best solution for all type of instances with $\alpha = 2$, $\beta = 2$. The points on the figures represent found solutions. As can be seen, for the medium and large instance EM-RW and EM-LP methods find only one solution during the run, because these methods didn't have time to initialize the population.

Instance	OVERLAP	HM-RW	HM-LP	EM-RW	EM-LP
ft06	59.23 ± 3.501	59.91 ± 3.085	59.47 ± 3.389	59.38 ± 3.407	59.29 ± 3.462
la01	1035 ± 131.8	1035 ± 126.8	1031 ± 127.4	1056 ± 137.5	1056 ± 137.7
la05	815.9 ± 96.18	815.6 ± 93.94	811.8 ± 92.6	823 ± 111.6	831.2 ± 103.5
swv06	1976 ± 103.5	2009 ± 75.58	1983 ± 66.62	4637 ± 201.2	4541 ± 231.9
swv10	2057 ± 83.48	2073 ± 79.1	2044 ± 71.46	4746 ± 230.9	4759 ± 226.5
la31	2639 ± 272.5	2651 ± 268.7	2642 ± 269.8	4724 ± 298.2	4725 ± 260.4
la35	2695 ± 274.4	2704 ± 272.5	2695 ± 274.7	4885 ± 245.4	4917 ± 193.5

Table 5.13: The objectives of the OVERLAP and the heuristic methods for small and medium instances

Instance	OVERLAP [s]	HM-RW [s]	HM-LP [s]	EM-RW [s]	EM-LP [s]
ft06	6.156 ± 4.293	3.497 ± 13.09	0.9041 ± 1.392	201.4 ± 194.3	187.6 ± 178.9
la01	240.5 ± 176.3	300.5 ± 154.9	310.1 ± 158.6	593.4 ± 28.71	596.7 ± 8.041
la05	278 ± 164.9	253.8 ± 156.6	306.7 ± 162.7	580.8 ± 83.16	594 ± 28.86
swv06	587.5 ± 12.6	560.2 ± 42.47	568.6 ± 34.14	598.4 ± 1.212	598.4 ± 1.199
swv10	591.8 ± 10.92	556.4 ± 40.61	562 ± 39.2	598.7 ± 1.346	598.3 ± 1.606
la31	324.6 ± 193.4	405.8 ± 123.1	362.4 ± 147.2	598.1 ± 1.732	597.7 ± 1.718
la35	497.5 ± 108.5	422.2 ± 110.3	343.2 ± 158.2	598 ± 1.736	598.1 ± 1.614

Table 5.14: The time of the OVERLAP and the heuristic methods for small and medium instances

Instance	HM-RW	HM-LP	EM-RW	EM-LP
ta41	2759 ± 234.5	2707 ± 249.5	13311 ± 466.3	13226 ± 536.1
ta49	2647 ± 213.9	2590 ± 223.1	12562 ± 477.9	12518 ± 495
ta51	4412 ± 438.3	4358 ± 451.4	16344 ± 486	16275 ± 528.2
ta57	4458 ± 447.6	4418 ± 456.3	16631 ± 457.2	16724 ± 466.9

Table 5.15: The objectives of the heuristic methods for large instances

Instance	HM-RW [s]	HM-LP [s]	EM-RW [s]	EM-LP [s]
ta41	542.1 ± 54.96	486.1 ± 104.9	598 ± 1.83	598.3 ± 1.792
ta49	540.9 ± 58.25	517.4 ± 86.82	597.9 ± 1.6	597.9 ± 1.768
ta51	540 ± 59.81	454.5 ± 114	598.2 ± 1.865	597.5 ± 1.856
ta57	528.2 ± 59.87	461.5 ± 98.66	598 ± 1.841	598 ± 1.798

Table 5.16: The time of the heuristic methods for large instances

	ta41	ta49	ta51	ta59
Count found solutions (%)	33	23	0	0

Table 5.17: The number of solutions obtained the OVERLAP method for the large instances

Instance	Generation	α	OVERLAP	HM-RW	HM-LP	EM-RW	EM-LP
ta41	0	1.8	3013	2957	2921	13296	13227
ta41	0	2	3146	3134	3110	13641	13490
ta41	1	1.8	2999	2954	2926	13035	13120
ta41	1	2	3146	3138	3110	13849	13237
ta41	2	1.8	3003	2963	2921	13451	13027
ta41	2	2	3146	3132	3111	13187	13492
ta41	3	1.8	2994	2945	2917	13157	13105
ta41	3	2	3146	3142	3111	13011	13016
ta41	4	1.8	2990	2959	2919	13159	13195
ta41	4	2	3146	3132	3110	13329	13577
ta49	0	2	3018	2985	2957	12399	12481
ta49	1	2	3018	2995	2958	12695	12169
ta49	2	2	3018	2986	2957	12920	12535
ta49	3	1.8	2872	2827	2780	12296	12070
ta49	3	2	3018	2991	2958	13116	13095
ta49	4	1.8	2872	2829	2777	13225	12560
ta49	4	2	3018	2984	2958	12560	12887

Table 5.18: The objective of the Overlap and the heuristic methods for large instances, where the Overlap method obtained the solution

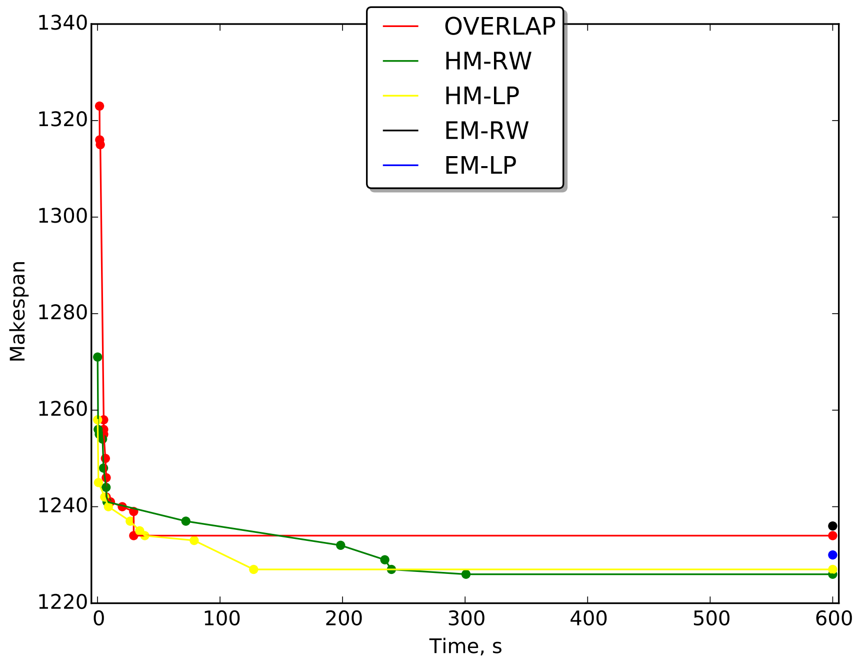

Figure 5.1: LA 01, $\alpha = 2$, $\beta = 2$

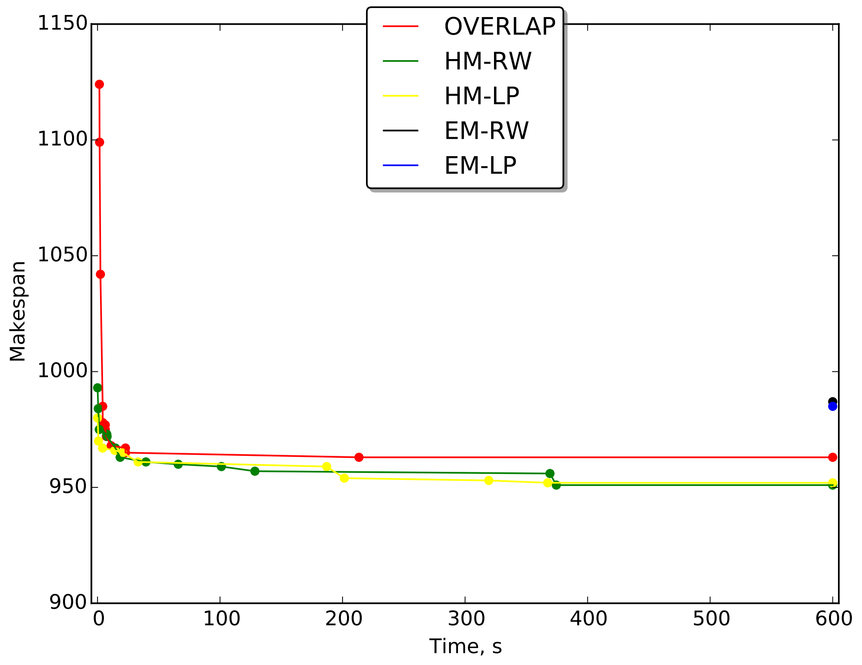

Figure 5.2: LA 05, $\alpha = 2$, $\beta = 2$

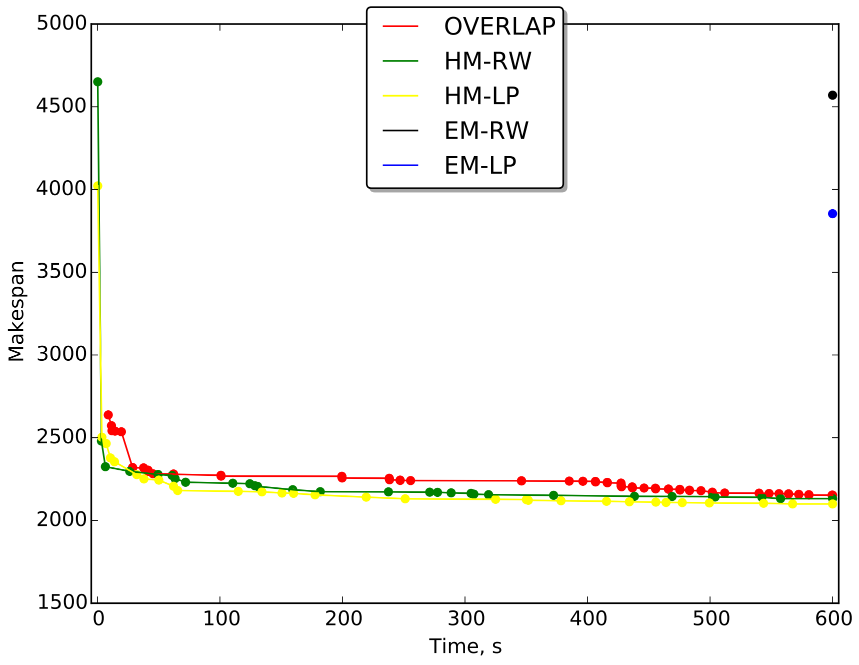

Figure 5.3: SWV 06, $\alpha = 2$, $\beta = 2$

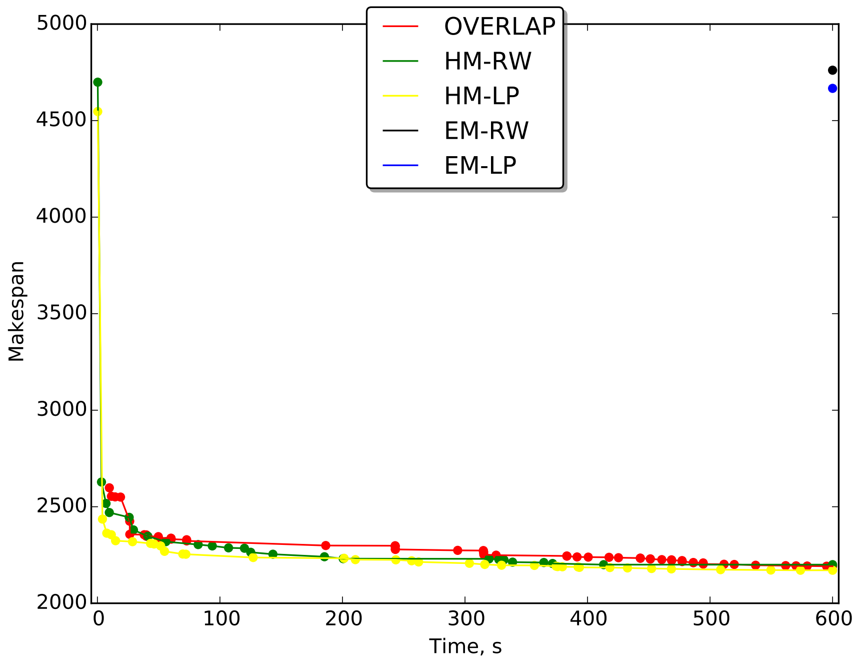

Figure 5.4: SWV 10, $\alpha = 2$, $\beta = 2$

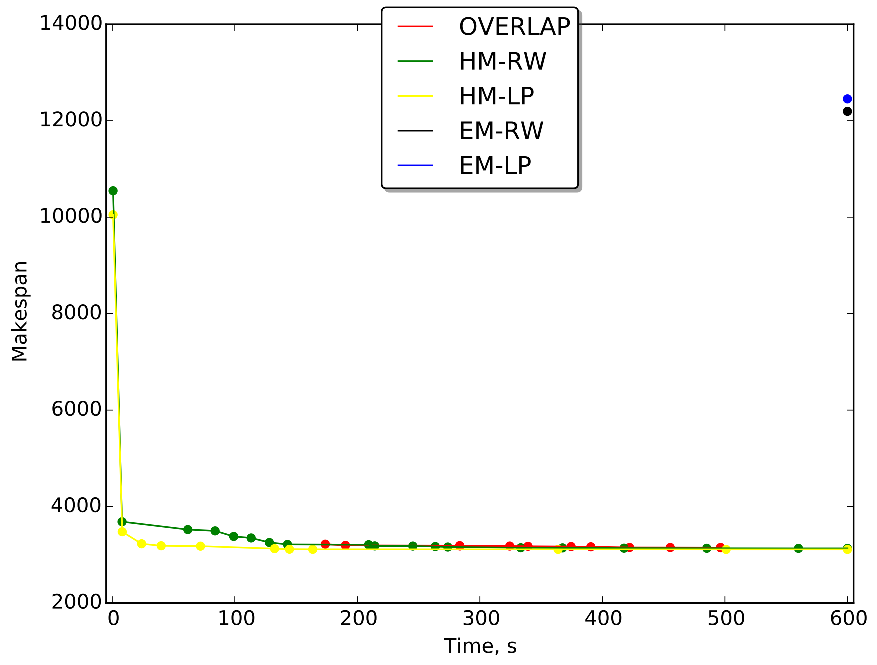

Figure 5.5: TA 41, $\alpha = 2$, $\beta = 2$

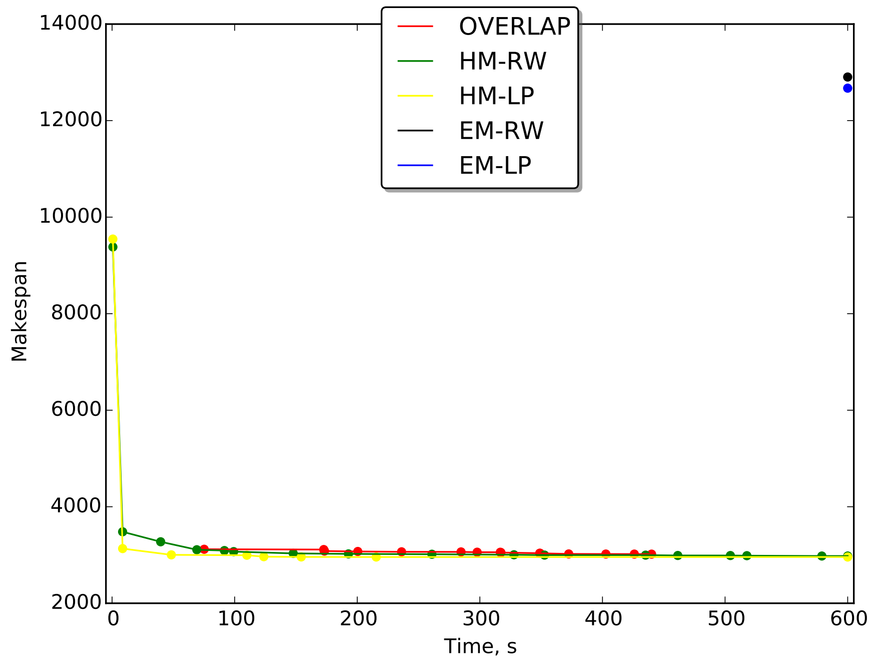

Figure 5.6: TA 49, $\alpha = 2$, $\beta = 2$

5.5 Analysis of the heuristic methods

5.5.1 Comparison of the heuristic methods

As can be seen, from tables 5.13 and 5.15 the best heuristic method is the HM-LP method. It has the best objectives among all heuristic methods because HM-LP method has the more accurate suggestion (i.e., rank), which operation is more critical when algorithm calculates makespan for fixed order.

EM-RW method and EM-LP method found worse solutions than HM-LP method, which don't use IBM CP Optimizer, because makespan calculation is strongly NP-hard and IBM CP Optimizer can't find the exact solution for fixed order during the reasonable time.

5.5.2 Comparison of the best heuristic and the best exact method

OVERLAP method and HM-LP method have almost the same results on the small and medium instances. As can be seen, from table 5.17 for the large instances OVERLAP method in the most case can't find any solution since large instances have a large horizon for which IBM CP Optimizer solver generates larger model. For the remaining large instances, OVERLAP method found the solutions, which are worse by 2%, than the HM-LP method.

HM-LP method and HM-RW method need less memory, than OVERLAP method. Thus heuristic methods can run for the large instances on a common customer computer. For TA instances, OVERLAP method needs about 20 GB RAM, whereas HM-LP method needs only about 30 MB RAM.

Chapter 6 Conclusion

This master thesis deals with the Job Shop Scheduling problem (JSSP) with energy limits, which is NP-hard. The energy limits constraint the total energy consumption of the machines within the so-called metering intervals. The motivation for solving this problem is that the JSSP is important for manufacturing where the energy limits are contracted with the energy system. The manufacturing company are financially penalized if they violate the contracted energy limits since overconsumption leads to instability of the electrical grid.

The literature review shows that very few articles exist which consider scheduling together with energy limits. Existing articles consider scheduling with the energy limits on the fewer fixed number of machines.

In this master thesis, we created two exact and four heuristic methods. The exact methods are fully our contribution. For implementation, these methods we used IBM CP Optimizer. The presented heuristic methods are the extension of the existing approaches by the energy limits. Specially, we designed a procedure that, given a fixed-operation ordering on the machines, finds start times of the operation that don't violate the energy limits. Moreover, we proved that finding the optimal start times that don't violate energy limits for the fixed operation ordering on the machines is NP-hard.

During the experiments methods were tested and benchmarked. The largest instances, which were used in the experiments, have size 50×15, where 50 is number of jobs, 15 number of machines. The experiments show that even for small instances JSSP with energy limits is very hard. Any exact method didn't find the optimal solutions for all small instances, which were tested. The experiments suggest that the OVERLAP method is more efficient than the OPTIONAL method.

Moreover, we compared heuristic methods and the OVERLAP method. The experiments show that the best heuristic method is the HM-LP method. For the large instances, the OVERLAP method can't find any solutions and needs a lot of memory, whereas the HM-LP method doesn't need a lot of memory and finds better solutions. Unlike the OVERLAP method, the HM-LP method doesn't need IBM CP Optimizer, which is the advantage of the HM-LP method. Therefore, the best method for the solving JSSP with energy limits providing by this work is the HM-LP method.

Bibliography

[BJS92] Peter Brucker, Bernd Jurisch, and Bernd Sievers, *A branch and bound algorithm for the job-shop scheduling problem*, Elsiever (1992).

[BM14] Daniela Borissova and Ivan Mustakerov, *A parallel algorithm for optimal job shop scheduling of semi-constrained details processing on multiple machines*, ResearchGate (2014).

[BV98] Egon Balas and Alkis Vazacopoulos, *Guided local search with shifting bottleneck for job shop scheduling*, Management Science (1998).

[dejW17] Ren Qing dao-er ji and Yuping Wang, *A new hybrid genetic algorithm for job shop scheduling*, Elsevier (2017).

[GL97] Fred W. Glover and Manuel Laguna, *Tabu search*, Springer US, 1997.

[IBM] IBM, *Ibm knowledge center*, `www.ibm.com/support/knowledgecenter/`, Accessed: 2018-05-17.

[LC10] Ye LI and Yan CHEN, *A genetic algorithm for job-shop scheduling*, Journal of software (2010).

[MA16] S Afshin Mansouri and Emel Aktas, *Minimizing energy consumption and makespan in a two-machine flowshop scheduling problem*, Journal of the Operational Research Society (2016).

[MRX01] Yazid Mati, Nidhal Rezg, and Xiaolan Xie, *Scheduling problem of job-shop with blocking: A taboo search approach*, MIC'2001 (2001).

[MvH17] István Módos, Přemysl Šůcha, and Zdeněk Hanzálek, *Algorithms for robust production scheduling with energy consumption limits*, Elsevier (2017).

[PLC14] Bo Peng, Zhipeng Lü, and T.C.E. Cheng, *A tabu search/path relinking algorithm to solve the job shop scheduling problem*, Elsevier (2014).

[vH] van Hoorn, *Job shop instances and solutions*, `http://jobshop.jjvh.nl/`, Accessed: 2018-05-16.

[WLZW97] Jihua Wang, Peter B. Luh, Xing Zhao, and Jinlin Wang, *An optimization-based algorithm for job shop scheduling*, Sadhana (1997).

[ZLGR06] Chao Yong Zhang, PeiGen Li, ZaiLin Guan, and YunQing Rao, *A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem*, Elsevier (2006).

Appendix A

Conten of the CD

Figure A.1 represents CD structure. Folder **PDF document** contains thesis document and specification. Thesis document source code is in **Latex document** folder.

```
├── heuristic
│   └── modules
├── instances
├── Latex document
│   ├── experiments
│   └── fig
├── optional varialbles
│   └── modules
├── overlap
│   └── modules
└── PDF document
```

Figure A.1: Disk structure

Folder **instances** contains instance examples. Input instance format:

$n\ m\ E_{max}\ H$
$mu_{0,0}\ p_{0,0}\ P_{0,0}\ mu_{0,1}\ p_{0,1}\ P_{0,1}\ \ldots\ mu_{0,m-1}\ p_{0,m-1}\ P_{0,m-1}$
$mu_{1,0}\ p_{1,0}\ P_{1,0}\ mu_{1,1}\ p_{1,1}\ P_{1,1}\ \ldots\ mu_{1,m-1}\ p_{1,m-1}\ P_{1,m-1}$
$\ldots$
$mu_{n-1,0}\ p_{n-1,0}\ P_{n-1,0}\ mu_{n-1,1}\ p_{n-1,1}\ P_{n-1,1}\ \ldots\ mu_{n-1,m-1}\ p_{n-1,m-1}$
$P_{n-1,m-1}$

where:

- n number of job
- m number of machines
- E_{max} energy limit
- H horizon
- $mu_{j,i}$ machine index, on which operation i of job j has to be processed
- $p_{j,i}$ processing time of operation i of job j
- $P_{j,i}$ power consumption operation i of job j

After calculation programs return a schedule. Output schedule format:

$ord_{0,0}$ $s_{0,0}$ $ord_{0,1}$ $s_{0,1}$ ... $ord_{0,n-1}$ $s_{0,n-1}$
$ord_{1,0}$ $s_{1,0}$ $ord_{1,1}$ $s_{1,1}$ ... $ord_{1,n-1}$ $s_{1,n-1}$
...
$ord_{m-1,0}$ $s_{m-1,0}$ $ord_{m-1,1}$ $s_{m-1,1}$ ... $ord_{m-1,n-1}$ $s_{m-1,n-1}$

where:

- $ord_{m,k}$ index of a job that is on position k in processing order on machine m
- $s_{m,k}$ start time of the operation that is on position k in processing order on machine m

The remaining folder contain C++ source code of the methods. Folder name represent method name. Heuristic methods located in **heuritic** folder. For compiling a program go to the folder *method name* and run commands

```
cmake ./
make
```

File *heuristic\Algorithm_parameters.h* contains define directives for compiling certain heuristic methods. For all binary the first parameter is instance file, the second parameter is timeout (sec). Tables A.1 shows set of directives for compiling corresponding methods.

Method	LONGEST PATH	EXACT CALCULATION	ONE WORKER
HM-RW			
HM-LP	●		
EM-RW		●	●
EM-LP	●	●	●

Table A.1: Compilation parameter