



**CZECH TECHNICAL
UNIVERSITY
IN PRAGUE**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

Master's Thesis

Energy Optimization of Robotic Cells with Single Robot and m Machines

Bc. Jakub Chmel

**Cybernetics and Robotics - Robotics
chmel.jakub@seznam.cz**

May 2018

Supervisor: doc. Ing. Přemysl Šůcha, Ph.D.

Acknowledgement / Declaration

I would like to thank my advisor Přemysl Šůcha for his invaluable guidance and advice, patience and support throughout the year. I would also like to thank Libor Bukata for his help while formulating a mathematical model. Finally, I would like to thank my family for their endless support.

Author statement for undergraduate thesis:

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

In Prague, 25 May 2018

.....

Abstrakt / Abstract

Práce se zaměřuje na optimalizaci spotřeby energie robotické buňky typu *flow shop*. Uvažujeme robotickou buňku skládající se z m strojů a jednoho transportního robota, která produkuje jeden typ výrobku. Robot přenáší výrobky mezi stroji a zajišťuje naložení a vyložení strojů. Uvažujeme cyklický rozvrh pohybů robota s cílem minimalizovat spotřebu energie.

Problém formulujeme jako smíšené celočíselné lineární programování (MILP). Popisujeme postup, kterým ověřujeme správnost MILP formulace. Dále jsme vyvinuli heuristický přístup založený na genetickém algoritmu pro tento NP těžký problém.

Výsledky práce naznačují, že běžné výrobní *flow shop* linky mohou být efektivně optimalizovány exaktním algoritmem. Nicméně větší robotické buňky s více stroji mohou dosáhnout v rozumném čase téměř optimálního řešení pomocí heuristického přístupu. Hlavní přínos práce je odlišná kriteriální funkce oproti ostatním pracím, které se zabývají optimalizací výrobních linek typu *flow shop*.

Klíčová slova: Rozvrhování robotických buněk, flow shop, smíšené celočíselné lineární programování (MILP), genetický algoritmus (GA), minimalizace spotřeby energie

Překlad titulu: Optimalizace spotřeby robotických buněk s jedním robotem a m stroji

This thesis focuses on the energy optimization of flow shop type manufacturing cells. We consider cells consisting of m machines and a material handling robot producing one type of a part. The robot transfers parts between machines and ensures machines loads and unloads. We consider the cyclic scheduling of the robot moves with the objective of minimizing the energy consumption.

We develop a mixed integer linear programming formulation (MILP) of the problem. We describe a verification procedure to evaluate the correctness of the MILP formulation. We also develop a heuristic approach based on genetic algorithm for this NP-hard problem.

The result of this thesis indicates that a common flow shop type manufacturing cells can be effectively optimized by the exact algorithm. However, larger robotic cells with more machines can achieve a near-optimal solution by the heuristic approach in a reasonable time. The main contribution of the thesis is a different objective function compared to other studies that deal with the optimization of the flow shop type manufacturing cells.


Keywords: Robotic cell scheduling, flow shop, mixed integer linear programming formulation (MILP), genetic algorithm (GA), energy consumption minimization

Contents /

1 Introduction	1	6 Benchmark Instances	32
1.1 Related work	1	6.1 Generating of Benchmarks	32
1.2 Our research	2	6.2 Set of Benchmarks	35
1.3 Thesis outline	3	7 Experiments	37
2 Problem Statement	4	7.1 Exact Algorithm - Depen-	
2.1 Machine Environment	4	dence of an Execution Time	
2.1.1 Number of Machines		on a CT Value	37
per Stage	4	7.2 Exact Algorithm - Depen-	
2.1.2 Number and Type of		dence of Energy Consump-	
Robots.....	5	tion on a CT Value	38
2.1.3 Cell Layout.....	5	7.3 Genetic Algorithm - Param-	
2.2 Processing Characteristics.....	5	eter Calibration	38
2.2.1 Robot Operations.....	5	7.4 Comparison of Exact and	
2.2.2 Pickup Criterion	6	Heuristic Algorithm.....	39
2.2.3 Number of Part-Types....	7	8 Conclusion	43
2.2.4 Processing Strategy.....	7	References	44
2.3 Objective Function.....	7	A Specification	47
2.4 Example	8	B Abbreviations	49
3 Mixed Integer Linear Program-		C Enclosed CD	50
ming Formulation	13		
3.1 Sets	13		
3.2 Constant Parameters	13		
3.3 Decision Variables	14		
3.4 Constraints	15		
3.5 Objective Function.....	19		
3.6 Model Summary	20		
4 Verifying Correct Mathemati-			
cal Formulation in MILP Model ...	21		
4.1 Alternative Model Descrip-			
tion	21		
4.2 Comparative and Verifica-			
tion Procedure	24		
5 Heuristic Algorithm	27		
5.1 Genetic Algorithm	27		
5.1.1 Scheme of Genetic Al-			
gorithm	27		
5.2 Genetic Algorithm Aplication .	28		
5.2.1 Solution Encoding	28		
5.2.2 Population Size and			
Zero Population Gen-			
eration.....	28		
5.2.3 Fitness Computation	28		
5.2.4 Elite Solutions	30		
5.2.5 Crossover	30		
5.2.6 Mutation	30		
5.2.7 Termination Criterion ...	30		

Tables / Figures

<p>2.1. All other needed parameters for description of the robotic cell in Example 1 10</p> <p>2.2. The optimal RAS from Example 1 11</p> <p>6.1. Measured data from the simulated movement 33</p> <p>6.2. Summary of parameters and their values used for generating benchmark instances 35</p> <p>7.1. The levels of the parameters used in our GA formulation ... 39</p> <p>7.2. Summary of the results 40</p> <p>7.3. Summary of the results - continue 41</p>	<p>1.1. Robotic cell 2</p> <p>2.1. Classification scheme for robotic flow shops 4</p> <p>2.2. Definition of the robot activity .. 6</p> <p>2.3. Definition of the possible robot moves and waits of the robot in the idle state 7</p> <p>2.4. Robot operation energy consumption 8</p> <p>2.5. Demonstration of different energy-intensive robot configurations in the robot idle state 9</p> <p>2.6. Model of the robotic cell for Example 1 9</p> <p>2.7. Energy functions of time duration of loaded and unloaded robot movements, Example 1 10</p> <p>2.8. A solution to the problem from Example 1 11</p> <p>2.9. Gantt chart of the solution from Example 1 11</p> <p>3.1. A principle how we convert convex energy functions to piecewise-linear functions 14</p> <p>3.2. The relationship of two immediately following activities which transfer the same part .. 16</p> <p>3.3. The relationship of two arbitrary activities in the RAS 17</p> <p>4.1. Relations between activities for using Floyd-Warshall algorithm 22</p> <p>4.2. Visualization of the process of searching the part sequence numbers 23</p> <p>4.3. Searching the part sequence numbers for more complicated RAS example 23</p> <p>6.1. Dependence of energy consumption on a robot speed 34</p> <p>7.1. Exact algorithm - dependence of an execution time on a CT value 37</p>
---	---



7.2. Exact algorithm - dependence of energy consumption on a CT value	38
7.3. The parameter calibration result for the GA	39

Chapter 1

Introduction

The manufacturing companies are using more and more industrial robots. According to Robotic Industries Association, robot sales increased by 16 % to 294,312 units valued at \$ 13,1 billion in 2016, IFR (2017) [1]. The companies use industrial robots especially to increase productivity. However, raising the cost of electricity causes the pressure to reduce energy consumption. For example, industrial robots consume in average 8 % of the energy needed during the production of vehicles (see [2]). The primary goal of the optimization is to minimize the energy consumption of a robotic cell for a given production rate. For example, a real optimized robotic cell from Škoda Auto can save up to 20 % of energy by changing the robot speeds and applying power-saving modes (see [3]).

Manufacturing companies often use a Flow Shop (FS) type production system. This system consists of an input device denoted by M_0 , some set of machines denoted by M_1, \dots, M_m , an output device denoted by M_{m+1} , and a material handling robot [4], see Figure 1.1. We called this production system as a robotic cell. The robot transfers one part at a time between machines. All unprocessed parts are available on machine M_0 (input buffer) and follow the same operational sequence from M_1 to M_m . After that, finished parts are stored on machine M_{m+1} (output buffer). Such a robotic cell can produce identical parts and is referred to as a single-part-type cell. In contrast, a multiple-part-type cell processes different types of parts. These different part types require different processing times on a given machine. Dawande et al. (2007) [5], describe much diverse industrial application such as semiconductor manufacturing, production of printed circuit boards, testing board used in mainframe, textile mills or engine block manufacturing. More complex robotic cells need more sophisticated models and algorithms. The classical problem of FS is to determine the part sequence (PS) and the handling robot move sequence (RMS) that maximize throughput or minimize the cycle time (CT) [5, 4]. Minimize the CT means that robotic cell process the same set of parts indefinitely. The state of the system is described by the position of the robot and the status of all machines. The machine can be in the idle state or loaded with a part. The final state of a cycle schedule must be identical to its initial state to satisfy repetitiveness. However, the init state need not be an empty state in which all machines are idle. Cycle schedule repeats the same sequence robot moves indefinitely. This approach is easy to implement and control for real-life application.

1.1 Related work

Existing studies assume several variations of the FS. Sethi et al. (1992) [6] define an n -unit cycle as a sequence of robot activities in which the system returns to the same state after production n parts. Many studies use a multi-unit cycle according to the fact that multi-unit cycle can lead to a CT that is smaller than the smallest one-unit CT which proved Brauner and Finke (2001) [7]. Brauner et al. (2003) [8] studied a complexity of a one-cycle robotic flow-shops, and proved that the problem is NP-hard.

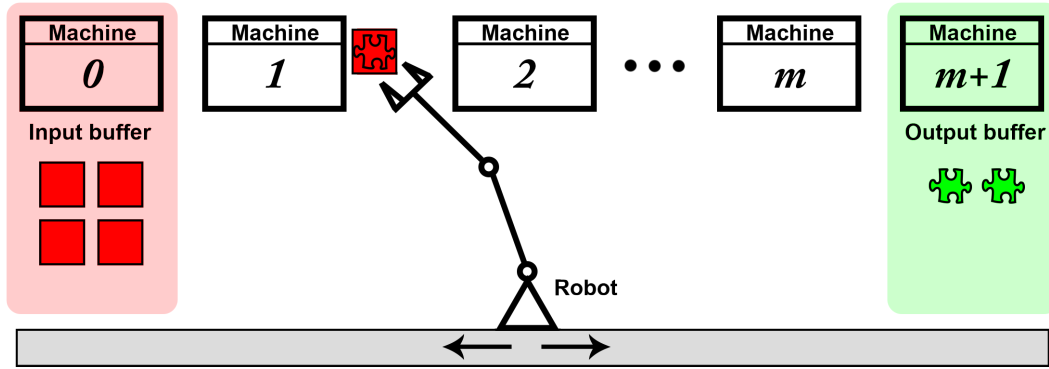


Figure 1.1. Robotic cell.

Some studies assume Hybrid Flow Shops (HFSs), sometimes called as a robotic cell with parallel machines, which differs from the classical FS that allows identical parallel machines. This technique increases the capacity of a single stage and enables parallel operations. The last extensibility of the FS is a number of used manipulation robots (single-robot resp., multiple-robot) or number of parts that can robot transfer at one time (single-gripper or dual-gripper robot). The FS robotic cells have not temporary storage between the machines. It causes that all parts must be either in the input buffer, on one of the machines, transfer by the robot, or in the output buffer. There are three types of pickup criteria — free pickup, no wait, and interval. In free pickup criteria, there is no limit for remove a part that has completed processing on a machine. Compared to that a part must be pickup immediately as soon as machine completes processing that part in no wait criteria. In interval robotic cells, each machine has defined time interval (also called time window) in which a completed processing part must be removed from that.

Batur et al. (2011) [9] developed Mixed Integer Linear Formulation (MILP) and heuristic algorithm based on Longest Processing Time (LPT) for model with multiple-part-type and two machines. Elmi and Topaloglu (2017) [10] considered the cyclic scheduling with multiple-part-type and two single-gripper robots. They developed Ant Colony Optimization (ACO) algorithm. Batur et al. (2016) [11] assume HFSs where is parallel machines, and some parts can skip some operations (machines). They model this as Traveling Salesman Problem (TSP) and formulate heuristic approach based on Simulated Annealing (SA). The great inspiration for our work is the article from Gultekin et al. (2018) [4]. They assume FS with multiple-part-type, m -machines, free pickup criteria, and single-gripper robot. They consider cycle scheduling of the parts and the robot move with the objective of maximalizing the throughput rate. They published a very detailed review of the related scheduling problems in robotic cells. Other older thorough reviews of the scheduling problems in robotic cells can be found in article by Dawande et al. (2005) [12] and in book by Dawande et al. (2007) [5].

1.2 Our research

An overview of the existing literature shows that there are many variants of the scheduling problem in FS robotic cells, but all these studies have the same objective: maximize the throughput or minimize the CT.

In this thesis, we consider a simpler variant of FS — single-part-type, m -machines, free pickup criteria, and single-gripper robot. The main contribution of our work is different objective. Our objective is to find optimal energy consumption schedule for

a fixed CT. We develop a MILP formulation of this problem and implement it in the Gurobi Optimization solver¹. We tested the MILP model by a different approach, but this approach can only be used for small robotic cells because it uses a brute force method. We also developed a heuristic approach based on a Genetic Algorithm (GA). We compare both the exact and the heuristic algorithm on generated benchmark instances that we provide publicly for others.

1.3 Thesis outline

The remainder of the thesis is organized as follows. The problem statement and the notation are presented in next Chapter 2. The mathematical programming formulations are explained in Chapter 3. Chapter 4 describes the method that verifies the correctness of the mathematical formulation of the problem. Chapter 5 presents the proposed heuristic algorithm. In Chapter 6 the generating of benchmarks and the set of tested instances are summarized. Chapter 7 represent the results of the thesis on the experiments. Chapter 8 concludes the thesis.

¹ For more information see the software web page: <http://www.gurobi.com>

Chapter 2

Problem Statement

In this section, we provide the formal definition of the problem. We use a classification scheme for sequencing, and scheduling problems in robotic cells described in a book and an article, written by Dawande et al. [5, 12]. See Figure 2.1 for better orientation in classification of the FS. The following text discusses the inclusion of our model under consideration in the classification scheme.

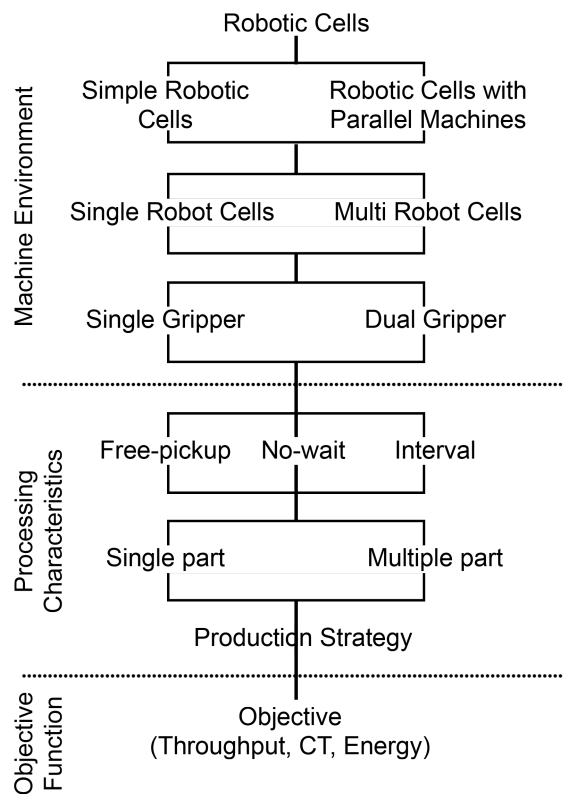


Figure 2.1. The classification scheme for robotic flow shops [12].

2.1 Machine Environment

In this section, we describe the first part of the classification schema that considers a number of machines, a number of robots and its types, and their layout in the cell.

2.1.1 Number of Machines per Stage

Each processing stage of our robotic cell has only one machine, and in the literature is this type called as a simple robotic cell. Because in simple robotic cells the number of processing stages is equal to the number of machines, we use the word **machine** with the same meaning as the word **stage**. We consider the robotic cell contains $m+2$ machines: M_0, M_1, \dots, M_{m+1} . Let $\mathcal{M} = 0, 1, \dots, m+1$ be the set of indices of these machines. The

robot takes a part from the input buffer (denoted as machine M_0), transports the part to the first processing machine (M_1), and loads the part. M_1 process the part and after that, the robot unloads the part from M_1 , then transports it to the next processing machine (M_2), on which it loads the part. The robot repeats this procedure on others processing machines (M_2, M_3, \dots, M_m). After the last processing machine (M_m) has finished its operations on the part, the robot unloads the part and gives it to the output buffer (denoted as M_{m+1}). Each machine provides some operation (drilling, cutting, welding, grinding, painting, ...) with the part. Each part to be processed goes through each machine $M_i, i \in \mathcal{M}$ in the same sequence, and cannot skip any machines.

Be careful that the previous paragraph is not misunderstood. It does not specify that the robot must wait nearby the machine until the machine has completed its processing on the part. The robot can move to another processing machine or to the input buffer to transport another part to its next place. In other words, a robot move sequence (RMS) may not be the same as the sequence of the part processing on the machines.

2.1.2 Number and Type of Robots

We assume the robotic cell that contains one robot only (called *single-robot* cell). This robot has one gripper (*single-gripper* robot), and it can hold only one part at a time. A single-gripper robot has one disadvantage versus dual-gripper robot. In a single-gripper simple robotic cell, the robot cannot unload a part from machine $M_i, i \in 0, \dots, m-1$, unless the next machine M_{i+1} is ready to process it. This condition is commonly referred to as a blocking condition (see chapter 2.1.3 in [5]).

2.1.3 Cell Layout

In our model, we consider both cell layout: *linear* and *circular*. The linear cell model (Figure 1.1) allows more processing machines while circular cell model has the robot in the center and machines are around. The second variant is more transparent for drawing pictures, so we draw the pictures for circular cells without detracting from generality.

2.2 Processing Characteristics

In this section, we describe robot operations first, then the considered pickup criterion, and a number of part-types. In the end, we write about used processing strategy.

2.2.1 Robot Operations

The robot can do two types of movements: *loaded* and *unloaded* or it can be in the *idle state*. At any point time, the robot must be moving, or be in the idle state and simultaneously meets two constraints:

- No two identical types of operations (loaded, unloaded and idle state) can not follow immediately.
- Between two unloaded moves, or two idle states must be one loaded move.

Loaded move

We define the set of robot activities A_0, A_1, \dots, A_m , see Figure 2.2. Let $\mathcal{A} = 0, 1, \dots, m$ be the set of indices of these activities. Each activity A_i contains several steps, $i \in \mathcal{A}$. The robot unloads a part from the machine M_i , transfers it from machine M_i to the next processing machine M_{i+1} , and loads the part to machine M_{i+1} , $i \in \mathcal{A}$. Each loading and unloading time is assumed to be zero. The loaded robot move is characterized by

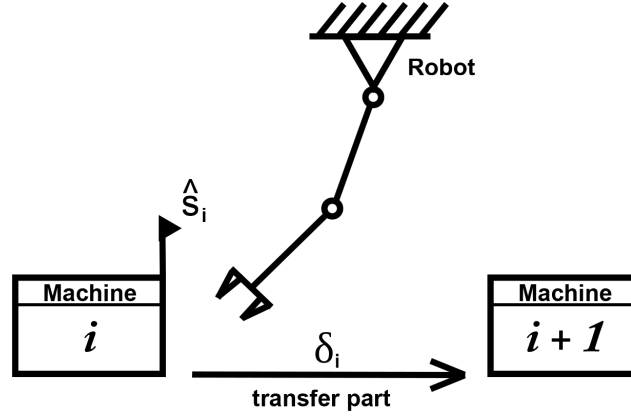


Figure 2.2. Definition of the robot activity A_i , $i \in \mathcal{A}$.

transfer time duration denoted by δ_i , and it means the duration that is needed for the activity A_i , $i \in \mathcal{A}$. Each loaded move duration δ_i has defined its lower bound duration $\underline{\delta}_i$ that meets physical constraints of this robot move, $i \in \mathcal{A}$.

Unloaded move

The unloaded robot move is the second type of the movements. Let $\theta_{i+1,j}$, where $i, j \in \mathcal{A}$, is the duration of the unloaded robot move from machine M_{i+1} after its loaded to machine M_j for its unloaded, see Figure 2.3. Each unloaded move duration $\theta_{i+1,j}$ has defined its lower bound duration $\underline{\theta}_{i+1,j}$ that meets physical constraints of this robot move. Some unloaded robot moves defined above are not acceptable. In these situations, the lower bound duration $\underline{\theta}_{i+1,j}$ is set to zero. It is in following two combinations of indices i, j :

- $i + 1 = j$; move from the one place to the same place, $i, j \in \mathcal{A}$.
- $i = j$; this unloaded move cause that the next loaded movement does not respect the blocking condition, $i, j \in \mathcal{A}$.

Robot idle state

The robot can wait (to be in the idle state) in three situations. The first wait can occur when the robot loads the machine with a part and waits unless the machine completed its processing on the part. Other two waits can occur before respectively after the unloaded move. We define $w_{i+1,j}$, where $i, j \in \mathcal{A}$, that is equal to the duration of the robot waiting between machines M_{i+1} and M_j , see Figure 2.3. During this waiting, the robot is in the idle state between two activities: A_{i+1} and A_j . Some robot waits in the idle state defined above are not acceptable as in the cases of unloaded moves. In these situations, the $w_{i+1,j}$ is set to zero. It is in following combinations of indices i, j :

- $i = j$; this wait means that there is also the unloaded move with those indices combination. This unloaded move causes that the next loaded movement does not accept the blocking condition.

2.2.2 Pickup Criterion

From the three types of pickup criterion, we selected free-pickup criterion. A part which has complete a process on a machine can stay on this machine for an infinite amount of time until it has been transported to the next place. This type of criterion is not suitable for parts that need limited time operations such as a galvanic plating, etc.

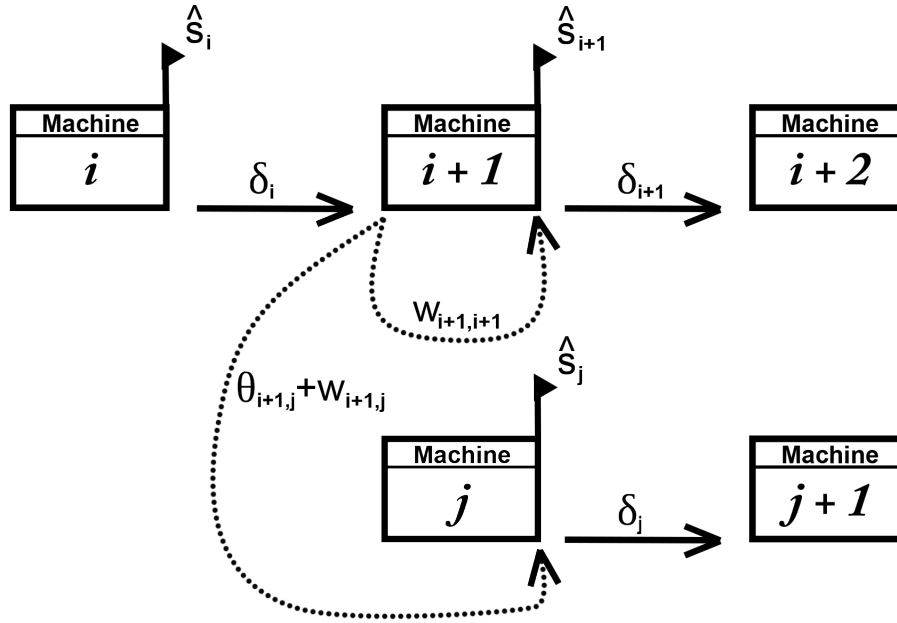


Figure 2.3. Definition of the possible robot moves and waits of the robot in the idle state - Solid arrows present loaded robot movements while dot arrows present a combination of unloaded robot movements with durations of the robot idle state.

2.2.3 Number of Part-Types

We consider that our robotic cell is producing identical parts (called *single-part-type*). We define the processing time p_i , $i \in \mathcal{A}$. It means duration of the operation on machine M_i and not how long the part occupied this machine. The processing time is equal to zero for the input buffer (machine M_0), $p_0 = 0$.

2.2.4 Processing Strategy

We assume a cyclic scheduling using an one-unit cycle. Brauner et al. (2003) [8] describe the one-unit cycle as a permutation of $m + 1$ activities starting with activity A_0 . This last assumption is made without loss of generality. In each repetition of the one-cycle one part leaves the cell, and one enters it.

2.3 Objective Function

Our optimization problem uses energy consumption as its objective. The problem addresses in this thesis is to minimize energy consumption for a fixed CT value. It requires a definition of energy functions for individual movements and waits in the robot idle state.

Let E_i^δ denotes energy function for the loaded move of the robot, $E_{i+1,j}^\theta$ denotes energy function for the unloaded move from machine M_{i+1} to machine M_j , and $E_{i+1,j}^w$ denotes energy function for waiting robot in the idle state. Vergnano et al. (2012) [13] describe a method how to model and parametrize energy consumption for each robot operation as a function of an execution time. Mohammeda et al. (2014) [14] apply this approach for a six-axis manipulation robot. We use definitions, which are described in the previous literature [13–14]. The loaded and unloaded robot movements are convex functions by the description in the literature, and their approximations can be described

by equations:

$$E_i^\delta(\delta_i) = \sum_{c=-3}^1 a_c \delta_i^c, \quad \forall i \in \mathcal{A}; \quad (2.1)$$

$$E_{i+1,j}^\theta(\theta_{i+1,j}) = \sum_{c=-3}^1 a_c \theta_{i+1,j}^c, \quad \forall i, j \in \mathcal{A}. \quad (2.2)$$

Figure 2.4 sees such two functions which demonstrate the robot operation energy consumption as the function of its execution time.

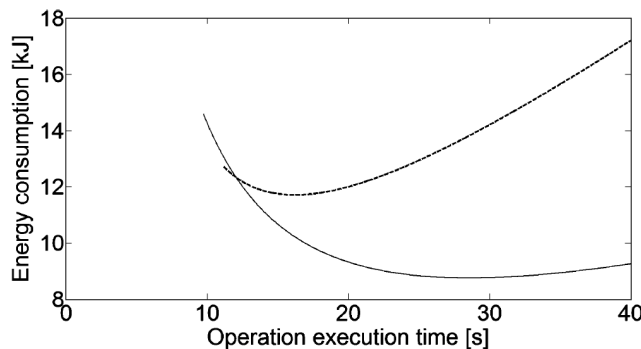


Figure 2.4. Robot operation energy consumption as a function of its execution time for two operations, taken from [13].

Let $E_{i+1,j}^w$ denotes a linear energy function for waiting robot in the idle between machines M_{i+1} and M_j . When the robot is in the idle state, its arms are not moving and stay in the same position. In this situation, motors consume only energy which is needed to compensate gravity force, but not consumes energy needed for accelerations. Thanks to this, we can consider that the dependence of consumed energy on time is a linear function. The linear coefficient of the function depends only on the configuration of robotic arms. See Figure 2.5 which demonstrates different energy-intensive robot configurations in the robot idle state. In the described robotic cell, we assume two possible places for each wait duration $w_{i+1,j}$ where the robot can be in the idle state. For the wait duration $w_{i+1,j}$, the first place is by machine M_{i+1} after its loading, and the second place is by machine M_j before its unloading. We prefer the place where the robot configuration consumes less value of energy. The following equation describes the function of the preferred place for each possible wait duration $w_{i+1,j}$:

$$E_{i+1,j}^w(w_{i+1,j}) = a_{i+1,j} w_{i+1,j}, \quad \forall i, j \in \mathcal{A}; \quad a_{i+1,j} \in \mathbb{R}_0^+; \quad (2.3)$$

where $a_{i+1,j}$ is the linear coefficient of this function.

The outcome of the problem described above is the Robot Activities Sequence (RAS) that the robot cyclically repeats for fixed the CT value so that it consumes the smallest amount of energy.

2.4 Example

The following example can help to understand the problem and operations of the robotic cell.

Example 2.1. In the example, we consider a single-gripper simple robotic cell with five machines M_0, M_1, M_2, M_3, M_4 , see Figure 2.6. Machine M_0 is an input buffer

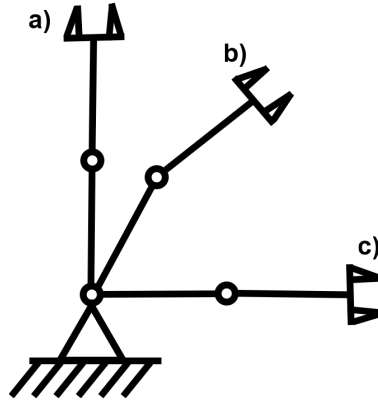


Figure 2.5. Demonstration of different energy-intensive robot configurations in the robot idle state. The configuration a) consume the least amount of energy. The second extrem is the robot configuration c) which consumes the most amount of energy. Something between the robot configurations a) and c) is the variant b).

which contains an infinite number of unprocessed parts, and these parts are immediately available. Machines M_1 , M_2 and M_3 are processing machines. Machine M_4 is an output buffer which stores entirely processed parts. For simplicity reason, we assume that all energy functions for loaded and unloaded robot movements in this example have the same formula:

$$E_i^\delta = 5000\delta_i^{-3} + 200000\delta_i^{-2} + 1000\delta_i^{-3} + 4000 + 650\delta_i, \quad i \in 0, 1, \dots, 3, \quad (2.4)$$

$$E_{i+1,j}^\theta = 5000\theta_{i+1,j}^{-3} + 200000\theta_{i+1,j}^{-2} + 1000\theta_{i+1,j}^{-3} + 4000 + 650\theta_{i+1,j}, \quad i, j \in 0, 1, \dots, 3. \quad (2.5)$$

The minimal value of the previous energy function is around 8.5 s, see Figure 2.7.

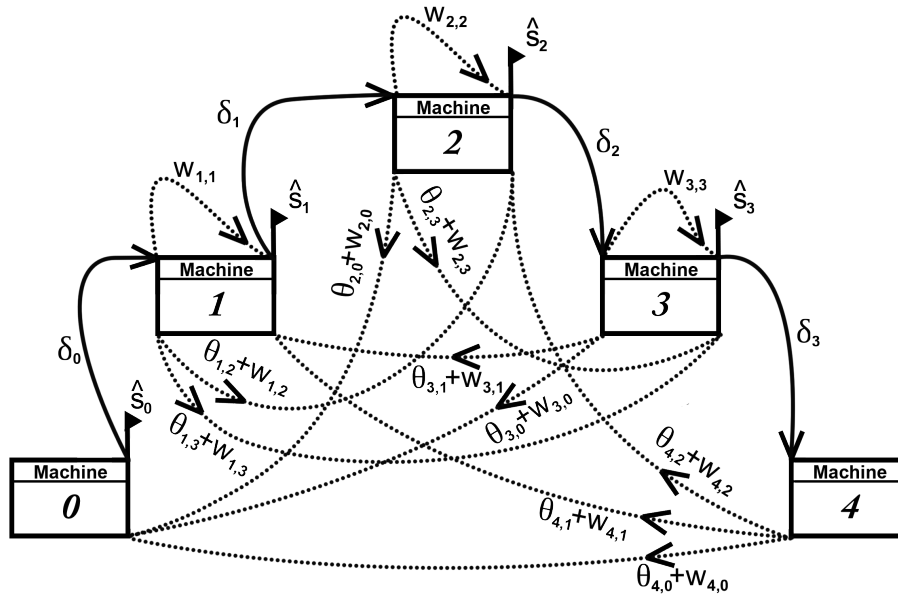


Figure 2.6. Model of the robotic cell that is considered in Example 1 with all possible moves and durations of waits.

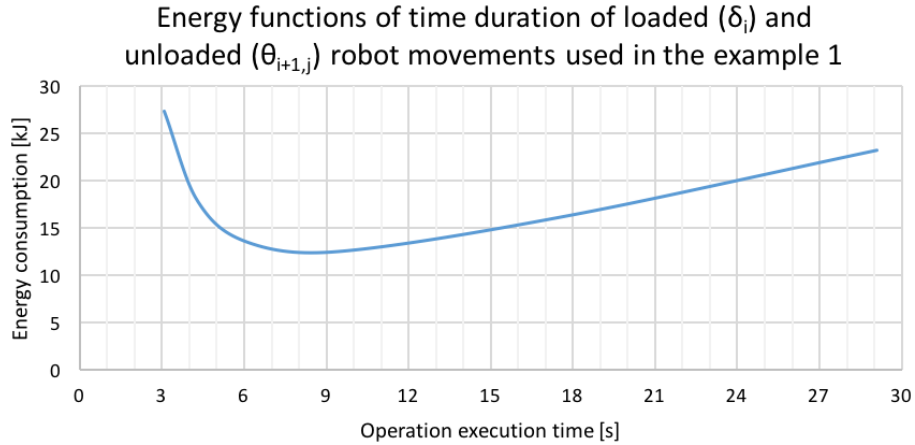


Figure 2.7. The graph corresponding to the equation (2.4) and (2.5). The minimum function value is around 8,5s.

The Table 2.1 contains all other needed parameters for the description of the robotic cell:

- Processing times,
- Lower bounds for loaded robot movements,
- Lower bounds for unloaded robot movements,
- Coefficients a of linear functions for robot idle states. There are available two coefficients for each machine because the robot can wait after loading the machine or before unloading the machine.

The cycle time is given as $CT = 70$.

<i>Processing times:</i>					
Machine	M_0	M_1	M_2	M_3	
Processing times	0	25	5	40	
<i>Lower bounds δ_i:</i>					
Activity	A_0	A_1	A_2	A_3	
δ_i	4	3	1	6	
<i>Lower bounds $\theta_{i+1,j}$:</i>					
Machine [\downarrow from/ \rightarrow to]	M_0	M_1	M_2	M_3	
M_1	0	0	2	3	
M_2	5	0	0	2	
M_3	3	7	0	0	
M_4	2	3	5	0	
<i>Coefficients a of linear functions:</i>					
Machine	M_0	M_1	M_2	M_3	M_4
after loading	—	60	40	100	80
before unloading	20	120	160	80	—

Table 2.1. All other needed parameters for description of the robotic cell in Example 1.

The MILP model, which is developed in the next Chapter 3, found the optimal solution, see Figure 2.8. The optimal RAS $\sigma_{RAS} = \{0, 3, 1, 2\}$ together with the starting times of each activities are provided in Table 2.2. According to that, we use the one-unit cycle processing strategy, in each repetition of the σ_{RAS} one part leaves the cell, and one enters into it. Figure 2.9 shows the corresponding Gantt chart. We can see that part with label 1 enters into the cell, and the part with label 0 leaves the cell during one-unit cycle processing strategy. Next, we can see that the example contains all three possible variants where the robot can wait in the idle state.

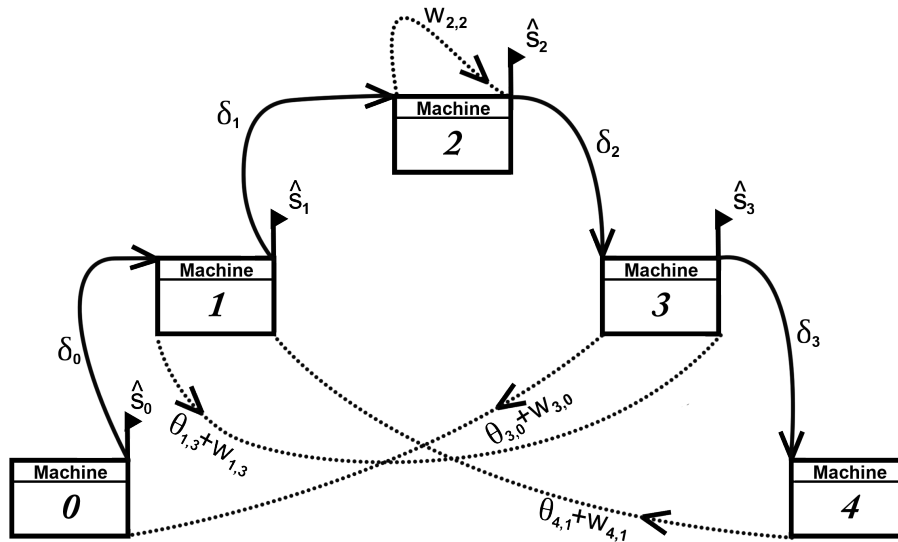


Figure 2.8. A solution to the problem from Example 1.

Activity	A_0	A_3	A_1	A_2
\hat{s}_i	0	21.3	33.9	45.2

Table 2.2. The optimal RAS $\sigma_{RAS} = \{0, 2, 1, 3\}$ and start times of activities from Example 1.

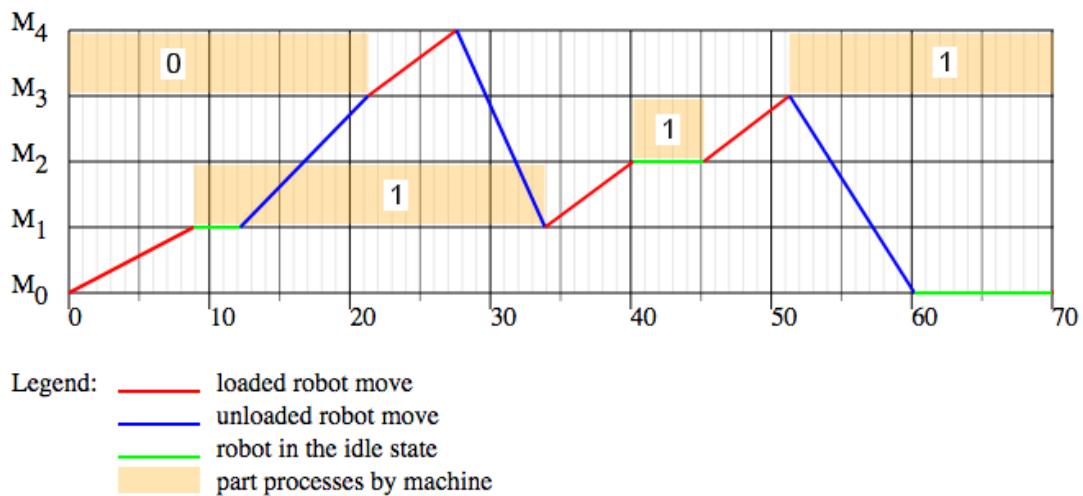


Figure 2.9. Gantt chart of the solution from Example 1.

This chapter has formally describe the problem addressed in this thesis. In the end, we give the example of the single-gripper simple robotic cell with five machines. This example should help to understand the model parameters and possible movements of the manipulation robot. We used the same energy-consuming function for all movements to simplified the example compared to a real model. In the next section, we formulate a solution to the addressed problem using a MILP.

Chapter 3

Mixed Integer Linear Programming Formulation

In this chapter, we describe a solution to the problem that we explained in the previous section using Mixed Integer Linear Programming (MILP) formulation. First, we define sets, constant parameters, and decision variables. Then, we formulate model constraints. A summary of the whole model is given at the end of the chapter.

3.1 Sets

We define two following sets in the MILP formulation:

- $\mathcal{M} = \{0, 1, \dots, m + 1\}$: The set of machine indices, $(M_0, M_1, \dots, M_{m+1})$. M_0 is the input buffer and M_{m+1} is the output buffer;
- $\mathcal{A} = \{0, 1, \dots, m\}$: The set of activity indices, (A_0, A_1, \dots, A_m) .

3.2 Constant Parameters

In this section, we describe parameters which are fixed during the optimization. We divide those parameters into three categories.

In the first category are parameters which give physical restraints for robot movements:

- $\underline{\delta}_i$: The lower bound for the loaded robot move duration δ_i , $\underline{\delta}_i \in \mathbb{R}_0^+$; $i \in \mathcal{A}$;
- $\underline{\theta}_{i,j}$: The lower bound for the unloaded robot move duration $\theta_{i,j}$, $\underline{\theta}_{i,j} \in \mathbb{R}_0^+$; $i, j \in \mathcal{A}$.

The second category contains parameters which are needed for energy functions description. Energy consuming functions defined by Equations (2.1) and (2.2) in the previous Chapter 2 are convex functions. However, we need a linear representation of these functions in the MILP formulation. Therefore, we sample these convex functions with a uniform distribution to create piecewise-linear functions, see Figure 3.1. A number of the linear function used for the approximation is defined by a parameter k , $k \in \mathbb{N}$. The k value is set according to the CT value. The increasing k value gives a better approximation of convex functions. However, more time is needed to compute a solution. Now, we can define parameters which contain an amount of energy for each robot movements and each segment of the piecewise-linear functions:

- $e_{i,k}^\delta$: Energy which consumes the robot during its loaded movement duration δ_i if we assume k -th linear approximation of the convex function, $e_{i,k}^\delta \in \mathbb{R}$; $i \in \mathcal{A}$; $k \in \mathbb{N}$;

$$e_{i,k}^\delta(\delta_i) = a_k \delta_i + b_k, \quad \forall i \in \mathcal{A}; \quad a_k, b_k \in \mathbb{R}; \quad (3.1)$$

- $e_{i+1,j,k}^\theta$: Energy which consumes the robot during its unloaded movement duration $\theta_{i+1,j}$ if we assume k -th linear approximation of the convex function, $e_{i+1,j,k}^\theta \in \mathbb{R}$; $i, j \in \mathcal{A}$; $k \in \mathbb{N}$;

$$e_{i+1,j,k}^\theta(\theta_{i+1,j}) = a_k \theta_{i+1,j} + b_k, \quad \forall i, j \in \mathcal{A}; \quad a_k, b_k \in \mathbb{R}; \quad (3.2)$$

- $e_{i,l}^w$: Energy which consumes the robot during its waiting in the idle state which is localized nearby machine M_i . The indice l specifies where the robot waits. If $l = 0$ the robot waits in front of the machine M_i after the robot loaded it or if $l = 1$ the robot waits behind the machine M_i to unload it, $e_{i,l}^w \in \mathbb{R}_0^+$; $i \in \mathcal{M}$; $l \in \{0, 1\}$;

$$e_{i,0}^w(w_{i,x}) = a_{k_0} w_{i,x}, \quad \forall i, x \in \mathcal{M}; \quad a_k \in \mathbb{R}; \quad (3.3)$$

$$e_{i,1}^w(w_{x,i}) = a_{k_1} w_{x,i}, \quad \forall i, x \in \mathcal{M}; \quad a_k \in \mathbb{R}. \quad (3.4)$$

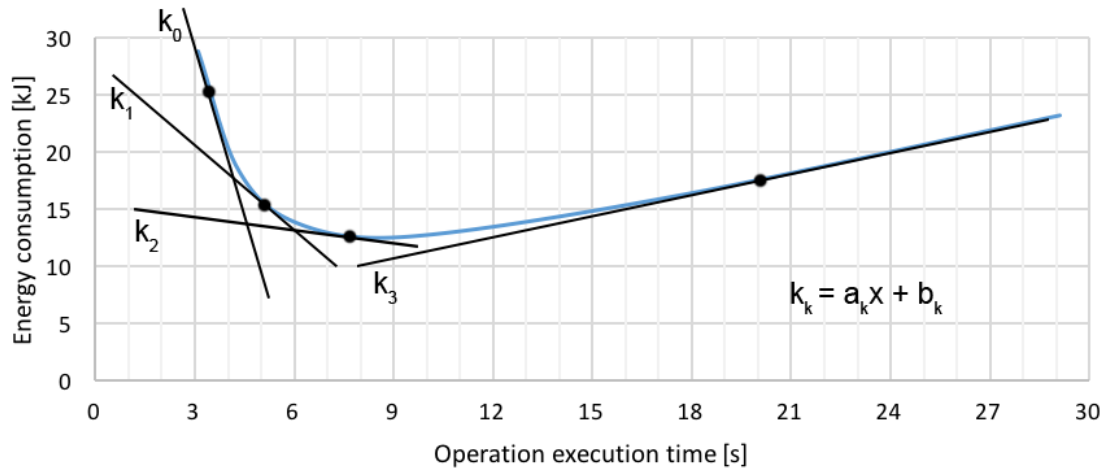


Figure 3.1. A principle how we convert convex energy functions to piecewise-linear functions.

Now, we can describe the third category which contains other constant parameters:

- p_i : The processing time of parts on machines M_i , $p_i \in \mathbb{R}^+$; $i \in \mathcal{A} \setminus \{0\}$;
- CT : The cycle time, $CT \in \mathbb{R}^+$. Duration of the RAS is equal to the CT value;
- B : A very large number.

3.3 Decision Variables

Now we describe decision variables describing the model. Those variables can be changed during the optimization.

Variables describing the time dependencies of activities:

- s_i : The start time of the activity A_i , $i \in \mathcal{A}$, which transfers the first part from the first one-unit cycle, $s_i \in \mathbb{R}_0^+$;
- \hat{s}_i : The start time of the activity A_i $i \in \mathcal{A}$ at the first RAS, $\hat{s}_i \in [0, CT)$;
- q_i : An integer variable which shifts the activity start time s_i from a n -execution of RAS to the first execution of RAS about $n - 1$ times CT . The value of q_i variable

is count to $n - 1$, $q_i \in \mathbb{N}_0$. It is because, the previous three variables are bound by equation $s_i = \widehat{s}_i + CT \cdot q_i$, $\forall i \in \mathcal{A}$.

Variables needed for robot movements:

- δ_i : Traveling time of the loaded robot between adjacent machines M_i and M_{i+1} , $\delta_i \in \mathbb{R}_0^+$; $i \in \mathcal{A}$;
- $\theta_{i,j}$: Traveling time of the unloaded robot between machines M_i and M_j , $\theta_{i,j} \in \mathbb{R}_0^+$; $i, j \in \mathcal{A}$;
- $w_{i,j}$: Waiting time of the robot between machines M_i and M_j , $w_{i,j} \in \mathbb{R}_0^+$; $i, j \in \mathcal{A}$.

Variables which contain the amount of energy for each robot movement:

- \underline{E}_i^δ : Energy which consumes the robot during its loaded movement expressed as the maximum of piecewise-linear functions of E_i^δ , $\underline{E}_i^\delta \in \mathbb{R}_0^+$; $i \in \mathcal{A}$;

$$\underline{E}_i^\delta(\delta_i) = \max(e_{i,k}^\delta(\delta_i)), \forall i \in \mathcal{A}; k \in \mathbb{N}; \quad (3.5)$$

- $\underline{E}_{i+1,j}^\theta$: Energy which consumes the robot during its valid unloaded movement expressed as the maximum of piecewise-linear functions of $E_{i+1,j}^\theta$, $\underline{E}_{i+1,j}^\theta \in \mathbb{R}_0^+$; $i, j \in \mathcal{A}$;

$$\underline{E}_{i+1,j}^\theta(\theta_{i+1,j}) = \max(e_{i+1,j,k}^\theta(\theta_{i+1,j})), \forall i, j \in \mathcal{A}; k \in \mathbb{N}; \quad (3.6)$$

- $\underline{E}_{i,j}^w$: Energy which consumes the robot during its waiting in the idle, $\underline{E}_{i,j}^w \in \mathbb{R}_0^+$; $i, j \in \mathcal{M}$;

$$\underline{E}_{i,j}^w(w_{i,j}) = \min(e_{i,0}^w, e_{j,1}^w), \forall i, j \in \mathcal{M}. \quad (3.7)$$

The last type of variables are the following binary decision variables:

$$x_{i,j} = \begin{cases} 1 & \text{if } A_i \rightarrow A_j, \forall i, j \in \mathcal{A}; \\ 0 & \text{otherwise} \end{cases}$$

$$y_{i,j} = \begin{cases} 1 & \text{if } A_i \rightarrow A_j \text{ immediately, } \forall i, j \in \mathcal{A}; \\ 0 & \text{otherwise} \end{cases}$$

- $x_{i,j}$: $x_{i,j}$ is equal to 1 if activity A_i occurs before activity A_j or $x_{i,j}$ is equal to 0 in the opposite case;
- $y_{i,j}$: $y_{i,j}$ is equal to 1 if activity A_i occurs immediately before activity A_j or $y_{i,j}$ is equal to 0 in the opposite case.

3.4 Constraints

In this section, we describe all constraints which define the mathematical model of our robotic cell. All variables and parameters, which we use, are explained in the previous paragraphs of this chapter.

The following constraints define the order of robot activities in the RAS:

$$x_{0,j} = 1, \forall j \in \mathcal{A} : j \neq 0; \quad (3.8)$$

$$x_{i,i} = 0, \forall i \in \mathcal{A}; \quad (3.9)$$

$$x_{i,j} = 1 - x_{j,i}, \forall i, j \in \mathcal{A} : i < j; \quad (3.10)$$

$$y_{i,j} \leq x_{i,j}, \forall i, j \in \mathcal{A} : j \neq 0; \quad (3.11)$$

$$\sum_{\forall i \in \mathcal{A} : i \neq j} y_{i,j} = 1, \forall j \in \mathcal{A}; \quad (3.12)$$

$$\sum_{\forall j \in \mathcal{A} : i \neq j} y_{i,j} = 1, \forall i \in \mathcal{A}; \quad (3.13)$$

$$y_{0,0} = 0. \quad (3.14)$$

The constrain (3.8) ensures that activity A_0 is the first activity of the RAS. The constrain (3.9) defines the value of $x_{i,j}$ when $i = j$. If activity A_i is before activity A_j in the RAS, then the A_j must occur after the A_i . This sentence imposes constrain (3.10). Next constrain (3.11) ensures that if activity A_i occurs immediately before activity A_j , then the A_i must be before the A_j in the RAS. The previous sentence is correct except the last activity in the RAS. This last activity is before activity A_0 from the next cycle of the RAS. The described issue is solved by the constraint (3.11) which is defined for $\forall j \in \mathcal{A}$ except $j \neq 0$. The constraint (3.12) respectively constraint (3.13) ensures that each activity A_j has only one predecessor, respectively each activity A_i has only one successor. The previous equations do not specify the state of decision variable $y_{0,0}$, so the constraint (3.14) defines its state.

The following constraints define the start time s_i of activity A_i which transfers the first part from the first one-unit cycle:

$$s_i + \delta_i + p_{i+1} \leq s_{i+1}, \forall i \in \mathcal{A} : i \neq m; \quad (3.15)$$

$$s_{i+1} + \delta_{i+1} - CT \leq s_i, \forall i \in \mathcal{A} : i \neq m. \quad (3.16)$$

The constraint (3.15) say that between two start times of activities A_i and A_{i+1} , $i \in \mathcal{A} : i \neq m$, which transfer the same part must be enough amount of time for the loaded robot move δ_i plus the processing time of a part p_{i+1} on machine M_{i+1} . However, the maximum amount of that time is limited by the constraint (3.16). For a better understanding, see Figure 3.2 that illustrates these two equations.

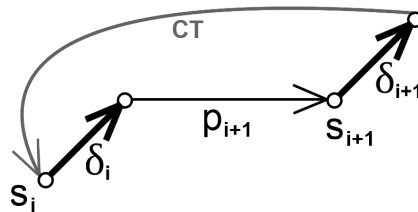


Figure 3.2. The relationship of two immediately following activities which transfer the same part.

The previous constraints (3.15) and (3.16) describe dependencies of activities which transfer the same part. However, activities can transfer several parts in the one-unit cycle of the RAS. Constraint (3.17) describes the relationship between activity A_i which transfers the first part from the first one-unit cycle (start time s_i) and activity A_i from the RAS (start time \hat{s}_i), $i \in \mathcal{A}$. We commented this constraint in Section 3.3 when we

defined the used variables. We can describe the constraint (3.17) by other words: \widehat{s}_i is the remainder after the division of s_i by CT and q_i is the integer part of this division:

$$s_i = \widehat{s}_i + CT \cdot q_i, \forall i \in \mathcal{A}. \quad (3.17)$$

The constraint (3.18) defines the start time of the schedule:

$$s_0 = 0. \quad (3.18)$$

Now we describe constraints that specified the order of the activities in the RAS. Similar problem solves Sucha et al. (2004) [15] in Section 5.2. We used this approach and transformed it into our problem. A robot activity schedule repeats a one-unit cycle of the RAS infinitely. Each RAS, which is neither in the head nor the tail of the schedule, contains all activities even if they can transfer parts from different one-unit cycles. See Figure 3.3, which contains a piece of the final schedule (three one-unit cycles) for a robotic cell with three activities. In the u -th occurrence of the one-unit cycle, the activity A_i and A_j start at time s'_i and s'_j respectively and meet Equation (3.17): $s'_i = \widehat{s}_i + CT \cdot u$ and $s'_j = \widehat{s}_j + CT \cdot u$. We derive constraints based on \widehat{s} because those constraints do not depend on q from Equation (3.17). Based on the order of activities in the RAS, two disjoint cases can occur (lines a) and b) in Figure 3.3):

	The (u - 1)-th one-unit cycle			The u-th one-unit cycle			The (u + 1)-th one-unit cycle		
	A_0	A_i	A_j	A_0	A_i	A_j	A_0	A_i	A_j
					s'_i	s'_j			
a) A_0, A_1, A_2 $i < j$	u-1	u-1	u-1	u	u	u	u+1	u+1	u+1
b) A_0, A_2, A_1 $i > j$	u-1	u-2	u-1	u	u-1	u	u+1	u	u+1

Figure 3.3. The piece of schedule on which we derive the relationship of two arbitrary activities in the RAS.

In the first case, we consider activity A_i to be followed by activity A_j and both transfer the u -th part in the same cycle, i.e. the situation in Figure 3.3 a), therefore

$$\widehat{s}_j - \widehat{s}_i \geq \delta_i + \theta_{i+1,j} + w_{i+1,j}. \quad (3.19)$$

At the same time, the previous occurrence of A_j , which transfer the $(u - 1)$ -th part, is followed by the A_i , which transfer the u -th part, therefore

$$\widehat{s}_i - (\widehat{s}_j - CT) \geq \delta_j + \theta_{j+1,i} + w_{j+1,i}. \quad (3.20)$$

The conjunction into one double-inequality is:

$$\delta_j + \theta_{j+1,i} + w_{j+1,i} \leq \widehat{s}_i - \widehat{s}_j + CT \leq CT - (\delta_i + \theta_{i+1,j} + w_{i+1,j}). \quad (3.21)$$

In the second case, we consider activity A_j to be followed by activity A_i but they transfer the u -th part in different cycles, i.e. the situation in Figure 3.3 b). To derive constraint for the second case, it is enough to exchange index i with index j in the double-inequality:

$$\delta_i + \theta_{i+1,j} + w_{i+1,j} \leq \widehat{s}_j - \widehat{s}_i + CT \leq CT - (\delta_j + \theta_{j+1,i} + w_{j+1,i});$$

$$\begin{aligned} \delta_i + \theta_{i+1,j} + w_{i+1,j} - CT &\leq \widehat{s}_j - \widehat{s}_i \leq -(\delta_j + \theta_{j+1,i} + w_{j+1,i}); \\ \delta_j + \theta_{j+1,i} + w_{j+1,i} &\leq \widehat{s}_i - \widehat{s}_j \leq CT - (\delta_i + \theta_{i+1,j} + w_{i+1,j}). \end{aligned} \quad (3.22)$$

The first case, Equation (3.21), differs from the second case, i.e. Equation (3.22), only in CT in the middle of the double-inequality. This CT indicates whether activity A_i is before activity A_j within the same one-unit cycle or not. According to that, Equations (3.21) and (3.22) can be reduced into one double-inequality, while using decision variable $x_{i,j}$ ($x_{i,j} = 1$ when activity A_i is followed by activity A_j and $x_{i,j} = 0$ when activity A_j is followed by activity A_i):

$$\delta_j + \theta_{j+1,i} + w_{j+1,i} \leq \widehat{s}_i - \widehat{s}_j + CT \cdot x_{i,j} \leq CT - (\delta_i + \theta_{i+1,j} + w_{i+1,j}), \quad \forall i, j \in \mathcal{A} : i < j. \quad (3.23)$$

The next constraint (3.24) specified that sum of all possible loaded and unloaded durations and durations of all possible wait the robot in idle states is equal to the CT value. A combination of indices $i = j$ ignore unloaded movements from machine to the same machine and the blocking condition [5] is met by ignoring indices where $i = j + 1$.

$$\sum_{i \in \mathcal{A}} \delta_i + \sum_{i \in \mathcal{A}: i \neq 0, j \in \mathcal{A}: i \neq j+1} w_{i,j} + \sum_{i \in \mathcal{A}: i \neq 0, j \in \mathcal{A}: i \neq j \wedge i \neq j+1} \theta_{i,j} = CT. \quad (3.24)$$

The following set of constraints describe lower and upper bounds for durations of the loaded and the unloaded robot movements and idle states durations:

$$\delta_i \geq \underline{\delta}_i, \quad \forall i \in \mathcal{A}; \quad (3.25)$$

$$\theta_{i+1,j} \geq y_{i,j} \cdot \underline{\theta}_{i+1,j}, \quad \forall i, j \in \mathcal{A}; \quad (3.26)$$

$$\theta_{i+1,j} \leq 0, \quad \forall i, j \in \mathcal{A} : i + 1 = j; \quad (3.27)$$

$$\theta_{i+1,j} \leq y_{i,j} \cdot CT, \quad \forall i, j \in \mathcal{A} : i + 1 \neq j; \quad (3.28)$$

$$w_{i+1,i+1} \geq y_{i,i+1} \cdot p_{i+1}, \quad \forall i \in \mathcal{A}; \quad (3.29)$$

$$w_{i+1,j} \leq y_{i,j} \cdot CT, \quad \forall i, j \in \mathcal{A}. \quad (3.30)$$

The constraint (3.25) applies lower bound for loaded robot move durations δ_i . The constraint (3.26) using decision variable $y_{i,j}$ sets the lower bound for unloaded robot move durations $\theta_{i+1,j}$. This lower bound is set to $\underline{\theta}_{i+1,j}$ if $y_{i,j} = 1$ or is pushed to zero if $y_{i,j} = 0$. The upper bound for durations θ are divided into two inequalities. The first constraint (3.27) pushes durations of prohibited movements to zero (unloaded move from a machine to the same machine). The second constraint (3.28) sets the upper bound of $\theta_{i+1,j}$ to CT if $y_{i,j} = 1$ or is pushed to zero if $y_{i,j} = 0$. The constraint (3.29) set the lower bound for idle state durations $w_{i+1,i+1}$ when the robot waits by the machine M_{i+1} until that machine finished the process on the part. The constraint (3.30) sets the upper bound for idle state durations $w_{i+1,j}$ to CT if $y_{i,j} = 1$ or is pushed to zero if $y_{i,j} = 0$.

The last two constraints (3.31) and (3.32) ensure the correct value of consuming energy for loaded and unloaded robot movement durations. Those constraints select the maximum of piecewise-linearized convex functions for fixed durations δ_i respectively $\theta_{i+1,j}$ which is assumed in the schedule. The problem is to ensure that unused unloaded robot moves do not influence the objective. In the linear programming, we can not use multiplication of two variables (multiply energy variable by $y_{i,j}$ variable), so we use a trick with the subtraction of a large number, i.e. the part $(1 - y_{i-1,j}) \cdot B$ in constraint (3.32).

$$\underline{E}_i^\delta(\delta_i) \geq e_{i,k}^\delta(\delta_i), \quad \forall i \in \mathcal{A}; \quad k \in \mathbb{N}; \quad (3.31)$$

$$\underline{E}_{i+1,j}^\theta(\theta_{i+1,j}) \geq e_{i+1,j,k}^\theta(\theta_{i+1,j}) - (1 - y_{i-1,j}) \cdot B, \quad \forall i \in \mathcal{M}; \quad j \in \mathcal{A}; \quad k \in \mathbb{N} : i \neq 0. \quad (3.32)$$

3.5 Objective Function

The goal of the optimization is to find a feasible schedule having the given CT that minimize the overall energy consumption. We assume the following objective function in our model of the robotic cell:

$$\min \left(\sum_{i \in \mathcal{A}} \underline{E}_i^\delta(\delta_i) + \sum_{i \in \mathcal{M}, j \in \mathcal{A}: j \neq 0 \wedge i \neq j+1 \wedge i \neq j} \underline{E}_i^\theta(\theta_{i,j}) + \sum_{i \in \mathcal{M}, j \in \mathcal{A}} \underline{E}_i^w(w_{i,j}) \right). \quad (3.33)$$

3.6 Model Summary

Minimize

$$\sum_{i \in \mathcal{A}} \underline{E}_i^\delta(\delta_i) + \sum_{i \in \mathcal{M}, j \in \mathcal{A}: j \neq 0 \wedge i \neq j+1 \wedge i \neq j} \underline{E}_i^\theta(\theta_{i,j}) + \sum_{i \in \mathcal{M}, j \in \mathcal{A}} \underline{E}_i^w(w_{i,j})$$

subject to

$$\begin{aligned} x_{0,j} &= 1, & \forall j \in \mathcal{A} : j \neq 0 \\ x_{i,i} &= 0, & \forall i \in \mathcal{A} \\ x_{i,j} &= 1 - x_{j,i}, & \forall i, j \in \mathcal{A} : i < j \\ y_{i,j} &\leq x_{i,j}, & \forall i, j \in \mathcal{A} : j \neq 0 \\ \sum_{\forall i \in \mathcal{A} : i \neq j} y_{i,j} &= 1, & \forall j \in \mathcal{A} \\ \sum_{\forall j \in \mathcal{A} : i \neq j} y_{i,j} &= 1, & \forall i \in \mathcal{A} \\ y_{0,0} &= 0, \\ s_i + \delta_i + p_{i+1} &\leq s_{i+1}, & \forall i \in \mathcal{A} : i \neq m \\ s_{i+1} + \delta_{i+1} - CT &\leq s_i, & \forall i \in \mathcal{A} : i \neq m \\ s_i &= \widehat{s}_i + CT \cdot q_i, & \forall i \in \mathcal{A} \\ s_0 &= 0, \\ \delta_j + \theta_{j+1,i} + w_{j+1,i} &\leq \widehat{s}_i - \widehat{s}_j + CT \cdot x_{i,j} \leq CT - (\delta_i + \theta_{i+1,j} + w_{i+1,j}), & \forall i, j \in \mathcal{A} : i < j \\ \sum_{i \in \mathcal{A}} \delta_i + \sum_{i \in \mathcal{A} : i \neq 0, j \in \mathcal{A} : i \neq j+1} w_{i,j} + \sum_{i \in \mathcal{A} : i \neq 0, j \in \mathcal{A} : i \neq j \wedge i \neq j+1} \theta_{i,j} &= CT, \\ \delta_i &\geq \underline{\delta}_i, & \forall i \in \mathcal{A} \\ \theta_{i+1,j} &\geq y_{i,j} \cdot \underline{\theta}_{i+1,j}, & \forall i, j \in \mathcal{A} \\ \theta_{i+1,j} &\leq 0, & \forall i, j \in \mathcal{A} : i+1 = j \\ \theta_{i+1,j} &\leq y_{i,j} \cdot CT, & \forall i, j \in \mathcal{A} : i+1 \neq j \\ w_{i+1,i+1} &\geq y_{i,i+1} \cdot p_{i+1}, & \forall i \in \mathcal{A} \\ w_{i+1,j} &\leq y_{i,j} \cdot CT, & \forall i, j \in \mathcal{A} \\ \underline{E}_i^\delta(\delta_i) &\geq e_{i,k}^\delta(\delta_i), & \forall i \in \mathcal{A}; k \in \mathbb{N} \\ \underline{E}_{i+1,j}^\theta(\theta_{i+1,j}) &\geq e_{i+1,j,k}^\theta(\theta_{i+1,j}) - (1 - y_{i-1,j}) \cdot B, & \forall i \in \mathcal{M}; j \in \mathcal{A}; k \in \mathbb{N} : i \neq 0 \end{aligned}$$

Chapter 4

Verifying Correct Mathematical Formulation in MILP Model

The model described in the previous chapter is complicated because it contains a lot of variables, parameters, and constraints. Primarily, there are many exceptions and conditions which depend on values of indices. For this reason, we developed an alternative approach in order to compare its solution with the solution of the [MILP].

In this chapter, we describe the alternative approach to define the model of our robotic cell based on the Floyd-Warshall Algorithm [16–17]. The model is simpler than the previous one because we assume an extra knowledge as the activity order in the RAS. At the end, we describe a pseudo code of a comparative and verification procedure that compares both MILP and alternative solutions. From the outset, it is necessary to say that this alternative procedure can only be used to verify the correctness of the mathematical formulation of the MILP model, as we can only solve small instances with this approach.

4.1 Alternative Model Description

In this section, we formulate the alternative model of our robotic cell. We describe the cell as a graph and use the Floyd–Warshall algorithm for finding the longest path in it.

Assume that the robotic cell is represented by an oriented graph G . Vertices of the graph G are activities A_0, A_1, \dots, A_m , and weights of edges are equal to the minimal value of durations that is needed between start times of activities. Let's say we know the order of activities in the RAS. This is the same as knowledge of values of all $y_{i,j}$ in the MILP model. According to that, we know which unloaded movements occur. It means that values of those variables can be nonzero and others must be zero. Then the longest path between two vertices (i, j) , $i, j \in \mathcal{A}$, is equal to the duration that the robot needs between start time of activity A_i and start time of activity A_j .

Let us consider the weight matrix C [$m \times m$] where the matrix elements correspond to the minimal value of duration that the robot need between start times of activities. In the beginning, the C matrix is initialized follows:

$$c(i, j) = \begin{cases} 0 & \text{if } i = j, \forall i, j \in \mathcal{A}; \\ -\infty & \text{otherwise} \end{cases}$$

Now we need to mathematically describe elements $c(i, j)$ in the weight matrix C . See Figure 4.1, which graphically represents relations between activities, and can be described by the following equations:

$$\underline{s}_i + \underline{\delta}_i + p_{i+1} \leq \underline{s}_{i+1}, \quad \forall i, j \in \mathcal{A}; \quad (4.1)$$

$$\underline{s}_i + \underline{\delta}_i + \underline{\theta}_{i+1,j} + (U_i - U_j) \cdot CT \leq \underline{s}_j, \quad \forall i, j \in \mathcal{A} : y_{i,j} = 1. \quad (4.2)$$

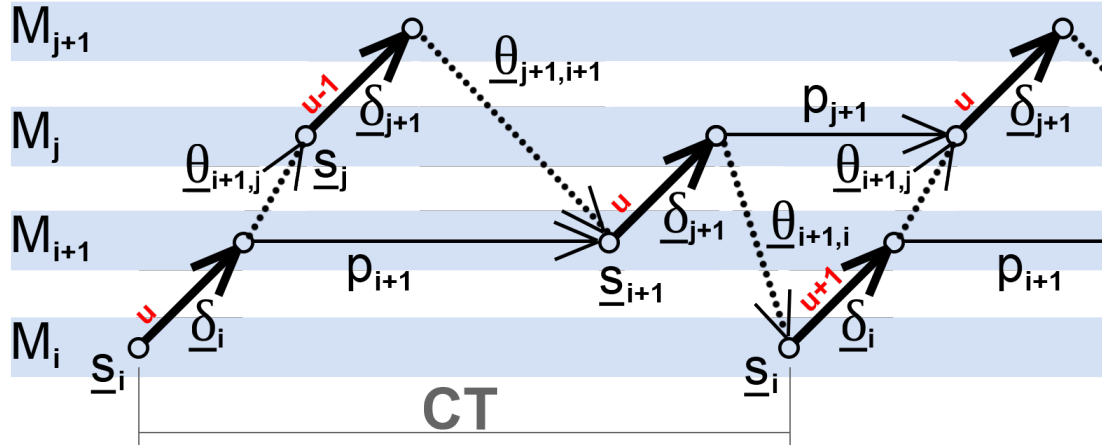


Figure 4.1. Relations between activities for using Floyd–Warshall algorithm: wide arrows represent minimal a time duration of a loaded robot move δ compared to that dot arrows represent a minimal duration which is needed for an unloaded robot movement θ ; thin arrows visualize a process time duration p on a machine; red labels notice a sequence number U of a part which is transferred by an activity.

where U_i respectively U_j is a sequence number of a part. This number starts from 1 and is incremented by one with each part which the cell begins to process. It may be difficult to determine values of sequence numbers. Therefore, we give the following example.

Example 4.1. Let's have a fixed RAS, for example $\{A_0, A_2, A_1\}$. The goal is to find sequence numbers U of parts for each activity in the RAS. The first following pseudocode describes, how to achieve the goal.

Let have variable n initialized to a u value. The initial value of u can be a general number so that it can be set up to $u = 1$. We want to find all the activities in the RAS in ascending order. Therefore, we define the variable a (initialized to zero), which keeps the index of the activity being searched. Then we periodically go through the RAS cycle until a temporary variable a is less than the number of activities in the RAS. If we find an activity in RAS which the indice number is equal to the value in the a variable, then we write the n value to the a position on the part sequence array and increment the a by one. We decrement variable n by one after each throughput of the RAS.

```

1 let U be an array of sequence numbers
2 let RAS be an array of robot activity sequence
3 let a be a temporary variable
4 let n be a counter of RAS cycles
5
6 a := 0 // activity A_0 is the first activity in the RAS
7 n := u // u is a general number, can be set up to u := 1
8
9 while a < len(RAS)
10   for i from 0 to len(RAS)
11     if RAS[i] = a
12       U[a] := n
13       a := a + 1
14     end if
15   n := n - 1

```

Figure 4.2 illustrates the solution of the part sequence numbers for the RAS specified in this example. A solution for more complicated RAS $\{A_0, A_4, A_3, A_5, A_2, A_1\}$ is shown in Figure 4.3. Now, it should be clear, how to find sequence numbers of parts for each activities in the RAS, and we can continue with model description.

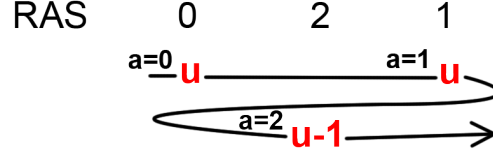


Figure 4.2. Visualization of the process of searching the part sequence numbers for the first example: The arrow visualizes algorithm throughputs of the RAS; black labels a represent an indice value of an activity which the algorithm actually finding; red labels mean a sequence number value of part that an activity transfer it.

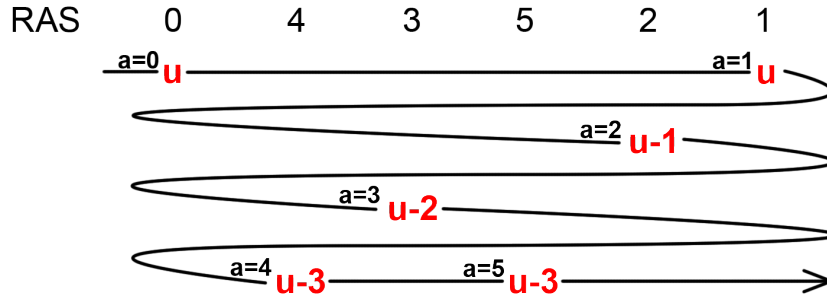


Figure 4.3. Searching the part sequence numbers for more complicated RAS example: descriptions of used symbols are same as in the Figure 4.2

From the previous two equations (4.1) and (4.2), we can edit the weight matrix C of the maximal durations where

$$c(i, i + 1) = \underline{\delta}_i + p_{i+1}, \quad \forall i \in \mathcal{A}; \quad (4.3)$$

$$c(i, j) = \underline{\delta}_i + \underline{\theta}_{i+1, j} + (U_i - U_j) \cdot CT, \quad \forall i, j \in \mathcal{A} : y_{i, j} = 1. \quad (4.4)$$

Equation (4.3) specifies the minimum duration between two activities which index differs by one (the A_i activity followed by the A_{i+1}). Equation (4.4) sets up the minimum duration between two activities which immediately follow in the RAS.

The initialized weight matrix C by equations (4.3) and (4.4) is an input for the Floyd–Warshall algorithm. After the algorithm ends its process, we must check values on the diagonal in output matrix C . If some of those values are positive, it means that algorithm does not find a feasible solution, because it contains a positive cycle. In other words, there does not exist a path which executes each activity just one time for the given RAS and CT. However, if the diagonal values stay zero, it says that the algorithm finds a feasible solution. Moreover, if we focus on values in the first line of the output matrix C , we have minimal start times of activities which transfer the same part (the part with the same sequence number). Be careful that these start times are not misinterpreted as compared to the samely labeled MILP start times. The difference is that MILP searches values of δ , θ and w that give the energy-optimal solution, however, the described alternative algorithm using only lower bounds ($\underline{\delta}$ and $\underline{\theta}$).

4.2 Comparative and Verification Procedure

From the previous section, we have defined an alternative model, and we know how the Floyd–Warshall algorithm works when is applied to our type of the robotic cell. Now we describe the comparative and verification procedure that compares both MILP and alternative solutions.

Let have an instance *cell* of a robotic cell. This instance contains all needed constants like delta, theta, processing times, energy functions, etc. The goal of the verification procedure is to find an interval of CT values for which the MILP formulation gives a feasible solution. The goal is achieved as follow:

- If the MILP formulation gives a feasible solution for the defined CT, the same RAS must be feasible in the Floyd procedure described previously for the same instance;
- or if the MILP formulation does not give a solution for the defined CT, the Floyd procedure cannot find any solutions for the same instance. In this case, the Floyd procedure tries to find solutions for all permutations of the RAS.

The problem is that the feasibility depends on the CT value. The goal of the procedure is to find the smallest CT value, denoted as $CT_{feasible}$, for which we can find at least one feasible solution. We assume a lower bound for the feasible CT interval equal to zero, denoted as CT_{lower} . We define an upper bound for the feasible CT interval, denoted as CT_{upper} , by the following equation:

$$CT_{upper} = 10 \times \sum_{i \in \mathcal{A}} \underline{\delta}_i + \sum_{i \in \mathcal{A} \setminus \{m\}} p_{i+1} + 10 \times \underline{\theta}_{m+1,0}. \quad (4.5)$$

The robot does $m + 1$ loaded movements δ and m times waits by machines while the waits durations are equal to process times p . At the end of the RAS, the robot does one unloaded move $\theta_{m+1,0}$ from the output buffer machine (M_{m+1}) to the input buffer machine (M_0). The constant multiplication 10 is used because we do not know the optimal move durations δ and θ but only their lower bounds.

See the following pseudo-code which we comment on the following text. The code cyclically does following steps until a difference between CT_{upper} and CT_{lower} is not small (less than CT_{res}):

- A MILP model is built and $MILP()$ procedure returns its feasibility status S_{ilp} ($S_{ilp} = true$ if the MILP model find feasible solution);
- If $MILP()$ returns a feasible solution, the $getRAS()$ method gets the RAS, and the $floyd()$ procedure enumerates that solution;
- Then two situations can occur. If the $floyd()$ procedure returns true (feasible solution), then both approaches give the same feasible result. It is the correct behavior and algorithm can continue. In the other case, there is a mistake in some approach and the algorithm reports an error;
- If $MILP()$ do not find a feasible solution. All permutations of RAS is built tested by $floyd()$ procedure. It returns feasibility S_e as OR product of feasibility of all permutations;
- Then two situations can occur. If the $floyd()$ procedure does not find any solution (returns false), then both approaches are not able to find a solution for the defined value of the CT. It is the correct behavior in this case, and the algorithm can continue. But if S_e is true, there is a mistake in some approach and algorithm reports an error;

- If the algorithm does not find an error, bounds for CT value are updated in the second phase of the algorithm;
- If $MILP()$ procedure gives a feasible solution, we save the value of the CT to the variable $CT_{feasible}$ first. Then the CT_{upper} is set to the value in the CT . The new CT value for the next cycle is count by splitting the interval $\langle CT_{lower}; CT_{upper} \rangle$;
- If $MILP()$ procedure gives a unfeasible solution, we save the value of CT to the variable CT_{lower} first. The CT_{upper} is not change in this situation. The new CT value for the next cycle is count by splitting the interval $\langle CT_{lower}; CT_{upper} \rangle$;
- Then the algorithm repeats the previous steps with the new CT value. When the $CT_{upper} - CT_{lower} < CT_{res}$ the verification algorithm returns value of the $CT_{feasible}$.

```

1 procedure bf_test(Instance cell)
2   let CT, CT_lower, CT_upper be temporary variables
3   let CT_feasible be the smallest feasible cycle time
4   let CT_res be the smallest resolution of cycle time
5   let S_ilp, S_e be boolean variables
6   let RAS be an array of robot activity sequence
7
8   CT_upper := "number" // CT that ensure feasible solution
9   CT := CT_upper
10  CT_lower := 0
11
12  while (CT_upper - CT_lower > CT_res)
13    S_ilp := MILP(cell, CT) // MILP model procedure
14    if(S_ilp)
15      RAS := getRAS() // Returns MILP RAS solution as the array
16      // Enumeration of RAS using Floyd-Warshall algorithm
17      S_e := floyd(cell, CT, RAS)
18      if(S_e) ok_feasible() // Both feasible
19      else error()
20    else
21      // Brute force enumeration using Floyd-Warshall algorithm
22      S_e := floyd(cell, CT)
23      if(!S_e) ok_unfeasible() // Both unfeasible
24      else error()
25    end if
26    if S_ilp
27      CT_feasible := CT
28      CT_upper := CT
29      CT := (CT_upper + CT_lower) / 2
30    else
31      CT_lower := CT
32      CT := (CT_upper + CT_lower) / 2
33    end if
34  return CT_feasible

```

In this chapter, we formulated the alternative approach which was selected for verifying of the correct mathematical formulation in the MILP model. We can not ensure that the MILP model is absolutely corrected using this approach, but it can minimalize possible mistakes to the minimum. For a randomly generated instance of our robotic cell and still decreasing CT value, we check all permutations of the RAS if the MILP model does not find a solution. If the feasibility of verification algorithm is the same as

the feasibility of the *MILP* () procedure, then we update the *CT* value and repeats the algorithm. In the end, we find the smallest *CT* value which gives the feasible solutions.

The problem with this approach is that the *floyd* () procedure must generate all permutations of the RAS in situations when the *MILP* () procedure does not find a solution. So it is possible to check models with up to 11 activities in reasonable time. Then the time needed for verification rapidly grow. However it is very likely, if the algorithm does not detect mistakes on small instances, it will not detect them in large ones.

Chapter 5

Heuristic Algorithm

In previous Chapters 3 and 4, we design and verify an exact algorithm for scheduling problem based on the MILP. This approach always finds an optimal solution, but it may take a long time, especially if the number of machines is large. For this reason, we design and implement an alternative algorithm exploiting the structure of the objective function. We chose the Genetic Algorithm (GA) [18] as the alternative approach. In the following part of the text, we first describe the principle of the genetic algorithm and then we apply it to our robotic cell.

5.1 Genetic Algorithm

The GA is a heuristic approach which uses a principle of natural selection. The GA uses evolution techniques which are known from biology as selection, inheritance, crossover, and mutation. The principle of the GA is a creation of generations which represents different solutions to a problem. The GA stores a set of solutions (called population) where each individual of the population is one solution to the problem.

The goal of the GA is to get high-quality solutions by an interactively applying principle of the natural selection. To do that, the GA needs to measure a solution quality by a fitness function for each individual in each generation. The fitness function expresses the solution quality represented by the individual. Individuals are partially randomly selected based on the fitness function value to a modification (crossover and mutation), thanks to this, the new individuals are creating for the next generation. Some high-quality solutions may achieve the next generation without these modifications. This procedure is interactively repeated thereby the quality of the solutions is gradually increasing. The GA is stopped after the solution quality reaches the defined value or after a time limit is exceeded.

5.1.1 Scheme of Genetic Algorithm

The genetic algorithm can be described by the following sequence of steps:

- 1) Create a zero population which contains randomly generated individuals;
- 2) Select several individuals with a high value of fitness function from the population.
Use some partially random procedure for the selection;
- 3) Generate a new population from the selected individuals using followings techniques:
 - crossover - exchange part of two individuals between them;
 - reproduction - copy some individuals without modification;
 - mutation - slightly modify some new individuals;
- 4) Evaluate fitness function for each new individual;
- 5) If a termination criteria is not achieved continue with step 2;
- 6) After the GA fulfill the termination criteria, the individual with the highest value of the fitness function is the best solution which the GA found.

So far we described general features and principle of the GA. In the next section, we apply these principles to our robotic cell, and we detailed explain each step of the GA.

5.2 Genetic Algorithm Application

In this section, we design a genetic algorithm which solves the energy optimization problem of our robotic cell. We gradually describe each part of the GA schema. At first, we explain a solution encoding. Then we describe a selection of the population size, and how we generate a zero population. One subsection is dedicated to a fitness computation. Then three subsections define three procedures how to generates individuals to a new generation. At the end of the section, we mention a termination criterion of the GA.

5.2.1 Solution Encoding

The effectivity of the genetic algorithm depends on a suitable encoding of a solution into the set of properties. Every properties has a defined set of possible values. The solution is then represented as combinations of values from all properties.

In our case, the goal of the optimization algorithm is to find the RAS and then durations of each movement and waiting in the idle state. So that the suitable encoding can be a RAS without the first activity which is always A_0 . Let us define the solution encoding as an array of activity indices. Thanks to this definition, the total number of solutions is equal to all permutations of activities A_1, A_2, \dots, A_m i.e. $m!$ solutions. The first part of the goal ensures solutions encoding and the second part measure a quality of a solution by the fitness function.

5.2.2 Population Size and Zero Population Generation

The number of distinct solutions depends on m . So it is not suitable to use a constant population size for all instances of robotic cells. We determine the population size as a function of the number of activities, $\alpha \times m$. The value of the α parameter is set by computational tests, see Section 7.3.

If we do not restrict the value of CT, all permutation of activities are feasible in the one-unit processing strategy. Therefore the zero population can be initialized randomly. However, we need to ensure that at least one individual from population gives the feasible solution for defined CT. Otherwise, we can not create a new generation through crossing and mutation. If no feasible permutation is generated, then the algorithm is restarted. Further for this case, there need to be defined the maximal execution time duration in the termination criterion.

5.2.3 Fitness Computation

The genetic algorithm needs to evaluate the quality of individuals. Additionally, the solution encoding which we use does not fully describe the solution of the robotic cell. In this subsection, we define a fitness computation that meets the previous requirements.

We define the fitness function using linear programming as in the case of the exact algorithm. However, the model is much simpler thanks to RAS knowledge from solution encoding. The output of the fitness computation is the quality of the solution, which is the most energy-efficient solution for the given RAS.

Now let's define the fitness computation as a Linear Programming (LP) problem. All variables, which we use, have the same meaning and are described in Chapter 3.

Minimize

$$\sum_{i \in \mathcal{A}} \underline{E}_i^\delta(\delta_i) + \sum_{i,j \in \mathcal{A}: y_{i,j}=1 \wedge i \neq j} \frac{E_{i+1,j}^\theta(\theta_{i+1,j})}{y_{i,j}=1} + \sum_{i,j \in \mathcal{A}: y_{i,j}=1} \frac{E_{i+1,j}^w(w_{i+1,j})}{y_{i,j}=1}; \quad (5.1)$$

subject to

$$s_i + \delta_i + p_{i+1} \leq s_{i+1}, \quad \forall i \in \mathcal{A}: i \neq m; \quad (5.2)$$

$$s_{i+1} + \delta_{i+1} - CT \leq s_i, \quad \forall i \in \mathcal{A}: i \neq m; \quad (5.3)$$

$$s_i = \widehat{s}_i + CT \cdot q_i, \quad \forall i \in \mathcal{A}; \quad (5.4)$$

$$s_0 = 0; \quad (5.5)$$

$$\widehat{s}_i + \delta_i + \theta_{i+1,j} + w_{i+1,j} \leq \widehat{s}_j, \quad \forall i, j \in \mathcal{A}: y_{i,j} = 1 \wedge j \neq 0; \quad (5.6)$$

$$\widehat{s}_i + \delta_i + \theta_{i+1,0} + w_{i+1,0} \leq \widehat{s}_0 + CT, \quad \forall i \in \mathcal{A}: y_{i,0} = 1; \quad (5.7)$$

$$\sum_{i \in \mathcal{A}} \delta_i + \sum_{i,j \in \mathcal{A}: y_{i,j}=1} w_{i+1,j} + \sum_{i,j \in \mathcal{A}: y_{i,j}=1} \theta_{i+1,j} = CT; \quad (5.8)$$

$$\delta_i \geq \underline{\delta}_i, \quad \forall i \in \mathcal{A}; \quad (5.9)$$

$$\underline{\theta}_{i+1,j} \leq \theta_{i+1,j} \leq CT, \quad \forall i, j \in \mathcal{A}: y_{i,j} = 1 \wedge i+1 \neq j; \quad (5.10)$$

$$w_{i+1,i+1} \geq p_{i+1}, \quad \forall i \in \mathcal{A}: y_{i,i+1} = 1 \wedge i \neq m; \quad (5.11)$$

$$w_{i+1,j} \leq CT, \quad \forall i, j \in \mathcal{A}: y_{i,j} = 1; \quad (5.12)$$

$$\underline{E}_i^\delta(\delta_i) \geq e_{i,k}^\delta(\delta_i), \quad \forall i \in \mathcal{A} \wedge k \in \mathbb{N}; \quad (5.13)$$

$$\underline{E}_{i+1,j}^\theta(\theta_{i+1,j}) \geq e_{i+1,j,k}^\theta(\theta_{i+1,j}), \quad \forall i, j \in \mathcal{A}, k \in \mathbb{N}: y_{i,j} = 1 \wedge i+1 \neq j. \quad (5.14)$$

Equations (5.2)-(5.5) are the same as equations used in the exact formulation. One difference is in the Equation (5.4), where the q_i is not a decision variable now, but it is a known constant. The value of the q_i is equal to a difference of part sequence numbers. If A_i and A_{i+1} activities transfer same part in one execution of the RAS, then the value of the q_i is equal to zero. But those activities can transfer different parts, and then it is needed to count the part sequence number, see Example in Section 4.1. Equations (5.6) and (5.7) are based on the double-inequality (3.23) used in the exact formulation when the order of the activities is known.

From Equation (5.8) implies, that a sum of durations of robot operations durations must be equal to the CT value. Equations (5.9)-(5.12) describe bounds for decision variable. Those bounds are used only for variables that are used in the robot operations defined by the RAS.

The last two Equations (5.13) and (5.14) describe a relation between duration of robot moves and energy which is consumed during a movement. Equation (5.13) is same as constraint used in the exact algorithm, while the Equation (5.14) can be simpler because we do not need to shut down this constraint for unused moves. That is possible thanks to the knowledge which movements are used at the time when we define this constraint.

The Equation (5.1) describes the objective function of the previous MILP formulation and ensures that the algorithm finds the minimal value of consuming energy for a defined order of activities in the RAS. The value of this objective we used as our fitness value.

■ 5.2.4 Elite Solutions

Elite solutions are some individuals which have the highest fitness value. Those individuals automatically move to the next generations. The best individuals are maintained through different generations by this techniques. The number of elite solutions which we move to the next generation is determined as a percentage β of the population size. The value of β parameter we set by computational tests, see Section 7.3. All the elite solutions can also be used as a parent for crossover or can be a little bit mutated. Mentioned techniques (crossover and mutation) are described in the following sections.

■ 5.2.5 Crossover

Crossover is a process when we produce a child solution from more than one parent solutions. Parent solutions must be selected by a partly random process.

In our GA we use a roulette wheel selection method [18]. The method is based on a real roulette wheel, which contains same size fields for each number. Then each number on the roulette has the same probability selection. However, we need the partly random process which prefers individuals with the highest fitness to be selected as parents. So we use the roulette wheel with the number of fields equal to a population size where a size of an individual field is proportional to an inverted value of its fitness. Probabilities of fields are normalized so that their sum is equal to one. Thanks to that we can randomly generate two numbers in an interval from 0 to 1 and then find two intervals which are assigned to individuals. By this process, we select two individuals as parents for the crossover technique. We create two children from those two parents by the following process. At first, we randomly generate a cutting point. Then the RAS is copied from the first (second) parent to the first (second) child until the cutting point. The second part beyond the cutting point is completed by adding the same sequence of missing activities from the other parent RAS.

■ 5.2.6 Mutation

Mutation is very useful technique. It enables to achieve solutions which cannot be built only by the crossover technique from the initial population. Let assume that no individual from the initial population does not start with the first activity. Then the crossover technique is not able to generate a child which start with the first activity. That is one example when the mutation is useful.

The mutation interchanges the order of two activities in an individual with probability γ . If the individual is selected for mutation, then the activities for interchange are selected randomly. The value of parameter γ is set by computational tests, see Section 7.3. The mutation is the last operation changing the new population, and it is applied to both previously generated individuals by elite solutions and the crossover technique.

■ 5.2.7 Termination Criterion

The genetic algorithm is not able to determine if it found an optimal solution. Thus it is essential to define one or more heuristic termination criterion.

In our case, the GA terminates after λ generation which returns the same best solution. The value of parameter λ is set by computational tests, see Section 7.3. The previous termination criterion may not be sufficient, because of a time convergence to a local or a global minimum may take not defined time especially for large robotic cells with many machines. Other problem can occur when no solution exists for defined CT value. However, the similar situation occurs when the CT value is nearby the minimal

CT value then only a few solutions are feasible from all permutation of activities. In these situations, the algorithm repeatedly generates an initial population until it creates some feasible solution. Thus we need a second termination criterion based on a duration of execution for situations when the GA executes its computation for a long time.

Chapter 6

Benchmark Instances

In this chapter, we describe how we generate instances of our robotic cell, and which benchmark sets we have generated for evaluation of our algorithms. However, it is necessary to say that no test instances do not appear for our problem in the literature. Some benchmark instances exist for the robotic flow shop problem, but they are generated for models that minimize CT or maximize throughput, i.e., the benchmark used by Gultekin et al. (2018) [4] from the source written by Carlier (1978) [19]. So we can not use those benchmark instances because, as opposed to others, we minimize the energy consumption of the robotic cell with the fixed CT value. That is the reason why we generate own benchmarks.

In the first section, we explain the parameters which is necessary to generate our benchmark instances. In the second section, we describe the generated instances on which we perform experiments.

6.1 Generating of Benchmarks

In the following text, we describe which parameters are needed for an initial description of our robotic cell. If we need to create a new instance, we must define approach with which we generate values for all constant parameters. These constant parameters include:

- lower bound durations for loaded and unloaded movements,
- processing times of parts on machines,
- value of CT,
- parameters which describe energy functions,
- and some other parameters, i.g., a very large number B , a sample period used for sampling energy functions and number of samples.

Let's first explain the generation of lower bound durations for loaded and unloaded moves. We do not assume specific machine layout, thanks to this we do not expect a relationship between machine distance and duration of movement between them. This approach we also select because of that duration of move between two nearby machines can require time-consuming changes of the robot configuration. These problematic configurations are located close to a maximum motor rotation value which is used in robot arms or near singular robot areas. If we consider the previous description, we can randomly generate values for both types of lower bound durations $\underline{\delta}$ and $\underline{\theta}$ from the interval (0.1 s - 2 s) with the uniform distribution. With this approach, we do not add to the model the dependence on machine layout of the robotic cell.

Other generated parameters are the process times of parts on machines. We randomly generate values of those parameters p from the interval (5 s - 60 s) with the uniform distribution. The resulting robotic cell can thus include a combination of both short and long manufacturing operations.

Before we describe the generation of energy functions, let's explain a few general robotic cell parameters. The most important parameter is the number of production machines. This parameter is equal to the value of the index m , which we first introduced in Chapter 2, and we retain its importance in the whole thesis. However, it is necessary to remind that m means the number of production machines while the robotic cell contains $m + 2$ machines (do not forget to the input and output buffers). The parameter m can take integer value greater than 1 which is evident from the problem statement. Another parameter is B which is a large number. This number must be large at least as the highest possible value of energy functions because the number B is used to eliminate the effect of unused unloaded movements in the MILP formulation. So we set this parameter to the largest value which can be represented in an operating system.

In this paragraph, we explain how to generate a CT value. Other work is usually looking for a minimum CT value. So, in real use of our model, the user knows the minimum CT value or at least knows the current CT in a non-optimized robotic cell. For the industry, interesting instances of robotic cells are those which have the CT value close to the minimal possible CT value. The minimum CT value of our robotic cell model can be determined using the verification procedure described in the Chapter 4 but only for a cell with 10 production machines ($m = 10$). For larger instances, we need to estimate the CT value. However, all instances we have generated allow us to find a feasible schedule.

Now, we describe how we generate parameters for energy functions. This work is based on data measured by Libor Bukata¹. He has simulated a move of a six-axis manipulator to obtain dependence of duration and energy consumption on speed. His simulated move was concatenated from four smaller movements. He simulated the concatenate move in the Tecnomatix Process Simulates² software developed by Siemens company and using the Kuka Robot RCS module. See the Table 6.1 which contains measured data for robot Kuka kr150 r2700 extra³, and the Figure 6.1 show a graph from this data.

<i>speed [%]</i>	<i>duration of all four moves [s]</i>	<i>energy [J]</i>
100	5.19	23 821
80	6.17	20 248
50	9.45	16 632
25	17.73	17 706
15	29.11	22 697

Table 6.1. Measured data from the simulated movement.

From observation of the Figure 6.1 and from the literature [13–14], we can say that each movement of a robot can be split into three sections (descent, minimum, and growth). The robot consumes a lot of energy when it uses the maximum speed to move. The simple consideration is to let the robot can move slower, so it will consume less energy. However, the previous consideration is not entirely correct. A slowly moving robot must also compensate gravity force for a longer time. Thanks to this, energy curves have a global minimum after which the energy consumption increases almost linear. Vergnato et al. (2012) [13] describe an equation how to parametrize

¹ Postgraduate student and research worker in Industrial Informatics Research Center, CTU in Prague; <http://industrialinformatics.fel.cvut.cz/member/libor-bukata>

² More information: <https://www.plm.automation.siemens.com/global/en/products/tecomatix/>

³ More information: <https://www.robots.com/robots/kuka-kr-150-r2700-extra>

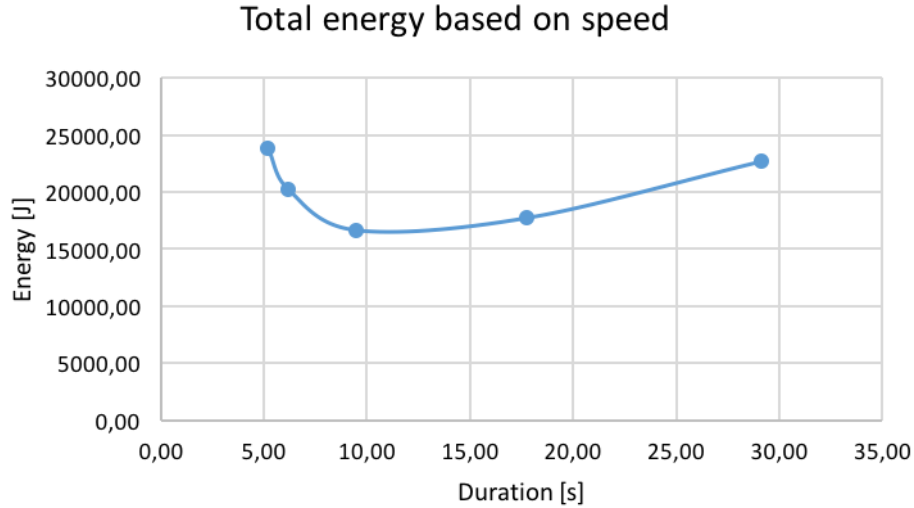


Figure 6.1. The graph shows the dependence of energy consumption on a robot speed for the simulated movement.

those energy functions:

$$E(t) = \sum_{c=-3}^1 a_c t^c = a_{-3} t^{-3} + a_{-2} t^{-2} + a_{-1} t^{-1} + a_0 + a_1 t. \quad (6.1)$$

If we approximate the measured data from the Table 6.1 using the Equation (6.1) we obtain the following coefficients:

$$a_{-3} = 300; \quad a_{-2} = 361400; \quad a_{-1} = 3000; \quad a_0 = 7200; \quad a_1 = 515. \quad (6.2)$$

In this way, we have one possible option of the energy function. In order not to use the same curve for all movements, we generate the individual curve coefficients for each move. Values of coefficients are randomly generated with the uniform distribution from following intervals:

$$\begin{aligned} a_{-3} &\in [100; 5000); \\ a_{-2} &\in [300; 370000); \\ a_{-1} &\in [3000; 7000); \\ a_0 &\in [3000; 8000); \\ a_1 &\in [200; 800). \end{aligned} \quad (6.3)$$

We need to be careful with one problem that can occur if we independently generate a lower bound duration of a motion and its associated energy function. The lower bound duration of the movement must be less than the argument of the energy function in the global minimum. If we break this condition described in the previous sentence, the movement will not correspond to the real movement of the robot. In this case, the solver would only use the growing part of the energy function, and it is not possible. Therefore, it is necessary to check this condition when we generate energy functions. The global minimum can be found by using the first derivative of the energy function as:

$$E(t)' = 0; \quad (6.4)$$

$$-3a_{-3}t^{-4} - 2a_{-2}t^{-3} - a_{-1}t^{-2} + a_1 = 0. \quad (6.5)$$

Intervals defined in the Equation (6.3) together with the interval for generating lower bound durations $\underline{\delta}$ and $\underline{\theta}$ described above allow creating only suitable combinations.

So far we have described the generation of energy functions for loaded and unloaded movements. In these cases, we were able to come out from the simulated motion. However, the used simulation program does not allow to simulate the robot's consumption when it does not move. Thanks to this, we can not estimate consumes in idle states without measuring on a real robot. For this reason, we expecte the robot's consumption in the idle states as a linearly increasing dependence on wait time duration, $E(t) = a_w t + 0$. The interval of values for the a_w coefficient is created like 20 % of the interval for the a_1 parameter, $a_w \in [40; 160]$.

The last two parameters for generating benchmark instances are sampling period and number of samples. Energy functions are convex functions, and [MILP] formulation allows only linear functions. Therefore, we sample energy functions with a sampling period of 0.25 s. At these points, which are one hundred, the energy function is approximated by the linear function. So that, each energy curve is sampled in the interval (0.25 s - 25 s).

The Table 6.2 shows a summary of all parameters and their values that we used to generate benchmark instances.

<i>parameter</i>	<i>value or interval</i>
lower bound duration $\underline{\delta}$	0.1 s - 2.0 s
lower bound duration $\underline{\theta}$	0.1 s - 2.0 s
process time p	5 s - 60 s
number of production machines m	min 2
large number B	max double value
CT	—
coefficient a_{-3}	100 - 5000
coefficient a_{-2}	300 - 370000
coefficient a_{-1}	3000 - 7000
coefficient a_0	3000 - 8000
coefficient a_1	200 - 800
coefficient a_w	40 - 160
sampling period	0.25 s
number of samples	100

Table 6.2. Summary of parameters and their values or value intervals that are used to generate benchmark instances.

6.2 Set of Benchmarks

In this section, we describe the generated instances on which we perform experiments. Each generated instance is stored in a separate plain text file. The file structure is as follows:

The first line contains two numbers, the first is the number of production machines m , and the second number is the CT value. Following lines always contain the name of the parameter (p, delta, or theta) and its value. Next rows contain information about parameters of the linear straight line used to approximate the energy curve. All of these test instances are stored on the enclosed CD.

We generate test instances for several robotic cell size for evaluation of our method, i.e., robotic cells with 2-15 processing machines. The benchmark set always contains 5

instances for each size of the considered robotic cell. So the benchmark set includes 70 instances of our robotic cell. We also generate some instances for a specific experiment, but those instances are stored in a separate directory.

Chapter 7

Experiments

This chapter deals with experiments which we have done. The chapter purpose is to show the usability of the exact and the heuristic approaches, and their comparison. Experiments are conducted on benchmark instances. See the previous Chapter 6 where we explain a process of generating those instances.

All experiments are performed on a machine with Intel Core i5 2,3 GHz (2 cores), 8 GB of DDR3 RAM under a 64-bit operating system. Mathematical models are solved with a Gurobi Optimizer 7.5.1 solver with 600 s time limit. The genetic algorithm is executed with 900 s time limit.

7.1 Exact Algorithm - Dependence of an Execution Time on a CT Value

We select two instances for this experiment. The first cell contains 7 processing machines ($m = 7$) and the second contains 10 processing machines ($m = 10$). Because a size of cells is smaller than 11 we can compute the minimal CT value which gives a feasible solution by the verification procedure, see Chapter 4.

At first, we compute the smallest CT value which gives the feasible solution. Then we repeatedly execute the exact algorithm with the same instance but with the increasing CT value, and measure the CPU execution time.

The Figure 7.1 shows the dependence of a CPU execution time on a CT value for two tested instances. We can see that instances, where their CT value is near to the least CT value for those instances, need a more CPU execution time.

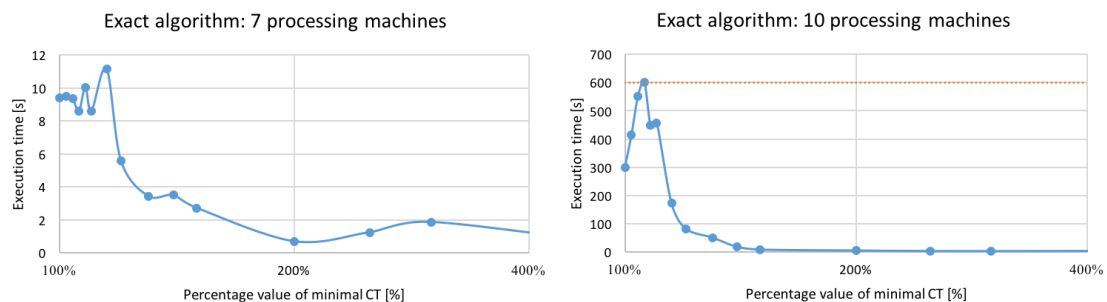


Figure 7.1. Dependence of an execution time on a CT value. The x-axis is in the logarithmic scale, and the dot orange line represents the time limit for the CPU execution time.

Thus, we can say that difficult instances are those which have its CT value close to the lowest CT value. It follows that most industrial applications would be among difficult instances. Because the goal of an industrial production is often to maximize production in a minimum time.

7.2 Exact Algorithm - Dependence of Energy Consumption on a CT Value

We select the same two instances as in Section 7.1 for this experiment to demonstrate the other one dependence. We make the same cells with the difference that now we measure the amount of consumed energy.

The Figure 7.2 shows the dependence of energy consumption on a CT value for two tested instances.

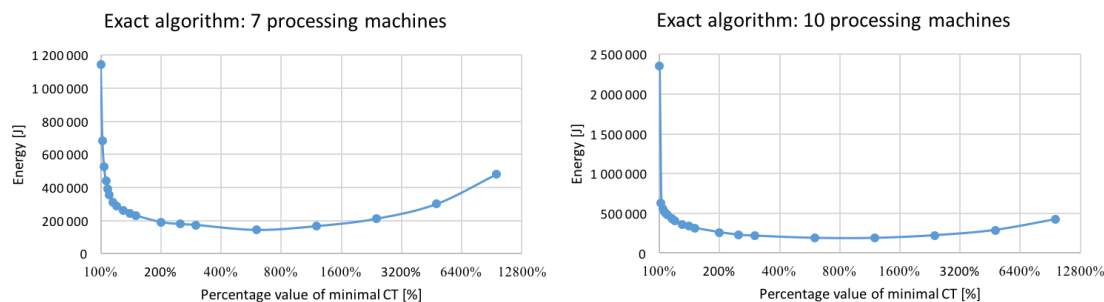


Figure 7.2. Dependence of an energy consumption on a CT value. The x-axis is in the logarithmic scale.

From observation of above graphs, i.e., Figure 7.2, we can say that the concatenate robot movement has a similar dependence as any single robot movement operation. If we do not require on the schedule with a minimum CT value, energy consumption is rapidly decreasing, i.g., the difference between the minimal CT value and about 2 % greater CT value can save 40 % of energy on each RAS cycle. If we consider the difference between 104 % and 110 % of the minimal CT value, it is possible to save 32 % of energy. Similarly, energy consumption gradually decreases with increasing of the CT value up to the value where it is possible to achieve the optimum for each movement, then energy consumption linearly increasing¹. From that CT value, the robot uses the excess time for the least energy-intensive operation, which can be assumed to be waiting in an idle state, specifically in the least energy-consuming one.

7.3 Genetic Algorithm - Parameter Calibration

In this section, we discuss calibration of the heuristic algorithm. Then we make computation test to determine the best parameter values of the genetic approach which we describe in the Chapter 5.

We use a robotic cell with 10 processing machine ($m = 10$) for parameter calibration. To select the best parameter values, we define two levels for each parameter, see Table 7.1. Each combination, which is 2^4 , is solved for three times to minimize the randomness process used in the GA. As a result, we solve $2^4 \times 3 = 48$ problems, and it is our dataset for parameters calibration.

The performance of each parameter combinations is measured by two approaches. The first is a relationship of the maximum % error to its average % error. The second approach is base on a relationship of the average CPU time to the average % error. In the first phase of the parameter calibration, we compute the optimal solution for the

¹ The linear dependence may not be straightforward since the x-axis is logarithmic.

Parameter	Levels	
α	10	15
β	0.1 (10 %)	0.2 (20 %)
γ	0.05 (5 %)	0.1 (10 %)
λ	10	15

Table 7.1. The levels of the parameters used in our GA formulation.

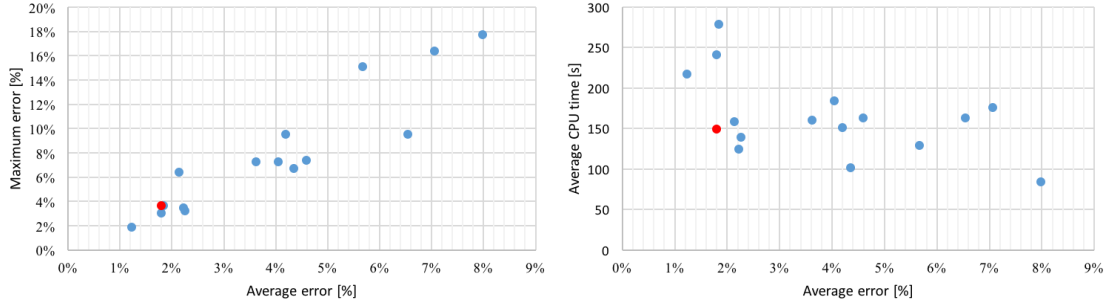


Figure 7.3. The parameter calibration result for the GA. The red mark is one result which we selected, and parameter combination of this result we use for next evaluation tests of the GA.

tested instance by the exact algorithm. By this way, we get the optimal solution that is used to calculate errors. In the next phase, the average CPU time, the average and the maximum error of the three replication are calculated. The measured results for each of the 48 parameter combinations are shown in Figure 7.3.

We selected the parameter combination which achieves the average error of 1.81 % and the maximum error of 3.62 %. This combination is highlighted by the red mark in Figure 7.3. We selected that despite the fact that exist better combinations with the smaller of the error value, but the selected one has the smaller average CPU time (149.04 s). The selected parameter values are $\alpha = 10$, $\beta = 0.2$, $\gamma = 0.1$, and $\lambda = 15$. We use those parameters for next evaluation tests of the GA.

7.4 Comparison of Exact and Heuristic Algorithm

In this section, we describe the last experiment which gives a comparison of two approaches that we implemented, i.e., the exact and the heuristic algorithm. The performance of both algorithms is measured at same instances. In Chapter 6, we describe the method with which we generate benchmark instances, and we also explain the dataset structure. Our use benchmark datasets consist of the 70 randomly generated instances which model robotic cells with m producing machines where m is the number between 2 and 15. We give those datasets publicly available.

The experiment procedure is as follows:

- One solution (amount of consumed energy) is counted by the exact algorithm using the MILP formulation by the Gurobi Optimization solver. It gives us the best-found solution and the gab. The gab is a percentage value that indicates the smallest value of a potentially optimal solution. In other words, the solver need not find the optimal solution in a defined time limit, but it gives us a certainty that the optimal solution is not less than the best-found solution minus the gab.

- The same instance is solved by the heuristic algorithm three times to minimize the randomness process used in the GA. The best and the average of solutions, and the best and the average CPU time are saved.
- Differences between the solution from the exact algorithm and the best respectively the average of solutions from the GA are counted by the follows equations:

$$\text{diff} = 100 \times (\text{Min. of 3 replication GA} - \text{Best found MILP}) / \text{Best found MILP}, \quad (7.1)$$

$$\text{avgdiff} = 100 \times (\text{Avg. of 3 replication GA} - \text{Best found MILP}) / \text{Best found MILP}. \quad (7.2)$$

The results are summarized in Tables 7.2 and 7.3.

<i>file</i>	MILP			GA			<i>diff [%]</i>	<i>avg diff [%]</i>
	<i>obj [J]</i>	<i>gab [%]</i>	<i>CPU [s]</i>	<i>obj [J]</i>	<i>CPU [s]</i>			
m02.1	357855	0.00	0.01	357855	3.23	0.00	0.00	
m02.1	357 855	0.00	0.01	357 855	3.23	0.00	0.00	
m02.2	115 345	0.00	0.02	115 345	3.51	0.00	0.00	
m02.3	750 607	0.00	0.01	750 607	2.98	0.00	0.00	
m02.4	129 526	0.00	0.01	129 526	3.30	0.00	0.00	
m02.5	97 523	0.00	0.01	97 523	3.42	0.00	0.00	
m03.1	169 832	0.00	0.02	169 832	5.88	0.00	0.00	
m03.2	723 040	0.00	0.02	723 040	5.47	0.00	0.00	
m03.3	143 788	0.00	0.05	143 788	6.16	0.00	0.00	
m03.4	264 416	0.00	0.02	264 416	5.87	0.00	0.00	
m03.5	158 166	0.00	0.30	158 166	6.36	0.00	0.00	
m04.1	751 237	0.00	0.32	751 237	9.95	0.00	0.00	
m04.2	388 612	0.00	0.18	388 612	9.90	0.00	0.00	
m04.3	428 052	0.00	0.40	428 052	9.32	0.00	0.00	
m04.4	207 670	0.00	0.49	207 670	11.04	0.00	0.00	
m04.5	225 334	0.00	0.22	225 334	9.89	0.00	0.00	
m05.1	204 894	0.00	0.64	219 753	16.84	0.00	4.61	
m05.2	306 745	0.00	1.20	434 762	18.24	0.00	21.77	
m05.3	185 460	0.00	0.63	190 172	15.93	2.48	2.48	
m05.4	414 018	0.00	0.73	414 018	16.69	0.00	0.55	
m05.5	319 234	0.00	0.75	322 484	14.20	1.01	1.01	
m06.1	336 578	0.00	9.13	344 750	50.26	0.00	1.59	
m06.2	224 743	0.00	2.39	224 743	24.32	0.00	3.26	
m06.3	195 212	0.00	2.45	195 212	36.75	0.00	1.10	
m06.4	205 777	0.00	3.02	205 777	26.99	0.00	0.00	
m06.5	199 063	0.00	1.03	199 063	24.13	0.00	0.24	
m07.1	336 578	0.00	9.15	341 664	31.97	1.49	1.83	
m07.2	278 695	0.00	2.99	278 695	50.72	0.00	3.84	
m07.3	308 145	0.00	4.57	320 732	45.97	3.92	4.95	
m07.4	242 004	0.00	3.98	247 394	58.28	0.00	3.06	
m07.5	218 531	0.00	5.75	218 531	50.98	0.00	1.58	

Table 7.2. Summary of the results which compare the implementation of the exact and the heuristic algorithm. Other results continue in Table 7.3

<i>file</i>	MILP			GA			
	<i>obj [J]</i>	<i>gab [%]</i>	<i>CPU [s]</i>	<i>obj [J]</i>	<i>CPU [s]</i>	<i>diff [%]</i>	<i>avg diff [%]</i>
m08_1	277 065	0.00	5.95	297 078	60.61	0.00	4.20
m08_2	259 188	0.00	10.27	267 947	72.84	3.27	3.41
m08_3	317 811	0.00	5.45	317 811	40.26	0.00	2.66
m08_4	287 385	0.00	8.23	287 385	80.03	0.00	1.29
m08_5	298 237	0.00	15.37	308 280	42.72	0.00	3.22
m09_1	354 815	0.00	31.07	399 005	106.77	0.00	4.55
m09_2	300 610	0.00	44.30	314 248	98.51	0.00	4.03
m09_3	318 616	0.00	41.28	333 985	118.60	2.83	3.86
m09_4	395 570	0.00	16.44	410 370	82.62	0.00	1.23
m09_5	396 471	0.00	33.00	423 237	99.14	2.02	4.69
m10_1	457 908	0.00	51.49	471 881	179.23	2.96	4.51
m10_2	392 724	0.00	64.19	416 061	140.83	2.08	5.59
m10_3	430 437	0.00	152.82	467 250	176.06	0.00	5.03
m10_4	307 226	0.00	88.53	326 478	212.22	3.83	4.61
m10_5	340 653	0.00	182.86	370 727	119.99	4.10	7.96
m11_1	367 593	0.00	49.97	369 325	189.51	0.47	0.71
m11_2	351 105	0.00	5.41	382 287	262.70	7.53	8.77
m11_3	389 213	0.00	6.26	398 364	141.85	0.00	4.23
m11_4	418 244	0.00	205.17	444 192	152.89	5.18	5.63
m11_5	476 768	0.00	350.54	493 234	155.17	1.33	3.38
m12_1	441 029	0.00	221.99	454 933	394.68	3.06	7.85
m12_2	481 796	0.00	196.33	584 840	298.52	4.53	11.42
m12_3	529 998	0.00	219.28	588 689	304.68	4.40	7.46
m12_4	478 099	0.00	520.69	528 551	212.91	6.65	8.26
m12_5	513 181	3.44	TL	537 337	595.13	3.53	5.79
m13_1	634 997	5.92	TL	768 320	240.82	9.94	15.02
m13_2	881 638	6.27	TL	1 037 471	271.15	6.29	9.72
m13_3	550 192	6.21	TL	587 171	557.32	5.56	6.57
m13_4	809 354	20.93	TL	785 343	286.81	-3.06	3.04
m13_5	711 191	14.43	TL	749 400	499.95	0.77	7.70
m14_1	795 797	10.81	TL	887 668	307.45	6.04	8.74
m14_2	777 523	12.78	TL	794 471	391.81	2.13	6.94
m14_3	685 192	15.13	TL	762 532	484.76	6.13	9.50
m14_4	809 610	19.42	TL	720 840	579.51	-12.31	-7.49
m14_5	674 889	6.26	TL	851 220	453.35	5.47	13.57
m15_1	757 470	18.92	TL	824 162	410.77	-0.24	3.04
m15_2	933 222	23.34	TL	847 115	841.75	-10.16	-2.21
m15_3	902 659	15.30	TL	1 027 537	906.68	0.32	6.39
m15_4	1 117 970	32.58	TL	1 025 268	903.148	-19.41	-9.04
m15_5	883 172	18.55	TL	911 475	733.742	-7.17	-2.60

Table 7.3. Continuation of the summary of the results from the previous Table 7.2

In 16 of the instances, the exact algorithm based on the MILP formulation hit the ten-minute time limit, and so it did not find an optimal solution, but it found a feasible integer solution with its gab in all tested cases. It is observed that exact algorithm can effectively deal with our variants of the robotic cell with up to 11 processing machines in time less than 10 minutes. Therefore, it makes no sense to use the heuristic approach

in smaller robotic cells. However, the implemented heuristic found the same optimal solution in 35 out of 50 cases where $m < 12$. In the rest of the 15 instances, the GA finds a solution that is maximal of 7.53 % worse compared to the optimal solution, and on average of just 2.96 % worse.

The Heuristic approach is beginning to be interesting for our robotic cell variants where $m > 11$. From the observation of the results, we can see that the GA algorithm find the worse solution than the exact algorithm with the ten-minutes time limit in 14 from 20 solutions (on average of 4.63 % worse), but it needs mostly less computation time. In the 6 tested instances, the GA algorithm finds the better solution than computes the exact algorithm.

Chapter 8

Conclusion

Optimizing energy consumption is undoubtedly an important problem for the industry because optimization can significantly reduce energy consumption and thus reduce the fixed cost of manufacturing the product. Another advantage of such optimization is that it can be implemented into existing robotic cells.

This thesis presents an optimization approach for flow shop (FS) robotic cells, which are cells consisting of m producing machines and a material handling robot producing one type of a part. The robot transfers parts between machines and ensures machine loads and unloads. We consider the cyclic scheduling of the robot moves with the objective of minimizing the energy consumption.

We developed a mixed integer linear programming formulation (MILP) that determines the energy optimal robot activity sequence (RAS). We described a verification procedure to evaluate the correctness of the MILP formulation to minimize mistakes in the mathematical model. Since the problem complexity increases concerning the number of machines, and it belongs into the category of NP-hard problems, we developed a heuristic approach based on a genetic algorithm (GA).

In the experiments, we have described and verified which instances are complicated to solve. As a result, we have generated a dataset of benchmark instances from just these complicated instances. No other benchmark datasets are available yet for the problem defined by us. Therefore we give our datasets publicly available. Furthermore, we have shown that by the energy optimization we can save up to 32 % of energy if we use by 2 % longer CT value and do not require the maximum speed of the robot. In the end, we compared both approaches as the exact algorithm based on the MILP formulation so the heuristic approach based on the GA. We have shown that the exact algorithm proposed by us can very effectively and quickly solve most of the robotic cells considered by us. Therefore, it seems unnecessary to create the heuristic approach, but we designed it for all these reasons: We used the very good Gurobi Optimization solver in the implementation of the exact algorithm that can be too expensive for non-commercial use; Other cheaper solvers would probably not reach so good results; Another reason is that other studies consider significantly larger cells, i.g., Gultekin et al. (2018) [4] describe in their computation study cells used in chemical and electroplating industry that contains up to 28 machines.

The main contribution of the thesis is the different objective function compared to other studies that deal with the optimization of the flow shop type manufacturing cells. Our study is the first which optimizing energy consumption of the robotic flow shop cell.

This thesis may be extended in the future by more complicated flow shop robotic cells. So far we have proposed a procedure for optimizing of the single-gripper simple robotic cell, which processes single part type, but we can arbitrarily reformulate the mathematical model to another combination in the future.

References

- [1] IFR International Federation of Robotics. *Executive Summary World Robotics 2017 Industrial Robots*. 2017.
https://ifr.org/downloads/press/Executive_Summary_WR_2017_Industrial_Robots.pdf.
- [2] D. Meike, and L. Ribickis. *Energy Efficient Use of Robotics in the Automobile Industry*. The 15th International Conference on Advanced Robotics, 2011.
- [3] L. Bukata, P. Šůcha, Z. Hanzálek, and P. Burget. *Energy Optimization of Robotic Cells*. IEEE Transactions on Industrial Informatics, volume 13, issue 1, 2017.
- [4] H. Gultekin, B. Coban, and V. E. Akhlaghi. *Cyclic scheduling of parts and robot moves in m-machine robotic cells*. Computers and Operations Research, 2018.
- [5] M. W. Dawande, H. N. Geismar, S. P. Sethi, and C. Sriskandarajah. *Throughput Optimization in Robotic Cells*. 1 edition. Springer US, 2007. ISBN 978-0-387-70988-8.
- [6] S. P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak. *Sequencing of parts and robot moves in a robotic cell*. International Journal of Flexible Manufacturing Systems, volume 4, issues 3-4, 1992.
- [7] N. Brauner, and G. Finke. *Cycles and permutations in robotic cells*. Mathematical and Computer Modelling, volume 34, issues 5-6, 2001.
- [8] N. Brauner, G. Finke, and W. Kubiak. *Complexity of one-cycle robotic flow-shops*. Journal of Scheduling, volume 6, issue 4, 2003.
- [9] G. D. Batur, O. E. Karasan, and M. S. Akturk. *Multiple part-type scheduling in flexible robotic cells*. Int. J. Production Economics, 2011.
- [10] A. Elmi, and S. Topaloglu. *Cyclic job shop robotic cell scheduling problem: Ant colony optimization*. Computers & Industrial Engineering, 2017.
- [11] G. D. Batur, S. Erol, and O. E. Karasan. *Robot move sequence determining and multiple part-type scheduling in hybrid flexible flow shop robotic cells*. Computers & Industrial Engineering, 2016.
- [12] M. W. Dawande, H. N. Geismar, S. P. Sethi, and C. Sriskandarajah. *Sequencing and Scheduling in Robotic Cells: Recent Developments*. Journal of Scheduling, volume 8, issue 5, 2005.
- [13] A. Vergnano, C. Thorstensson, B. Lennartson, P. Falkman, M. Pellicciari, F. Leali, and S. Biller. *Modeling and Optimization of Energy Consumption in Cooperative Multi-Robot Systems*. IEEE Transactions on Automation Science and Engineering, volume 9, issue 2, 2012.
- [14] A. Mohammeda, B. Schmidt, L. Wanga, and L. Gaoc. *Minimizing Energy Consumption for Robot Arm Movement*. 8th International Conference on Digital Enterprise Technology, 2014.

-
- [15] P. Šůcha, Z. Pohl, and Z. Hanzálek. *Scheduling of Iterative Algorithms on FPGA with Pipelined Arithmetic Unit*. Real-Time and Embedded Technology and Applications Symposium, 2004.
- [16] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, pp. 133-165, 1993.
- [17] B. Korte, and J. Vygen. *Combinatorial Optimization*. Springer, pp. 157-163, 2006.
- [18] M. Melanie. *An Introduction to Genetic Algorithms*. A Bradford Book The MIT Press, 1999.
- [19] J. Carlier. *Ordonnancements a contraintes disjonctives*. RAIRO-Operations Research, volume 12, issue 4, 1978.

Appendix A

Specification



MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Chmel Jakub** Personal ID number: **406450**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Cybernetics**
Study program: **Cybernetics and Robotics**
Branch of study: **Robotics**

II. Master's thesis details

Master's thesis title in English:

Energy Optimization of Robotic Cells with Single Robot and m Machines

Master's thesis title in Czech:

Optimalizace spotřeby robotických buněk s jedním robotem a m stroji

Guidelines:

This thesis addresses production systems with m machines and a single robotic manipulator. The aim is to find a cyclic schedule of the manipulator moves that guarantees the required production rate and minimizes energy consumption of the manipulator. The goal is to propose an efficient algorithm for this scheduling problem. The particular objectives of the thesis are:

- 1) Review the existing works in the robotic cell scheduling domain.
- 2) Design an exact algorithm for the scheduling problem based on Mixed Integer Linear Programming.
- 3) Implement the exact algorithm and evaluate its performance on randomly generated benchmark instances.
- 4) Design and implement an alternative algorithm exploiting the structure of the objective function.
- 5) Compare both algorithms on the benchmark instances.

Bibliography / sources:

- [1] Hakan Gultekin, Betül Coban, Vahid Eghbal Akhlaghi, Cyclic scheduling of parts and robot moves in m-machine robotic cells, In Computers & Operations Research, volume 90, 2018, pages 161-172.
- [2] Libor Bukata, Přemysl Šůcha, Zdeněk Hanzálek and Pavel Burget, Energy Optimization of Robotic Cells, In IEEE Transactions on Industrial Informatics, volume 13, issue 1, 2017, pages 92-102.
- [3] Jacques Carlier, Mohamed Haouari, Mohamed Kharbeche, Aziz Moukrim, An optimization-based heuristic for the robotic cell problem, In European Journal of Operational Research, volume 202, issue 3, 2010, pages 636-645.

Name and workplace of master's thesis supervisor:

doc. Ing. Přemysl Šůcha, Ph.D., Optimization, CIIRC

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

Date of master's thesis assignment: **09.01.2018** Deadline for master's thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

doc. Ing. Přemysl Šůcha, Ph.D.
Supervisor's signature

doc. Ing. Tomáš Svoboda, Ph.D.
Head of department's signature

prof. Ing. Pavel Ripka, CSc.
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

Appendix B

Abbreviations

ACO	■	Ant Colony Optimization
CT	■	Cycle Time
FS	■	Flow Shop
GA	■	Genetic Algorithm
HFSs	■	Hybrid Flow Shops
LP	■	Linear Programming
LPT	■	Longest Processing Time
MILP	■	Mixed Integer Linear Programming
PS	■	Part Sequence
RAS	■	Robot Activity Sequence
RMS	■	Robot Move Sequence
SA	■	Simulated Annealing
TSP	■	Traveling Salesman Problem



Appendix C

Enclosed CD

- [energy_optimization]
Contains all source codes.
 - [benchmarks]
 - [build]
 - [include]
 - [src]
- [Thesis]
Contains the sources for this thesis.
- thesis.pdf
PDF version of this thesis.