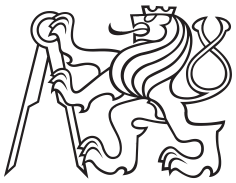
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Periodic Scheduling with Precedence Constraints

Karolína Rezková

Supervisor: prof. Dr. Ing. Zdeněk Hanzálek
May 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Rezková** Jméno: **Karolina** Osobní číslo: **420132**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Periodické rozvrhování s relacemi následností

Název diplomové práce anglicky:

Periodic Scheduling with Precedence Constraints

Pokyny pro vypracování:

1) nastudujte problematiku periodického rozvrhování s relacemi následností pro neharmonické i harmonické množiny period a výpočetních časů úloh
2) vyberte vhodnou metodu (např. meta□heuristiku) a navrhněte algoritmy pro řešení tohoto rozvrhovacího problému s minimalizací end□to□end latence jednotlivých řetězců úloh
3) implementujte několik verzí algoritmu
4) vyhodnoťte kvalitu řešení a dobu běhu jednotlivých algoritmů na benchmarkových instancích

Seznam doporučené literatury:

1. Minaeva, A. □ Hanzálek, Z.: Survey on Periodic Scheduling for Time□Triggered Hard Real□Time Systems, ACM Computing Surveys, article in press.
2. Jan Korst, Emile Aarts, and Jan Karel Lenstra. 1996. Scheduling periodic tasks. INFORMS journal on Computing 8, 4 (1996), 428–435.
3. Hladík, R.; Minaeva, A.; Hanzalek, Z.: On the Complexity of a Periodic Scheduling Problem with Precedence Relations, In: 14th Annual International Conference on Combinatorial Optimization and Applications, COCOA 2020, Dallas. Lecture Notes in Computer Science, vol 12577, Springer.

Jméno a pracoviště vedoucí(ho) diplomové práce:

prof. Dr. Ing. Zdeněk Hanzálek katedra řídicí techniky FEL

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

Datum zadání diplomové práce: **21.02.2021** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **19.02.2023**

prof. Dr. Ing. Zdeněk Hanzálek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. Datum převzetí zadání

Podpis studentky

Acknowledgements

Děkuji ČVUT, že mi je tak dobrou *alma mater*.

Declaration

Prohlašuji, že jsem předloženou práci vypracovala samostatně, a že jsem uvedla veškerou použitou literaturu.

V Praze, 10. May 2022

Abstract

In this thesis, we address periodic scheduling problem with precedence constraints, which is strongly $\mathcal{NP}$ hard. We show the connection of this problem to guillotine packing, which is a special case of 2D packing. We present algorithm, that searches for its feasible solution and prove its completeness. We also suggest a search algorithm for finding the optimal solution. Based on the tests, we evaluate the quality of solution.

Keywords: tabu search, local search, periodic scheduling, 2D bin packing, guillotine packing, periodic scheduling with precedence constraints, level algorithms

Supervisor: prof. Dr. Ing. Zdeněk Hanzálek

Abstrakt

V této práci se zabýváme problémem periodického rozvrhování s relacemi následnosti a jeho souvislostí se specifickým případem dvojdimenzionálního bin packing problému. Prezentujeme vlastní algoritmus, který hledá jeho řešení bez ohledu na hodnotu objektivní funkce a dokážeme jeho správnost. Navrhneme také metodu lokálního prohledávání pro nalezení jeho optimálního řešení. Na základě testů vyhodnotíme kvalitu řešení.

Klíčová slova: tabu search, lokální prohledávání, periodické rozvrhování, 2D bin packing, guillotine packing, periodické rozvrhování s relacemi následností, level algorithms

Překlad názvu: Periodické rozvrhování s relacemi následnosti

Contents

1 Introduction **1**

2 Problem Statement **3**

2.1 Specification of Tasks 3

2.2 Schedule 4

2.3 Optimization Criterion 6

3 Harmonic PSP with Precedence Constraints and 2D Bin Packing Problem **7**

3.1 Equivalence of Periodic Scheduling and 2D Bin Packing 7

3.2 Derivation of 2D Bin Packing Problem Associated with the Periodic Scheduling Problem 9

3.3 Other Properties of PSP and 2D Bin Packing Problem 11

4 Finding a Feasible Solution **15**

4.1 The Queue Approach 15

4.1.1 Initial Queue 16

4.1.2 Reconstruction of Schedule out of the Queue 16

4.1.3 Properties of the Approach . . 17

4.2 Level Algorithms 17

4.2.1 The Next-Fit Algorithm 17

4.2.2 The First-Fit Algorithm 18

4.2.3 The Initial Order of Rectangles 19

4.2.4 Algorithm Customization . . . 19

4.2.5 Implemented Versions of Level Algorithm 19

4.2.6 Properties of the Approach . . 20

4.3 Guillotine Packing Approach . . . 20

4.3.1 Basic Idea of the Algorithm for PSP with Harmonic Periods and Processing Times 21

4.3.2 The Associated 2D Bin Packing Problem as a Special Case of Guillotine Packing Problem 21

4.3.3 Search Algorithm 29

4.3.4 Properties of the Approach . . 35

5 Finding an Optimal Solution **39**

5.1 Overview of Existing Local Search Heuristic 39

5.2 Tabu Search 40

5.2.1 Neighbor Function 41

5.2.2 Tabu List 42

5.2.3 Picking the Best Neighbor . . 42

5.2.4 Terminal Condition 43

5.3 Tabu Search with the Queue Approach 44

5.3.1 Generating Neighboring Solutions 44

5.3.2 Tabu List Items 44

5.3.3 Tabu List Length 45

5.3.4 Aspiration Criterion 46

5.3.5 Terminal Conditions 46

5.3.6 Proposed Modification 46

5.4 Search Algorithm from [HMH20] with Aspiration Criterion 47

6 Experimental Part **49**

6.1 Generated Test Instances 49

6.2 Test Results 50

6.2.1 The Comparison of Algorithms for Feasibile solution 50

6.2.2 The Comparison of Algorithms for Optimality 51

7 Conclusions **53**

A Attachments **55**

B Nomenclature **57**

C Bibliography **61**

Figures

4.1 An example of a schedule that can not be represented by a queue. . . . 17

4.2 The examples of possible packings obtained by level algorithms on the same list of rectangles. 18

4.3 An example of bin packing that can be (left) and can not be (right) achieved by guillotine cutting. 20

4.4 Possible positions of the rectangle R_1 with the necessary cuts. 23

4.5 An example of a guillotine packing of an instance of 2D bin packing problem. 26

4.6 Cutting rectangle R_1 by one vertical guillotine cut. 26

4.7 Cutting rectangle R_2 by two vertical guillotine cuts. 27

4.8 Cutting each sub-bin by two horizontal guillotine cuts. 27

4.9 Cutting rectangle R_6 by one vertical guillotine cut. 28

4.10 Cutting each sub-bin by one horizontal guillotine cut. 28

4.11 Cutting rectangle R_9 by one vertical guillotine cut. 28

4.12 Each sub-bin contains either a rectangle with no unoccupied space or an unoccupied space only. 29

4.13 Transformation of a 2D bin packing solution. 33

4.14 An example of search tree of proposed DFS (part 1). 36

4.15 An example of search tree of proposed DFS (part 2). 37

5.1 The main loop of the algorithm [Glo90]. 41

5.2 Selection of the best neighbor [Glo90]. 43

Tables

6.1 Parameters of the generated test sets [HMH20] 50

6.2 Number of solved instances (up to 3 minutes) by test set and algorithm. L1 – Next-fit, rectangles ordered by width; L2 – Next-fit, rectangles ordered by height; L3 – Next-fit, rectangles ordered by width, rotated; L4 – Next-fit, rectangles ordered by height, rotated; L5 – First-fit, rectangles ordered by width; L6 – First-fit, rectangles ordered by height; L7 – First-fit, rectangles ordered by width, rotated; L8 – First-fit, rectangles ordered by height, rotated; Q – Reconstruction from queue of tasks; DFS – Depth-first search. . . 51

6.3 The comparison of an average value of degeneracy on instances solved by all compared algorithms SA – the original search algorithm from [HMH20]; SA with AC – search algorithm from [HMH20] with the usage of aspiration criterion; TS – tabu search with queue implementation. 51

Chapter 1

Introduction

Periodic Scheduling Problem (PSP) is an important class of various optimization problems with different purposes. In general, the problem deals with assigning resources to given set of jobs or tasks in time. It has numerous applications in different areas, such as information technology, transportation, production and many others.

There are multiple variations of this common problem given by a different nature of tasks, resources or some other additional conditions. Many of that may significantly affect the time complexity of the solution. The identical aspects of the problem may be addressed by a different nomenclature or notation in literature. In order to deal with this inconvenience, [GLLK79] comes up with a universal notation of this class of tasks. Its structured overview may be found in [MH21]. The problem studied in this work may therefore be addressed as $PD|T_i^{\text{harm}}, \text{jit}_i = 0, \text{chains}|\sum \delta$, but we will provide its detailed formal definition in the following chapter.

A simplified version of this problem with harmonic execution times and no objective function was proved by [KAL96] to be solvable in polynomial time, but [HMH20] shows that this problem is strongly $\mathcal{NP}$ hard. [MH21] mentions, that non-harmonic zero jitter case with constant processing times on a single machine with no objective function has also been proven to be strongly $\mathcal{NP}$ hard.

Chapter 2

Problem Statement

In this chapter, we provide a formal description of the scheduling problem considered in this work.

2.1 Specification of Tasks

We are given an unempty set of tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ and an unempty set of machines $\mathcal{M} = \{\mu_1, \ldots, \mu_{|\mathcal{M}|}\}$. Each task $\tau_i \in \mathcal{T}$ is further described by three mappings

$$\begin{aligned} m &: \mathcal{T} \mapsto \mathcal{M}, \\ p &: \mathcal{T} \mapsto \mathbb{Z}^+, \\ T &: \mathcal{T} \mapsto \mathbb{Z}^+. \end{aligned}$$

For any task $\tau_i \in \mathcal{T}$, mapping m specifies a machine $\mu_j \in \mathcal{M}$ on which τ_i is to be executed. We say that machine μ_j is dedicated to task τ_i or that task τ_i is assigned to machine μ_j.

Mapping p assigns each task τ_i the number of time units needed to its single execution $p(\tau_i)$ and is referred to as the processing time of task τ_i.

Mapping T gives the exact number of time units $T(\tau_i)$ after that the execution of task τ_i needs to be repeated. $T(\tau_i)$ is referred to as the period of task τ_i. Note that $T(\tau_i) \geq p(\tau_i)$ for any $\tau_i \in \mathcal{T}$.

Let $T(\mathcal{T}) = \{T(\tau_i) \mid \tau_i \in \mathcal{T}\}$ denote the set of all possible periods of tasks $\tau_i \in \mathcal{T}$. Set $T(\mathcal{T})$ is harmonic, formally

$$\forall i, i' \in \{1, \ldots, n\} : \ (T(\tau_i) \mid T(\tau_{i'})) \ \vee \ (T(\tau_{i'}) \mid T(\tau_i)).$$

Apart from that, we are also given a partition $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$ of the set of tasks $\mathcal{T}$ into pairwise disjoint unempty ordered sets $C_1, \ldots, C_{|\mathcal{C}|}$, such that for every $k \in \{1, \ldots, |\mathcal{C}|\}$, any two tasks $C_k^l, C_k^{(l')}$ from the ordered set $C_k = \left(C_k^1, \ldots, C_k^{|C_k|}\right)$ have the same period. Sets C_k are referred to as chains or precedence chains. The order of task C_k^l in chain C_k indicates the order in which the task C_k^l is executed with respect to other tasks of chain C_k. We say that task C_k^{l-1} is the predecessor of task C_k^l for any $k \in \{1, \ldots, |\mathcal{C}|\}$ and $l \in \{2, \ldots, |C_k|\}$. Similarly, the task C_k^{l+1} may be referred to as the successor of task C_k^l for any $k \in \{1, \ldots, |\mathcal{C}|\}$ and $l \in \{1, \ldots, (|C_k| - 1)\}$. Denote $T(C_k) = T(C_k^1) = \cdots = T(C_k^{|C_k|})$ the common period of tasks in chain C_k.

2.2 Schedule

In order to solve the given problem, we need to find mapping $s : \mathcal{T} \mapsto \mathbb{Z}_0^+$ assigning each task $\tau_i \in \mathcal{T}$ its first starting time $s(\tau_i)$ in time units, so that the following conditions are met. Note that time 0 denotes the beginning of execution of the set of tasks $\mathcal{T}$ and the starting time of any task is measured in time units from this point of time.

(C1) Task τ_i is executed by machine $m(\tau_i)$.

(C2) Each execution of task τ_i takes $p(\tau_i)$ time units.

(C3) Each execution of task τ_i is restarted after exactly $T(\tau_i)$ time units.

(C4) Once a machine starts to execute a task, its execution cannot be interrupted by execution of any other task.

(C5) There is at most one task being executed by each machine at any moment.

(C6) The execution of task C_k^l can only begin after the execution of task C_k^{l-1} has ended for any $k \in \{1, \ldots, |\mathcal{C}|\}$ and $l \in \{2, \ldots, |C_k|\}$.

Mapping s is referred to as a schedule.

Conditions (C1) –(C3) guarantee that s is a schedule for the given set of tasks $\mathcal{T}$ specified by mappings m, p and T respectively. Condition (C3) can also be called a zero jitter condition, since it requires a strictly periodic execution of tasks.

Condition (C4) guarantees that given scheduling task is non-preemptive. As a consequence of conditions (C2) –(C4) the time of z-th execution of task $\tau_i \in \mathcal{T}$ for any $z \in \mathbb{Z}^+$ is given by the interval

$$E(\tau_i, z) = \big[s(\tau_i) + (z-1) \cdot T(\tau_i),\ s(\tau_i) + p(\tau_i) + (z-1) \cdot T(\tau_i)\big).$$

Condition (C5) says that there are no collisions in schedule s. In other words, if we choose the time intervals of arbitrary executions of a pair of different tasks assigned to the same machine, its intersection should be empty, formally

$$\forall \tau_i, \tau_{i'} \in \mathcal{T},\ m(\tau_i) = m(\tau_{i'}),\ \tau_i \neq \tau_{i'},\ \forall z_i, z_{i'} \in \mathbb{Z}^+ : \\ E(\tau_i, z_i) \cap E(\tau_{i'}, z_{i'}) = \emptyset. \quad (2.1)$$

Statement (2.1) gives an infinite number of conditions for any pair of tasks assigned to the same machine. According to [KALW91] and due to harmonic property of $T(\mathcal{T})$, statement (2.1) is equivalent to

$$\forall \tau_i, \tau_{i'} \in \mathcal{T},\ m(\tau_i) = m(\tau_{i'}),\ \tau_i \neq \tau_{i'},\ T(\tau_i) \ \leq\ T(\tau_{i'}) : \\ p(\tau_i) \leq \big(s(\tau_{i'}) - s(\tau_i)\big) \pmod{T(\tau_i)} \ \leq\ T(\tau_i) - p(\tau_{i'}), \quad (2.2)$$

which gives only one condition for any pair of tasks assigned to the same machine.

Condition (C6) defines the precedence relations and can be formally denoted as

$$\forall k \in \{1, \ldots, |\mathcal{C}|\},\ |C_k| > 1,\ \forall l \in \{2, \ldots, |C_k|\} :\ s(C_k^l) + p(C_k^l) \leq s(C_k^{l+1}).$$

Schedule s fulfilling conditions (C1) –(C6) above is called a feasible schedule.

2.3 Optimization Criterion

For any feasible schedule s and a chain $C_k = \left(C_k^1, \ldots, C_k^{|C_k|}\right) \in \mathcal{C}$, we define the end-to-end latency of the chain C_k as

$$L(C_k) = \left(s(C_k^{|C_k|}) + p(C_k^{|C_k|})\right) - (C_k^1).$$

It can be understood as the amount of time in time units needed to complete all of tasks in the chain C_k according to the schedule s with respect to the given precedence constraints.

The degeneracy $\delta_s(C_k)$ of the chain C_k is defined as $\delta_s(C_k) = \left\lceil \frac{L(C_k)}{T(C_k)} \right\rceil - 1$. Similarly to end-to-end latency, the degeneracy of the chain C_k can also be understood as the amount of time needed to complete all tasks in the chain C_k, measured in number of periods of the chain C_k and lowered by 1. Furthermore, we define degeneracy $\delta(s)$ of the schedule s as

$$\delta(s) = \begin{cases} \sum\limits_{C_k \in \mathcal{C}} \delta_s(C_k), & \text{if the schedule } s \text{ is feasible,} \\ +\infty, & \text{otherwise.} \end{cases} \tag{2.3}$$

The goal of the periodic scheduling problem is to find the optimal schedule $s^* : \mathcal{T} \mapsto \mathbb{Z}_0^+$ assigning each task $\tau_i \in \mathcal{T}$ its starting time $s^*(\tau)$ so that the degeneracy of the schedule is minimal:

$$s^* = \underset{s:\mathcal{T}\mapsto\mathbb{Z}_0^+}{\arg\min}\, \delta(s).$$

[HMH20] shows, that this problem is strongly $\mathcal{NP}$ hard.

Chapter 3

Harmonic PSP with Precedence Constraints and 2D Bin Packing Problem

In this chapter, we discuss the connection between the original harmonic periodic scheduling problem and the 2D bin packing problem. We show how to derive the 2D bin packing problem associated with the original harmonic periodic scheduling problem and discuss some of its properties.

3.1 Equivalence of Periodic Scheduling and 2D Bin Packing

According to section 3 in [HH20], finding a feasible solution to the periodic scheduling problem with harmonic periods on a single machine without precedence constraints is equivalent to finding a solution to a specific case of 2D bin packing problem. Let us show that precedence constraints have no effect on the problem feasibility.

Property 3.1. *Suppose we have an instance of periodic scheduling problem with harmonic periods and a schedule s fulfilling conditions (C1) –(C5) . Schedule s can then be modified so that condition (C6) also holds.*

Proof. If schedule s fulfills the condition (C6) , there is no need for modification. For contradiction, suppose the condition (C6) does not hold. That means, there exists a chain having two consecutive tasks, such that the execution of the successor begins before the execution of the predecessor is finished.

That is,

$$\exists C_{k'} \in \mathcal{C}, \exists l' \in \{2, \ldots, |C_{k'}|\} : s(C_{k'}^{l'-1}) + p(C_{k'}^{l'-1}) > s(C_{k'}^{l'}).$$

Denote

$$z = \left\lceil \frac{s(C_{k'}^{l'-1}) + p(C_{k'}^{l'-1}) - s(C_{k'}^{l'})}{T(C_{k'})} \right\rceil. \tag{3.1}$$

Note that z is a positive integer, because both numerator and denominator are positive. Let us define a new schedule s', such that

$$s'(C_k^l) = \begin{cases} s(C_k^l) + z \cdot T(C_k), & \text{if } k = k' \text{ and } l = l' \\ s(C_k^l), & \text{otherwise,} \end{cases} \tag{3.2}$$

and show that $s'(C_{k'}^{l'-1}) + p(C_{k'}^{l'-1}) \geq s'(C_{k'}^{l'})$ and that s' still fulfills conditions (C1) –(C5) . Postponing execution of one particular task has clearly no effect on conditions (C1) –(C4) , but it might cause a collision of $C_{k'}^{l'}$ with some other task. From 2.2, we have

$$\begin{aligned} &\forall \tau_i \in \mathcal{T},\ m(\tau_i) = m(C_{k'}^{l'}),\ \tau_i \neq C_{k'}^{l'},\ T(\tau_i) \ \leq\ T(C_{k'}) : \\ &\qquad p(\tau_i) \leq \left(s(C_{k'}^{l'}) - s(\tau_i)\right) \pmod{T(\tau_i)} \ \leq\ T(\tau_i) - p(C_{k'}^{l'}), \end{aligned} \tag{3.3}$$

and

$$\begin{aligned} &\forall \tau_{i'} \in \mathcal{T},\ m(C_{k'}^{l'}) = m(\tau_{i'}),\ C_{k'}^{l'} \neq \tau_{i'},\ T(C_{k'}) \ \leq\ T(\tau_{i'}) : \\ &\qquad p(C_{k'}^{l'}) \leq \left(s(\tau_{i'}) - s(C_{k'}^{l'})\right) \pmod{T(C_{k'})} \ \leq\ T(C_{k'}) - p(\tau_{i'}). \end{aligned} \tag{3.4}$$

We need to show that the same inequalities hold also for s'. From 3.4, we get $\forall \tau_{i'} \in \mathcal{T},\ m(C_{k'}^{l'}) = m(\tau_{i'}),\ C_{k'}^{l'} \neq \tau_{i'},\ T(C_{k'}) \ \leq\ T(\tau_{i'})$:

$$s'(\tau_{i'}) - s'(C_{k'}^{l'}) \overset{(3.2)}{=} s(\tau_{i'}) - s(C_{k'}^{l'}) - z \cdot T(C_{k'}) \tag{3.5}$$

and therefore also

$$\begin{aligned} \left(s'(\tau_{i'}) - s'(C_{k'}^{l'})\right) \pmod{T(C_{k'})} &\overset{(3.5)}{=} \\ &\overset{(3.5)}{=} \left(s(\tau_{i'}) - s(C_{k'}^{l'}) - z \cdot T(C_{k'})\right) \pmod{T(C_{k'})} \\ &= \left(s(\tau_{i'}) - s(C_{k'}^{l'})\right) \pmod{T(C_{k'})}. \end{aligned}$$

which proves that 3.4 holds for s'. Similarly, from 3.3 and harmonic property of periods, we can easily obtain that

$$\exists z_1 \in \mathbb{N} : T(C_{k'}) = z_1 \cdot T(\tau_i). \tag{3.6}$$

Then $\forall \tau_i \in \mathcal{T},\ m(\tau_i) = m(C_{k'}^{l'}),\ \tau_i \neq C_{k'}^{l'},\ T(\tau_i) \ \leq\ T(C_{k'})$:

$$\begin{aligned} s'(C_{k'}^{l'}) - s'(\tau_i) &\overset{(3.2)}{=} s(C_{k'}^{l'}) + z \cdot T(C_{k'}) - s(\tau_i) \\ &\overset{(3.6)}{=} s(C_{k'}^{l'}) + z \cdot z_1 \cdot T(\tau_i) - s(\tau_i), \end{aligned} \tag{3.7}$$

and therefore also

$$\begin{aligned}(s'(C_{k'}^{l'}) - s'(\tau_i)) \pmod{T(\tau_i)} &\overset{(3.7)}{=} \\ &\overset{(3.7)}{=} (s(C_{k'}^{l'}) + z \cdot z_1 \cdot T(\tau_i) - s(\tau_i)) \pmod{T(\tau_i)} \\ &= (s(C_{k'}^{l'}) - s(\tau_i)) \pmod{T(\tau_i)},\end{aligned}$$

which proves that 3.3 holds for s'. By

$$\begin{aligned}s'(C_{k'}^{l'}) &\overset{(3.2)}{=} s(C_{k'}^{l'}) + z \cdot T(C_{k'}) \\ &\overset{(3.1)}{\geq} s(C_{k'}^{l'}) + \frac{s(C_{k'}^{l'-1}) + p(C_{k'}^{l'-1}) - s(C_{k'}^{l'})}{T(C_{k'})} \cdot T(C_{k'}) \\ &= s(C_{k'}^{l'}) + s(C_{k'}^{l'-1}) + p(C_{k'}^{l'-1}) - s(C_{k'}^{l'}) \\ &= s(C_{k'}^{l'-1}) + p(C_{k'}^{l'-1}) \\ &\overset{(3.2)}{=} = s'(C_{k'}^{l'-1}) + p(C_{k'}^{l'-1}),\end{aligned}$$

we prove that s' fulfills condition (C6) for consecutive tasks $C_{k'}^{l'-1}$ and $C_{k'}^{l'}$. Note that if we iterate through each chain form the first task to the last and perform schedule modification 3.2 in case condition (C6) is not met, we do not influence already checked relations, because postponing the successor has no effect on any preceding task. □

Property 3.1 shows that precedence relations do not have to be considered during construction of any feasible schedule and finding feasible schedule can therefore be reformulated equivalently as a set of 2D bin packing problems for each separate machine. But still, according to [MH21] or [HH20], the problem of finding feasible solution is also $\mathcal{NP}$ hard.

3.2 Derivation of 2D Bin Packing Problem Associated with the Periodic Scheduling Problem

As already mentioned above in the Problem Statement chapter, the periodic scheduling problem with harmonic periods is defined by a set of tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ and a set of machines $\mathcal{M} = \{\mu_1, \ldots, \mu_{|\mathcal{M}|}\}$. Each task $\tau_i \in \mathcal{T}$ is further specified by its dedicated machine $m(\tau_i)$, processing time $p(\tau_i)$, and period $T(\tau_i)$. Let $\mathcal{T}_{\mu_j} = \{\tau_i \in \mathcal{T} \mid m(\tau_i) = \mu_j\}$ denote the set of tasks assigned to machine μ_j and $T(\mathcal{T}_{\mu_j}) = \{T(\tau_i) \mid \tau_i \in \mathcal{T}_{\mu_j}\}$ the set of all possible periods of tasks in $\mathcal{T}$ assigned to machine μ_j. Since $T(\mathcal{T})$ is harmonic and $\mathcal{T}_{\mu_j} \subseteq \mathcal{T}$ for any $\mu_j \in \mathcal{M}$, $T(\mathcal{T}_{\mu_j})$ is also harmonic.

The 2D bin packing problem associated with the original periodic scheduling problem on machine $\mu_j \in \mathcal{M}$ is defined by a bin B_j of width $W(B_j)$ and height $H(B_j)$ and a set of rectangles $\mathcal{R}_j = \{R_\tau \mid \tau \in \mathcal{T}_{\mu_j}\}$, each rectangle of width $w(R_\tau)$ and height $h(R_\tau)$. According to [HH20], the dimensions of any rectangle $R_\tau \in \mathcal{R}_j$ would be

$$w(\tau) = p(\tau), \tag{3.8}$$

$$h(\tau) = \frac{\max\left(T(\mathcal{T}_{\mu_j})\right)}{T(\tau)}. \tag{3.9}$$

The dimensions of the bin B_j would be

$$W(B_j) = \min\left(T(\mathcal{T}_{\mu_j})\right), \tag{3.10}$$

$$H(B_j) = \frac{\max\left(T(\mathcal{T}_{\mu_j})\right)}{\min\left(T(\mathcal{T}_{\mu_j})\right)}. \tag{3.11}$$

Note that values $H(B_j), W(B_j), w(\tau_i)$ and $h(\tau_i)$ are positive integers for any $j \in \{1, \ldots, |\mathcal{M}|\}$ and any $\tau_i \in \mathcal{T}$ due to the problem definition and the fact that $T(\mathcal{T}_{\mu_j})$ is harmonic for any $\mu_j \in \mathcal{M}$. Also, $\max\left(T(\mathcal{T}_{\mu_j})\right)$ is referred to as hyperperiod of machine μ_j.

The rectangles $R_\tau \in \mathcal{R}_j$ are to be positioned into the bin B_j. Let (x_τ, y_τ) denote the coordinates of the lower-left vertex of rectangle R_τ in bin B_j for $R_\tau \in \mathcal{R}_j$. Note that the lower-left vertex of bin B_j has coordinates $(0, 0)$ and the upper-right vertex has coordinates $(W(B_j), H(B_j))$, so for any $\tau \in \mathcal{T}_{\mu_j}$, we have $0 \leq x_\tau < x_\tau + w(\tau) \leq W(B_j)$ and $0 \leq y_\tau < y_\tau + h(\tau) \leq H(B_j)$. For any rectangle R_τ that is placed in bin B_j, it must also hold the following:

$$y_\tau \equiv 0 \pmod{h(\tau)}. \tag{3.12}$$

The goal is to position rectangles from set $\mathcal{R}_j$ into the bin B_j so that condition (3.12) holds and no collision occurs.

For better clarity, the indices and arguments referring to the tasks and machines of the original periodic scheduling problem will be omitted in the following text. So the definition of the 2D bin packing problem on one machine will be newly denoted as packing the set of n rectangles $\mathcal{R} = \{R_i \mid i = 1, \ldots, n\}$ of width w_i and height h_i for $i \in \{1, \ldots, n\}$ into bin B of width W and height H so that no collision occurs. The coordinates of the lower-left vertex of rectangle R_i in the bigger rectangle B will be denoted as (x_i, y_i) for any $i \in \{1, \ldots, n\}$ and

$$\forall i \in \{1, \ldots, n\} : \; y_i \equiv 0 \pmod{h_i}. \tag{3.13}$$

According to [HH20], collision between rectangles R_i and $R_{i'}$ such that $h_{i'} \leq h_i$ occurs if and only if $(y_i \leq y_{i'} < y_i + h_i) \wedge (x_i < x_{i'} + w_{i'}) \wedge (x_{i'} < x_i + w_i)$.

From that we can derive that there is no collision in 2D bin packing if and only if

$$\forall R_i, R_{i'} \in \mathcal{R},\ R_i \neq R_{i'},\ h_{i'} \leq h_i :$$
$$(y_i > y_{i'})\ \vee\ (y_{i'} \geq y_i + h_i)\ \vee\ (x_i \geq x_{i'} + w_{i'})\ \vee\ (x_{i'} \geq x_i + w_i). \quad (3.14)$$

3.3 Other Properties of PSP and 2D Bin Packing Problem

Property 3.2. *Suppose we have an instance of a 2D bin packing problem. If the sequence of rectangles $\{R_1, \ldots, R_n\}$ is sorted by its height in non-increasing order, it is also sorted in non-decreasing order by the periods of tasks associated with those rectangles.*

Proof. The sequence of rectangles $\{R_1, \ldots, R_n\}$ is sorted by its height in non-decreasing order, so

$$\forall i, i' \in \{1, \ldots, n\},\ i < i' :\ h_i \geq h_{i'}.$$

Using the definition (3.9), we get

$$\frac{\max_{\tau \in \mathcal{T}} T(\tau)}{T(\tau_i)} \ \geq\ \frac{\max_{\tau \in \mathcal{T}} T(\tau)}{T(\tau_{i'})}.$$

By the definition of processing time, it holds that $\forall i'' \in \{1, \ldots, n\} :\ T(\tau_{i''}) > 0$, so we can multiply the inequality by $\left(T(\tau_i) \cdot T(\tau_{i'})\right)$:

$$T(\tau_{i'}) \cdot \max_{\tau \in \mathcal{T}} T(\tau)\ \geq\ T(\tau_i) \cdot \max_{\tau \in \mathcal{T}} T(\tau),$$

and finally, since the maximum of finite set of positive integers is a positive integer, we can also divide the inequality by $\left(\max_{\tau \in \mathcal{T}} T(\tau)\right)$ getting

$$T(\tau_{i'})\ \geq\ T(\tau_i).$$

□

Property 3.2 also implies

$$T(\tau_1) = \min_{\tau \in \mathcal{T}} T(\tau) \quad (3.15)$$

for a sequence of rectangles sorted by its height in non-increasing order.

Property 3.3. *Suppose we have an instance of periodic scheduling problem and the associated 2D bin packing problem. If the set of periods of all the tasks* $\{T(\tau) \mid \tau \in \mathcal{T}\}$ *is harmonic, then the set of heights* $\{h_i \mid i = 1, \ldots, n\}$ *of rectangles* $\mathcal{R} = \{R_1, \ldots, R_n\}$ *is also harmonic.*

Proof. Suppose the set of periods of the tasks $\{T(\tau) \mid \tau \in \mathcal{T}\}$ is harmonic. That means,

$$\forall i, i' \in \{1, \ldots, n\}\ \exists z \in \mathbb{Z}^+ :\ \big(T(\tau_i) = z \cdot T(\tau_{i'})\big)\ \vee\ \big(T(\tau_{i'}) = z \cdot T(\tau_i)\big).$$

Using (3.9), we get

$$\left(\frac{\max_{\tau\in\mathcal{T}} T(\tau)}{h_i} = z \cdot \frac{\max_{\tau\in\mathcal{T}} T(\tau)}{h_{i'}}\right)\ \vee\ \left(\frac{\max_{\tau\in\mathcal{T}} T(\tau)}{h_{i'}} = z \cdot \frac{\max_{\tau\in\mathcal{T}} T(\tau)}{h_i}\right).$$

We can multiply both equations by positive integers h_i and $h_{i'}$ and divide them by positive integer $\big(\max_{\tau\in\mathcal{T}} T(\tau)\big)$ resulting in

$$\forall i, i' \in \{1, \ldots, n\}\ \exists z \in \mathbb{Z}^+ :\ \big(h_{i'} = z \cdot h_i\big)\ \vee\ \big(h_i = z \cdot h_{i'}\big),$$

which means that the set of heights $\{h_i \mid i = 1, \ldots, n\}$ of rectangles in $\mathcal{R}$ is harmonic. □

Property 3.4. *Suppose we have an instance of periodic scheduling problem with harmonic periods and the associated 2D bin packing problem defined as described in subsection 3.1. Then at least one rectangle of the set of rectangles* $\mathcal{R} = \{R_1, \ldots, R_n\}$ *has the same height as bin* B.

Proof. Without loss of generality, we can assume that the sequence of rectangles $\{R_1, \ldots, R_n\}$ is sorted in non-increasing order by heights. Using the previous results and the definitions, we can see that the height h_1 of the first rectangle R_1 is equal to the height H of the bin B, formally

$$h_1 \overset{(3.9)}{=} \frac{\max_{\tau\in\mathcal{T}} T(\tau)}{T(\tau_1)} \overset{(3.15)}{=} \frac{\max_{\tau\in\mathcal{T}} T(\tau)}{\min_{\tau\in\mathcal{T}} T(\tau)} \overset{(3.10)}{=} \frac{\max_{\tau\in\mathcal{T}} T(\tau)}{L} \overset{(3.11)}{=} H.$$

We have shown that the rectangle R_1 has the same height as the bin B, so the statement holds. □

Property 3.4 also implies that the second coordinate of this rectangle of height H is 0 because otherwise, the rectangle would not fit into the bin.

Property 3.5 (Necessary condition for feasibility 1)**.** *Suppose we have a feasible instance of periodic scheduling problem. Then*

$$\forall \mu_j \in \mathcal{M} :\ \sum_{\substack{\tau_i\in\mathcal{T} \\ m(\tau_i)=\mu_j}} \frac{p(\tau_i)}{T(\tau_i)} \leq 1. \tag{3.16}$$

Proof. According to the definitions in chapter 2, the execution of task τ_i takes $p(\tau_i)$ time units and is restarted every $T(\tau_i)$ time units. Since τ_i is assigned to μ_j, its execution takes up $\frac{p(\tau_i)}{T(\tau_i)}$ of the total time of machine μ_j since its first execution. If we sum up proportional expressions of time consumption for all the tasks assigned to the machine μ_j, we get exactly

$$\forall \mu_j \in \mathcal{M} : \sum_{\substack{\tau_i \in \mathcal{T} \\ m(\tau_i)=\mu_j}} \frac{p(\tau_i)}{T(\tau_i)},$$

which therefore can not exceed 1, because that would cause a contradiction to condition (C5) . □

The sum from Property 3.5 is sometimes referred to as utilization of machine μ_j.

Property 3.6 (Necessary condition for feasibility 2)**.** *Suppose we have a feasible instance of periodic scheduling problem. Then*

$$\forall \tau_i, \tau_{i'} \in \mathcal{T},\ m(\tau_i) = m(\tau_{i'}) :\ T(\tau_i) \geq p(\tau_{i'}).$$

Consequently,

$$\forall \mu_j \in \mathcal{M} : \min_{\substack{\tau_i \in \mathcal{T} \\ m(\tau_i)=\mu_j}} T(\tau_i) \geq \max_{\substack{\tau_i \in \mathcal{T} \\ m(\tau_i)=\mu_j}} p(\tau_{i'}). \tag{3.17}$$

Proof. For contradiction, suppose

$$\exists \tau_i, \tau_{i'} \in \mathcal{T},\ m(\tau_i) = m(\tau_{i'}) :\ T(\tau_i) < p(\tau_{i'}). \tag{3.18}$$

From the definition in chapter 2, we can derive that execution of task τ_i takes exactly $p(\tau_i)$ time units of any time interval of length $T(\tau_i)$ since the first occurrence of task τ_i. Any execution of task $\tau_{i'}$ takes $p(\tau_{i'})$ time units and can not be interrupted. Since $T(\tau_i) < p(\tau_{i'})$ according to 3.18, there exists a time interval I of length $T(\tau_i)$ such that $I \subsetneq E(\tau_{i'}, z)$ for some positive integer z. During the whole interval I, the machine $m(\tau_{i'})$ executes task $\tau_{i'}$, but since interval I is of length $T(\tau_i)$, the same machine $m(\tau_i) = m(\tau_{i'})$ should also execute task τ_i for $p(\tau_i)$ time units during I. In case $\tau_{i'} \neq \tau_i$, we get a contradiction to condition (C5) . Otherwise, we get $\tau_{i'} = \tau_i$. From 3.18, we get $T(\tau_i) < p(\tau_i)$, which is also a contradiction – no task can have period shorter than execution time. We have therefore proven that the proposed formula holds.

For any $\mu_j \in \mathcal{M}$, $\mathcal{T}_{\mu_j}$ is a finite set and therefore, we can pick $\hat{\tau}_i$ so that $T(\hat{\tau}_i) = \min\{T(\tau) | \tau \in \mathcal{T}_{\mu_j}\}$ and $\hat{\tau}_{i'}$ so that $p(\hat{\tau}_{i'}) = \min\{p(\tau) | \tau \in \mathcal{T}_{\mu_j}\}$ and get 3.17. □

Note that the equality in 3.17 holds only if $\mathcal{T}_{\mu_j} = \{\hat{\tau}\}$ and $T(\hat{\tau}) = p(\hat{\tau})$.

Chapter 4

Finding a Feasible Solution

There are several approaches proposed in literature to solve problems similar to the one considered in this work. [HMH20] proposes a local search algorithm, that considers the exact same problem as was introduced in chapter 2. [Nv09] mentions several heuristics for 2D oriented offline strip packing problem for rectangular items, which is similar to the 2D bin packing problem associated with our original PSP. [KAL96] proposes an approach to solving the PSP if both periods and processing times of tasks are powers of 2. [KAL96] also claims that the proposed approach can be generalized for solving the PSP if periods and processing times are harmonic.

In this chapter we describe the mentioned approaches and some of their properties. We also look for modifications that would allow us to use similar approach for our case of PSP.

4.1 The Queue Approach

We primarily consider the approach from [HMH20], because there is no modification needed. The goal of the proposed local search heuristics is to generate the optimal schedule. The process is iterative. At first, an initial schedule is created. The current schedule is then modified in order to find a schedule with a lower degeneracy.

4.1.1 Initial Queue

Schedule is represented by a queue given by an ordered list of tasks of the instance. The initial queue meets the following conditions

- the task with the shorter period is queued earlier than the task with the longer period,
- in case the periods are equal, the task with the longer processing time is queued earlier than the task with the shorter processing time.

4.1.2 Reconstruction of Schedule out of the Queue

In order to reconstruct the schedule, we iterate through the queue and schedule the tasks in the given order. Once the task is scheduled, its position can not be changed. Each task is scheduled at the earliest possible time on the dedicated machine so that

- its first execution does not start earlier than the first execution of any already scheduled task on the same machine finishes,
- its execution does not collide with the execution of any other task on the same machine (periodic execution of tasks is considered),
- its first execution does not start earlier than the first execution of its predecessor finishes, if the predecessor of the task has already been scheduled.

Note that this reconstruction does not always find a feasible solution (in such case, the queue is slightly modified). It also allows relaxation of some precedence constraints. Specifically, if there are two consecutive tasks queued in reverse order, it may happen that the precedence constraint for these tasks will not be fulfilled. To fix this, we can delay the execution of the successor task as proposed in 3.1.

4.1.3 Properties of the Approach

The queue representation provides a simple way of modifying the schedule by rearranging the queue. According to [HMH20], the reconstruction function should guarantee that a minor adjustment of the queue of tasks causes only a small change in the respective schedule, which is the advantage of this approach. On the other hand, there is no bijection between the state space of all possible schedules and the queues for a given problem instance. In Figure 4.1, we provide an example of a simple schedule that can not be represented by a queue in the proposed way. Also a small adjustment of

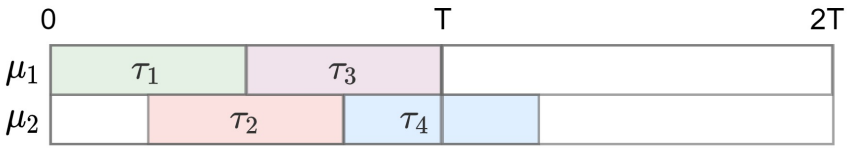


Figure 4.1: An example of a schedule that can not be represented by a queue.

the queue may result in no change in the schedule (for example permutation of tasks consecutive in the queue that are pairwisely dedicated to different machines and are not consecutive in any chain). As a result, this approach may not be able to find a feasible solution for every feasible instance.

4.2 Level Algorithms

Several methods that can be slightly modified and used to find a solution of the associated 2D bin packing problem have been mentioned in [Nv09]. The purpose of original algorithms is to solve the offline strip packing problem. In the strip packing problem, the strip (corresponding to the initial bin in our concept) has only one fixed dimension. The other dimension (the height) is not bounded. The general idea of the approaches is to sort the rectangles and position them to horizontal planes of the strip. There are several different variants of the algorithm, but we will focus on two of its basic variants.

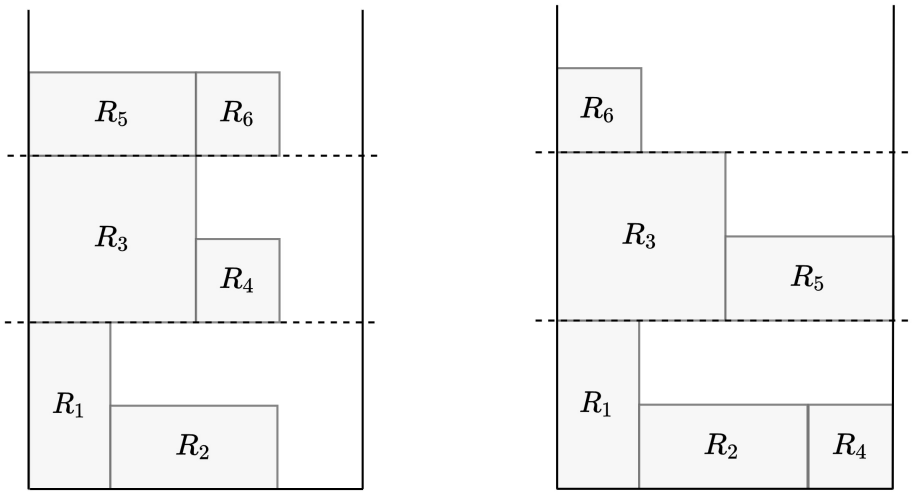
4.2.1 The Next-Fit Algorithm

The next-fit algorithm processes one rectangle at the time. The first level of the strip corresponds to its lower side. The first rectangle is positioned

to the left side of the first level – to the lower left vertex of the strip. Every other rectangle is then positioned to the current level right next to the right side of the previously positioned rectangle. If this is not possible due to the insufficient width of the unoccupied space on a current level, a new level is created in the height given by the highest rectangle and the currently processed rectangle is placed to the left side of this level. The lower levels are not revisited any more. The possible packing obtained by this algorithm is depicted in Figure 4.2a.

4.2.2 The First-Fit Algorithm

The first-fit algorithm works very similarly to the next-fit algorithm. The main difference is that it allows to revisit the lower levels with unoccupied space. In general, the currently processed rectangle is placed leftmost possible on the lowest level into which it fits. The possible packing obtained by this algorithm is depicted in Figure 4.2b.



(a) : Next-Fit packing **(b) :** First-Fit packing

Figure 4.2: The examples of possible packings obtained by level algorithms on the same list of rectangles.

4.2.3 The Initial Order of Rectangles

As it was already mentioned at the beginning of this section, the initial order of rectangles affects the results. We will further explore the most intuitive variants, which are decreasing height and decreasing width ordering.

4.2.4 Algorithm Customization

Since the problem addressed in this work differs from the problem for which the algorithms were originally designed, it is necessary to adapt the algorithm.

The strip unbounded in one dimension is replaced by a bin with with specified finite dimensions. This fact allows us to create levels in the bin in two different dimensions (or equivalently rotate the entire instance 90 degrees).

Apart from the no collision constraint, the 2D oriented offline strip packing problem has no other requirements on the position of the rectangle. However, the 2D bin packing problem associated to our original PSP requires meeting the condition 3.13. Whenever we find a possible position for rectangle according to the original algorithm, we consult it with the condition 3.13 and shift the rectangle up (or to the right in the rotated case).

4.2.5 Implemented Versions of Level Algorithm

Given the options we propose above, we get eight possible versions of the algorithm. The differences are in the basic strategy of positioning the rectangles (either next-fit or first-fit), the order of rectangles (descending either in height or in width) and the instance rotation. All eight options have been implemented and tested. Note that in the rotated version of any strategy with rectangles sorted by height in the descending order, there is no need to consult the condition 3.13. It is automatically met due to the harmonic property of heights.

4.2.6 Properties of the Approach

The advantage of this approach is its simplicity. The methods can be easily implemented. However, the algorithms are also not complete and therefore do not provide a feasible solution for every feasible instance.

4.3 Guillotine Packing Approach

Section 2 in [KAL96] introduces an algorithm solving a special case of the periodic scheduling problem where both periods of the given tasks and their execution times are powers of 2. It is also mentioned that this algorithm can be generalized for the case of the periodic scheduling problem where both periods of the given tasks and their execution times are harmonic. According to [KAL96], the time complexity of this special case with harmonic periods and execution times is $\mathcal{O}(nl)$, where $n = |\mathcal{T}|$, which is the number of the tasks, and $l = \left|\{T(\tau) \mid \tau \in \mathcal{T}\}\right|$, which is the number of different periods of the tasks. Note that the addressed problem is a special case of 2D bin packing considered in this work.

In this section, we state that the solution of problem considered in this work is a special case of guillotine packing and propose a complete algorithm based on [KAL96]. The guillotine cut is formally described in [FMT16]. The basic idea of this principle is to divide the bin into two sub-bins by an edge-to-edge cut parallel to one of the sides of the divided bin. Iteratively, any of the sub-bins created from the original bin may be also divided by another guillotine cut. An example of bin packing that can and can not be achieved by guillotine cuts is provided in Figure 4.3.

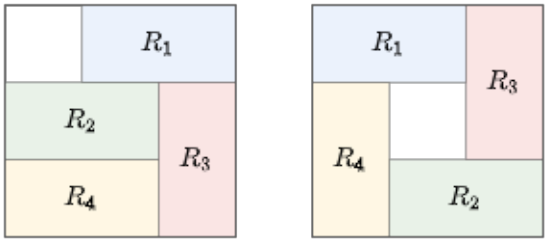


Figure 4.3: An example of bin packing that can be (left) and can not be (right) achieved by guillotine cutting.

4.3.1 Basic Idea of the Algorithm for PSP with Harmonic Periods and Processing Times

If we consider the associated 2D bin packing problem for PSP with harmonic periods and processing times (which is possible since it is a special case of PSP considered in this work), the algorithm proposed in [KAL96] can then be demonstrated as applying the guillotine cuts on the bin in order to divide it into sub-bins and to place the rectangles into those sub-bins so that each sub-bin contains either a single rectangle with no unoccupied space or an unoccupied space only.

4.3.2 The Associated 2D Bin Packing Problem as a Special Case of Guillotine Packing Problem

The case studied in this work is more general than the special cases considered in [KAL96], because the processing times of the tasks are not harmonic. However even this more general problem is a specific case of guillotine packing problem. Let us show that if an instance of periodic scheduling problem with harmonic periods considered in this work is feasible, the solution can be obtained by guillotine cutting applied on the set of associated 2D bin packing problems for single machines.

Lemma 4.1. *If an instance of periodic scheduling problem with harmonic periods on a single machine is feasible, the solution can be obtained as the guillotine packing of associated 2D bin packing problem.*

Proof. Let us consider an instance of a periodic scheduling problem with harmonic periods on one machine. That is a set of tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, with processing time $p(\tau_i)$ and period $T(\tau_i)$ for any $\tau_i \in \mathcal{T}$ and a set of machines containing only a single machine, $\mathcal{M} = \{\mu_1\}$. All the tasks $\tau_i \in \mathcal{T}$ are assigned to machine μ_1.

The 2D bin packing problem associated with the mentioned periodic scheduling problem is given by a bin B of width $W = \min\{T(\tau_i) \mid \tau_i \in \mathcal{T}\}$ and height $H = \frac{\max\{T(\tau_i)|\tau_i \in \mathcal{T}\}}{\min\{T(\tau_i)|\tau_i \in \mathcal{T}\}}$; and a set of n rectangles $\mathcal{R} = \{R_1, \ldots, R_n\}$, each rectangle R_i for $i \in \{1, \ldots, n\}$ having width of $w_i = p(\tau_i)$ and height of $h_i = \frac{\max\{T(\tau)|\tau \in \mathcal{T}\}}{T(\tau_i)}$. Let us denote $\mathcal{H} = \{h_1, \ldots, h_n\}$ the set of all possible heights of rectangles $R_i \in \mathcal{R}$. By Property 3.3, we know that set of heights $\mathcal{H}$ is harmonic. According [HH20], a feasible periodic schedule on one machine defines a feasible 2D packing such that $y_i \equiv 0 \pmod{h_i}$ for any $i \in \{1, \ldots, n\}$. Our periodic scheduling problem is feasible by assumption, so there are known

coordinates for lower left vertex of each rectangle R_i denoted (x_i, y_i) defining a packing with no collision, such that $y_i \equiv 0 \pmod{h_i}$ for any $i \in \{1, \ldots, n\}$.

Without loss of generality, assume the sequence of rectangles $\{R_1, \ldots, R_n\}$ is sorted by its height in non-ascending order. By Property 3.2, we also know the sequence is sorted in non-descending order by the periods of tasks associated with corresponding rectangles. We will construct the sequence of guillotine cuts that will separate the rectangles one by one in order of sequence given above using induction on the cardinality of the set of the heights of the rectangles. Note that the number of heights is the same as the number of possible periods of the corresponding tasks.

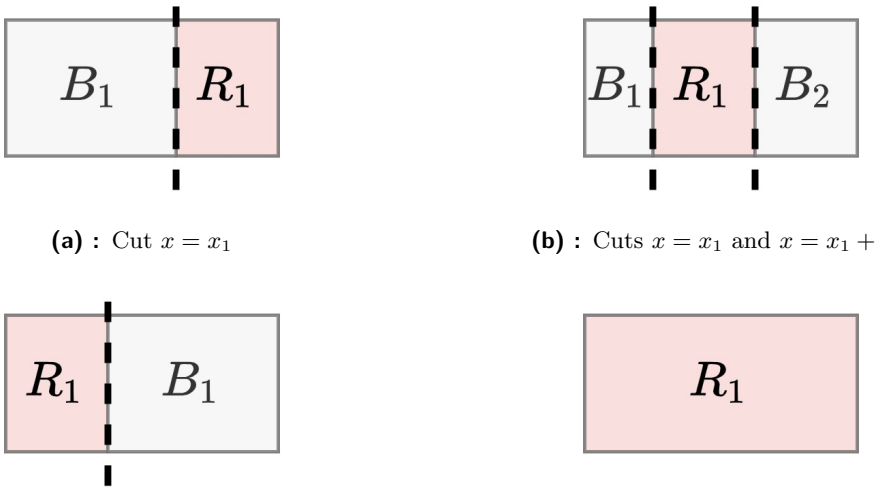
(Trivial case.) If $|\mathcal{H}| = 0$, the set of the heights of the rectangles is empty, therefore also the set of rectangles $\mathcal{R}$ is empty and the set of tasks $\mathcal{T}$ is empty, which contradicts the Problem Statement. Accordingly, such a case is not possible.

(Base step.) If $|\mathcal{H}| = 1$, all the rectangles are of the same height. According to Property 3.4, there is at least one rectangle of height H. Consequently, all the rectangles are of height H. We show that this case can be solved by a set of vertical guillotine cuts.

By at most two vertical guillotine cuts of bin B made in coordinates of vertical sides of rectangle R_1 (that is $x = x_1$ and $x = x_1 + w_1$), we divide rectangle R_1 from the rest of bin B. Notice that in case the side of the rectangle coincides with any side of the bin, the corresponding cut is skipped. All the possible positions of the first rectangle and the necessary cuts are depicted in Figure 4.4. Since the packing is collision-free, the proposed cuts do not interfere with other rectangles.

As a result, we are left with up to two smaller sub-bins of height H. Each of the rectangles $\{R_2, \ldots, R_n\}$ (if there are any) is positioned in one of the new sub-bins. Although each of the sub-bins can be considered separately (because the cuts made in one sub-bin do not affect the second sub-bin), they can not be treated as a separate instance, because they might not contain a rectangle of height H The coordinates of the lower-left vertex of rectangles $\{R_2, \ldots, R_n\}$ in the original bin B can be recomputed with respect to the newly created sub-bins as follows.

If $x_i < x_1$ for any $i \in \{2, \ldots, n\}$, the coordinates in the new sub-bin stay the same, because the lower left vertex of the new sub-bin corresponds to the lower left vertex of the original bin B. This is the case of sub-bin B_1 in



(a) : Cut $x = x_1$

(b) : Cuts $x = x_1$ and $x = x_1 + w_1$

(c) : Cut $x = x_1 + w_1$

(d) : No cuts

Figure 4.4: Possible positions of the rectangle R_1 with the necessary cuts.

Figure 4.4a and the sub-bin B_1 in Figure 4.4b.

If $x_i \geq x_1 + w_1$ for any $i \in \{2, \ldots, n\}$, the new coordinates $(\tilde{x}_i, \tilde{y}_i)$ in the new sub-bin can be recalculated as $(\tilde{x}_i, \tilde{y}_i) = (x_i - (x_1 + w_1), y_i)$, because the lower left vertex of the new sub-bin corresponds to the position $(x_1 + w_1, y_1)$ in the original bin B, which is as a consequence of Property 3.4 equal to $(x_1 + w_1, 0)$ in bin B. This is the case of sub-bin B_2 in Figure 4.4b and the sub-bin B_1 in Figure 4.4c.

Clearly, the case of $x_1 \leq x_i < x_1 + w_1$ for any $i \in \{2, \ldots, n\}$ causes a collision of rectangles R_1 and R_i and therefore is not possible. It contradicts the feasibility of given instance.

The process that was described for cutting the rectangle R_1 from the bin B can be repeated to cut any of the other rectangles R_i of the same height as R_1 from the relevant sub-bin in which R_i is located. So the case for $|\mathcal{H}| = 1$ can be solved using only vertical guillotine cuts.

(Induction step.) To prove the induction step, suppose it is possible to construct the sequence of guillotine cuts for all the instances having $|\mathcal{H}| = m$. Note that each instance has at least one rectangle having the height of the bin as was shown in Property 3.4. Let's show that we can also solve any instance having $|\mathcal{H}| = m + 1$.

Consider an instance having $|\mathcal{H}| = m + 1$. Suppose there are exactly k rectangles having the same height as the first rectangle and the bin B. According to Property 3.4, $k \geq 1$. We have

$$H = h_1 = h_2 = \cdots = h_k > h_{k+1}.$$

Applying the same reasoning as was used in the base step of this proof, we can cut the first k rectangles using only vertical guillotine cuts. Note that

$$|\{h_{k+1}, \ldots, h_n\}| = m,$$

but rectangle R_{k+1} is of height $h_{k+1} < h_k = H$, so we can not use the induction hypothesis yet.

Cutting off the first k rectangles leaves us with a sequence of $(n - k)$ rectangles to be cut and a sequence of sub-bins of height H. Let $z \in \mathbb{Z}^+$ denote the number of sub-bins and $\{B_1, \ldots, B_z\}$ the actual sequence of the sub-bins.

We already know, that $h_k > h_{k+1}$. Using Property 3.3, we get

$$\exists q \in \mathbb{Z}^+,\ q > 1 :\ H = h_k = q \cdot h_{k+1}.$$

At this point, we perform $(q - 1)$ horizontal cuts on all of the sub-bins in heights $y = i \cdot h_{k+1}$ for $i = 1, \ldots, (q - 1)$. That results in $(q \cdot z)$ new sub-bins of height h_{k+1}, which allows us to use the induction hypothesis.

We show by contradiction that none of the performed vertical cuts interferes with any of the rectangles $R_{k+1}, \ldots, R_n$. For contradiction, suppose that exists $i' \in \{1, \ldots, (q - 1)\}$ such that vertical cut $y = i' \cdot h_{k+1}$ interferes with a rectangle $R' \in \{R_{k+1}, \ldots, R_n\}$.

Denote h' the height of rectangle R'. Since it has not been cut off yet, it is positioned in one of the sub-bins of set $\{B_1, \ldots, B_z\}$, denote this sub-bin B'. By performing only vertical guillotine cuts, the coordinate y' of its lower side in bin B could not be changed, so it is also y' in sub-bin B', the coordinate of its upper side is $y' + h'$.

The fact that vertical cut $y = h_{k+1} \cdot i'$ interferes with a rectangle R' can be formally described as

$$y' \ < \ h_{k+1} \cdot i' \ < \ y' + h'. \tag{4.1}$$

Since the sequence of rectangles was sorted in non-ascending order by their heights, we can claim that $h' \leq h_{k+1}$. Since the set of periods of tasks in the

original problem was harmonic, the set of heights of the rectangles is also harmonic according to Property 3.3, so we can claim that

$$\exists c \in \mathbb{Z}^+ : h_{k+1} = c \cdot h'.$$

Using the latter on (4.1), we get

$$\exists c \in \mathbb{Z}^+ : \; y' < c \cdot h' \cdot i' < y' + h'. \tag{4.2}$$

Further, we can use the condition (3.13) given by definition, saying that

$$\exists d \in \mathbb{Z} : \; y' = d \cdot h'$$

and transform the inequality (4.2) such that

$$\exists c \in \mathbb{Z}^+ \; \exists d \in \mathbb{Z} : \; d \cdot h' < c \cdot h' \cdot i' < d \cdot h' + h'. \tag{4.3}$$

By dividing the inequality by positive integer h' and subtracting d, we get

$$\exists c \in \mathbb{Z}^+ \; \exists d \in \mathbb{Z} : \; 0 < c \cdot i' - d < 1. \tag{4.4}$$

Since $c \in \mathbb{Z}^+$, $d \in \mathbb{Z}$ and $i' \in \{1, \ldots, (q-1)\}$, the expression $c \cdot i' - d \in \mathbb{Z}$. That means (4.4) is a contradiction, because there is no integer greater than 0 and smaller than 1 at the same time.

We have shown by contradiction, that the proposed horizontal guillotine cuts on all the sub-bins $\{B_1, \ldots, B_z\}$ in heights $y = h_{k+1} \cdot i$ for $i = 1, \ldots, (q-1)$ do not interfere with any of the rectangles $R_{k+1}, \ldots, R_n$. Performing the cuts results in dividing each sub-bin B_i for $i \in \{1, \ldots, z\}$ into q new sub-bins of height h_{k+1}. At this point, we can use the induction hypothesis. Note that the coordinates of the lower left vertex of rectangles in the new sub-bins need to be recomputed with respect to the new related sub-bins by a formula $(\tilde{x}, \tilde{y}) = (x, (y \mod h_{k+1}))$. □

We have shown that any solution of the 2D bin packing considered in this work is a guillotine packing. The proof of Lemma 4.1 also shows how to use guillotine cutting to separate the rectangles of any solution of any instance of the 2D bin packing problem associated with the periodic scheduling problem with harmonic periods. This procedure will be demonstrated in the following example of packing depicted in Figure 4.5. Also, the notation of coordinates of the lower-left vertex of a rectangle R with respect to a new sub-bin denoted as $(\tilde{x}, \tilde{y})$ will be only used when explaining its computation. At any point, notation (x, y) will be used for the coordinates of the lower-left vertex of rectangle R in the sub-bin B and will be referred to as the coordinates of R in B.

Example 1. Figure 4.5 depicts a solution of a feasible instance of 2D bin packing problem associated with a PSP with harmonic periods on a single

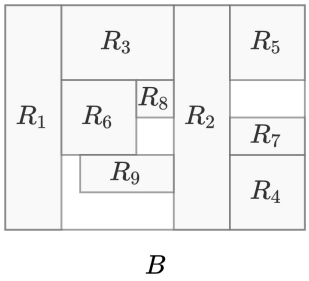


Figure 4.5: An example of a guillotine packing of an instance of 2D bin packing problem.

machine. We will show the sequence of guillotine cuts that divides bin B into sub-bins containing either a single rectangle or an unoccupied space.

Bin B of height H and width L contains 9 rectangles. The sequence of rectangles $\{R_1, \ldots, R_9\}$ is already ordered by its heights in non-ascending order, so we can process the rectangles in this specified order.

There are 2 rectangles of the same height as bin B marked in Figure 4.6 in red color. The first coordinate of rectangle R_1 in bin B is 0. The left

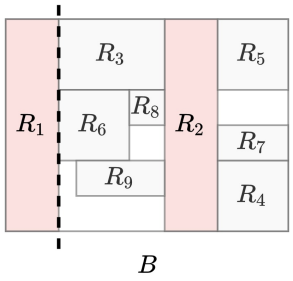


Figure 4.6: Cutting rectangle R_1 by one vertical guillotine cut.

cut can be omitted, because it coincides with the left side of the bin B, so the only cut needed to cut off rectangle R_1 is $x = x_1 + w_1 = 0 + w_1 = w_1$ as shown in Figure 4.6 in a dashed line. By this vertical guillotine cut, we obtain 2 new sub-bins as can be seen in Figure 4.7 – a sub-bin containing rectangle R_1 and nothing else (on the left), and sub-bin B_1 (on the right). The first mentioned sub-bin needs no further action, so it is marked by green color. Sub-bin B_1 contains all other rectangles, since the coordinates of the lower left vertex of B_1 are $(w_1, 0)$ with respect to the original bin B, the new coordinates $(\tilde{x}_i, \tilde{y}_i)$ of all the rectangles $\{R_2, \ldots, R_9\}$ in sub-bin B_1 need to be recalculated as $(\tilde{x}_i, \tilde{y}_i) = (x_i - l_1, y_i)$ for all $i \in \{2, \ldots, 9\}$.

Next, we need to cut off rectangle R_2 from sub-bin B_1. The coordinates of

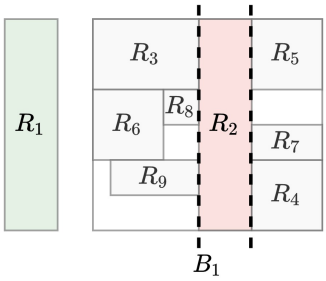


Figure 4.7: Cutting rectangle R_2 by two vertical guillotine cuts.

the lower left vertex of R_2 in B_1 are $(x_2, y_2) = (x_2, 0)$. For this, two vertical guillotine cuts are needed, because none of its vertical sides coincides with any of the sides of the sub-bin B_1. The cuts $x = x_2$ and $x = x_2 + w_2$ in B_1 are depicted in Figure 4.7 in a dashed line. Those cuts divide sub-bin B_1 from Figure 4.7 into 3 new sub-bins as can be seen in Figure 4.8. The

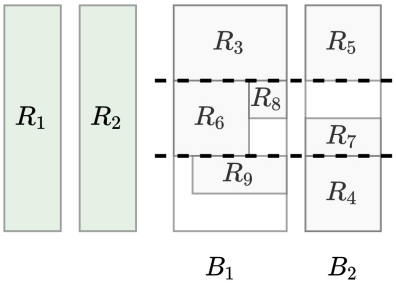


Figure 4.8: Cutting each sub-bin by two horizontal guillotine cuts.

coordinates of rectangles in the new sub-bin B_1 need no recalculation, and the first coordinates of the rectangles in the new sub-bin B_2 need to be lowered by $x_2 + w_2$.

At this point, all the rectangles of height H are positioned in a sub-bin containing only that particular rectangle. Since $h_2 = 3h_3$, two horizontal guillotine cuts need to be performed on each of the sub-bins B_1 and B_2. The cuts are in heights $y = h_3$ and $y = 2h_3$ as depicted in dashed lines in Figure 4.8. The result of this cutting can be seen in Figure 4.9. Making 2 horizontal cuts divides each of 2 sub-bins in Figure 4.8 into 3 new sub-bins, so at this moment, we are left with 6 new sub-bins. Sub-bin B_1 in Figure 4.8 was divided into sub-bins B_1, B_2 and B_3 in Figure 4.9 and sub-bin B_2 in Figure 4.8 was divided into sub-bins B_4, B_5 and B_6 in Figure 4.9. The coordinates of all the rectangles with respect to the new sub-bins need to be recalculated by formula $(\tilde{x}_i, \tilde{y})_i = (x_i, (y_i \mod h_3))$ for $i \in \{3, \ldots, 9\}$. At this moment, there are 4 rectangles having the same height as the new sub-bins and we can proceed with vertical cuts.

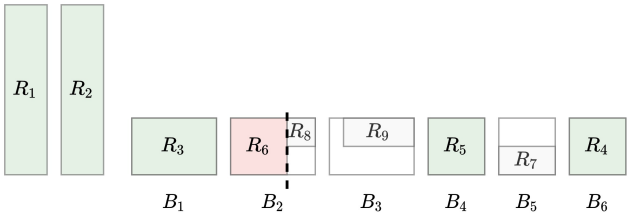


Figure 4.9: Cutting rectangle R_6 by one vertical guillotine cut.

Both cuts needed to cut off the rectangle R_3 coincide with vertical sides of sub-bin B_1 in Figure 4.9, so the cuts can be omitted and the rectangle R_3 is already placed in a sub-bin containing nothing else. The same situation occurs for rectangle R_4 in sub-bin B_6 and rectangle R_5 in sub-bin B_4 in the same Figure. The only rectangle that needs a cut is R_6 in B_2 - its left side coincides with the side of B_2, so only cut $x = x_6 + w_6 = w_6$ needs to be performed on B_2 from Figure 4.9. The situation after this cut is depicted in Figure 4.10. Note that the coordinates of rectangle R_8 in the new B_2 need to be recomputed as $(\tilde{x}_8, \tilde{y}_8) = (x_8 - w_6, y_8)$.

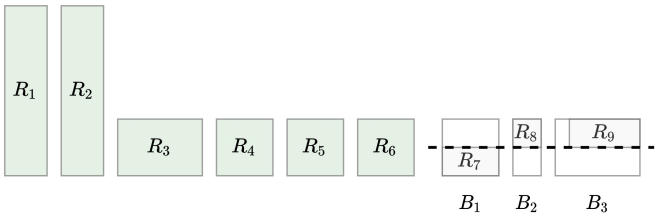


Figure 4.10: Cutting each sub-bin by one horizontal guillotine cut.

Once again, all the rectangles of the same height as the sub-bins have already been processed, so the horizontal cutting follows. Since $h_6 = 2h_7$, only one cut $y = h_7$ needs to be done on each of the sub-bins B_1, B_2 and B_3 in Figure 4.10. The result of this cutting can be seen in Figure 4.11. We get $3 \cdot 2 = 6$ new sub-bins, the coordinates of all rectangles R_7, R_8 and R_9 need to be recomputed with respect to the new sub-bins from Figure 4.11 by formula $(\tilde{x}_i, \tilde{y})_i = (x_i, (y_i \mod h_7))$ for $i \in \{7, 8, 9\}$ and we can proceed with vertical cuts.

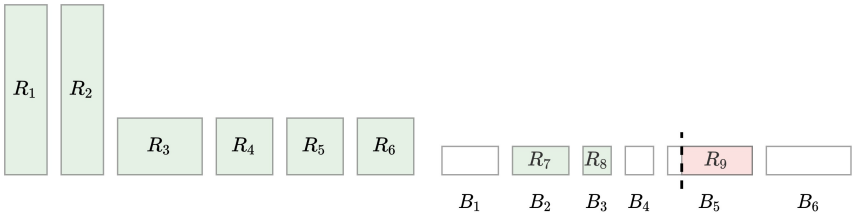


Figure 4.11: Cutting rectangle R_9 by one vertical guillotine cut.

As we can see in Figure 4.11, rectangles R_7 and R_8 are perfectly fitted into sub-bins B_2 and B_3 so that all of the sides of rectangles coincide with the sides of sub-bins. The last cut needed is $x = x_9$ on sub-bin B_5 as depicted in the dashed line in Figure 4.11 No recomputation of coordinates is needed after this cut. The result can be seen in Figure 4.12. First 9 sub-bins in Figure 4.12

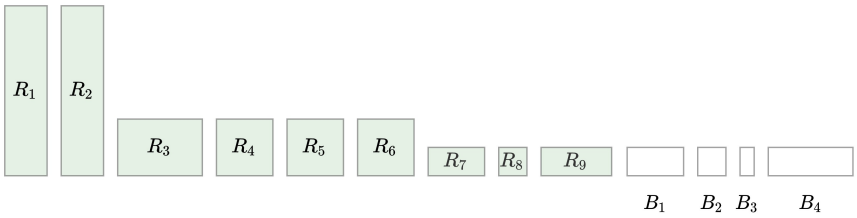


Figure 4.12: Each sub-bin contains either a rectangle with no unoccupied space or an unoccupied space only.

contain exactly one rectangle each and no unoccupied space. The remaining 4 sub-bins B_1, B_2, B_3 and B_4 contain only unoccupied space, so the original bin was divided into sub-bins of desired forms using only guillotine cuts. △

4.3.3 Search Algorithm

We have proved that the rectangles of the set of 2D bin packing problems on single machine associated with the original periodic scheduling problem with harmonic periods can be separated by guillotine cutting. In order to be able to solve instances of the given problem, we propose an algorithm determining the coordinates of the rectangles in the bins for each machine. The algorithm is based on the depth-first search and generalizes the algorithm from [KAL96].

At the beginning, we are given a set of n rectangles $\mathcal{R} = \{R_1, \ldots, R_n\}$, each rectangle $R_i \in \mathcal{R}$ has a specified width w_i and height h_i (those were derived from the tasks of the original periodic scheduling problem with harmonic periods dedicated to one of the machines). By formulas (3.10) and (3.11), we can derive the width W and the height H of the initial bin B. The goal is to find the coordinates (x_i, y_i) of each rectangle $R_i \in \mathcal{R}$ so that formula (3.13) holds and no collision occurs, which means also formula (3.14) holds.

State space

Each state s of the search space is specified by a tuple $\langle \mathcal{B}, \hat{\mathcal{R}} \rangle$, where $\mathcal{B} = \{B_1, \ldots, B_{|\mathcal{B}|}\}$ is an ordered sequence of currently available sub-bins and $\hat{\mathcal{R}} = \{R_1, \ldots, R_{|\hat{\mathcal{R}}|}\}$ is an ordered sequence of rectangles without a specified position. Every rectangle $R_i \in \hat{\mathcal{R}}$ is specified by a tuple $R_i = \langle w_i, h_i \rangle$, where h_i denotes the height and w_i the width of rectangle R_i for $i \in \{1, \ldots, |\hat{\mathcal{R}}|\}$. Every sub-bin $B_i \in \mathcal{B}$ is specified by a tuple $B_i = \langle W_i, H_i \rangle$, where H_i and W_i denote the height and the width of sub-bin B_i for $i \in \{1, \ldots, |\mathcal{B}|\}$. Heights of all the sub-bins of a given state s are the same as the height of the first rectangle $R_1 \in \hat{\mathcal{R}}$, if such exists (otherwise, the heights correspond to the minimal height of rectangles from $\hat{\mathcal{R}}$ in case there are any sub-bins left). Note that the algorithm does not distinguish between two sub-bins of the same dimensions (as well as two rectangles of the same dimensions).

Let us describe the search tree more closely. The initial state (and the root of the search tree) is specified by a tuple $s_0 = \langle \mathcal{B}^0, \hat{\mathcal{R}}^0 \rangle$, where the sequence of sub-bins $\mathcal{B}^0 = \{B\}$ contains only the original bin $B = \langle W, H \rangle$ and the sequence of rectangles to be positioned $\hat{\mathcal{R}}^0$ is the sequence of rectangles from set $\mathcal{R}$ ordered primarily by their heights in non-ascending order and secondarily (in case the heights are equal) by their widths in non-ascending order.

Each forward edge form state $s_i = \langle \mathcal{B}^i, \hat{\mathcal{R}}^i \rangle$ to state $s_{i'} = \langle \mathcal{B}^{i'}, \hat{\mathcal{R}}^{i'} \rangle$ in this search tree represents placing the first rectangle from $\hat{\mathcal{R}}^i = \{R^i_1, \ldots, R^i_{|\hat{\mathcal{R}}^i|}\}$ into one of sub-bins of $\mathcal{B}^i$. The rectangle is placed to the leftmost position in the chosen sub-bin. If there is any unoccupied space left in this sub-bin, a vertical cut is made, which divides the sub-bin into two new sub-bins - one fully occupied by the rectangle and one fully unoccupied.

Therefore the i-th level of search tree represents a set of guillotine packings of rectangles $R^0_1, \ldots, R^0_i$ to sub-bin B, $|\hat{\mathcal{R}}^i| = n - i$, and if $|\hat{\mathcal{R}}^i| \geq 2$, then $\hat{\mathcal{R}}^{i'} = \{R^i_2, \ldots, R^i_{|\hat{\mathcal{R}}^i|}\}$ (otherwise rectangle R^i_1 was the last one to position and $\hat{\mathcal{R}}^{i'}$ is empty). The number of direct successors of state $s_i = \langle \mathcal{B}^i, \hat{\mathcal{R}}^i \rangle$ can be computed as the cardinality of set $\{B^i_k \in \mathcal{B}^i \mid w^i_1 \leq W^i_k\}$, a set of sub-bins that the rectangle R^i_1 fits into. Note that $\{B^i_k \in \mathcal{B}^i \mid w^i_1 \leq W^i_k\}$ is a set – not a sequence – and also that any two sub-bins are considered to be the same whenever they have the same dimensions. Hence, if $\mathcal{B}^i$ is a set of any number of sub-bins with the same dimension, the cardinality of set $\{B^i_k \in \mathcal{B}^i \mid w^i_1 \leq W^i_k\}$ would be at most one. Direct successors of any state are ordered in the same order as the sub-bins, in which we have placed the rectangle R^i_1.

Deriving the exact form of $\mathcal{B}^{i'}$ from $\mathcal{B}^i$ is more complicated. Suppose rectangle R_1^i is to be positioned into sub-bin $B_{k'}^i$ (and it is possible to place it there). In the statement below we will use the following notation:

- H^i denotes the common height of sub-bins in $\mathcal{B}^i$,
- q denotes $\frac{h_1^i}{h_2^i}$, $q \in \mathbb{Z}^+$ because of the harmonic property of heights of rectangles,
- $\hat{\mathcal{B}^{i'}}$ denotes a sequence containing the same items as $\mathcal{B}^{i'}$, but possibly in a different order,
- B_{new}^i denotes a new sub-bin having width $(W_{k'}^i - w_1^i)$ and the same height as sub-bins in $\mathcal{B}^i$,
- $n \times B_{(k,n)}^i$ denotes n new sub-bins having the same width as sub-bin B_k^i and height $\frac{H^i}{n}$ (assuming $n \in \mathbb{Z}^+$, $n \mid H^i$),
- $n \times B_{(new,n)}^i$ denotes n new sub-bins having width $(W_{k'}^i - w_1^i)$ and height $\frac{H^i}{n}$ (assuming $n \in \mathbb{Z}^+$, $n \mid H^i$).

Then

$$\hat{\mathcal{B}^{i'}} = \begin{cases} \{B_1^i, \ldots, B_{k'-1}^i, B_{k'+1}^i, \ldots, B_{|\mathcal{B}^i|}^i\}, & \text{if } (|\hat{\mathcal{R}}^i| = 1 \ \wedge \ W_{k'}^i = w_1^i) \\ & \quad \vee (h_1^i = h_2^i \ \wedge \ W_{k'}^i = w_1^i), \\ \{B_1^i, \ldots, B_{k'-1}^i, B_{new}^i, B_{k'+1}^i, \ldots, B_{|\mathcal{B}^i|}^i\}, & \text{if } (\hat{\mathcal{R}}^i| = 1 \ \wedge \ W_{k'}^i > w_1^i) \\ & \quad \vee (h_1^i = h_2^i \ \wedge \ W_{k'}^i > w_1^i), \\ \{q \times B_{(1,q)}^i, \ldots, q \times B_{(k'-1,q)}^i, & \\ \quad q \times B_{(k'+1,q)}^i, \ldots, q \times B_{(|\mathcal{B}^i|,q)}^i\}, & \text{if } (h_1^i > h_2^i \ \wedge \ W_{k'}^i = w_1^i), \\ \{q \times B_{(1,q)}^i, \ldots, q \times B_{(k'-1,q)}^i, q \times B_{(new,q)}^i, & \\ \quad q \times B_{(k'+1,q)}^i, \ldots, q \times B_{(|\mathcal{B}^i|,q)}^i\}, & \text{if } (h_1^i > h_2^i \ \wedge \ W_{k'}^i > w_1^i). \end{cases}$$

In order to get the exact form of $\mathcal{B}^{i'}$, sequence $\hat{\mathcal{B}^{i'}}$ needs to be sorted by width of sub-bins in non-ascending order.

An example of the proposed search tree for a specific instance can be found in Figures 4.14 and 4.15 at the end of this chapter. The root of the tree is on the top of in Figures 4.14. Each node is depicted as two columns, the left column contains all available sub-bins and the right column all available rectangles. Indices of both sub-bins and rectangles give their proper order. The rectangle is in red color if it should be placed in current level, but there is no suitable sub-bin. The dashed arrows at the bottom of Figure 4.14 and at the top of Figure 4.15 depict the same edge.

Completeness

The leaves of the described search tree in level n (which is the number of rectangles) represent all guillotine packings meeting the following conditions:

$$\forall R_i, R_{i'} \in \mathcal{R},\ y_i < (y_{i'} + h_{i'}),\ y_{i'} < (y_i + h_i),\ x_i < x_{i'} :\ h_i \geq h_{i'}, \tag{4.5}$$

$$\forall R_i, R_{i'} \in \mathcal{R},\ x_i = x_{i'},\ h_i = h_{i'},\ w_i > w_{i'} : y_i < y_{i'}. \tag{4.6}$$

$$\forall R_i, R_{i'} \in \mathcal{R},\ x_i = x_{i'},\ h_i > h_{i'} : y_i < y_{i'}, \tag{4.7}$$

Condition 4.5 claims, that if we consider any rectangle and explore rectangles that are packed in the corresponding height in the original bin, all rectangles of greater height are packed on its left side and all rectangles of smaller height are packed on its right side. Conditions 4.6 and 4.7 claim that rectangles having the same first coordinate are packed so that their order from the bottom of the original bin to its top corresponds to the original order of the rectangles. All of these conditions are consequences of the ordering of rectangles (4.5, 4.7) and sub-bins (4.6, 4.7). Apart from that, there are also conditions on the free space. It holds that on the left side of any rectangle $R_i \in \mathcal{R}$, there is no free space (which is a direct consequence of the fact, that each rectangle is positioned on the leftmost place in chosen sub-bin).

Lemma 4.2 (Completeness). *For any feasible instance of 2D bin packing problem, there exists a feasible guillotine packing meeting conditions 4.5, 4.6, 4.7 and having all the unoccupied space on right side of all rectangles from $\mathcal{R}$.*

Proof. In order to prove the lemma, it is sufficient to provide a transformation of arbitrary packing into such guillotine packing, that meets the required conditions.

The process is iterative and handles the rectangles in the order given by $\hat{\mathcal{R}}$ but at this time, the coordinates of rectangles are taken into account. Each rectangle is first shifted horizontally so that 4.5 holds and also that no unoccupied space is left on the left side of any sub-bin (the horizontal order of the rectangles is correct). After this point, we perform only vertical movements.

We iterate through $\hat{\mathcal{R}}$ once again and move rectangles vertically, so that 4.6 and 4.7 hold. Hence we want to keep the horizontal ordering, we perform the shift on whole blocks of rectangles (all the rectangles on the right side of the misplaced rectangle). □

Lemma 4.2 claims that an arbitrary packing of 2D bin packing problem can be transformed into a guillotine packing meeting conditions 4.5, 4.6, 4.7 and having all the unoccupied space on right side of all rectangles form $\mathcal{R}$. Since the proposed search tree traverses all possible guillotine packings meeting conditions 4.5, 4.6, 4.7 and having all the unoccupied space on right side of all rectangles form $\mathcal{R}$, it is also the proof of completeness of the depth-first search traversal of proposed tree structure.

Example 2 (Demonstration of the transformation from 4.2)**.** Figure 4.13 depicts the process of transformation of an arbitrary packing into such guillotine packing which can be obtained via traversing the search tree.

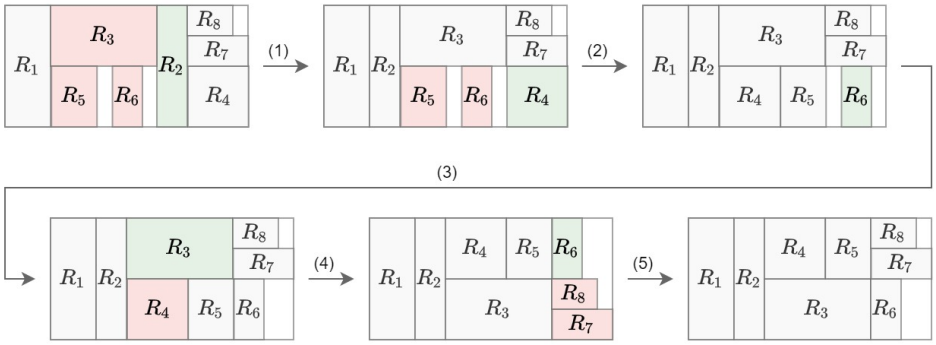


Figure 4.13: Transformation of a 2D bin packing solution.

During the transformation, we iterate through the ordered sequence of rectangles of the instance and shift some of them. The instance in Figure 4.13 contains 8 rectangles and their indices correspond to the order of $\hat{\mathcal{R}}$.

- Horizontal movements
  - (1) The first horizontally misplaced rectangle is R_2, as can be seen at the top left packing in Figure 4.13. Condition 4.5 is not met for three pairs of rectangles – $(R_2, R_3), (R_2, R_5)$ and (R_2, R_6). Rectangle R_2 is marked in green and all the rectangles positioned on the incorrect side of R_2 are marked in red. To fix the order of the rectangles, R_2 needs to be shifted by $(x_2 - x_3)$ to the left and rectangles R_3, R_5, R_6 need to be shifted by w_2 to the right.
  - (2) A similar case for R_4 (in green at the top middle packing in Figure 4.13), shift R_4 by $(x_4 - x_5)$ to the left and R_5 and R_6 (in red on the same Figure) by w_4 to the right.
  - (3) Rectangle R_6 (in green at the top right packing in Figure 4.13) has an unoccupied space on its left side and therefore needs to be shifted by $(x_6 - x_5 - w_5)$ to the left.

  The horizontal order of the rectangles has been corrected. Let us iterate through the $\hat{\mathcal{R}}$ once more to fix the vertical order.

- Vertical movements
  - (4) The first vertically misplaced rectangle is R_3, as can be seen at the bottom left packing in Figure 4.13. Condition 4.6 is not met for the pair of rectangles – (R_3, R_4). Rectangle R_3 is marked in green and all the rectangles of the same height and greater width having the first coordinate equal to x_3 positioned lower than R_3 are marked in red. To fix the order of the rectangles, R_3 needs to be shifted by $(y_3 - y_4)$ to the bottom, together with R_7 and R_8. On the other hand, rectangle R_4 needs to be shifted by $(y_3 - y_4)$ to the top, together with R_5 and R_6.
  - (5) The second vertically misplaced rectangle is R_6, as can be seen at the bottom middle packing in Figure 4.13. Condition 4.7 is not met for two pairs of rectangles – (R_6, R_7) and (R_6, R_8). Rectangle R_6 is marked in green and all the rectangles of the lower height having the first coordinate equal to x_6 positioned lower than R_6 are marked in red. To fix the order of the rectangles, R_6 needs to be shifted by $(y_6 - y_7)$ to the bottom and rectangles R_7 and R_8 need to be shifted by $(y_6 - y_7)$ to the top.

The guillotine packing meeting the required conditions is depicted at the bottom right part in Figure 4.13. Note that during transformation, the indices of misplaced rectangles are always lower than the indices of rectangles with which they are in conflict. That is caused by the fact that we iterate over $\hat{\mathcal{R}}$ in the given order and once we fix position of a rectangle in one dimension, it is never broken. △

The example above also helps us to derive the proper formulae for two of the shifts introduced in the proof of Lemma 4.2.

If condition 4.5 does not hold for some rectangle $R_{i'}$, we first need to identify the set of rectangles

$$\mathcal{R}^\star_{i'} = \{R_i \in \mathcal{R} \mid y_i < (y_{i'} + h_{i'}),\ y_{i'} < (y_i + h_i),\ x_i < x_{i'},\ h_i < h_{i'}\}.$$

Denote $x^\star_{i'} = \min\{x_i \mid R_i \in \mathcal{R}^\star_{i'}\}$ the minimal first coordinate of rectangle in $\mathcal{R}^\star_{i'}$. Then rectangle $R_{i'}$ needs to be shifted by $(x_{i'} - x^\star_{i'})$ to the left and all the rectangles from set $\mathcal{R}^\star_{i'}$ need to be shifted by $w_{i'}$ to the right. Note that shifting rectangles in horizontal direction does not effect feasibility.

If condition 4.6 does not hold for some rectangle R_i, we first need to identify the set of rectangles

$$\mathcal{R}^\star_i = \{R_{i'} \in \mathcal{R} \mid\ x_i = x_{i'},\ h_i = h_{i'},\ w_i > w_{i'},\ y_i \geq y_{i'}\}.$$

Denote $R_i^\star = \underset{R_{i'} \in \mathcal{R}_i^\star}{\arg\min}(y_{i'})$ the rectangle form $\mathcal{R}_i^\star$ having minimal second coordinate. Then rectangle R_i needs to be shifted by $(y_i - y_i^\star)$ to the bottom and rectangle $R_i^\star$ needs to be shifted by $(y_i - y_i^\star)$ to the top. Together with R_i, we also need to shift all rectangles $R_{i''}$ such that $x_{i''} > x_i$ and $y_i \leq y_{i''} \leq (y_i + h_i)$ by $(y_i - y_i^\star)$ to the bottom and similarly, together with $R_{i'}$, we also need to shift all rectangles $R_{i''}$ such that $x_{i''} > x_{i'}$ and $y_{i'} \leq y_{i''} \leq (y_{i'} + h_{i'})$ by $(y_i - y_i^\star)$ to the top. Hypothetically, moving rectangles in vertical direction might effect feasibility, but since we are shifting only rectangles of height $h_i = h_{i'}$ or smaller by a multiple of h_i, 3.13 still holds and even the shifted packing is feasible.

If condition 4.7 does not hold for some rectangle R_i, the situation is similar to the previous case, but more complex since we have to consult 3.13. We need to identify the set of rectangles

$$\mathcal{R}_i^\star = \{R_{i'} \in \mathcal{R} \mid \ x_i = x_{i'},\ h_i > h_{i'},\ y_i \geq y_{i'}\}.$$

Denote

$$y_i^\star = \max\{\hat{y} \in \mathbb{Z}^+ \mid \hat{y} \equiv 0 \pmod{h_i},\ \forall R_{i'} \in \mathcal{R}_i^\star : \hat{y} \leq y_{i'}\}.$$

Then rectangle R_i needs to be shifted by $(y_i - y_i^\star)$ to the bottom. Together with R_i, we also need to shift all rectangles $R_{i''}$ such that $x_{i''} > x_i$ and $y_i \leq y_{i''} \leq (y_i + h_i)$ by $(y_i - y_i^\star)$ to the bottom. To avoid collision, we need to shift rectangles $R_{i''}$ such that $x_{i''} \geq x_i$ and $y_i^\star \leq y_{i''} \leq (y_i^\star + h_i)$ by $(y_i - y_i^\star)$ to the top.

4.3.4 Properties of the Approach

The algorithm discussed in this section is complete, which is an advantage. At the same time, it is complicated. The depth-first search algorithm is known to run in $\mathcal{O}(|V| \cdot |E|)$ time, where $|V|$ is the number of vertices and $|E|$ is the number of edges of the search graph. Our search tree has $n + 1$ levels and each vertex has at most H descendants. From that point of view, the time complexity of the worst case might be exponential in number of rectangles. On the other hand, the algorithm does not traverse the whole search tree every time, feasible solution can be constructed from any leaf in the n-th level. Also, the root is known to have only one descendant, so the above approximation is probably a loose upper bound.

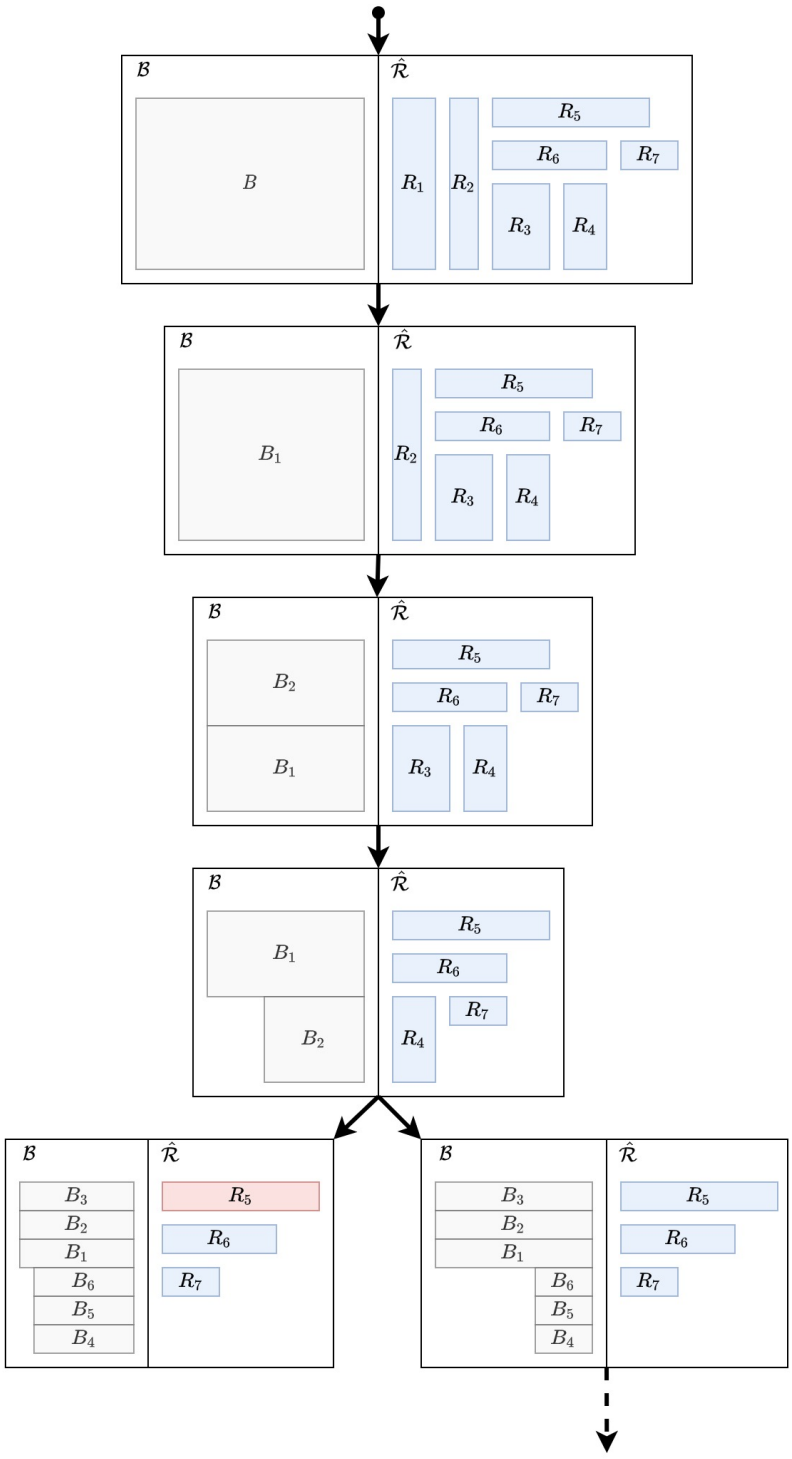


Figure 4.14: An example of search tree of proposed DFS (part 1).

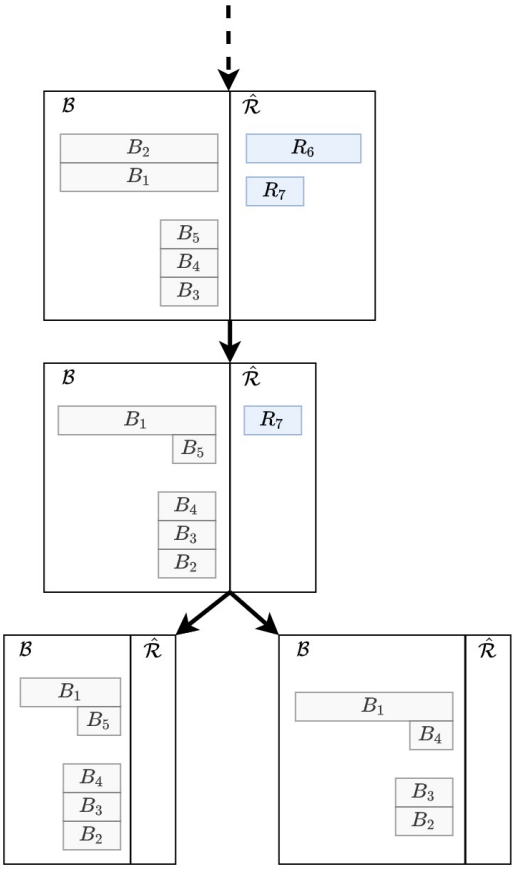


Figure 4.15: An example of search tree of proposed DFS (part 2).

Chapter 5

Finding an Optimal Solution

In the previous chapter, we have discussed finding a feasible solution for the presented variant of PSP. In this chapter, we will concentrate on finding the optimal solution. [HMH20] proposes a local search heuristic for this particular PSP based on the Queue Approach described in section 4.1. We provide an overview of this heuristic. Then, we also provide description of local search method called the *tabu search* and use this method to search for the optimal solution of given PSP.

5.1 Overview of Existing Local Search Heuristic

As we have already mentioned, [HMH20] proposes a local search heuristic which works with the queue of tasks described in section 4.1. The heuristic is presented in Algorithm 1. The algorithm works in iterations. At first it initializes a queue of tasks Q, reconstructs a schedule (as was described in subsection 4.1.2) and determines the degeneracy of current solution $\delta(Q)$.

At every iteration, we call the Neighbor function, which takes the current queue of tasks Q and generates a random neighbouring queue Q'. Based on queue Q', we reconstruct a new schedule and determine its degeneracy $\delta(Q')$. If the degeneracy of the new schedule is lower or equal to the degeneracy of the previous schedule based on queue Q, we set the current queue to Q'. Otherwise, current queue stays the same and we proceed to the next iteration.

Algorithm 1 Local search heuristic [HMH20]

```
1:  s_act ← InitializeSolution()
2:  δ_act ← ReconstructAndComputeDegeneracy(s_act)
3:  if δ_act = 0 then return s_act
4:  while elapsedTime < timeLimit do
5:      s_new ← DeriveNewSolution(s_act)
6:      δ_new ← ReconstructAndComputeDegeneracy(s_new)
7:      if δ_new ≤ δ_act then
8:          s_act ← s_new
9:          δ_act ← δ_new
10:         if δ_new = 0 then return s_new
    return s_act
```

The algorithm terminates either when the time limit expires or when it finds a schedule with degeneracy of 0, which can not be further optimized. [HMH20] claims that this solution solves on average 92% of test cases optimally or almost-optimally.

Due to the design, the algorithm is not able to escape the local optima, because it always proceeds to better solution. Therefore, the process needs to be restarted multiple times to reach the mentioned results.

5.2 Tabu Search

Tabu search (TS) is a single-solution local search algorithm introduced in [Glo86] and described and explained in detail in [Glo90]. It is a metaheuristic with memory allowing to escape the local optima. Its main idea is to avoid the states that have been visited recently or the ones that have the same features as the visited states. The only exception is made when there is an aspiration for a better result. To remember the recently visited states, the algorithm uses a tabu list – a queue of restricted features. To evaluate if it is worth breaking the restriction, a condition called aspiration criterion is used.

The basic run of the tabu search algorithm is depicted in Figure 5.1. In the beginning, the best solution so far $\hat{s}$ and an empty tabu list are initialized. To enter the main loop, a current solution is needed. Therefore the input of the algorithm is an initial solution, which is also considered to be the best solution so far $\hat{s}$. At first, a list of neighbor solutions is derived from the current solution. The neighbors are the candidates for the current solution of the following iteration. All the candidates are evaluated and confronted with

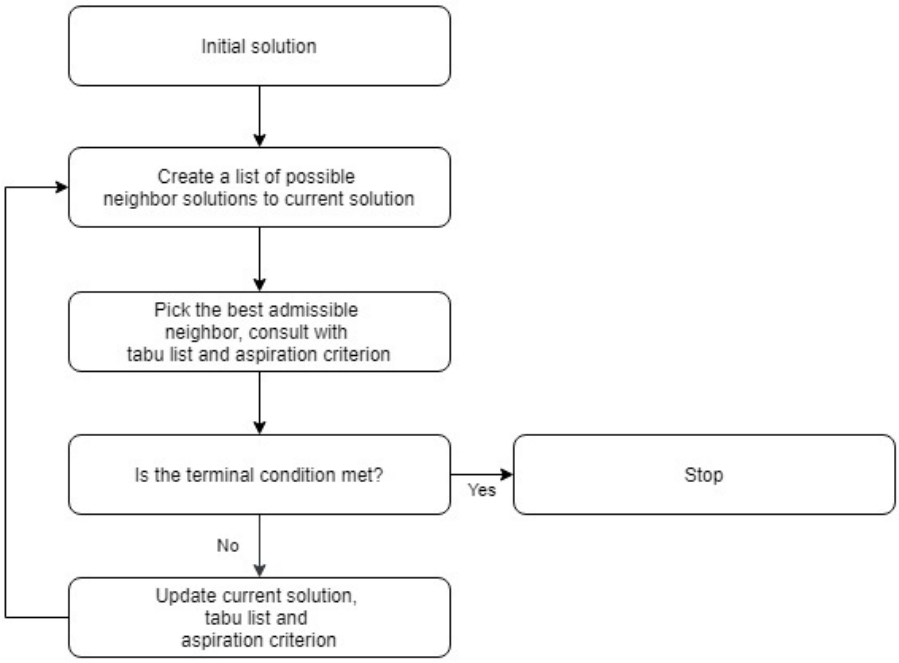


Figure 5.1: The main loop of the algorithm [Glo90].

a tabu list and aspiration criterion. Out of the suitable neighbors, the best one is picked. In case the algorithm did not meet the terminal condition, the current move needs to be recorded to the tabu list, the aspiration criterion is updated and the chosen neighbor is considered to be the current solution of the next iteration. Also, the best so far solution $\hat{s}$ might be updated. If the terminal condition is met, the run ends and the output is the best solution so far $\hat{s}$.

5.2.1 Neighbor Function

The very first step in the loop is generating a list of neighbors from a current solution. A state is considered to be the neighbor of the current solution if it is somehow similar to it. Its particular concept depends on the given problem and implementation.

5.2.2 Tabu List

There are many parameters that need to be set during the implementation of the tabu search. Among them are the properties of the tabu list, such as the structure of its items and its length.

Whenever we move from the current solution to its neighbor, we should track the features of the change in a new tabu list item. Therefore the nature of the change should be captured well by the selected feature. We also have to keep in mind that every tabu list item possibly decreases the number of admissible solutions in the state space. If we pick a feature that is too restrictive, we might come across the situation that no neighbor of the current solution is admissible.

The length of the tabu list indicates the memory range of the algorithm. Technically, the tabu list is a queue of the given maximal length. In case of an attempt to insert an item that would increase its length over the maximum, the earliest inserted item is removed from the queue. If the list is too short, the run of the program is more likely to loop and get stuck in the local optima. On the other hand, if the tabu list is too long, it might prevent the program from returning to the promising part of state space.

5.2.3 Picking the Best Neighbor

One of the most computationally demanding parts of the whole algorithm might be the choice of the best neighbor $n^\star$. The process is depicted in Figure 5.2.

We iterate over the list of neighbors. At first, the current neighbor is evaluated. Its value is compared with $n^\star$ – the best neighbor found so far within the current round of the main loop. If the value of the current neighbor is worse than the value of $n^\star$, we continue to the next iteration (if there is any left). Otherwise, we proceed to consult the tabu list. The current neighbor is checked if it contains any feature stored in the tabu list. In case it does not, we claim that the current neighbor is so far the best admissible neighbor of this main loop. The current neighbor becomes the new $n^\star$ and we continue to the next iteration. Otherwise, we found out that the current neighbor has some restricted feature and we can consider it to be admissible (and in this phase also the best admissible neighbor of this main loop found so far, the

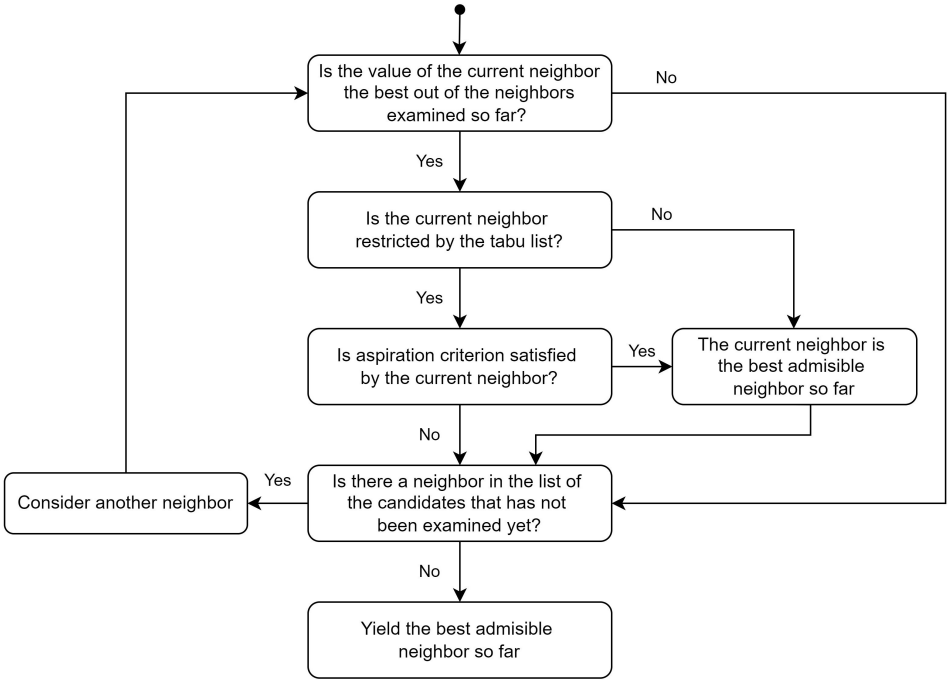


Figure 5.2: Selection of the best neighbor [Glo90].

new value of $n^\star$) if and only if the aspiration criterion is satisfied. After that, we can once more continue to the next iteration or step out of the cycle.

As we might have already noticed, the aspiration criterion is a way to overcome the restrictions given by the tabu list. Hence it is important to specify it in a way that corresponds to the nature of the given task. The aspiration criterion is usually related to the calculation of the solution value.

5.2.4 Terminal Condition

The terminal condition in the main loop is not strictly specified and might be also defined in several ways based on the specific problem. The program might be limited by time, the number of iterations, etc. The terminal condition might be also met if the program found a solution that is known to be optimal.

5.3 Tabu Search with the Queue Approach

Representation of the schedule by the queue brings the possibility of simple manipulation and fast generation of neighbouring solutions, which seems to be a good choice for tabu search, since it searches the whole set of neighbouring solutions in every iteration. Therefore, we tried to use the queue representation of schedule in tabu search method. This section provides details on the implementation.

5.3.1 Generating Neighboring Solutions

When generating the list of neighbors from a current solution given by a queue, we include the queues derived from the current solution

- by swapping two of its tasks,
- by swapping tasks of a chain so that they are ordered according to the given precedence.

Note that at least two tasks changed their positions in the queue. The size of the neighbourhood is given by formula $\binom{n}{2} + |\mathcal{C}|$.

5.3.2 Tabu List Items

To track the changes of the current solution during the program run in the tabu list, we propose to use the original position of the swapped tasks in the task queue with respect to the adjacent tasks. Each tabu list item is therefore represented by two triplets of tasks,

$$\{(pred_1, moved_1, succ_1), (pred_2, moved_2, succ_2)\},$$

where

- $moved_1$ and $moved_2$ denote the tasks that changed their positions,

- $pred_1$ and $pred_2$ denote the tasks that were directly in front of the tasks $moved_1$ and $moved_2$ respectively in the original queue,
- $succ_1$ and $succ_2$ denote the tasks that were directly behind the tasks $moved_1$ and $moved_2$ respectively in the original queue.

If any of the tasks which changed their position was the very first or the very last task in the queue, we use keep $pred_i$ or $succ_i$ respectively empty. If there are more tasks than two that changed their positions, we use the first two of such tasks.

Example 3. To track the change from queue

$$(\tau_1, \tau_{\mathbf{2}}, \tau_3, \tau_4, \tau_{\mathbf{5}}, \tau_6, \tau_7, \tau_8)$$

to the neighbor queue

$$(\tau_1, \tau_{\mathbf{5}}, \tau_3, \tau_4, \tau_{\mathbf{2}}, \tau_6, \tau_7, \tau_8),$$

we insert

$$\{(\tau_1, \tau_2, \tau_3), (\tau_4, \tau_5, \tau_6)\}$$

into tabu list. △

Example 4. To track the change from queue

$$(\tau_{\mathbf{1}}, \tau_5, \tau_3, \tau_4, \tau_2, \tau_{\mathbf{6}}, \tau_7, \tau_{\mathbf{8}})$$

to the neighbor queue

$$(\tau_{\mathbf{6}}, \tau_5, \tau_3, \tau_4, \tau_2, \tau_{\mathbf{8}}, \tau_7, \tau_{\mathbf{1}}),$$

we insert

$$\{(null, \tau_1, \tau_5), (\tau_2, \tau_6, \tau_7)\}$$

into tabu list. △

5.3.3 Tabu List Length

The research of available literature on determination of the tabu list length did not show any paper addressing optimal tabu list length for PSP. Experiments described in [ZSRQ01] showed a positive correlation of a proper tabu list length and the ratio of number of jobs to the number of machines in the relative job-shop scheduling problem (JSSP).

5.3.4 Aspiration Criterion

As the aspiration criterion, we use a condition based on the degeneracy of the current schedule. If value of degeneracy of the current schedule is strictly lower than the value of degeneracy of $\hat{s}$ (which is the best found solution so far within the whole program run), the restrictions given by the tabu list can be broken.

5.3.5 Terminal Conditions

The terminal condition of our algorithm is the time limit of three minutes, program also terminates if it finds a schedule having degeneracy equal to zero, because such value can clearly not be further optimized.

5.3.6 Proposed Modification

During the algorithm implementation, the tests showed that the program is able to perform only a few iterations in the given time limit of three minutes. The majority of time was spent on reconstruction of the schedules from a queue. Testing one specific instance, the algorithm presented in [HMH20] performed about 21 000 iterations in the given time limit. The implementation of tabu search on the other hand performed only a few iterations on the very same instance.

In [KTTH06], several methods are proposed to deal with this issue. The random neighborhood method (in which the tasks that can be swapped are randomly chosen and their number is bounded) and the first improving neighborhood method (where we go for the first neighbor having a smaller degeneracy) were implemented.

5.4 Search Algorithm from [HMH20] with Aspiration Criterion

Since the tabu search algorithm implementation spends too much time on neighbourhood generation, we propose to enclose the tabu list and the aspiration criterion to the Local search heuristic introduced in [HMH20]. The original Local search heuristic is described in algorithm 1. Its modification consists of recording the moves to the tabu list and consulting the tabu list and the aspiration criterion in case equality occurs on line 7.

Chapter 6

Experimental Part

In this chapter, we provide the experimental results on the implemented versions of algorithms mentioned in this work. In order to test the quality of the implemented algorithms, a set of benchmark tests was randomly generated by the same instance generator as was used in [HMH20]. At first, let us describe the nature of generated test instances. Every algorithm is given time limit of 3 minutes.

6.1 Generated Test Instances

We test the implementation on 7 different sets of 800 instances. Each particular instance is feasible and its degeneracy is equal to 0. The number of tasks of each instance is at least 100 and at most 9 000. Each set contains

- 200 instances having $T(\mathcal{T}) = \{2^0, 2^1, 2^2, \ldots, 2^{10}\}$ and $|\mathcal{M}| = 5$,
- 200 instances having $T(\mathcal{T}) = \{2^0, 2^1, 2^2, \ldots, 2^{10}\}$ and $|\mathcal{M}| = 10$,
- 200 instances having $T(\mathcal{T}) = \{2, 10, 20, 100, 200, 1\,000, 2\,000, 4\,000\}$ and $|\mathcal{M}| = 5$,
- 200 instances having $T(\mathcal{T}) = \{8, 16, 64, 256, 1\,024, 2\,048\}$ and $|\mathcal{M}| = 5$.

Each set can be further described by the lower and upper bound of the chain length – $min\{|C_k| \mid C_k \in \mathcal{C}\}$ and $max\{|C_k| \mid C_k \in \mathcal{C}\}$ respectively – and the upper bound of machine utilization:

$$\sigma_{max} = \max_{\mu \in \mathcal{M}} \sum_{\tau : m(\tau) = \mu} \frac{p(\tau)}{T(\tau)}.$$

The exact values of those characteristics are given in Table 6.1

	σ_{max}	$min\{\|C_k\| \mid C_k \in \mathcal{C}\}$	$max\{\|C_k\| \mid C_k \in \mathcal{C}\}$
Set 1	0.8	5	15
Set 2	0.9	2	5
Set 3	0.9	5	15
Set 4	0.9	15	25
Set 5	0.93	5	15
Set 6	0.96	5	15
Set 7	1	5	15

Table 6.1: Parameters of the generated test sets [HMH20]

In addition the utilization of 95% generated instances in sets 1–6 is at the same time greater or equal to $(\sigma_{max} - 0.2)$. All the instances in set 7 have utilization greater or equal to 0.96 and at least 90% of them have utilization equal to 1.

6.2 Test Results

6.2.1 The Comparison of Algorithms for Feasibile solution

Table 6.2 provides the results in absolute numbers of instances, because the differences between algorithms are significant.

	L1	L2	L3	L4	L5	L6	L7	L8	Q	DFS
Set 1	16	16	261	363	16	16	261	530	793	799
Set 2	8	8	117	117	8	8	117	242	785	795
Set 3	10	10	93	118	10	10	93	242	795	792
Set 4	5	5	108	118	5	5	108	234	791	798
Set 5	6	6	69	62	6	6	69	138	788	798
Set 6	8	8	52	47	8	8	52	82	783	793
Set 7	1	1	6	7	1	1	6	7	436	782
$\sum$	54	54	706	832	54	54	706	1475	5 171	5 557

Table 6.2: Number of solved instances (up to 3 minutes) by test set and algorithm. L1 – Next-fit, rectangles ordered by width; L2 – Next-fit, rectangles ordered by height; L3 – Next-fit, rectangles ordered by width, rotated; L4 – Next-fit, rectangles ordered by height, rotated; L5 – First-fit, rectangles ordered by width; L6 – First-fit, rectangles ordered by height; L7 – First-fit, rectangles ordered by width, rotated; L8 – First-fit, rectangles ordered by height, rotated; Q – Reconstruction from queue of tasks; DFS – Depth-first search.

6.2.2 The Comparison of Algorithms for Optimality

	SA	SA with AC	TS
Set 1	0.0	0.41	14.1
Set 2	10.4	1.29	17.2
Set 3	0.1	3.29	49.9
Set 4	1.5	5.11	7.1
Set 5	2.7	4.39	47.0
Set 6	17.7	11.02	17.9
Set 7	50.8	129.23	90.3

Table 6.3: The comparison of an average value of degeneracy on instances solved by all compared algorithms SA – the original search algorithm from [HMH20]; SA with AC – search algorithm from [HMH20] with the usage of aspiration criterion; TS – tabu search with queue implementation.

Chapter 7

Conclusions

According to the results of feasibility algorithms tests depicted in Table 6.2, we can conclude that, given any other settings, first-fit algorithm solves at least as many instances as next-fit algorithm on any test set. From more detailed results, we could even claim, that this holds for every tested instance (which meets the assumption). Out of the tested level algorithms, rotated first-fit algorithm with rectangles ordered by height solves the largest number of test instances. Also, the level algorithms return the results immediately. On the other hand, the queue algorithm with reconstructive function and the depth-first search algorithm run longer, the depth-first search found a feasible solution in more than 99 % of instances.

Consulting the results of tested algorithms for finding the optimal solution, none of the tested algorithms returns strictly better results then the others. in all tested sets of instances. The local search algorithm gets still closer to the optima. Consulting with the tabu list and aspiration criterion helps only in some sets of instances.

In this work, we have studied a specific variation of PSP with precedence constraints. We have shown and demonstrated some of its important properties. We have also described a relation between the addressed PSP and a specific variation of 2D bin packing problem. We have proposed several approaches for finding a feasible and optimal solution. We have implemented some of the proposed algorithm and tested their performance.

As further work, it might be interesting to dig deeper into the guillotine packing properties.

I consider the introduction of dept-first search algorithm finding the feasible solution in over 99 % of benchmark instances to be the greatest contribution of this work.

Appendix A

Attachments

- *psp_queue.zip* – a modified source code from Richard Hladík containg his implementation of Search Algorithm from [HMH20], my proposed modifications and my implementation of tabu search.
- *pspFeasible.zip* – a source code of implemented feasible algorithms.

Appendix B

Nomenclature

Indices

i an index of a task
$i \in \{1, \ldots, n\}$

j an index of a machine
$j \in \{1, \ldots, |\mathcal{M}|\}$

k an index of a chain
$k \in \{1, \ldots, |\mathcal{C}|\}$

l an index of a task in chain C_k for any $k \in \{1, \ldots, |\mathcal{C}|\}$
$l \in \{1, \ldots, |C_k|\}$

Operators

$\delta(s)$ degeneracy of schedule s
$\delta : (s : \mathcal{T} \mapsto \mathbb{Z}_0^+) \mapsto \mathbb{Z}_0^+ \cup \{+\infty\}$

$\delta_s(C_k)$ degeneracy of chain C_k with respect to schedule s
$\delta_s : \mathcal{C} \mapsto \mathbb{Z}_0^+$

$E(\tau_i, z)$ the time interval of the z-th execution of task $\tau_i \in \mathcal{T}$, $z \in \mathbb{Z}^+$
$E(\tau_i, z) \subseteq \mathbb{R}_0^+$

$H(B_j)$ a height of bin B_j
$H : \mathcal{B} \mapsto \mathbb{Z}^+$

$h(R_\tau)$ a height of rectangle R_τ
$h : \mathcal{R} \mapsto \mathbb{Z}^+$

$L(C_k)$ the end-to-end latency of chain C_k
$L : \mathcal{C} \mapsto \mathbb{Z}^+$

$m(\tau_i)$ a machine dedicated to task τ_i
$m : \mathcal{T} \mapsto \mathcal{M}$

$p(\tau_i)$ a processing time of a task τ_i
$p : \mathcal{T} \mapsto \mathbb{Z}^+$

$s(\tau_i)$ a starting time of task τ_i (schedule)
$s : \mathcal{T} \mapsto \mathbb{Z}_0^+$

$s^*(\tau_i)$ the optimal starting time of task τ_i
$s^* : \mathcal{T} \mapsto \mathbb{Z}_0^+$

$T(\tau_i)$ a period of task τ_i
$T : \mathcal{T} \mapsto \mathbb{Z}^+$

$T(C_k)$ the common period of tasks in chain C_k
$T : \mathcal{C} \mapsto \mathbb{Z}^+$

$W(B_j)$ a width of bin B_j
$W : \mathcal{B} \mapsto \mathbb{Z}^+$

$w(R_\tau)$ a width of rectangle R_τ
$w : \mathcal{R} \mapsto \mathbb{Z}^+$

Sets

$\mathcal{B}$ a set of bins

$\mathcal{C}$ a set of chains

$\mathcal{H}$ a set of heights of the rectangles

$\mathcal{M}$ a set of machines

$\mathcal{R}$ a set of rectangles

$\mathcal{R}_j$ a set of rectangles to be positioned in bin B_j

$\mathcal{T}$ a set of tasks

$\mathcal{T}_{\mu_j}$ a set of tasks assigned to machine μ_j

$T(\mathcal{T})$ a set of all possible periods of tasks $\tau_i \in \mathcal{T}$

Other symbols

μ_j a machine
$\mu_j \in \mathcal{M}$

τ an arbitrary task from a domain specified in the text

τ_i	a task $\tau_i \in \mathcal{T}$
B	a bin $B \in \mathcal{B}$
B_j	a bin $B_j \in \mathcal{B}$
C_k	a chain $C_k \in \mathcal{C}$
C_k^l	the l-th task of chain C_k $C_k^l \in C_k$
H	a height of bin B $H \in \mathbb{Z}^+$
h_i	a height of rectangle R_i $h_i \in \mathbb{Z}^+$
n	the cardinality of set $\mathcal{T}$
$n^\star$	the best neighbour in tabu search
Q	a queue of tasks $\tau_i \in \mathcal{T}$
R_τ	a rectangle associated to task τ $R_\tau \in \mathcal{R}$
R_i	a rectangle $R_i \in \mathcal{R}$
W	a width of bin B $W \in \mathbb{Z}^+$
w_i	a width of rectangle R_i $w_i \in \mathbb{Z}^+$
x_τ	the horizontal coordinate of the lower-left vertex of rectangle R_τ in a specified bin
x_i	the horizontal coordinate of the lower-left vertex of rectangle R_i in a specified bin
y_τ	the vertical coordinate of the lower-left vertex of rectangle R_τ in a specified bin
y_i	the vertical coordinate of the lower-left vertex of rectangle R_i in a specified bin

Appendix C

Bibliography

[FMT16] Fabio Furini, Enrico Malaguti, and Dimitri Thomopulos, *Modeling two-dimensional guillotine cutting problems via integer programming*, INFORMS Journal on Computing **28** (2016), no. 4, 736–751.

[GLLK79] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling: a survey*, Discrete Optimization II (P.L. Hammer, E.L. Johnson, and B.H. Korte, eds.), Annals of Discrete Mathematics, vol. 5, Elsevier, 1979, pp. 287–326.

[Glo86] Fred Glover, *Future paths for integer programming and links to artificial intelligence*, Computers & Operations Research **13** (1986), no. 5, 533–549.

[Glo90] Fred Glover, *Tabu search: A tutorial*, Interfaces **20** (1990), no. 4, 74–94.

[HH20] Claire Hanen and Zdeněk Hanzálek, *Periodic scheduling and packing problems*, 2020.

[HMH20] Richard Hladík, Anna Minaeva, and Zdeněk Hanzálek, *On the complexity of a periodic scheduling problem with precedence relations*, Combinatorial Optimization and Applications (Cham), Springer International Publishing, 2020, pp. 107–124.

[KAL96] Jan Korst, Emile Aarts, and Jan Karel Lenstra, *Scheduling periodic tasks*, INFORMS Journal on Computing **8** (1996), no. 4, 428–435.

[KALW91] Jan Korst, Emile Aarts, Jan Karel Lenstra, and Jaap Wessels, *Periodic multiprocessor scheduling*, Parle '91 Parallel Architectures and Languages Europe (Berlin, Heidelberg), Springer-Verlag Berlin Heidelberg, 1991, pp. 166–178.

[KTTH06] Farhad Kolahan, Ahmad Tavakoli, Behrang Tajdin, and Modjtaba Hosayni, *Analysis of neighborhood generation and move selection strategies on the performance of tabu search*, Proceedings of the 6th WSEAS International Conference on Applied Computer Science (Tenerife), vol. 6, World Scientific and Engineering Academy and Society, 2006, pp. 503–507.

[MH21] Anna Minaeva and Zdeněk Hanzálek, *Survey on periodic scheduling for time-triggered hard real-time systems*, ACM Comput. Surv. **54** (2021), no. 1.

[Nv09] N. Ntene and J.H. van Vuuren, *A survey and comparison of guillotine heuristics for the 2d oriented offline strip packing problem*, Discrete Optimization **6** (2009), no. 2, 174–188.

[ZSRQ01] Chaoyong Zhang, Xinyu Shao, Yunqing Rao, and Haobo Qiu, *Some new results on tabu search algorithm applied to the job-shop scheduling problem*, Tabu Search, I-Tech Education and Publishing, 2008-09-01, pp. 175–198.