# Combinatorial algorithms for the PNS and online scheduling problems

Doctoral thesis

Csanád Imreh

Dept. of Informatics

Szeged

Supervisor: Dr János Csirik

**2001**

# Contents

# Chapter 1

# Introduction

For many optimization problems their structures make it possible to develop fast algorithms for their solution by using combinatorial ideas. On the other hand, the original problem is often too difficult to find efficient algorithms. In these cases, some combinatorial ideas can be used to establish algorithms for solving some particular cases of the problem, or algorithms which do not solve the problem, but give an approximate solution which is close to the optimal one in some sense.

In the first part of this work we study the following hard optimization problem, called PNS problem. In a manufacturing system, materials of different properties are consumed through various mechanical, physical and chemical transformation to yield desired products. Devices in which these transformations are carried out are called operating units, e.g. a lathe or a chemical reactor. Thus, a manufacturing system can be considered as a network of operating units which is called process network. A process design problem in general, and flowsheeting in particular mean to construct a manufacturing system. A design problem is defined from a structural point of view by the raw materials, the desired products, and the available operating units, which determine the structure of the problem as a process graph containing the corresponding interconnections among the operating units. Thus, the appropriate process networks can be described by some subgraphs of the process graph belonging to the design problem under consideration. Our goal is to find a process network with minimal cost. The importance of process network synthesis arises from the fact that such networks are ubiquitous in the chemical and allied industries.

This minimization yields a combinatorial optimization problem. In [9] it is proven that this optimization problem is NP-hard.

In general there are three basic approaches to attack NP-hard problems. The first approach is to develop exponential time algorithms for solving the problem. In case of PNS problem, some exponential time algorithms based on the Branch and Bound technique were developed and studied in [25] and [26]. Another approach is to investigate specially structured instances which can be solved efficiently. These classes are called *well-solvable classes.* In case of PNS problem, well-solvable classes have not been developed earlier. In this thesis we present some of them. The third approach is to establish fast (polynomial time) algorithms which do not guarantee an optimal solution in general, but always result in a feasible solution which is close to the optimal solution in some sense. Such algorithms, called *heuristic algorithms* or *heuristics*, are important for several reasons. The feasible solutions found by heuristics can be used in procedures based on the Branch and Bound technique. Moreover, in practical problems there is often not enough time to find an optimal solution by an exponential algorithm, or the size of the problem is too large to use an exponential algorithm. In these cases, heuristic algorithms can be useful again. It can also occur that one does not need an optimal solution, it is sufficient to find a feasible solution the cost of which is not far from the optimal cost. For the PNS problem, heuristic algorithms have not yet been studied. Now we introduce and analyse some heuristics for the PNS problem.

Another field where algorithms based on combinatorial ideas are very useful is the online computation. The theory of online algorithms and competitive analysis is a new, rapidly developing area. In the second part of this work, we investigate three different online problems which are closely related to online machine scheduling. We develop and analyse some online algorithms for these problems, and we also present some general lower bounds for the possible competitive ratios.

The first problem considered is a particular scheduling problem. Usually in scheduling problems, the number of the available machines is a fixed parameter of the problem. We study the problem of scheduling with machine cost. Here we also have to purchase the machines, and the total cost, which

we want to minimize, is the sum of the cost of purchasing the machines and the cost of the produced schedule. The second problem we investigated is a scheduling problem where a two-layer multiprocessor architecture is given. In this problem there are two sets of machines, and the decision maker has to make an online assignment of jobs to one of the machine sets. The jobs are scheduled in an optimal offline preemptive way within a set. We study two models here. In the first one the goal is to minimize the sum of the makespans of the machine sets, in the second model we want to minimize the maximum of these makespans. The third problem which we consider is a strip packing problem with modifiable items. We investigate the online strip packing problem, where the sizes of the items are not fixed, so we can lengthen them.

The thesis is organized as follows. In the following chapter we give the mathematical model of the PNS problem, furthermore, we recall the most fundamental definitions and results concerning this problem. Then, in Chapter 3, some new results on the well-solvable classes of the PNS problem are presented. The material of this chapter is based on [7], [29] and [30]. In Chapter 4, two heuristic algorithms are introduced for the PNS problem, and they are studied by the worst-case analysis. This chapter is based on [8].

We start to present our results on online algorithms in Chapter 5. This chapter contains the most fundamental definitions of the competitive analysis, and we also present some basic scheduling models. Later, in Chapter 6, the online scheduling problem with machine cost is studied. We investigate this problem in two different models. This chapter is based on [32]. Chapter 7 contains our results on the scheduling problem with two-layer multiprocessor architecture (cf. [27], [28]). Finally, in Chapter 8, a strip packing problem with modifiable items is investigated. This chapter is based on [31].

# Chapter 2

# The mathematical model of the PNS problem

This chapter is devoted to the foundation of PNS problem. First in Section 2.1, the necessary notions and notation are introduced. Section 2.2 contains the corresponding optimization problem, and we recall here the proof which shows that this optimization problem is NP-hard.

## 2.1 The structural model of PNS problem

The foundations of PNS and the background of the combinatorial model studied here can be found in [18], [19], [20].

In the combinatorial approach, the structure of a process can be described by the process graph (see [19]) defined as follows.

Let $M$ be a finite nonempty set, the set of the materials. Furthermore, let $\emptyset \neq O \subseteq \wp'(M) \times \wp'(M)$ with $M \cap O = \emptyset$, where $\wp'(M)$ denotes the set of all nonempty subsets of $M$. The elements of $O$ are called *operating units* and for an operating unit, $u = (\alpha, \beta) \in O$, $\alpha$ and $\beta$ are called the *input-set* and *output-set* of the operating unit, respectively. The pair $(M, O)$ is defined to be a *process graph* or P-*graph* in short. The set of vertices of this directed graph is $M \cup O$, and the set of arcs is $A = A_1 \cup A_2$, where $A_1 = \{(X, Y) : Y = (\alpha, \beta) \in O \text{ and } X \in \alpha\}$, and $A_2 = \{(Y, X) : Y = (\alpha, \beta) \in O \text{ and } X \in \beta\}$.

Now, let $o \subseteq O$ and $S \subseteq M$ be arbitrary. Let us define the following functions on the sets $o$ and $S$:

$$mat^{in}(o) = \bigcup_{(\alpha,\beta)\in o} \alpha, \qquad mat^{out}(o) = \bigcup_{(\alpha,\beta)\in o} \beta,$$

$$mat(o) = mat^{in}(o) \cup mat^{out}(o),$$

and

$$\Delta(S) = \{u : \ u \in O \ \& \ S \cap mat^{out}(u) \neq \emptyset\}.$$

Let process graphs $(m, o)$ and $(M, O)$ be given. The P-graph $(m, o)$ is defined to be a *subgraph* of $(M, O)$, if $m \subseteq M$ and $o \subseteq O$.

Now, we can define the structural model of PNS for studying the problem from a structural point of view. For this reason, let $M^*$ be an arbitrarily fixed possibly infinite set, the set of the available materials. By *structural model* of PNS, we mean a triplet $\mathrm{M} = (P, R, O)$, where $P$, $R$, $O$ are finite sets, $\emptyset \neq P \subseteq M^*$ is the set of the *desired products*, $R \subseteq M^*$ is the set of the *raw materials*, and $O \subseteq \wp'(M^*) \times \wp'(M^*)$ is the set of the available operating units. It is assumed that $P \cap R = \emptyset$ and $M^* \cap O = \emptyset$, moreover, $\alpha$ and $\beta$ are finite sets for every $(\alpha, \beta) = u \in O$.

Then, the process graph $(M, O)$, where $M = \cup\{\alpha \cup \beta : (\alpha, \beta) \in O\}$, presents the interconnections among the operating units of $O$. Furthermore, every feasible process network, producing the given set $P$ of products from the given set $R$ of raw materials by using operating units from $O$, corresponds to a subgraph of $(M, O)$. Investigating the corresponding subgraphs of $(M, O)$, therefore, we can determine the feasible process networks. If we do not consider further constraints such as material balance, then the subgraphs of $(M, O)$ which can be assigned to the feasible process networks have common combinatorial properties. They are studied in [19] and their description is given by the following definition.

A subgraph $(m, o)$ of $(M, O)$ is called a *solution-structure* of $(P, R, O)$ if the following conditions are satisfied:

$(A1)$ $P \subseteq m$,

$(A2)$ $\forall X \in m$, $X \in R \Leftrightarrow$ no $(Y, X)$ arc in the process graph $(m, o)$,

$(A3)$ $\forall Y_0 \in o$, $\exists$ path $[Y_0, Y_n]$ in $(m, o)$ with $Y_n \in P$,

$(A4)$ $\forall X \in m$, $\exists(\alpha, \beta) \in o$ such that $X \in \alpha \cup \beta$.

The set of the solution-structures of $\mathbf{M} = (P, R, O)$ will be denoted by $S(P, R, O)$ or $S(\mathbf{M})$.

Checking the desired conditions $(A1), \ldots, (A4)$, one can prove the following statement, which is presented in [19].

**Proposition 2.1.1** *Let* $\mathbf{M} = (P, R, O)$ *be a structural model of* PNS. *If* $(m, o)$ *and* $(m', o')$ *are solution-structures of* $\mathbf{M}$, *then* $(m, o) \cup (m', o') = (m \cup m', o \cup o')$ *is also a solution-structure of* $\mathbf{M}$.

Proposition 2.1.1 indicates that the set $S(\mathbf{M})$ of the solution-structures is closed under the operation of finite union. Since $S(\mathbf{M})$ is a finite set, the union of all its elements also yields a solution-structure which is the greatest with respect to the subgraph ordering relation. The significance of the greatest element is revealed in the following definition. Let $\mathbf{M} = (P, R, O)$ be a structural model of PNS. By the *maximal structure* of $\mathbf{M}$ we mean the solution-structure defined as

$$\mu(\mathbf{M}) = \bigcup_{(m,o) \in S(\mathbf{M})} (m, o).$$

If $S(\mathbf{M}) = \emptyset$, then $\mu(\mathbf{M}) = \emptyset$; as such, $\mu(\mathbf{M})$ is termed *degenerate.*

The above definition obviously leads to the following observation.

**Remark 2.1.2** *For any structural model of PNS, the set of solution-structures is nonempty if and only if the maximal structure of this structural model is not degenerate.*

One basic question is how we can obtain the maximal structure from a structural model. In [20] and [21] a simple polynomial time algorithm is presented which can evaluate if $S(\mathbf{M})$ is empty, moreover, if $S(\mathbf{M})$ is not empty, the algorithm generates the corresponding maximal structure. Since we will not use this algorithm, we do not give the details here, we just note the following theorem.

**Theorem 2.1.3** *For the structural model of PNS, a polynomial time algorithm exists, which generates the maximal structure.*

To show the importance of the maximal structure, we need the following definition. For any structural models $\mathrm{M} = (P, R, O)$ and $\mathrm{M}' = (P', R', O')$, M is defined to be *equivalent* to M' if the sets of desired products as well as the sets of solution-structures are identical for these structural models, *i.e.* $P = P'$ and $S(\mathrm{M}) = S(\mathrm{M}')$.

Let $\mathrm{M} = (P, R, O)$ be an arbitrary structural model with $S(\mathrm{M}) \neq \emptyset$. For $\mu(\mathrm{M}) = (\bar{M}, \bar{O})$, let us form the triplet $\bar{\mathrm{M}} = (P, \bar{R}, \bar{O})$, where $\bar{R} = R \cap \bar{M}$. This structural model is called the *reduced structural model* of M. An important property of this reduced model that its P-graph is a solution-structure. For the reduced model the following statement is valid.

**Proposition 2.1.4** $\bar{\mathrm{M}} = (P, \bar{R}, \bar{O})$ *is a structural model of PNS, and it is equivalent to* M.

With regard to Proposition 2.1.4, note that the structural model $\bar{\mathrm{M}}$ depends only on the maximal structure of M. Therefore, by Theorem 2.1.3, we obtain the following observation.

**Theorem 2.1.5** *For any structural model, the reduced structural model can be generated by a polynomial time algorithm.*

## 2.2 PNS problems with weight

Let us consider structural models of PNS problems in which each operating unit has a weight. We are to find a feasible process network with the minimal weight where by weight of a process network we mean the sum of the weights of the operating units belonging to the process network under consideration. Each feasible process network in such a class of PNS problems is determined uniquely from the corresponding solution-structure and vice versa. Thus, the problem can be formalized as follows.

### PNS problem with weights

Let a structural model of PNS problem $\mathrm{M} = (P, R, O)$ be given. Moreover, let $w$ be a positive real-valued function defined on $O$, the weight function. The basic model is then

$$\min\{\sum_{u \in o} w(u) : (m, o) \in S(P, R, O)\}.$$

In what follows, by PNS problem we mean this optimization problem, and the solution-structures are called *feasible solutions.*

It is known (see [9]) that this PNS problem is NP-hard. We recall here the basic idea of the proof since we will use it later. First we have to define a particular class of PNS problems. A PNS problem is called a PNS1 *problem* if every material is a raw material or desired product; moreover, every operating unit produces desired products from raw materials. In [9], it is proven that the PNS1 problem is equivalent to the set covering problem. These problems are not only equivalent from the complexity theoretical point of view, but they have the same mathematical model.

Now, we define the set covering problem, which is a well-known NP-hard problem (see [23]). In a *set covering problem*, a finite set $I$ and a system of its subsets $P_1, \ldots, P_m$ are given, where each subset has some positive cost. A set of indices $J^* \subset \{1, \ldots, m\}$ is called a *cover* if $\cup\{P_j : j \in J^*\} = I$. By the *cost of a cover* we mean the sum of the costs of the subsets belonging to it. The problem is then to find a cover with minimal cost.

Now, we outline the equivalence proof of the PNS1 and set covering problems. Let $(P, R, O)$ be an arbitrary PNS1 problem with weight function $w$. Let $u_j = (\alpha_j, \beta_j) \in \wp'(R) \times \wp'(P)$, $j = 1, \ldots, n$, denote the operating units. Then, one can prove that this problem is equivalent to the set covering problem in which the basic set is $P$ and the system of its subsets is $\beta_j$, $j = 1, \ldots, n$, and the costs are $w'(\beta_j) = w(u_j)$, $j = 1, \ldots, n$, respectively.

For the other direction, consider an arbitrary set covering problem. Let $P$ be the basic set and $\beta_j$, $j = 1, \ldots, n$ be the system of its subsets with costs $w'(\beta_j)$, respectively. Let $R$ be an arbitrary set with $R \cap P = \emptyset$. Let us consider the operating units $u_j = (R, \beta_j)$, $j = 1, \ldots, n$, and the weight function $w(u_j) = w'(\beta_j)$, $j = 1, \ldots, n$. Then, it is easy to see that the PNS1 problem $(P, R, O)$, where $O = \{u_1, \ldots, u_n\}$, is equivalent to the set covering problem under consideration.

# Chapter 3

# Some well-solvable PNS classes

In the previous chapter it is proven that the PNS problem is NP-hard. In general, when a problem is NP-hard or NP-complete, then the studies of some particular classes can result in effective procedures for solving the instances of these classes. Well-solvable classes were investigated for many optimization problem, a nice overview on them can be found in [11]. For the PNS problem, the first well-solvable PNS classes are developed in [7], [29], and [30]. In this chapter we present these particular classes and the corresponding polynomial algorithms for solving their instances.

## 3.1 Turning back PNS problem

The first well-solvable class of PNS problems is the class of turning back PNS problems. This class is introduced in [29].

A PNS problem is called *turning back* if for every material, there exists at most one operating unit producing this material. This restriction yields so nice structure of the process graph which admits a linear time algorithm for solving the problem. This is comprised in the following statement.

**Theorem 3.1.1** ([29]) *If a PNS problem* $\mathbf{M} = (P, R, O)$ *is turning back, then the following algorithm decides whether the problem has a feasible solution, and if it does, then the procedure provides an optimal solution.*

**Algorithm 3.1.1**

*Initialization*

Set $A_0 = P$, $K_0 = R$, $M_0 = \emptyset$, $O_0 = \emptyset$, $r := 1$.

*Iteration* (*r-th iteration*)

- *Step 1.* Choose one material from $A_{r-1}$; it is denoted by $X$. Terminate if there is no operating unit producing $X$ directly; no feasible solution exists. Otherwise, let $u = (\alpha, \beta)$ denote the operating unit producing $X$ directly and proceed to Step 2.

- *Step 2.* Terminate if there exists a $Z \in R$ such that $Z \in \beta$; no feasible solution exists. Otherwise, proceed to Step 3.

- *Step 3.* Let $K_r = K_{r-1} \cup \{X\}$, $A_r = (A_{r-1} \cup \alpha) \setminus K_r$, $O_r = O_{r-1} \cup \{u\}$, and $M_r = M_{r-1} \cup \alpha \cup \beta$. If $A_r \neq \emptyset$, then set $r := r + 1$ and proceed to the succeeding iteration. If $A_r = \emptyset$, then proceed to Step 4.

- *Step 4.* Terminate; the P-graph $(\bar{M}, \bar{O})$ is an optimal feasible solution where $\bar{M} = M_r$ and $\bar{O} = O_r$.

*Proof:* To validate the procedure, first of all, let us observe that the procedure is executed in linear time. Indeed, by $|K_r| > |K_{r-1}|$, we have that after at most $|M|$ iterations the procedure terminates where $M$ denotes the set of the materials appearing in the problem.

Prior to proving the validity of the procedure, we show the following lemma.

**Lemma 3.1.2** *During the procedure, $A_r \subseteq m$ and $O_r \subseteq o$ are valid, for all $r$ and for any feasible solution $(m, o)$.*

*Proof:* The statement concerning $A_r$ is proven by induction on the number of the iteration steps. Since Step 1 and Step 2 do not change $A_r$, it is sufficient to investigate Step 3. Before the first iteration step, $A_0 = P$, and hence, the statement follows from condition $(A1)$ of the feasible solutions. Now, let $r \geq 1$, and let us suppose that the statement is valid before the $r$-th

iteration step. Then, we show that it is valid after the $r$-th iteration step as well. In the iteration step considered, an element $X$ is chosen from $A_{r-1}$. Then, $X$ is contained in every feasible solution by the induction hypothesis, furthermore, $R \subseteq K_{r-1}$ and $A_{r-1} \cap K_{r-1} = \emptyset$ imply $X \notin R$. Therefore, by condition $(A2)$, every feasible solution contains an operating unit which produces $X$ directly. Since the considered PNS problem is turning back, there is exactly one operating unit $u$ producing $X$ which is chosen by the procedure. Consequently, $u$ is contained in every feasible solution. But each feasible solution is a P-graph, and thus, it contains both the input and output materials of $u$. On the other hand, by the induction hypothesis, each feasible solution contains also $A_{r-1}$, and thus, from $A_r \subseteq A_{r-1} \cup \alpha$ it follows that the statement is valid for $r$ as well.

To prove the statement regarding $O_r$, let $(m, o)$ be an arbitrary feasible solution, and $u$ be such an operating unit which was involved in $O_r$ during the $r$-th iteration phase. Then, there exists an $X \in A_{r-1}$ which is an output material of $u$. On the other hand, by the statement concerning $A_r$, we have $X \in m$, furthermore, by the definition of $A_r$, $X \notin R$. Therefore, by condition $(A2)$, there exists an operating unit in $o$ which has $X$ as its output material. Since the considered PNS problem is turning back, there exists exactly one operating unit which has $X$ as output material, namely, the operating unit $u$ which is chosen by the procedure. Consequently, $u \in o$ which ends the proof of Lemma 3.1.2.

□

Using Lemma 3.1.2, we prove the correctness of the algorithm. Firstly, consider the case when the algorithm terminates in the first step of the $r$-th iteration. Then, it is shown that no feasible solution exists. To prove by contradiction, let us suppose that the algorithm terminates in the first step of the $r$-th iteration and the problem has a feasible solution denoted by $(m, o)$. Lemma 3.1.2 yields that $A_{r-1} \subseteq m$, and thus, $X \in m$ for the material $X$ chosen in the $r$th iteration phase. Furthermore, by $R \subseteq K_{r-1}$ and $A_{r-1} \cap K_{r-1} = \emptyset$, we have $X \notin R$. This observation and condition $(A2)$ imply that there is an operating unit producing $X$ which contradicts our assumption on the termination.

Now, let us suppose that the algorithm terminates in Step 2 of the $r$-th iteration. Then, it is shown that no feasible solution exists. For verification

by contradiction, let us assume that $(m, o)$ is a feasible solution. Then, in the light of Lemma 3.1.2, $u \in o$ is valid for the operating unit $u$ selected in the $r$-th iteration phase. On the other hand, $u$ has an output material $Z$ with $Z \in R$ by our assumption on termination. This yields, that there is an edge $(u, Z)$ in $(m, o)$ where $Z \in R$, which contradicts condition $(A2)$.

Finally, we prove that the procedure provides an optimal feasible solution if it terminates in Step 4 of the $r$-th iteration phase. First, we show that the algorithm provides a P-graph $(\bar{M}, \bar{O})$ which is a feasible solution.

During the procedure, if an operating unit $u$ is involved into the set $O_r$, then we put its input and output materials into $M_r$, and thus, $\bar{O} \subseteq \wp'(\bar{M}) \times \wp'(\bar{M})$. On the other hand, $\bar{O} \neq \emptyset$ from our assumption. Consequently, $\bar{M}$ and $\bar{O}$ determine a P-graph. Therefore, we have to prove conditions $(A1)$, $(A2)$, $(A3)$, $(A4)$.

According to the procedure, in Step 3, we take the material chosen from $A_{r-1}$ into set $M_r$, and $K_r$ contains only materials from $P$ which are in $M_r$. These observations give that starting with $A_0 = P$, $A_r = \emptyset$ implies $P \subseteq M_r$. In addition, $\bar{M} = M_r$, thereby resulting in the validity of condition $(A1)$.

There is no operating unit in $\bar{O}$ with output material from $R$ since the procedure does not terminate in Step 2. This implies that there is no $(Y, X)$ arc in $(\bar{M}, \bar{O})$ if $X \in R$ and $X \in \bar{M}$. On the other hand, we put each material $X \notin R$ which is placed into $M_r$ and not produced by any operating unit from $O_r$ into $A_r$ in Step 3. Thus, if $A_r = \emptyset$, then each material from $M_r$ which is not a raw material is produced by an operating unit from $O_r$. The observations above provide the validity of condition $(A2)$.

Condition $(A3)$ is proven by induction. If $r = 1$, then $O_1$ contains only one operating unit which has an output material from $P$, and thus, condition $(A3)$ is satisfied. Now, let $r > 1$, and let us suppose that condition $(A3)$ is true for $O_{r-1}$. We show that condition $(A3)$ is valid for $O_r$ if the procedure has not yet been terminated in the $r-1$-th iteration phase. Since $O_r = O_{r-1} \cup \{u\}$, and by our induction hypothesis, condition $(A3)$ is true for $O_{r-1}$, it is enough to prove that there exists a path $[u, Z]$ in $(\bar{M}, \bar{O})$ with $Z \in P$ for the operating unit $u$ selected in the $r$-th iteration phase of the procedure. The operating unit $u$ has the output material $X$ from $A_{r-1}$. On the other hand, $X \in A_{r-1}$ is possible only if $X \in P$ or $X$ is an input material for some operating unit $v \in O_{r-1}$. In the first case, $[u, X]$ is a suitable path in $(\bar{M}, \bar{O})$. In the second



case, $(\bar{M}, \bar{O})$ contains a path $[v, Z]$ with $Z \in P$ by the induction hypothesis. Equipping this path with $u$ and $X$, we again get a suitable $[u, Z]$ path.

Finally, since we put only materials into $M_r$ which are input or output materials of the operating units from $O_r$, the validity of condition $(A4)$ is obvious.

Hence, we have proven that the algorithm gives a feasible solution. Now, it is shown that this feasible solution is optimal. Since the weight of each operating unit is positive, it is enough to prove that each operating unit from $\bar{O}$ is contained in every feasible solution. On the other hand, this statement is an obvious consequence of Lemma 3.1.2.

□

## 3.2 Hierarchical PNS problems

In this section we present our results on the hierarchical PNS problems which are introduced in [7]. These results were developed by a joint research with the coauthors, and it is impossible to separate the parts which belong to the author. The structure of these problems makes it possible to solve further particular classes by polynomial algorithms.

Let $l$ be a positive integer. A PNS problem $\mathbf{M} = (P, R, O)$ is called *$l$-hierarchical* if there exist a partition $M_0 = R, \ldots, M_l = P$ of $M$ and a partition $O_1, \ldots, O_l$ of $O$ that for each $i$, $i = 1, \ldots, l$, $O_i$ contains operating units which have all input materials from $M_{i-1}$ and output materials from $M_i$. If a PNS problem is $l$-hierarchical for some $l$, then it is called *hierarchical*. Such hierarchical PNS problems, which are thin in the sense that the size of $O_i$, $i = 1, \ldots, l$, and the size of $M_i$, $i = 1, \ldots, l$ are bounded by a fixed constant, are well-solvable. To form this statement more precisely, we need the following definitions. A PNS problem is called *$k$-wide $l$-hierarchical* if it is an $l$-hierarchical problem, moreover $|M_i| \leq k$ is valid for $i = 0, \ldots, l$; and $|O_i| \leq k$ holds for $i = 1, \ldots, l$.

**Theorem 3.2.1** ([7]) *If a PNS problem $\mathbf{M} = (P, R, O)$ is $k$-wide $l$-hierarchical, then the following algorithm gives an optimal feasible solution of the problem or it gives that the problem has no feasible solution. The time complexity of*

*the algorithm is $\mathcal{O}(l)$, where the constant in $\mathcal{O}$ may depend on $k$ exponentially.*

**Algorithm 3.2.2**
**Subprocedure 1** *(Computing functions $F_i$ and $G_i$)*

- *Initialization.* Let $N$ be a fixed number greater than $|O| \cdot q$, where $q$ denotes the maximum of the weights of the operating units.
- *Part 0.* Let $G_0(S) = 0$ and $F_0(S) = \emptyset$, for all $S \subseteq M_0$.
- *Part $i$ ($i = 1, \ldots, l$).*
  - *Step 1.1.* If there exists a set $S \subseteq M_i$ for which the functions $F_i$ and $G_i$ have not yet been determined, then choose one of them and perform the following steps for it. Otherwise, proceed to the $i+1$-th part if $i < l$, and proceed to Subprocedure 2 if $i = l$.
  - *Step 1.2.* Consider the subset $\Delta(S)$ of $O_i$, and for every set $Q \subseteq \Delta(S)$, investigate the validity of $S \subseteq mat^{out}(Q)$. If this relation is false for every $Q$, then proceed to Step 1.4. Otherwise, let the sets satisfying the relation above be denoted by $Q_1, \ldots, Q_t$ and proceed to Step 1.3.
  - *Step 1.3.* For every $Q_j$, $j = 1, \ldots, t$, calculate the following value:

    $$c_j = G_{i-1}(mat^{in}(Q_j)) + \sum_{u \in Q_j} w(u).$$

    Let us denote a set with minimal value by $Q_j$. If there are more sets with the same minimal value, then choose the set which has the smallest index. Furthermore, let $F_i(S) = Q_j$, $G_i(S) = c_j$, and proceed to Step 1.1.
  - *Step 1.4.* Let $F_i(S) = \emptyset$, $G_i(S) = N$, and proceed to Step 1.1.

**Subprocedure 2** *(Finding an optimal solution)*

- *Initialization.* If $G_l(P) \geq N$, then terminate; the problem has no feasible solution. Otherwise, let $A_0 = P$, $\bar{O}_0 = \emptyset$, and $r = 1$.

- *Iteration (r-th iteration)*

  - *Step 2.1.* Let
    $\bar{O}_r = \bar{O}_{r-1} \cup F_{l+1-r}(A_{r-1})$, $A_r = mat^{in}(F_{l+1-r}(A_{r-1}))$.
    If $r = l$, then proceed to Step 2.2, otherwise let $r := r + 1$ and proceed to the next iteration.

  - *Step 2.2.* Terminate; the optimal solution is the P-graph $(m, o)$, where $o = \bar{O}_l$, and $m = mat(o)$.

*Proof:* First, we prove that if the algorithm gives a solution, then the produced sets $m, o$ yield a P-graph, which is a feasible solution. By the definition of $m$, it is obvious that for the sets $m, o$, the P-graph $(m, o)$ exists and satisfies condition $(A4)$. Let us observe that if $i < l$, then for each element of $A_i$, there exists an operating unit in $o$ producing it. This observation follows from the definition of the functions $F_j$, $j = 1, \ldots, l$. Thus, by $A_0 = P$, we have that $(m, o)$ satisfies condition $(A1)$. Since in a hierarchical PNS problem there is no operating unit producing raw material, we get that in $(m, o)$ there is no edge leading into a raw material. To prove the second part of condition $(A2)$, let $X \in m$ be a material with $X \notin R$. Since $X \in m$, thus $X$ is an output or input material of some operating unit from $o$. In the first case, we get by the definition of the P-graph that there exists an edge leading into $X$. In the second case, let $u \in o$ be an operating unit which has $X$ as an input material. Since $u \in o$, there exists an index $r$ for which $u \in F_{l+1-r}(A_{r-1})$. This gives that $X \in A_r$. On the other hand, by induction on the number of iterations it is easy to see that $A_i \subseteq M_{l-i}$, for all $i$, $i = 0, \ldots, l$. This observation results in $r \neq l$. Thus, $X \in A_i$ for some $i < l$, which yields that there exists an edge in $(m, o)$ leading into it. Consequently, condition $(A2)$ is also valid for $(m, o)$. To prove condition $(A3)$, it is enough to show that for each operating unit from $\bar{O}_i$, $i = 1, \ldots, l$, there exists a path in $(m, o)$ leading from it into a desired product. We prove this statement by induction on $i$. For the case $i = 1$, we have $A_0 = P$, thus, by the definition of the function $F_l$, the validity of the statement follows. Now, let $1 \leq i < l$, and

let us suppose that the statement is valid for $i$. We show that it is also valid for $i+1$. Since $\bar{O}_{i+1} = \bar{O}_i \cup F_{l+1-(i+1)}(A_i)$, thus, by the induction hypothesis, it is enough to prove the statement for the operating units contained in $F_{l+1-(i+1)}(A_i)$. Let $u \in F_{l+1-(i+1)}(A_i)$ be arbitrary. By the definition of the function $F_{l+1-(i+1)}$, we obtain that $u$ has an output material from the set $A_i$. (Otherwise, during Step 1.2, $F_{l+1-(i+1)}(A_i) \subseteq \Delta(A_i)$ is not valid which is a contradiction.) Let such a material be denoted by $Z$. By the definition of $A_i$, it follows that $Z$ is an input material of some operating unit $v \in \bar{O}_i$. Then, by the induction hypothesis, there exists a path $[v, Y]$ in $(m, o)$ where $Y$ is a desired product. If we complete the beginning of this path with $u$ and $Z$, we get a path in $(m, o)$ leading from $u$ into the desired product $Y$. Thus, we have proven our statement for $i+1$, which yields that condition $(A3)$ is valid for the P-graph $(m, o)$. Consequently, the P-graph determined by the algorithm is a feasible solution.

Now, we prove the correctness of the procedure. To do this, first we show the following statement concerning $G_l$.

**Lemma 3.2.2** *For every feasible solution, the weight of the feasible solution is at least $G_l(P)$.*

*Proof:* Let $(m, o)$ be an arbitrary feasible solution of the problem. Let $o_i = O_i \cap o$, for $i = 1, \ldots, l$. Since $(m, o)$ is a feasible solution and the materials of $P$ can be produced only by operating units from $O_l$, by conditions $(A1)$ and $(A2)$, we have that $P \subseteq mat^{out}(o_l)$. The definition of the function $G_l$ and this observation yield the following inequality:

$$G_l(P) \leq G_{l-1}(mat^{in}(o_l)) + \sum_{u \in o_l} w(u).$$

On the other hand, $(m, o)$ is a feasible solution, thus $mat^{in}(o_l) \subseteq m$. The input materials of the operating units from $o_l$ are in the set $M_{l-1}$, thus if $l \neq 1$, then they are not contained in $R$. This yields that for each of them, there exists an operating unit in $o$ having it as an output material. Furthermore, the problem is hierarchical, and hence, the materials from the set $M_{l-1}$ are produced only by operating units from $O_{l-1}$. These observations yield that

$mat^{in}(o_l) \subseteq mat^{out}(o_{l-1})$. This relation and the definition of the function $G_{l-1}$ imply the following inequality:

$$G_{l-1}(mat^{in}(o_l)) \leq G_{l-2}(mat^{in}(o_{l-1})) + \sum_{u \in o_{l-1}} w(u).$$

In the same way as above, we obtain that the following inequality is valid, for all $i$ , $i = 1, \ldots, l-1$:

$$G_i(mat^{in}(o_{i+1})) \leq G_{i-1}(mat^{in}(o_i)) + \sum_{u \in o_i} w(u).$$

Summarizing the obtained inequalities, by $G_0(S) = 0$, we get the following inequality:

$$G_l(P) \leq \sum_{i=1}^{l} \sum_{u \in o_i} w(u),$$

which gives the required result.

□

By Lemma 3.2.2, we can prove the correctness of the procedure.

First, we prove that there is no feasible solution of the problem if $G_l(P) \geq N$. Contrary, let us suppose that there is a feasible solution of the problem. Let us denote the weight of this solution by $K$. By the definition of $N$, we have $N > K$. On the other hand, Lemma 3.2.2 states that $G_l(P) \leq K$, which results in the contradiction $N > N$.

Now, we show that the feasible solution produced by the algorithm is optimal if $G_l(P) < N$. First, let us observe that the weight of the produced feasible solution is $G_l(P)$. This observation follows immediately from the construction of the algorithm. Thus, by Lemma 3.2.2, we obtain that the weight of any feasible solution is at least as large as the weight of the produced feasible solution, which means that we get an optimal solution.

Let us investigate the time complexity of the procedure. In Subprocedure 1, we perform $l$ parts. During a part, we examine every subsets of $\Delta(S)$, for each subset $S$ of $M_i$. Since the problem is $k$-wide $l$-hierarchical, $M_i$ has at most $2^k$ subsets, and since for each such subset $S$, $\Delta(S) \subseteq O_i$, thus, $\Delta(S)$ can have only $2^k$ subsets. Consequently, we obtain that the number of operations performed in each iteration is independent of the size of the problem (it

depends only on $k$). In Subprocedure 2, which is based on functions $F_i$ and $G_i$, we perform $l$ iterations, and the number of operations in each iteration is a constant. This yields that the number of operations performed by the procedure is bounded by $\mathcal{O}(l)$.

□

Thus, for each fixed $k$, the above algorithm solves any $k$-wide hierarchical problem in linear time. However, we have to note that the constant in the complexity of the algorithm is exponential in $k$. This shows that our procedure can be really effective only for small $k$.

Investigating the presented algorithm, one can conclude an interesting observation on the solvability of the hierarchical PNS problems.

**Corollary 3.2.3** *For a hierarchical PNS problem, if every material, distinct from the raw materials, is produced by some operating units, then the problem has a feasible solution.*

*Proof:* Let us perform the algorithm for the problem. By the above assumption, we obtain that $S \subseteq mat^{out}(\Delta(S))$ for each subset $S$ of materials, which gives that Step 1.4 is not performed in Subprocedure 1. This yields that $G_l(P) < N$, and then the problem has a feasible solution.

□

## 3.3 Enlarging and $c$-ordered PNS problems

To close this chapter, we present a new method called *enlarging*, which is introduced in [30]. The idea of this method is to enlarge the problem with new dummy operating units having weight 0. However, this is not allowed in our model since each operating unit must have a positive weight. But if we assign a sufficiently small weight to the new operating units, then we can obtain the same result. By this method, we prove that more difficult PNS problems are reducible to hierarchical PNS problems. First, we introduce the following definitions.

A PNS problem $\mathrm{M} = (P, R, O)$ is called *reducible* to $\mathrm{M}' = (P', R', O')$ if there is a bijective mapping $\phi$ from $S(\mathrm{M})$ onto $S(\mathrm{M}')$ that the image

under $\phi^{-1}$ of any optimal solution of M' is an optimal solution of M. A PNS problem is called *cycle free* if the P-graph of the problem does not contain any directed cycle. Furthermore, a problem is called *integer* if every operating unit has a positive integer weight. Then, the following statement is valid.

**Theorem 3.3.1** ([30]) *Every integer cycle free PNS problem* $\mathbf{M} = (P, R, O)$ *is reducible to a hierarchical PNS problem.*

*Proof:* Let us consider an integer cycle free PNS problem. Without loss of generality, we may assume that the model is a reduced structural model since if it has a feasible solution, then by Theorem 2.1.5, one can obtain the equivalent reduced model. The case when the problem has no feasible solution is not interesting, since this fact also turns out by the maximal structure generation algorithm. Let the P-graph of our problem be $(M, O)$. Since we have a reduced model, $(M, O)$ is a feasible solution. Now, we construct a suitable hierarchical PNS problem in two phases.

**Phase 1.** *(Classifying)* In this phase, we do not change the model, we just order the sets of the materials and operating units in an appropriate way. Let $M_0$ be the set of the raw materials. Furthermore, let $O_1$ be the set of the operating units whose input materials are from $M_0$. At the $i$-th step of the ordering, when sets $M_0, \ldots, M_{i-1}$ and $O_1, \ldots, O_i$ are already determined, we construct the sets $M_i$ and $O_{i+1}$ as follows. Let $M_i$ be the set of the materials which are produced only by operating units from $O_1 \cup \ldots \cup O_i$, and which are not contained in $M_0 \cup \ldots \cup M_{i-1}$. If $\cup_{j=1}^{i} M_i = M$, then the ordering is finished, otherwise, let $O_{i+1}$ be the set of the operating units which have each input materials in $M_0 \cup \ldots \cup M_i$, and which are not contained in $O_1 \cup \ldots \cup O_i$. Some properties of this ordering are summarized in the following statement.

**Lemma 3.3.2** *The classifying is finished in a finite number of steps. Furthermore, if the ordering is finished after $l$ steps, the sets $M_0, \ldots, M_l$ and $O_1, \ldots, O_l$ form partitions of $M$ and $O$, respectively. Finally, every edge in $(M, O)$ leading from an operating unit into a material leads from $O_i$ into $M_j$ with $i \leq j$ for some $i, j \in \{1, \ldots, l\}$, and every edge in $(M, O)$ leading from a material into an operating unit leads from set $M_i$ into set $O_j$ with $i < j$ for some $i \in \{0, 1, \ldots l-1\}$ and $j \in \{1, \ldots, l\}$.*

*Proof:* The classifying phase of the algorithm is based on the same idea as the well-known procedure to assign indices to the vertices of a cycle free directed graph in a way that the starting vertex of every arc has index smaller than the ending vertex (see [33]). The difference is that our procedure does not choose only one source, but all of them in each step. The proof of Lemma 3.3.2 can be done in the same way as the correctness proof of this well-known procedure. Therefore, we omit the details here.

□

**Phase 2.** *(Enlarging)* First, we extend the set of the materials with a new material $P'$ and the set of the operating units with a new operating unit $u$, which has the original desired products as its input materials and $P'$ as its output material, furthermore, let the new desired product be $P'$. Let the weight of the new operating unit be 1. One can easily see that the original problem is reducible to this new problem $\mathrm{M}' = (\{P'\}, R, O \cup \{u\})$. (From a feasible solution of the original problem we can obtain a feasible solution of the new problem completing the first feasible solution with both the new material and new operating unit. On the other hand, from a feasible solution of the new problem we can obtain a feasible solution of the original problem deleting from the first feasible solution both the new material and new operating unit.) Furthermore, if we extend the partitions with $O_{l+1}$ and $M_{l+1}$ where $O_{l+1}$ contains the new operating unit, $M_{l+1}$ contains the new desired product, then the properties presented in Lemma 3.3.2 still hold. Now, we construct a hierarchical PNS problem to which this problem is reducible. For the edges in $(M, O)$, we define some new operating units and some new materials as follows. Consider first the edges leading from a material into an operating unit. Let $(X, Y)$ be such an edge. Then, by Lemma 3.3.2, $X \in M_i$ and $Y \in O_j$, where $j > i$. We do not define new operating units and materials if $j = i + 1$. Otherwise, we define $j - i - 1$ new operating units and $j-i-1$ new materials; let us denote them by $u(X, Y, i+1), \ldots, u(X, Y, j-1)$ and $A(X, Y, i + 1), \ldots, A(X, Y, j - 1)$, respectively. If $i + 1 < p \leq j - 1$, then let the input and output of $u(X, Y, p)$ be $A(X, Y, p - 1)$ and $A(X, Y, p)$, respectively, moreover, let $u(X, Y, i + 1) = (\{X\}, \{A(X, Y, i + 1)\}$. Furthermore, let us change the input set of $Y$ replacing $X$ by $A(X, Y, j - 1)$. In other words, replace the arc $(X, Y)$ by the path which contains the vertices $X, u(X, Y, i+1), A(X, Y, i+1), u(X, Y, i+2), \ldots, A(X, Y, j-1), Y$. Consider now the edges leading from an operating unit into a material. Let $(Y, X)$ be

such an edge. Then, by Lemma 3.3.2, $Y \in O_i$ and $X \in M_j$, where $i \leq j$. We do not define new operating units and materials if $j = i$. Otherwise, we define $j-i$ new operating units and $j-i$ new materials, let us denote them by $u(Y,X,i+1),\ldots,u(Y,X,j)$ and $A(Y,X,i),\ldots,A(Y,X,j-1)$, respectively. Let the input and output of $u(Y,X,p)$ be $A(Y,X,p-1)$ and $A(Y,X,p)$, respectively if $i+1 \leq p < j$, moreover, let $u(Y,X,j) = (\{A(Y,X,j-1)\},\{X\})$. Furthermore, let us change the output set of $Y$ replacing $X$ by $A(Y,X,i)$. In other words, replace the arc $(Y,X)$ by the path which contains the vertices $Y, A(Y,X,i), u(Y,X,i+1), A(X,Y,i+1),\ldots,u(Y,X,j), X$. Finally, let us assign the weight $\varepsilon = 1/(|M| \cdot |O| \cdot l)$ to each new operating unit. Let $\bar{M}$ be the set containing both the new materials and original materials, furthermore, let $\bar{O}$ be the set containing the new operating units and the original operating units. (If we changed the input or the output set of an original operating unit, then here we consider the modified operating unit.) In such a way, we obtain a new PNS model having the same raw materials and desired products as $\mathbf{M}'$. This is $\bar{\mathbf{M}} = (\{P'\}, R, \bar{O})$ and by equipping it with the defined weights, we obtain a hierarchical PNS problem as the following lemma shows.

**Lemma 3.3.3** *The new PNS problem defined above is hierarchical.*

*Proof:* For every $i$, $i = 0,1,\ldots,l+1$, let $N_i$ be the set of the new materials which are of the form $A(X,Y,i)$ or $A(Y,X,i)$, and let $\bar{M}_i = M_i \cup N_i$. On the other hand, for every $j$, $j = 1,\ldots,l$, let $U_j$ be the set of the new operating units which are of the form $u(X,Y,j)$ or $u(Y,X,j)$, and let $\bar{O}_j = O'_j \cup U_j$, where $O'_j$ contains the original or the modified version of the original operating units from $O_j$. Using sets $\bar{M}_0,\ldots,\bar{M}_{l+1}$ and $\bar{O}_1,\ldots,\bar{O}_{l+1}$, it follows immediately by the definition that the new problem is hierarchical.

□

Furthermore, the original PNS problem is reducible to this new hierarchical one as the following lemma states.

**Lemma 3.3.4** *The original PNS problem belonging to* $\mathbf{M}$ *is reducible to the new hierarchical PNS problem.*

*Proof:* Let us define a mapping $\varphi$ on $S(\mathbf{M})$ as follows. For every feasible solution $(m, o) \in S(\mathbf{M})$, let us assign to $(m, o)$ the subgraph $(\bar{m}, \bar{o})$ of $(\bar{M}, \bar{O})$, which we obtain by performing the enlarging on $(m, o)$. From the definition of the enlarging, it follows immediately that the subgraph $(\bar{m}, \bar{o})$ satisfies conditions $(A1), (A2), (A3), (A4)$, thus, the obtained subgraph is an element of $S(\bar{\mathbf{M}})$. On the other hand, since each new material is produced by only one operating unit in $\bar{\mathbf{M}}$, it follows by condition $(A2)$ that if a feasible solution of $\bar{\mathbf{M}}$ contains a new operating unit defined in the case of an edge $(X, Y)$, then it contains all of the new operating units and materials defined by considering $(X, Y)$. This yields that every element $(\bar{m}, \bar{o})$ of $S(\bar{\mathbf{M}})$ can be obtained by performing the enlarging on a suitable subgraphs of $(M, O)$. We can obtain this subgraph if we perform the inverse process of the enlarging on $(\bar{m}, \bar{o})$, we replace the sets of the new operating units and materials by the original edges and delete operating unit $u$ and material $P'$. Checking conditions $(A1), (A2), (A3), (A4)$, it is easy to see that the subgraph of $(M, O)$ obtained in such a way is a feasible solution of $\mathbf{M}$. Therefore, the mapping $\varphi$ defined above is a bijective mapping of $S(\mathbf{M})$ onto $S(\bar{\mathbf{M}})$. Finally, we prove that the image of an optimal solution of $\bar{\mathbf{M}}$ under $\varphi^{-1}$ is an optimal solution of $\mathbf{M}$. Let $(\bar{m}_1, \bar{o}_1)$ be an optimal solution of $\bar{\mathbf{M}}$, and let its image under $\varphi^{-1}$ be $(m_1, o_1)$. Then, $(m_1, o_1)$ is a feasible solution of $\mathbf{M}$. Let us suppose that it is not optimal. This yields that there exists a feasible solution of $\mathbf{M}$, denoted by $(m_2, o_2)$, which has smaller weight. Let $(\bar{m}_2, \bar{o}_2)$ be the image of $(m_2, o_2)$ under $\varphi$. Then, it is a feasible solution of $\bar{\mathbf{M}}$. On the other hand, since the number of the new operating units is smaller than $|M| \cdot |O| \cdot l$ and $w(u) = 1$, we get that $w(\bar{m}_2, \bar{o}_2) < w(m_2, o_2) + 2$. Furthermore, since the original problem is integer, $w(\bar{m}_1, \bar{o}_1) \geq w(m_1, o_1) + 1 \geq w(m_2, o_2) + 2$. Therefore, we obtain that $w(\bar{m}_2, \bar{o}_2) < w(\bar{m}_1, \bar{o}_1)$, which contradicts the optimality of $(\bar{m}_1, \bar{o}_1)$.

We have to mention the following remarks regarding Theorem 3.3.1.

**Remark 3.3.5** *The assumption that the PNS problem is integer is not a strong restriction from practical point of view. We can obtain integer PNS problems if we choose a sufficiently small unit of weight.*

**Remark 3.3.6** *If we consider a more general model in which some operating units can have* 0 *weights, then the integer assumption is not necessary since it is only used to define a sufficiently small weights for the new operating units.*

*This more general model is considered in* [30], *the results of this section are presented there without the integer assumption.*

**Remark 3.3.7** *Unfortunately, in the general case, the constructed hierarchical problem can be very wide, thus, the algorithm for solving hierarchical problems can be not effective enough to solve the general problem.*

However, if we make some further assumptions, then the problem constructed above is $k$-wide hierarchical for a constant $k$. Let us call a PNS problem $(c,l)$-*ordered* if there exist such partitions $M_0 = R, \ldots, M_l = P$ and $O_1, \ldots, O_l$, of $M$ and $O$, respectively, that $O_i$ contains only operating units which have input materials from $\cup\{M_j : i-c \leq j < i\}$ and output materials from $\cup\{M_j : i \leq j < i+c\}$, for every $i$, $i = 1, \ldots, l$. Furthermore, a $(c,l)$-ordered problem is called to be $k$-*wide* if $|M_i| \leq k$ and $|O_j| \leq k$ are valid for $i = 0, \ldots, l$, $j = 1, \ldots, l$. The following statement holds for the $(c,l)$-ordered problems.

**Proposition 3.3.8** *An integer $k_1$-wide $(c,l)$-ordered problem is reducible to a $k$-wide $l$-hierarchical problem where $k = k_1 + k_1^2(c-1)c$.*

*Proof:* Let us consider an arbitrary integer $k_1$-wide $(c,l)$-ordered problem, and denote it by $\mathbf{M} = (P,R,O)$. The problem considered is an integer cycle free problem, thus, we can reduce it to a hierarchical problem as it is presented in the proof of Theorem 3.3.1. Let us observe that the first part of the enlarging is necessary only to ensure that $M_{l+1}$ contains the desired products. On the other hand, in a $(c,l)$-ordered problem $M_l = P$, thus in case of ordered problems we can omit this part. Let the new PNS problem be $\bar{\mathbf{M}} = (P,R,\bar{O})$. We know that this problem is hierarchical with the sets $\bar{M}_j = M_j \cup N_j$, $j = 0, \ldots, l$, $\bar{O}_i = O_i' \cup U_i$, $i = 1, \ldots, l$, presented in Lemma 3.3.3. Let us consider the number of elements in $U_i$. We define a new operating unit which is placed into $U_i$ for each edge $(X,Y)$, where $X \in M_p$ and $Y \in O_q$ with $p < i < q$, and for each edge $(Y,X)$, where $Y \in O_q$ and $X \in M_p$ with $q < i \leq p$. Since the problem is $c$-ordered, it is easy to see that the pair $(q,p)$ can get in both cases at most $(c-1)c/2$ possible values. Furthermore, since the problem is $k_1$-wide, we can choose the edge $(X,Y)$ or $(Y,X)$ in ${k_1}^2$ ways for a fixed pair $p,q$. This yields that $|U_i| \leq k_1^2(c-1)c$.

Therefore, $|\bar{O}_i| \leq k_1 + k_1^2(c-1)c$. One can prove the same inequality for $|\bar{M}_j|$ in a similar way. This ends the proof of Proposition 3.3.8.

□

By Proposition 3.3.8 and by investigating the time complexity of the enlarging procedure, we can conclude the following result.

**Theorem 3.3.9** ([30]) *An integer $k_1$-wide $(c, l)$-ordered problem can be solved by an algorithm having time complexity $\mathcal{O}(l \cdot |M| \cdot |O|)$, where the constant may depend on $k_1 + k_1^2(c-1)c$ exponentially.*

*Proof:* First, we may construct the hierarchical problem of Theorem 3.3.1. Since the problem is in ordered form, we need not perform the classifying part. Furthermore, as it is mentioned in the proof of Proposition 3.3.8, we can also omit the first part of the enlarging phase. Therefore, to construct the suitable hierarchical problem, we have only to replace the edges with the new operating units and materials. This procedure can be performed in $2 \cdot l \cdot |M| \cdot |O|$ time. By Proposition 3.3.8, the obtained problem is $k_1 + k_1^2(c-1)c$-wide, thus, by Theorem 3.2.1, it can be solved by a $C \cdot l$ time algorithm where $C$ is a constant depending on $k_1 + k_1^2(c-1)c$ exponentially. Finally, from the optimal solution of the new problem we obtain the optimal solution of the original problem in at most $2 \cdot l \cdot |M| \cdot |O|$ time, by the inverse process of the enlarging.

□

# Chapter 4

# Heuristic algorithms for the PNS problem

For NP-hard problems, the construction and analysis of heuristic algorithms is a rapidly developing area. By heuristic algorithms we mean fast (polynomial time) algorithms which do not guarantee an optimal solution in general, but always result in a feasible solution which is close to the optimal solution in some sense. One can find more details on heuristic algorithms in [23] and [36].

For the PNS problem, heuristic algorithms have not yet been earlier studied. The first heuristic algorithms for the PNS problem are developed in [8]. This paper also contains an empirical analysis which belongs to the coauthors, the theoretical part contains mainly the results of the author. We present these algorithms in Section 4.1, and some theoretical worst-case bounds are proven for them in Section 4.2.

Throughout this chapter we will consider PNS problems which are in reduced form. By Theorem 2.1.5 we can do it without loosing the generality.

## 4.1 The heuristic algorithms

The basic ideas of both algorithms are the same. They can be considered as the generalizations of the well-known Chvatal's algorithm (see [14]) for the set covering problem. Both algorithms use a cost function defined on the materials, which gives a lower bound on the producing cost of the material

considered. By this function, the algorithms select operating units step by step, which finally form a feasible solution. First, we define the cost function, and then the algorithms are presented.

### 4.1.1 Definition of the cost function

The general definition of the cost function, denoted by $c$, is given in [25]. This definition is long and difficult. Moreover, in the worst-case bound proofs, only cycle free subclasses are investigated, and therefore, the general definition of the function $c$ is not recalled here. On the other hand, the definition of this function is very simple for the cycle free case. Thus, we present this particular definition here. We define the function $c$ by presenting the algorithm which determines it. Before the description of the algorithm, we outline its basic idea. In each iteration step, we have two sets $I^{(r)}$ and $J^{(r)}$. $I^{(r)}$ is the set of materials for which the costs have already been determined and $J^{(r)}$ is its complement. Initially, $I^{(0)}$ contains the raw materials. In the $r$-th step, we select a material $X$ from $J^{(r)}$ which is produced only by operating units having all inputs from the set $I^{(r)}$. We determine the cost for $X$ and move $X$ from $J^{(r)}$ into $I^{(r)}$. At the beginning of the procedure, zero is assigned as the cost of the elements of $I^{(0)}$. Now, the algorithm determining the function $c$ is presented in details.

**Algorithm 4.1.1**

- *Initialization*

  Set $I^{(0)} = R$, $J^{(0)} = M \setminus R$, $r = 0$,

  and $c(X) = 0$, for all $X \in M$.

- *Iteration ($r$-th iteration)*

  Terminate if $J^{(r)} = \emptyset$. Otherwise, choose a material $X \in J^{(r)}$, for which the input materials of all the operating units $u_1, \ldots, u_j$ producing $X$ directly are in $I^{(r-1)}$. For every $u_t$, calculate the value

$$c_t = w(u_t) + \max\{c(V) : V \in mat^{in}(u_t)\},$$

  and let $c(X)$ be the minimal $c_t$ value, moreover, let

$I^{(r+1)} = I^{(r)} \cup \{X\}$, $J^{(r+1)} = J^{(r)} \setminus \{X\}$. Set $r := r + 1$ and proceed to the next iteration.

To verify the correctness of the algorithm, we have to show that if $J^{(r)} \neq \emptyset$, then there exists an $X \in J^{(r)}$ for which the input materials of all the operating units $u_1, \ldots, u_j$ producing $X$ directly are in $I^{(r)}$. We prove this statement by contradiction. Let us suppose that for every $X \in J^{(r)}$, an operating unit exists which produces $X$ and has input material in $J^{(r)}$. Let us choose a material $A_0$ from $J^{(r)}$. Then, we have an operating unit $A_1$ that produces $A_0$ and has an input material $A_2 \in J^{(r)}$. The same statement is valid for $A_2$. Hence, we have an operating unit $A_3$ which produces $A_2$ and has an input material $A_4 \in J^{(r)}$. If we continue this list, we obtain a sequence of materials $A_0, A_2, \ldots$ and operating units $A_1, A_3, \ldots$ for which $A_{2i}$ and $A_{2i+2}$ are output and input materials of $A_{2i+1}$, respectively. On the other hand, the set of materials is finite, and consequently, $A_{2l} = A_{2k}$ for some $k > l$. Then, $A_{2k}, A_{2k-1}, \ldots, A_{2l+1}, A_{2l}$ is a cycle in the P-graph, which is a contradiction. Therefore, the algorithm is correct and it determines a nonnegative function $c$ in a finite number of steps.

Now, we can present our heuristic algorithms.

### 4.1.2 The algorithms

The algorithms select one operating unit in each iteration step. The difference between the two algorithms is in the rule for selecting the operating unit. The algorithms work with two sets, the *set of the selected operating units* and the *set of the required materials.* At the beginning of the procedure, the set of the selected operating units is empty, and the set of the required materials is $P$. Later, in each iteration step, we extend the set of the selected operating units with one operating unit and delete the output materials of this operating unit from the set of the required materials. Moreover, every input material of the operating unit considered, which is neither raw material nor input material of any of the selected operating units, is placed into the set of the required materials. The procedure terminates when the set of the required materials becomes empty. We obtain the feasible solution $(m, o)$, where $o$ is the set of the selected operating units, and $m = mat(o)$. For completing the description of the algorithms, we have to define the rules for

selecting the succeeding operating unit. We select the operating unit $v$ for which the quotient

$$\frac{w(v) + \text{the inputs' cost of } v}{\text{the number of the required outputs of } v}$$

is minimal. The difference between the two algorithms is in the calculation of the inputs' cost of an operating unit. In the first algorithm, called $\text{Asum}_c$, this cost is estimated by $\sum_{X \in mat^{in}(v)} c(X)$. In the second algorithm, called $\text{Amax}_c$, this cost is estimated by $\max\{c(X) : X \in mat^{in}(v)\}$. Since this is only the difference between the two algorithms, we present here only one of them, namely, the algorithm $\text{Asum}_c$.

**Algorithm $\text{Asum}_c$**

- *Initialization.* Set $N_0 = P$, $O_0 = \emptyset$, $K_0 = R$, and $i = 0$.
- *Iteration.* (*i-th iteration*)
  - *Step 1.* Proceed to Step 3 if $N_i = \emptyset$. Otherwise, for each operating unit $u \notin O_i$ producing material from $N_i$, take the quotient

$$\frac{w(u) + \sum_{X \in mat^{in}(u)} c(X)}{u(N_i)},$$

    where $u(N_i)$ denotes the number of those elements of $N_i$ which are produced by $u$. Select an operating unit for which this quotient is minimal and denote it by $v$.
  - *Step 2.* Let $O_{i+1} = O_i \cup \{v\}$, $K_{i+1} = K_i \cup mat^{out}(v)$, and $N_{i+1} = N_i \cup mat^{in}(v) \setminus K_{i+1}$. Increase the value of $i$ by 1, and proceed to the next iteration.
  - *Step 3.* Let $o = O_i$, and $m = mat(o)$.

First, let us consider the correctness and finiteness of the procedure. Though we consider only the algorithm $\text{Asum}_c$, the same proof is valid for $\text{Amax}_c$ as well. We have to show that for every material $A \in N_i$, there exists an operating unit producing $A$. Let us observe that for every $i$, $K_i$ contains the raw materials and the materials which are produced by some operating

units from $O_i$. Therefore, there are no raw materials in $N_i$. On the other hand, $(M, O)$ is the maximal structure, and hence, every material $A \notin R$ is produced by some operating unit. These observations yield the required statement. Moreover, we put one new operating unit into the set $O_i$ during each iteration, and therefore, the set $K_i$ will contain all the materials for some $i$, and the procedure terminates in the next step, provided that the procedure does not terminate earlier. Consequently, the algorithm terminates after finitely many steps and produces the sets $m$ and $o$. The definition of $m$ shows that these sets determine a P-graph. We prove now that this P-graph is a feasible solution.

**Theorem 4.1.1** *Algorithms $Asum_c$ and $Amax_c$ result in a feasible solution.*

*Proof:* We prove the statement for $\text{Asum}_c$; the same proof is valid for $\text{Amax}_c$ as well. We have to show that the P-graph $(m, o)$ satisfies the conditions given in the definition of the feasible solution. First, consider condition $(A1)$. By the earlier observation on $K_i$, it follows that the procedure removes from the set $N_i$ only those materials which are produced by some operating units from $O_i$. On the other hand, $N_0 = P$ at the beginning, and at the end of the procedure $N_i = \emptyset$, and therefore, for each desired product, $o$ contains some operating unit producing it. Hence, by the definition of $m$, the validity of condition $(A1)$ follows. For verifying condition $(A2)$, first observe that $(M, O)$ is a feasible solution, thus there is no operating unit producing some raw material in it. On the other hand, $o \subseteq O$, which yields that the same statement is valid for $o$. Moreover, in a similar way as in case of condition $(A1)$, one can prove that all other materials from $m$ are produced by some operating unit from $o$. As far as condition $(A3)$ is concerned, we show by induction on $i$ that for each operating unit in $O_i$, there is a path leading from it to some desired product. The statement is obvious for $i = 1$ since $O_1$ contains one operating unit producing some material from $N_0 = P$. Now, suppose that the statement is valid for $i \geq 1$. We show that it is also valid for $i + 1$, provided that $O_{i+1}$ exists. Since $O_{i+1} = O_i \cup \{v\}$, by the induction hypothesis, it is sufficient to prove that there is a path leading from $v$ into some desired product for the operating unit $v$ selected in the $i+1$-th iteration step. The operating unit $v$ has an output material $B$ in the set $N_i$. This material is a desired product or it is an input material of some operating

unit $u \in O_i$. In the first case, there is an edge from $v$ into a desired product and the statement is valid. In the second case, there is a path from $u$ into a desired product (by the induction hypothesis), and completing this path with $(B, u)$ and $(v, B)$, we obtain a path leading from $v$ into a desired product. Therefore, condition $(A3)$ is satisfied by $(m, o)$. The validity of condition $(A4)$ is obvious by the definition of $m$.

□

## 4.2 Worst-case bounds

In the worst-case analysis of a heuristic algorithm, we are to find a bound for the quotients calculated as follows. For every problem, we divide the cost of the produced solution by the cost of the optimal solution. To give the formal definition for the PNS problem, we need some further notation. Let $A$ be a heuristic algorithm for solving the PNS problem. For every PNS problem M, let the weight of the solution determined by $A$ and the weight of the optimal solution be denoted by $A(\mathrm{M})$ and $OPT(\mathrm{M})$, respectively. Then, $C$ is called a *worst-case bound* of algorithm $A$ if

$$A(\mathrm{M})/OPT(\mathrm{M}) \leq C$$

is valid for every M. $C$ is called *tight* if it is the smallest worst-case bound. We can also define the worst-case bounds for some subclasses. $C$ is called a *worst-case bound* of the algorithm for a class $\mathcal{P}$ of PNS problems if

$$A(\mathrm{M})/OPT(\mathrm{M}) \leq C$$

is valid for every $\mathrm{M} \in \mathcal{P}$. $C$ is called *tight* if it is the smallest worst-case bound on the class considered. Sometimes worst-case bounds are used in a more general sense. It is possible to use some functions depending on the problem instance instead of a general constant $C$ in the definition. In Theorem 4.2.1 we present such a result.

For some difficult problems, it has been proven that no heuristic algorithms exist with a constant worst-case bound under some complexity assumption (usually under the assumption $P \neq NP$). First, we prove that the PNS problem belongs to this class.

**Theorem 4.2.1** ([8]) *There is no polynomial time heuristic algorithm with constant worst-case bound for the class of PNS problems unless P=NP.*

*Proof:* It is proven in [6] that there is no polynomial time heuristic algorithm with constant worst-case bound for the set covering problem unless P=NP. This result and the equivalence between the set covering and PNS1 problems validates the statement of Theorem 4.2.1.

□

By Theorem 4.2.1 it is very unlikely to find a heuristic algorithm with a constant worst-case bound for the PNS problem. However, considering particular PNS classes, we can prove some tight worst-case bounds for the presented algorithms. First, we determine the worst-case bound for the PNS1 class.

**Theorem 4.2.2** ([8]) *For any problem from the PNS1 class, the algorithms $Asum_c$ and $Amax_c$ give the same result. Furthermore, they have the tight worst-case bound $\sum_{i=1}^{m} \frac{1}{i}$ on the PNS1 class where $m$ is the maximum size of the output sets.*

*Proof:* To prove this statement, first we have to recall Chvatal's following algorithm for the set covering problem.

**Algorithm 4.2.1** (Chvatal [14])

- *Initialization.* Let $J^* = \emptyset$.
- *Step 1.* Terminate if $P_j = \emptyset$, for all $j$; $J^*$ is the cover produced by the algorithm. Otherwise, choose an index for which the quotient $|P_j|/c_j$ is maximal. Denote it by $k$ and proceed to Step 2.
- *Step 2.* Put $k$ into $J^*$, replace every $P_j$ with $P_j \setminus P_k$ and proceed to Step 1.

For this algorithm the following worst-case bound is proven:

**Proposition 4.2.3** ([14]) *The tight worst-case bound of Algorithm* 4.2.1 *is $\sum_{i=1}^{d} 1/i$, where $d$ is the number of elements of the largest set from the subset system.*

Now, we can use the equivalence between the PNS1 and set covering problems. For PNS1 problems, each operating unit has only raw materials as input materials, and $c(X) = 0$ for every $x \in R$. This yields that the selection rules of both algorithms are reduced to the following rule. We always select the operating unit for which the ratio obtained by dividing the weight of the operating unit by the number of the required output materials is minimal. If we investigate the behaviour of this algorithm using the correspondence to the equivalent set covering problem, then it is easy to see that the algorithm selects the operating units corresponding to the sets chosen by Algorithm 4.2.1. Therefore, Proposition 4.2.3 implies the validity of Theorem 4.2.2.

Further classes of PNS problems for which we determine the worst-case bounds are the classes $S_k$, $k = 1, 2, \ldots$. For each fixed positive integer $k$, a PNS problem belongs to the class $S_k$ if every operating unit is separator type (it has only one input material), the graph of the problem does not contain a cycle, and the number of the desired products is equal to $k$.

**Theorem 4.2.4** ([8]) *For any problem from the class $S_k$, the algorithms $Asum_c$ and $Amax_c$ give the same result. Moreover, they have the tight worst-case bound $k$ on $S_k$, for every positive integer $k$.*

*Proof:* Let $k$ be an arbitrarily fixed positive integer. In what follows, we study the class $S_k$ for this fixed $k$. Consider an arbitrary reduced model from this class. Let $(M, O)$ be its P-graph. Since each operating unit is a separator,

$$\sum_{X \in mat^{in}(u)} c(X) = \max\{c(X) : X \in mat^{in}(u)\}$$

is valid for each operating unit $u$, which yields that the two algorithms are the same. In the remaining part of the proof, this algorithm is denoted by $A$. First, we prove a lemma on the function $c$.

**Lemma 4.2.5** *For every material $X$, the cost of every path leading from a raw material into $X$ is at least $c(X)$.*

*Proof:* We prove that this statement is valid for the elements of the set $I_r$ by induction on $r$. Since $I_0 = R$, the statement is obviously valid for $r = 0$. Now, let $r \geq 0$ and let us suppose that the statement is valid for each nonnegative integer which is not greater than $r$. We prove it for $r + 1$. During the $r + 1$-th iteration, we extend the set $I_r$ with one new material. Let us denote it by $Y$. Suppose that the statement is not valid. This yields that for some $Z \in R$ there is a path $[Z, Y]$ in $(M, O)$ with cost smaller than $c(Y)$. Consider the last two vertices in this path, denote them by $u$ and $V$. By the construction of the function c, it follows that $V \in I_r$ and $c(Y) \leq c(V) + w(u)$. On the other hand, since $V \in I_r$, by the induction hypothesis, each path leading from a raw material to $V$ has a cost at least $c(V)$. This yields that the cost of the path considered is at least $c(V) + w(u)$, which is a contradiction. Therefore, we proved the statement for $r + 1$, which ends the proof of Lemma 4.2.5.

□

Now, we prove that algorithm $A$ has the worst-case bound $k$. First, consider an optimal solution. Let $P_i$ be a desired product for which the value of $c$ is maximal. Since the optimal solution is a feasible solution, it must contain $P_i$. Furthermore, it follows immediately, by condition $(A2)$ and by the cycle free property of the P-graph of the problem, that in the P-graph of the optimal solution there is a path leading from a raw material into the material $P_i$. By Lemma 4.2.5, this yields that the cost of this path is at least $c(P_i)$. Therefore, we proved that

$$OPT(\mathrm{M}) \geq \max\{c(X) : X \in P\}.$$

Let us investigate the weight of the solution produced by algorithm $A$. We show the following lemma.

**Lemma 4.2.6** $A(\mathrm{M}) \leq \sum_{X \in P} c(X)$.

*Proof:* Consider the sequence $D_i = \sum_{X \in N_i} c(X) + \sum_{u \in O_i} w(u)$, $i = 0, 1, \ldots$. We prove that this sequence is monotone decreasing as $i$ is increasing. In the $i$-th step of the algorithm, we obtain $N_{i+1}$ by choosing an operating unit $u$, deleting its output materials from $N_i$ and enlarging $N_i \setminus mat^{out}(\{u\})$ by $mat^{in}(u)$ if $mat^{in}(u) \notin K_{i+1}$. Therefore, we have that

$$D_i - D_{i+1} \geq \sum_{i=1}^{l} c(X_i) - c(mat^{in}(u)) - w(u),$$

where $u$ is the operating unit selected in the $i$-th iteration step and $X_1, \ldots, X_l$ are the output materials of $u$ which are contained in $N_i$. On the other hand, by the selection rule of $A$, it follows that

$$c(mat^{in}(v)) + w(v) \geq \frac{c(mat^{in}(u)) + w(u)}{l}$$

is valid for every operating unit $v$ producing $X_i$. Consequently, by the construction of $c$, we obtain that

$$c(X_i) \geq \frac{c(mat^{in}(u)) + w(u)}{l}.$$

Summing up the inequalities concerning the values $c(X_i)$, we get the following inequality

$$\sum_{i=1}^{l} c(X_i) - c(mat^{in}(u)) - w(u) \geq 0.$$

Therefore, we proved the validity of $D_i - D_{i+1} \geq 0$.

On the other hand, $D_0 = \sum_{X \in P} c(X)$, and $D_s = A(\mathrm{M})$ holds at the end of the procedure, and hence, by the decreasing property of the sequence $D_i$, the statement of the lemma is valid.

□

Therefore, we proved that $OPT(\mathrm{M}) \geq \max\{c(X) : X \in P\}$ and $A(\mathrm{M}) \leq \sum_{X \in P} c(X)$ are valid. These inequalities and $|P| = k$ yield that

$$\frac{A(\mathrm{M})}{OPT(\mathrm{M})} \leq k,$$

which means that $k$ is a worst-case bound for $A$.

Now, we prove that this bound is tight. We show that for every $\varepsilon > 0$, there exists a PNS problem $\mathrm{M}_\varepsilon$ having maximal structure $(M_\varepsilon, O_\varepsilon)$ from the class $S_k$ for which

$$\frac{A(\mathrm{M}_\varepsilon)}{OPT(\mathrm{M}_\varepsilon)} > k - \varepsilon.$$

First, let us observe that it is sufficient to prove this for $\varepsilon < 1$. In what follows, let $0 < \varepsilon < 1$ be an arbitrarily fixed real number.

Moreover, let $0 < \delta < \frac{\varepsilon}{k-\varepsilon}$. Define the following PNS problem $\mathrm{M}_\varepsilon$ from the class $S_k$. Let the set of operating units be

$$O = \{u_0, u_1, \ldots, u_k, v_1, \ldots, v_{2k}\},$$

where $u_0 = (\{R_0\}, \{X_1, \ldots, X_k\})$, $u_i = (\{X_i\}, \{P_i\})$, $i = 1, \ldots, k$, furthermore, $v_i = (\{R_i\}, \{Y_i\})$, and $v_{k+i} = (\{Y_i\}, \{P_i\}$, $i = 1, \ldots, k$. The P-graph of the problem is shown in Figure 4.2.1.

In this problem, $R_0, \ldots, R_k$ are the raw materials, $P_1, \ldots, P_k$ are the desired products, moreover, $w(u_0) = 1$, $w(u_i) = \frac{\delta}{k}$, $i = 1, \ldots, k$, and $w(v_i) = \frac{1}{2}$, $i = 1, \ldots, 2k$.

Performing the algorithm $A$ on the problem considered, we obtain the feasible solution $(\bar{m}, \bar{o})$, where

$\bar{m} = \{R_1, \ldots, R_k, Y_1, \ldots, Y_k, P_1, \ldots, P_k\}$ and $\bar{o} = \{v_1, \ldots, v_{2k}\}$.

The weight of this solution is equal to $k$, and hence, $A(\mathrm{M}) = k$. On the other hand, $(m, o)$ is also feasible solution, where $m = \{R_0, X_1, \ldots, X_k, P_1, \ldots, P_k\}$ and $o = \{u_0, u_1, \ldots, u_k\}$. The weight of this solution is $1+\delta$, thus, $OPT(\mathrm{M})_\varepsilon \leq 1 + \delta$. Therefore, we obtain that

$$\frac{A(\mathrm{M}_\varepsilon)}{OPT(\mathrm{M}_\varepsilon)} \geq \frac{k}{1+\delta}$$

is valid. On the other hand,

$$\frac{k}{1+\delta} > \frac{k}{1+\frac{\varepsilon}{k-\varepsilon}} = k - \varepsilon.$$

This means that we proved for $\mathrm{M}_\varepsilon$ that

$$\frac{A(\mathrm{M}_\varepsilon)}{OPT(\mathrm{M}_\varepsilon)} \geq k - \varepsilon,$$

which shows that the bound $k$ is tight.

□

By the theorem above, we immediately obtain the following corollary.

**Corollary 4.2.7** *For the class $S_1$, $A$ results in an optimal solution.*

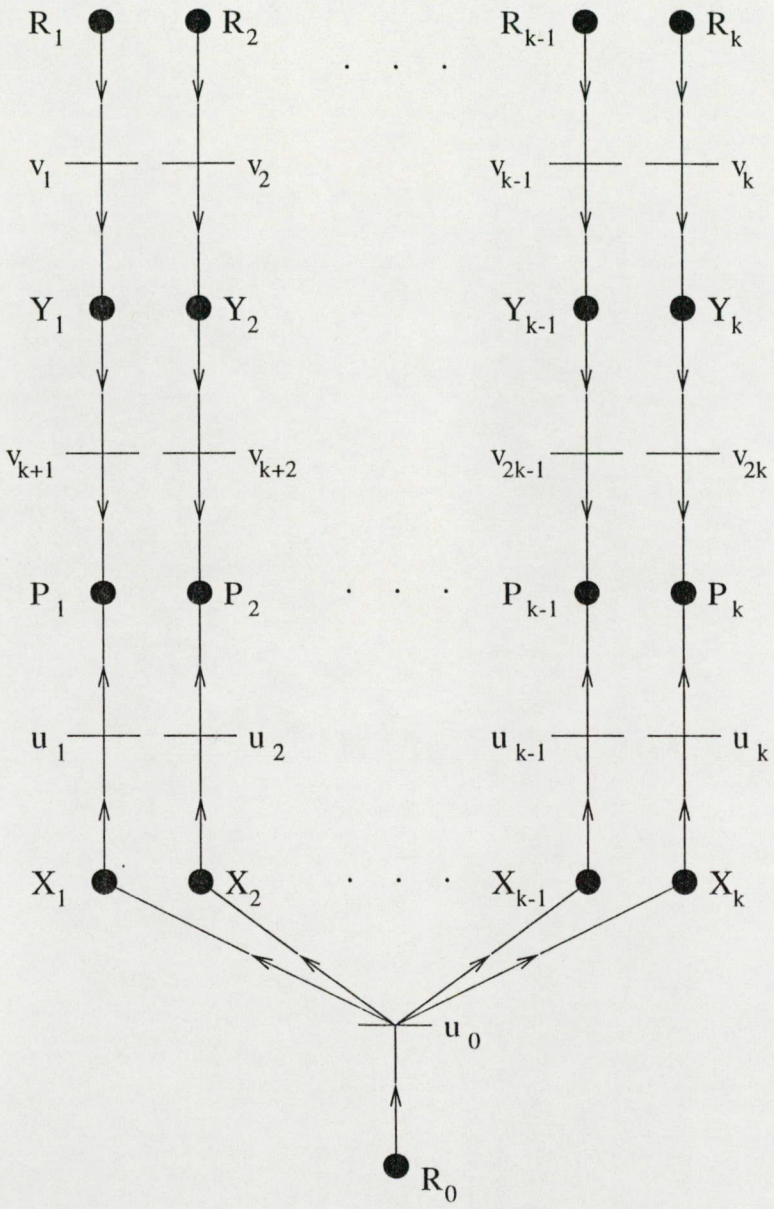



*Figure 4.2.1* The P-graph of the defined problem.

# Chapter 5

# Online scheduling problems

In the second part of the thesis we consider online problems which are closely related to online machine scheduling. First, in Section 5.1, the most important definitions of the competitive analysis are presented. Then, in Section 5.2, we introduce some basic scheduling and online scheduling models.

## 5.1 Competitive analysis

In online computation, an algorithm must produce decisions based only on past events without secure information on future. Such algorithms are called *online algorithms*. Online algorithms have many applications in different areas, such as computer science, economics and operations research.

One basic approach to studying online algorithms is the *average case analysis*, where we hypothesize some distribution on events, and we study the expected total cost. Another approach is the *competitive analysis*, where for each input sequence the cost produced by the online algorithm is compared to the offline (in the offline version we have the full knowledge of future) optimal value. Since we use the competitive analysis, we present here its most important definitions. One can find more details on the analysis of the online algorithms in [10] and [17].

We will consider minimization problems with nonnegative cost function, and therefore, we give the definitions only for this case. The definitions can be easily modified for the other optimization problems. Let a problem $P$ be given by the set of its instances $I$, and the cost function $w$. For each instance,

denote the cost of the optimal offline solution by $OPT(I)$. Let $A$ be an arbitrary online algorithm. Denote $A(I)$ the cost of the solution produced by $A$ on the problem instance $I$.

Algorithm $A$ is called *$C$-competitive* if for each instance,

$$A(I) \leq C \cdot OPT(I)$$

is valid. The *competitive ratio of an algorithm* is the least constant $C$ such that the algorithm is $C$-competitive. *The competitive ratio of a problem* is the best competitive ratio any online algorithm can achieve.

## 5.2 Scheduling problems

The problems which we investigate are closely related to parallel machine scheduling, therefore, we present some basic scheduling models here. In the simplest model, we have a set $J$ of jobs, each of them has a processing time $p_j$, and we have to process them on the available uniform machines. The number of machines is denoted by $m$. A schedule specifies for each job a machine and a time interval on the machine when the job is processed. The length of the time interval must be the processing time, the starting and ending point of the time interval are called the *starting and finishing time* of the job. A schedule is *feasible* if the time intervals do not overlap for each machine. Our goal is to minimize the maximal finishing time, which is called the makespan. In a more difficult model the jobs also have a parameter $r_j$, which is called release time. Then, we have the assumption that the starting time of job $j$ can not be smaller than $r_j$.

In the second model which will be used it is allowed to preempt the jobs. With preemption a job may be scheduled on multiple machines. A *time slot* is a non-empty interval $(q, t)$ with $q \geq 0$. In preemptive scheduling we have to assign time slots to each job on one or more machines. The sum of the sizes of the time slots must be the processing time of the job, and if time slots $(q_1, t_1), \ldots, (q_i, t_i)$ are assigned to a job, then $t_j \leq q_{j+1}$ must be valid for $j = 1, \ldots, i-1$. Furthermore, no two jobs may have overlapping time slots on the same machine. One can easily prove the following result for this problem.

**Proposition 5.2.1** *For any sets of jobs $J$ the makespan of the optimal schedule is*

$$\max\{\sum_{j\in J} p_j/m, \max_{j\in J} p_j\}.$$

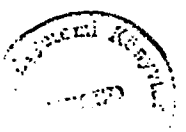
We introduced only the simplest parallel machine scheduling problems here. For details on the area of parallel machine scheduling we refer to [34], [40], and [37].

Now, we present the basic online parallel machine scheduling problems. Probably the most fundamental example of an online machine scheduling problem is where we schedule the jobs one by one. In this problem, we have a fixed number $m$ of identical machines. The jobs and their processing times are revealed to the online algorithm one by one. When a job is revealed, the online algorithm must irrevocably assign the job to a machine. There has been a great deal of work on this problem including [1], [12], and [22], but the best possible competitive ratio for this problem is still unknown for $m \geq 3$. The first result is due to Graham [24]. Although the terminology of competitive analysis was not used by him, it was shown that a simple algorithm, the List Scheduling, is $(2 - 1/m)$-competitive.

Another online machine scheduling problem is where the jobs arrive over time. Here again there are a fixed number of machines. Each job has a processing time and a release time. A job is revealed to the online algorithm at its release time. For each job the online algorithm must choose which machine the job will run on and assign a starting time. No machine may simultaneously run two jobs. Note that the algorithm is not required to immediately assign to a machine the job at its release time. However, if the online algorithm assigns a job at time $t$, then it cannot use information about jobs released after time $t$ and it cannot start the job before time $t$. The objective is to minimize the makespan. For details and results on this model, we refer to [35] and [39].

In what follows, we investigate three different problems which are closely related to online scheduling problems. The first problem, studied in [32], is a variant where we have to purchase the machines. In the second problem some generalized scheduling problems are considered, where the machines form a two-layer multiprocessor structure. These problems are investigated in [27] and [28]. Finally, in the third problem (cf. [31]) we investigate the problem





of scheduling with shared resources. This problem leads to a modified strip packing model, where it is allowed to lengthen the items which are packed into the strip. For the solution of this problem, we use some idea from the field of machine scheduling.

# Chapter 6

# Online scheduling with machine cost

In this chapter we investigate a scheduling problem, where the number of machines is not fixed, and the decision maker has to purchase the machines. First we define the mathematical model of the problem and introduce also the model where the jobs have release time. Then, we establish a class of online algorithms for this problem, and prove a theorem on the competitive ratio of some particular elements of this class. Some general lower bounds are also presented. These results are published in [32]. The paper was written while the author was visiting TU Graz. The published algorithms and lower bounds were developed through many ideas by a joint work with John Noga, therefore it is impossible to separate which results belongs to the author.

## 6.1 Problem definition

In machine scheduling, we typically have a fixed set of machines. The scheduling algorithm makes no decision regarding the initial set of machines nor is it allowed to change the set of machines later. It is usually assumed that the provided machines can be utilized without cost.

We investigate how scheduling problems change when machine costs are considered. We have several reasons for studying this idea. Most obviously, real machines have cost. If we do not have the necessary machines, then they must be obtained. Even if we already possess machines we may still incur

a fixed start up or conversion cost proportional to the number of machines used. Also, we still have an opportunity cost. By this cost we mean that if we use the machines for a given problem, we lose the chance to use them for something else. Further, in many cases it is desirable to buy or lease additional machines. A second reason we might allow the number of machines to be determined by the algorithm is that the performance of an algorithm on a given input can be highly dependent on the number of machines. A third reason is that by considering such a variant we may find other interesting problems and/or gain insight into the original.

We consider two scheduling problems with machine cost. The first one is a variant of online scheduling jobs one by one. The differences are that 1) no machines are initially provided, 2) when a job is revealed, the algorithm has the option to purchase new machines, and 3) the objective is to minimize the sum of the makespan and cost of the machines. We will refer to this problem as the *List Model* for scheduling with machine cost. The second problem which we consider is a variant of scheduling jobs arriving over time. The differences are that 1) no machines are initially provided and 2) the algorithm may purchase machines at any time, and 3) the objective is to minimize the sum of the makespan and the cost of the machines. We will refer to this problem as the *Time Model* for scheduling with machine cost. Throughout the remainder of the chapter we will use the following notations. The jobs will be labeled $j_1, \ldots, j_n$ and presented to the online algorithm in this order. We denote the processing time of the job $j_i$ by $p_i$ and the largest processing time by $L = \max\{p_i\}$. For a fixed algorithm, the starting time of the job $j_i$ is $s_i$ and its completion time is $c_i = s_i + p_i$. The total amount of processing needed by the first $\ell$ jobs is $P_\ell = \sum_{i=1}^{\ell} p_i$. In the Time Model the release time of the job $j_i$ is $r_i$. We will assume that the cost of purchasing a machine is 1. Since we could simply rescale the machine costs and job sizes, any other constant cost function is equivalent.

This scheduling problem is somewhere between the original scheduling and bin packing problems. In the original scheduling problem the number of machines is fixed and our goal is to minimize the time. In the bin packing problem the size of the bins (we can consider it as the time) is fixed and we want to minimize the number of bins (we can consider them as machines). In this scheduling problem with machine cost neither the time nor the number of machines is fixed, the goal is to minimize the sum of them.

The offline version of machine scheduling under both the List Model and

Time Model can easily be seen to be NP-complete by simple transformations from PARTITION. Since finding the exact optimal offline solution is a hard problem, in our upper bound proofs we will use the following lower bound on the optimal offline solution.

**Lemma 6.1.1** *For both the List Model and Time Model, the optimal offline cost is at least* $2\sqrt{P}$. *Further, if* $L \geq \sqrt{P}$, *then the optimal offline cost is at least* $L + P/L$.

*Proof:* Let $m$ be the number of machines and $M$ be the makespan of the optimal solution. Since the largest job must be placed on some machine, $L \leq M$. Since the total load on any machine is no more than $M$, the maximum amount of processing which can be accomplished is $mM$. So, $P \leq mM$. Therefore, the optimal offline cost must be greater than the solution to the following optimization problem: minimize $m + M$ subject to $P \leq mM$ and $L \leq M$. It is easy to see that this value is the one described. □

We study a class of online algorithms for the List Model. For an increasing sequence $\varrho = (0 = \varrho_1, \varrho_2, \ldots, \varrho_i, \ldots)$ we will define an online algorithm $A_\varrho$. When job $j_\ell$ is revealed, $A_\varrho$ purchases machines (if necessary) so that the current number of machines $i$ satisfies $\varrho_i \leq P_\ell < \varrho_{i+1}$. Algorithm $A_\varrho$ then assigns the job $j_\ell$ to the least loaded machine.

For the Time Model, we define a very similar class of online algorithms. For an increasing sequence $\varrho = (0 = \varrho_1, \varrho_2, \ldots, \varrho_i, \ldots)$ we will define an online algorithm $B_\varrho$. When job $j_\ell$ is revealed, $B_\varrho$ purchases machines (if necessary) so that the current number of machines $i$ satisfies $\varrho_i \leq P_\ell < \varrho_{i+1}$. Whenever there is at least one machine that is not processing a job and at least one job that has been released but not started, $A_\varrho$ assigns the job with the largest processing time to an idle machine.

## 6.2 List Model

### 6.2.1 Lower Bound

**Theorem 6.2.1** ([32]) *No online algorithm can have a competitive ratio smaller than* $\frac{4}{3}$.

*Proof:* Consider a very long sequence of jobs with each job having a very small processing time, $p_i = \varepsilon$ for all $i$. It is easy to see that any algorithm which never purchases a second machine is not $C$-competitive for any $C$. So assume that the algorithm purchases a second machine when the job $j_\ell$ is released. If $P_\ell \leq 2$, then the offline algorithm can serve all jobs with one machine and the competitive ratio can be no smaller than

$$\frac{P_\ell - \varepsilon + 2}{P_\ell + 1} \geq \frac{4 - \varepsilon}{3}.$$

If $P_\ell > 2$, then the offline algorithm can split the jobs nearly evenly between two machines and the competitive ratio can be no smaller than

$$\frac{P_\ell - \varepsilon + 2}{P_\ell/2 + 2 + \varepsilon} \geq \frac{4}{3 + \varepsilon}.$$

Since we can choose $\varepsilon$ to be arbitrarily small, we obtain the result.

□

### 6.2.2 Upper Bound

Throughout this section we will consider the algorithm $A = A_\varrho$ for $\varrho = (0, 4, 9, 16, \ldots, i^2, \ldots)$. The basic intuition for selecting $\varrho$ comes from Lemma 6.1.1. If the optimal cost is close to $2\sqrt{P}$, then the optimal algorithm uses approximately $\sqrt{P}$ machines. If $P \geq 4$, then $A$ tries to mimic this behavior by purchasing at most $\sqrt{P}$ machines.

**Theorem 6.2.2** ([32]) *The competitive ratio of $A$ is $(1 + \sqrt{5})/2$.*

*Proof:* For the sake of simplicity, in the following part of this chapter the number $(1 + \sqrt{5})/2$ is denoted by $\varphi$. We will first prove that $A$ is $\varphi$-competitive.

Consider an arbitrary sequence of jobs $\sigma = j_1, \ldots, j_n$ and fix an optimal schedule. Let $M^*$ be the optimal makespan, $m^*$ be the optimal number of machines, $M$ be the makespan of $A$, and $m$ be the number of machines used by $A$. Let $j_\ell$ be the last job that $A$ completes and $k$ be the number of machines that $A$ owns after $j_\ell$ is released.

*Case A:* If $A$ purchases only one machine, then the cost of the algorithm is $1 + P$ and $P < 4$. If the optimal offline schedule also uses one machine,

then the ratio of costs is 1. If the optimal offline schedule uses two or more machines, then the optimal cost is at least $2+\frac{P}{2}$. Since $P < 4$, the cost ratio is no more than $5/4$.

In the remaining cases we will repeatedly use several simple inequalities. By our choice of $A$, we know $m \leq \sqrt{P} < m+1$ and $k \leq \sqrt{P_\ell} < k+1$. We use Lemma 6.1.1 to estimate $OPT(\sigma)$. Since $A$ always assigns a job to the machine with the smallest load,

$$A(\sigma) \;\leq\; m + \frac{P_\ell - p_\ell}{k} + p_\ell \;=\; m + \frac{P_\ell}{k} + \frac{k-1}{k} p_\ell.$$

*Case B:* If $m > k$ and $p_\ell \leq \sqrt{P}$, then using the inequalities given above:

$$\frac{A(\sigma)}{OPT(\sigma)} \leq \frac{m + (k+1)^2/k + (k-1)\sqrt{P}/k}{2\sqrt{P}}$$

$$\leq \frac{m + k + 2 + 1/k + \sqrt{P} - \sqrt{P}/k}{2\sqrt{P}}$$

$$\leq \frac{3\sqrt{P} + (m - \sqrt{P})/(m-1)}{2\sqrt{P}} \leq 3/2 \leq \varphi.$$

*Case C:* If $m = k > 1$ and $p_\ell \leq \sqrt{P}$, then since $(m + P/m + \sqrt{P})/2\sqrt{P}$ is increasing in $P$, we have that

$$\frac{A(\sigma)}{OPT(\sigma)} \;\leq\; \frac{m + P/m + \sqrt{P}}{2\sqrt{P}} \;\leq\; \frac{3m + 3 + 1/m}{2(m+1)} \;\leq\; \frac{19}{12} \;\leq\; \varphi.$$

*Case D:* If $m > k$ and $p_\ell > \sqrt{P}$, then using the inequalities given above we obtain that

$$\frac{A(\sigma)}{OPT(\sigma)} \;\leq\; \frac{m + (k+1)^2/k + p_\ell(k-1)/k}{p_\ell + P/p_\ell} \;\leq\; \frac{2\sqrt{P} + p_\ell}{p_\ell + P/p_\ell}$$

$$= \;\frac{2 + p_\ell/\sqrt{P}}{p_\ell/\sqrt{P} + \sqrt{P}/p_\ell} \;\leq\; \varphi.$$

The last inequality follows from the fact that the maximum of $f(x) = (2 + x)(x + 1/x)$ is $\varphi$.

*Case E:* If $m = k > 1$ and $p_\ell > \sqrt{P}$, then since $(m + P/m - p_\ell/m)/2\sqrt{P}$ is increasing in $P$, we obtain that

$$\frac{m + P/m - p_\ell/m}{2\sqrt{P}} \leq \frac{m + (m+1)^2/m - p_\ell/m}{2(m+1)} = 1 + \frac{1 - p_\ell}{2m(m+1)} \leq 1.$$

Therefore, $m + P/m - p_\ell/m \leq 2\sqrt{P}$ and

$$\frac{A(\sigma)}{OPT(\sigma)} \leq \frac{2\sqrt{P} + p_\ell}{p_\ell + P/p_\ell} = \frac{2 + p_\ell/\sqrt{P}}{p_\ell/\sqrt{P} + \sqrt{P}/p_\ell} \leq \varphi,$$

where the final inequality follows from the fact that the maximum of $f(x) = (2 + x)(x + 1/x)$ is $\varphi$.

We now wish to show that $A$ is not $C$-competitive for any $C < \varphi$. Consider a sequence of $N^3$ jobs of size $1/N$ followed by one job of size $\varphi N$. $A$ will schedule the first $N^3$ by purchasing $N$ machines and putting $N^2$ jobs on each machine. The final job will be placed on an arbitrary machine. Therefore, $A$'s cost will be $N + N + \varphi N$. The optimal cost is no more than $\varphi N + \lceil (N + \varphi)/\varphi \rceil$. So, the competitive ratio of $A$ is at least

$$\frac{(2 + \varphi)N}{\varphi N + \lceil (N + \varphi)/\varphi \rceil} \xrightarrow{N \to \infty} \frac{2 + \varphi}{\varphi + 1/\varphi} = \varphi.$$

Consequently, the competitive ratio of $A$ is $\varphi$.

□

## 6.3 Time Model

The Time Model differs from the List Model in two respects. The online algorithm has the advantage of not having to immediately assign a job to a machine. However, the online algorithm has the disadvantage that if a machine is purchased at time $t$, then it cannot be used before time $t$. For these reasons neither our upper nor lower bounds from the List Model directly apply to the Time Model.

### 6.3.1 Lower Bound

**Theorem 6.3.1** ([32]) *No online algorithm can have a competitive ratio smaller than*

$$C = \frac{\sqrt{33}-1}{4} \approx 1.186.$$

*Proof:* Fix an online algorithm and let $S = C + 1/2$. Consider two jobs $(p_1, r_1) = (p_2, r_2) = (S, 0)$. Let $t = \max\{s_1, s_2\}$. If $t < S - 1$, then we will present a third job $(p_3, r_3) = (2S - t - \varepsilon, t + \varepsilon)$. The optimal offline cost is $S + 2$ for the first two jobs and $2S + 2$ if all three are given.

If $t \geq S$, then the algorithm's makespan is at least $2S$. So, the cost ratio can be no smaller than $(2S + 1)/(S + 2) = C$. If $S - 1 \leq t < S$, then the algorithm must run the two jobs on different machines and have makespan at least $2S - 1$. So, the cost ratio can be no smaller than $(2S+1)/(S+2) = C$. If $t < S - 1$, then the third job is presented. Once again the algorithm must run the first two jobs on different machines. If it purchases exactly two machines, then the makespan is at least $3S - t - \varepsilon$. If it purchases at least three machines then the makespan is at least $2S$. So, the cost ratio can be no smaller than $\min\{3S - t - \varepsilon + 2, 2S - \varepsilon + 3\}/(2S + 2)$. As $\varepsilon$ tends to zero this value tends to $C$.

Regardless of how the online algorithm schedules the first two jobs in this sequence, the cost ratio can be made arbitrarily close to $C$. Therefore, the competitive ratio must be at least $C$.

□

### 6.3.2 Upper Bound

Throughout this section we will consider the algorithm $B = B_\varrho$ for $\varrho = (0, 4, 9, 16, \ldots, i^2, \ldots)$. Once again we attempt to mimic the behavior of an offline algorithm which achieves a cost near $2\sqrt{P}$.

**Theorem 6.3.2** ([32]) *Algorithm $B$ is $\frac{6+\sqrt{205}}{12} \approx 1.693$-competitive.*

*Proof:* Consider an arbitrary sequence of jobs $\sigma = j_1, \ldots, j_n$ and fix an optimal schedule. Let $M^*$ be the optimal makespan, $m^*$ be the optimal number of machines, $M$ be the makespan of $B$, and $m$ be the number of machines used by $B$.

*Case A:* Suppose $m = 1$. If $B$'s machine is never idle, then let $t$=0. Otherwise, let $t$ be the latest time that $B$'s machine is idle. Let $W \leq P \leq 4$ be the total processing time of all jobs released at or after time $t$. The cost of $B$ is $1+t+W$. The optimal cost is at least $m^* + t + W/m^*$. For this case:

$$\frac{B(\sigma)}{OPT(\sigma)} \leq \frac{1+t+W}{m^*+t+W/m^*} \leq 5/4.$$

*Case B:* Suppose $m > 1$. Then we claim that $M - M^* \leq \frac{P}{m}$. Suppose that this was not true. Let $j_i$ be the last job to finish in $B$'s schedule and $\ell$ be the number of machines owned by $B$ at time $M^* - p_i$. Note that none of $B$'s machines can be idle during the time period $I = [M^* - p_i, M - p_i]$. So at least $(M - M^*)\ell$ processing is completed during $I$. Further, if $p_i \leq M - M^*$, then an additional $(M - M^* - p_i)(m - \ell)$ processing is completed during $[M^*, M - p_i]$ by the remaining $m - \ell$ machines that $B$ purchases.

If $\ell = m$, then more than $P$ processing is completed, which is a contradiction. If $\ell < m$, then $P_i < (\ell+1)^2$. Since the processing time of any job released after $M - p_i$ must be less than $p_i$, the job $j_i$ will start before any job released after $M - p_i$. Therefore,

$$\begin{aligned} P_i &\geq (M - M^*)\ell + p_i + \max\{(M - M^* - p_i)(m - \ell), 0\} \\ &\geq (M - M^*)(\ell+1) > P/m(\ell+1) \geq (\ell+1)^2, \end{aligned}$$

which is also a contradiction. So the claim must be true.

Since $m \geq 2$, $P < (m+1)^2$, and $(P/m + m)/\sqrt{P}$ is increasing in $P$, it is easy to verify that $P/m + m \leq 13\sqrt{P}/6$.

Putting these facts together we get:

$$\begin{aligned} B(\sigma) &\leq M^* + P/m + m \leq M^* + 2\sqrt{169P/144} \\ &\leq M^* + \frac{(-6+\sqrt{205})M^*}{12} + \frac{(6+\sqrt{205})P}{12M^*} \\ &= \frac{6+\sqrt{205}}{12}(M^* + P/M^*) \leq \frac{6+\sqrt{205}}{12} OPT(\sigma). \end{aligned}$$

The third inequality is an application of the arithmetic-geometric mean inequality. So we have shown that $B(\sigma)$ is $(6+\sqrt{205})/12$-competitive.

□

# Chapter 7

# Online scheduling with two-layer multiprocessor architecture

In this chapter we investigate a particular scheduling problem where the machines form a two-layer multiprocessor structure. We define two different but related problems: the SLS (Sum Layered Scheduling) and MLS (Maximum Layered Scheduling) problems. We establish two algorithms, and we determine their competitive ratios. Some general lower bounds are also presented.

## 7.1 Problem definition

Here we consider a scheduling problem where the machines have two-layer structure. In this problem we have two sets $\mathcal{P}$ and $\mathcal{S}$ of identical machines containing $k$ and $m$ machines with $k \leq m$. The jobs arrive one by one. Each job $j$ has two different processing times $p_j$ and $s_j$, one for each set of machines. We allow $\infty$ processing time, this means that the job cannot be executed on the machines of the set. We have to decide in an online way on which set of machines to schedule each job. Finally, when the stream of jobs has come to an end, we schedule in an offline way the jobs assigned to the set $\mathcal{P}$ (respectively, the jobs assigned to the set $\mathcal{S}$) on the machines of $\mathcal{P}$ (respectively, $\mathcal{S}$) so as to minimize the preemptive makespan. Let $C_P$ (respectively, $C_S$) denote this optimal makespan. In the first problem, which is investigated

in [28] and called *problem SLS(k,m)* (*Sum Layered Scheduling*), the cost of the constructed schedule is the sum $(C_P + C_S)$ of the two makespans. In the second problem, (cf. [27]) which is called *problem MLS(k,m)* (*Maximum Layered Scheduling*), the cost of the constructed schedule is the maximum $(\max\{C_P, C_S\})$ of the two makespans. The general problems without fixing the number of machines in the sets are denoted by *SLS* and *MLS*.

Problem SLS is a generalization of the semi online version of scheduling with rejection. In these scheduling problems, jobs arrive one by one, and we have to schedule each job in an online way or we can reject it at some penalty. The cost of the schedule is the makespan, and we are to minimize the sum of the cost of the schedule and the penalties of the rejected jobs. Nonpreemptive scheduling with rejection was introduced in [5], the preemptive version and randomized algorithms for it were studied in [38]. The main result of [5] is that they define a 2.61-competitive online algorithm for the problem, and prove that this algorithm is optimal. In [38] their algorithm is investigated for the problem where preemption is allowed, and a randomized version is also presented. In the semi online version we have to decide whether we reject or schedule the job in an online fashion, but we do an offline scheduling at the end. We denote this problem by *SOSR*. One can see immediately that SLS contains SOSR as a particular case (if one of the considered sets contains only a single machine). Problem MLS is a generalized version of the online two-machine scheduling problem with unrelated machines which is investigated in [2].

To present our results we need the following definitions.

For any subset $I$ of jobs, we use the following notations:

$$S_I = \sum_{j \in I} s_j, \quad P_I = \sum_{j \in I} p_j, \quad \text{Smax}_I = \max_{j \in I} s_j, \quad \text{Pmax}_I = \max_{j \in I} p_j.$$

Using these notations for the case of the problem SLS(k,m), the cost of a schedule $SC$ can be written in the form

$$w(SC) = \max\{\frac{P_R}{k}, \text{ Pmax}_R\} + \max\{\frac{S_Q}{m}, \text{ Smax}_Q\},$$

where $R$ and $Q$ are the sets of jobs assigned to $\mathcal{P}$ and $\mathcal{S}$, respectively. In the case of the problem $MLS(k, m)$, the cost of a schedule $SC$ can be written in the form

$$u(SC) = \max\{\frac{P_R}{k},\ \mathrm{Pmax}_R, \frac{S_Q}{m},\ \mathrm{Smax}_Q\},$$

where $R$ and $Q$ are the sets of jobs assigned to $\mathcal{P}$ and $\mathcal{S}$, respectively.

We study two online algorithms for these problems. First we present a simple greedy type algorithm. The second algorithm is a more difficult one, which is a generalized version of the reject total penalty algorithm from [5] and [38]. In fact, this algorithm is a class of algorithms since it depends on two parameters $0 < \alpha \leq 1$ and $0 < \gamma \leq 1$.

## 7.2 The load greedy algorithm

In this section we present and study the load greedy algorithm. The basic idea is to assign each job to the set where its load is smaller. We can define algorithm $LG$ as follows.

**Algorithm LG:** *If a job $j$ arrives, then it is assigned to $\mathcal{P}$ if $\frac{p_j}{k} \leq \frac{s_j}{m}$, otherwise it is assigned to $\mathcal{S}$.*

The competitive ratio of this algorithm on the problem $SLS(k,m)$ is determined by the following statement.

**Theorem 7.2.1** ([28]) *The competitive ratio of LG on $SLS(k,m)$ is*

$$C_{k,m}(LG) = \begin{cases} 1 + \frac{m}{k} & \text{if } k \geq 2, \\ m & \text{if } k = 1. \end{cases}$$

*Proof:* Consider first the case $k \geq 2$. Let $a$ and $b$ denote the makespans obtained by LG on $\mathcal{P}$ and $\mathcal{S}$, respectively. Furthermore, let us fix an optimal solution. First assume that $a \geq b$. Suppose that the makespan $a$ is defined by the maximal processing time. Denote the job with this maximal processing time by $j$. Then $p_j = a$, and since our algorithm assigned this job to $\mathcal{P}$, we get $s_j \geq \frac{m}{k}a$. Hence, the optimal cost is at least $a$, and this yields that for the competitive ratio, $C_{k,m}(LG) \leq \frac{a+b}{a} \leq 2 \leq 1 + \frac{m}{k}$. Now, suppose that the makespan is defined by the load of the jobs. Let $P$ denote the set of the jobs assigned by the algorithm to $\mathcal{P}$, and let $R$ and $Q$ denote the sets of the jobs from $P$ which are assigned to $\mathcal{P}$ and $\mathcal{S}$ in the optimal solution, respectively. Then the optimal cost is at least $\frac{P_R}{k} + \frac{S_Q}{m}$. Furthermore, by the definition of

LG, we get that $\frac{P_Q}{k} \leq \frac{S_Q}{m}$. This yields that the cost of the optimal solution is at least $\frac{P_R}{k} + \frac{P_Q}{k} = a$, and our statement follows in the same way as in the previous case.

Let us assume that $a < b$. Now, consider two cases depending on the makespan on $\mathcal{S}$. If the makespan is the load, then in the same way as above we obtain that the optimal cost is at least $b$, which yields that the algorithm is $1 + \frac{m}{k}$ competitive. Now, suppose that the makespan is defined by the maximal processing time. Denote the job with this maximal processing time by $j$. Then $s_j = b$ and since our algorithm assigned this job to $\mathcal{S}$, we have that $p_j > \frac{k}{m}b$. Hence, the optimal cost is at least $\frac{k}{m}b$. Furthermore, we proved above that if the makespan on $\mathcal{P}$ is $a$, then the optimal cost is at least $a$. This yields that our optimal cost is at least $\max\{\frac{k}{m}b, a\}$. Hence, the algorithm is

$$\frac{a+b}{\max\{\frac{k}{m}b, a\}} \leq 1 + \frac{m}{k}$$

competitive. We can handle the case $k = 1$ in a similar way. If $m = 1$, then the algorithm is obviously 1-competitive. If $m \geq 2$, we consider four cases, and we can obtain the upper bound 2 in the first three cases in the same way as we got it for $k \geq 2$. The only difference is in the case when $a < b$ and $b$ is defined by the processing time of a maximal job $j$. In this case, if the optimal algorithm assigns $j$ to $\mathcal{S}$, we obtain the upper bound 2. If it assigns $j$ to $\mathcal{P}$, then it causes $\frac{k}{m}b$ cost there. On the other hand, since $k = 1$, we cannot schedule the jobs which cause the makespan on $\mathcal{P}$ on the other machines from $\mathcal{P}$. This yields that these jobs will cause at least $a$ extra cost in the optimal solution. Thus, the algorithm is

$$\frac{a+b}{\frac{k}{m}b + a} \leq \frac{m}{k}$$

competitive.

To prove that the above analysis is tight in the case $k > 1$, consider the sequence of two jobs: $(1, \frac{m}{k} - \varepsilon)$ and $(1, m)$. Algorithm LG assigns the first job to $\mathcal{S}$ and the second job to $\mathcal{P}$ with cost $1 + \frac{m}{k} - \varepsilon$. Since the optimal cost is 1, choosing $\varepsilon$ to be sufficiently small, the competitive ratio on this sequence is arbitrarily close to $1 + \frac{m}{k}$. The tightness of the analysis for the case $k = 1$ follows if we consider the job $(1, \frac{m}{k} - \varepsilon)$.

□

For the $MLS(k,m)$ problem we obtain the following result.

**Theorem 7.2.2** ([27]) *Algorithm LG has the competitive ratio* $\max\{2, m/k\}$ *on problem* $MLS(k,m)$.

*Proof:* Consider an arbitrary list $L$ of jobs, and denote $a$ and $b$ the makespans obtained by LG on $\mathcal{P}$ and $\mathcal{S}$, respectively. Suppose that $a \geq b$. If the makespan is defined on $\mathcal{P}$ by the maximal completion time, then denote the job with this maximal processing time by $j$. Then $p_j = a$, and since our algorithm assigned this job to the set $\mathcal{P}$, we get $s_j > \frac{m}{k}a$. Hence, the optimal cost is at least $a$, and this yields that we have an optimal solution. Now, suppose that the makespan is defined by the load of the jobs. Let $P$ denote the set of the jobs assigned by LG to $\mathcal{P}$, and let $R$ and $Q$ denote the sets of the jobs from $P$ which are assigned to $\mathcal{P}$ and $\mathcal{S}$ in an optimal solution, respectively. Then the optimal cost is at least $\max\{\frac{P_R}{k}, \frac{S_Q}{m}\}$. Furthermore, by the definition of LG, we get that $\frac{P_Q}{k} \leq \frac{S_Q}{m}$. This yields that the cost of the optimal solution is at least $\max\{\frac{P_R}{k}, \frac{P_Q}{k}\} \geq a/2$, and our statement follows.

Suppose that $a < b$. Now, consider two cases depending on the makespan on $\mathcal{S}$. If the makespan is the load, then in the same way as above we obtain that the optimal cost is at least $b/2$, which yields that the algorithm is 2-competitive. Now, suppose that the makespan is defined by the maximal processing time. Denote the job with this maximal processing time by $j$. Then $s_j = b$ and since our algorithm assigned this job to set $\mathcal{S}$, we obtain that $p_j > \frac{k}{m}b$. Hence, the optimal cost is at least $\frac{k}{m}b$, which proves our statement.

To prove that the above analysis is tight, consider one job $(1, \frac{m}{k} - \varepsilon)$. Algorithm $LG$ assigns this job to $\mathcal{S}$ with cost $\frac{m}{k} - \varepsilon$. Since the optimal cost is 1, for a sufficiently small $\varepsilon$, the competitive ratio on this job is arbitrarily close to $\frac{m}{k}$. To prove that the bound 2 is tight, we have to consider a sequence of jobs where the load of each job is the same in the two sets.

□

The above results show that Algorithm LG works well only in the cases when $m/k$ is small. In the next section we present an algorithm which is also efficient in the cases, when $m/k$ is large.

## 7.3 Better algorithm for the general case

This algorithm can be considered as a generalization of the reject total penalty type algorithms for scheduling with rejection, presented in [5] and [38]. The algorithm has two parameters: $0 < \alpha \leq 1$ and $0 < \gamma \leq 1$. The competitive ratio depends on these parameters. We determine the competitive ratio on problem SLS an MLS for arbitrary pair $(\alpha, \gamma)$.

**Algorithm** $A(\alpha, \gamma)$

- 1. *Initialization.* Let $R := \emptyset$.
- 2. When a job $j$ arrives
  - (i) If $\frac{p_j}{k} \leq \frac{\gamma}{m} \cdot s_j$, then assign $j$ to $\mathcal{P}$.
  - (ii) If not (i), then let $r$ be the cost of the optimal offline preemptive scheduling of the set $R \cup \{j\}$ on $\mathcal{P}$. Formally, $r = \max\{\frac{P_{R\cup\{j\}}}{k}, \mathrm{Pmax}_{R\cup\{j\}}\}$. If $r \leq \alpha \cdot s_j$, then
    - (a) Assign $j$ to $\mathcal{P}$,
    - (b) Set $R = R \cup \{j\}$.
  - (iii) Otherwise, assign $j$ to $\mathcal{S}$.

Consider first the case of problem SLS. We prove the following statement.

**Theorem 7.3.1** ([28]) *Algorithm $A(\alpha, \gamma)$ has competitive ratio $C$ on SLS, where*

$$C = \max\{1 + \frac{1}{\alpha}, 1 + \alpha + \gamma, 1 + \frac{1}{\gamma}\}.$$

*Proof:* We prove the above theorem in two steps. First we show that the algorithm is $C$-competitive, and then we prove that this bound is tight. To prove the upper bound, let us consider an arbitrary sequence $L$ of jobs. Fix an optimal schedule of the jobs. Denote $P_{opt}$ the set of jobs which are assigned to $\mathcal{P}$ in this optimal schedule. Let $P_0$ be the set of the jobs with $\frac{p_j}{k} \leq \frac{\gamma}{m} \cdot s_j$, and $P$ be the set of the jobs assigned to $\mathcal{P}$ by our algorithm. Let us observe that, by the definition of the algorithm, we have $P_0 \subseteq P$. Define the following sets

$$X = L \setminus (P_{opt} \cup P), \quad Y = P_{opt} \setminus P,$$

$$Z = P_{opt} \cap (P \setminus P_0), \quad U = P_{opt} \cap P_0,$$

$$V = P_0 \setminus P_{opt}, \quad W = (P \setminus P_0) \setminus P_{opt}.$$

Then the algorithm gives the following cost on $L$:

$$A(L) = \max\{\frac{P_Z + P_U + P_V + P_W}{k}, \ \mathrm{Pmax}_{Z \cup U \cup V \cup W}\} + \max\{\frac{S_X + S_Y}{m}, \ \mathrm{Smax}_{X \cup Y}\}.$$

Furthermore, the optimal cost is $o_1 + o_2$, where

$$o_1 = \max\{\frac{P_Y + P_Z + P_U}{k}, \ \mathrm{Pmax}_{Y \cup Z \cup U}\}$$

$$o_2 = \max\{\frac{S_X + S_V + S_W}{m}, \ \mathrm{Smax}_{X \cup V \cup W}\}$$

are the makespans on the sets $\mathcal{P}$ and $\mathcal{S}$, respectively. To prove the upper bound, we have to show that $A(L) \leq C(o_1 + o_2)$. First let us observe that $C \geq 2$ is obviously valid for each $(\alpha, \gamma)$. Furthermore, let us prove the following inequalities, which will be used many times.

**Lemma 7.3.2** *The following inequalities are valid:*

$$(1) \quad \frac{P_W}{k} \leq \alpha \cdot \mathrm{Smax}_W,$$

$$(2) \quad \mathrm{Pmax}_W \leq \alpha \cdot \mathrm{Smax}_W,$$

$$(3) \quad \alpha \cdot \mathrm{Smax}_Y \leq \max\{\frac{P_Z + P_W + P_Y}{k}, \ \mathrm{Pmax}_{Z \cup W \cup Y}\}.$$

*Proof:* We first prove (1). Let $j$ be the last job from $W$. At the time when it was assigned to $\mathcal{P}$ by the algorithm, we had $r \leq \alpha \cdot s_j$. On the other hand, $j$ is the last job in $W$, thus $\frac{P_W}{k} \leq r$. Furthermore, obviously $s_j \leq \text{Smax}_W$, and the validity of (1) follows. We can prove (2) in the same way as (1). Indeed, let $j$ be a job from $W$ with $p_j = \text{Pmax}_W$. When it was assigned to $\mathcal{P}$, we had $p_j \leq r \leq \alpha \cdot s_j$. This inequality yields (2).

There exists a job $j \in Y$ with $s_j = \text{Smax}_Y$. At the time when it was assigned to $S$, we had $r > \alpha s_j$. On the other hand, $R \cup \{j\} \subseteq Z \cup W \cup Y$ was valid for the considered set $R$, thus $r \leq \max\{\frac{P_Z+P_W+P_Y}{k}, \text{Pmax}_{Z \cup W \cup Y}\}$ was also valid by definition. Hence, the required inequality follows.

□

Using this lemma we can prove the desired upper bound. For this reason consider the following cases.

*Case 1:* Suppose that $A(L) = \frac{P_Z+P_U+P_V+P_W}{k} + \frac{S_X+S_Y}{m}$. The definitions of the sets $V, Y$ and the algorithm yield the following inequalities

$$(4) \quad \frac{P_V}{k} \leq \frac{\gamma}{m} \cdot S_V,$$

$$(5) \quad \frac{S_Y}{m} \leq \frac{P_Y}{k \cdot \gamma}.$$

By inequality (5), we obtain that

$$\frac{P_Z + P_U}{k} + \frac{S_Y}{m} \leq o_1 + (\frac{1}{\gamma} - 1)\frac{P_Y}{k} \leq \frac{1}{\gamma} o_1.$$

On the other hand, by inequalities (1), (4) and by $\gamma < 1$, we obtain that

$$\frac{P_V + P_W}{k} + \frac{S_X}{m} \leq \frac{S_V + S_X}{m} + \alpha \cdot \text{Smax}_W \leq o_2(1 + \alpha).$$

Thus, we proved that $A(L) \leq C(o_1 + o_2)$ in this case.

*Case 2:* Assume that $A(L) = \frac{P_Z+P_U+P_V+P_W}{k} + \text{Smax}_{X \cup Y}$. If $\text{Smax}_{X \cup Y} = \text{Smax}_X$, then by inequalities (1) and (4), we obtain that

$$\frac{P_V + P_W}{k} + \text{Smax}_X \leq \frac{\gamma}{m} \cdot S_V + \alpha \cdot \text{Smax}_W + \text{Smax}_X \leq o_2(1 + \alpha + \gamma).$$

On the other hand, $\frac{P_Z+P_U}{k} \leq o_1$. Hence, we proved that $A(L) \leq C(o_1+o_2)$ in this case. Suppose that $\text{Smax}_{X\cup Y} = \text{Smax}_Y$. By inequality (3), we have the following two possibilities depending on the value of $\text{Smax}_Y$.

*Case 2.a:* Suppose that

$$\text{Smax}_Y \leq \frac{P_Z + P_W + P_Y}{k \cdot \alpha}.$$

Then we have

$$\frac{P_Z}{k}(1+\frac{1}{\alpha}) + \frac{P_Y}{k\cdot\alpha} + \frac{P_U}{k} \leq o_1(1+\frac{1}{\alpha}).$$

Furthermore, by (1) and (4),

$$\frac{P_W}{k}(1+\frac{1}{\alpha}) + \frac{P_V}{k} \leq \text{Smax}_W(1+\alpha) + \gamma\cdot\frac{S_V}{m} \leq o_2(1+\alpha+\gamma).$$

Consequently, we proved that $A(L) < C(o_1+o_2)$.

*Case 2.b:* Now suppose that $\text{Smax}_Y \leq \frac{\text{Pmax}_{Z\cup W\cup Y}}{\alpha}$. If $\text{Pmax}_{Z\cup W\cup Y} = \text{Pmax}_{Z\cup Y}$, then $\frac{P_Z+P_U}{k} + \text{Smax}_Y \leq o_1(1+\frac{1}{\alpha})$, and $\frac{P_V+P_W}{k} \leq o_2(\alpha+\gamma)$, hence our statement follows. If $\text{Pmax}_{Z\cup W\cup Y} = \text{Pmax}_W$, then by (2), $\text{Pmax}_W \leq \alpha\ \text{Smax}_W$, thus

$$\frac{P_V+P_W}{k} + \text{Smax}_Y \leq \frac{\gamma}{m}S_V + (\alpha+1)\ \text{Smax}_W \leq o_2(1+\alpha+\gamma),$$

and the desired bound on $A(L)$ follows.

*Case 3:* Suppose that $A(L) = \text{Pmax}_{Z\cup U\cup V\cup W} + \frac{S_X+S_Y}{m}$. Now, we have to distinguish three cases depending on the value of $\text{Pmax}_{Z\cup U\cup V\cup W}$.

*Case 3.a:* Suppose that $\text{Pmax}_{Z\cup U\cup V\cup W} = \text{Pmax}_{Z\cup U}$. In this case, by (5), we obtain that

$$\text{Pmax}_{Z\cup U} + \frac{S_Y}{m} \leq \text{Pmax}_{Z\cup U} + \frac{P_Y}{k\cdot\gamma} \leq o_1(1+\frac{1}{\gamma}).$$

Thus, by $\frac{S_X}{m} \leq o_2$, we obtain that $A(L) \leq C\cdot(o_1+o_2)$ in this case.

*Case 3.b:* Suppose that $\text{Pmax}_{Z\cup U\cup V\cup W} = \text{Pmax}_V$. Then, by the definition of $V$, we have $\frac{\text{Pmax}_V}{k} \leq \gamma\cdot\frac{\text{Smax}_V}{m}$, and thus

$$\frac{S_X}{m} + \text{Pmax}_V \leq (1+\frac{k\cdot\gamma}{m})o_2 < C\cdot o_2.$$

Furthermore, by (5), $\frac{S_Y}{m} \leq \frac{o_1}{\gamma}$, hence we proved our statement for this case.

*Case 3.c:* Suppose that $\text{Pmax}_{Z \cup U \cup V \cup W} = \text{Pmax}_W$. Then, by (2), $\text{Pmax}_W \leq \alpha o_2$, and thus, $A(L) \leq \frac{1}{\gamma} o_1 + (1+\alpha) o_2$, and the statement follows.

*Case 4:* Suppose that $A(L) = \text{Pmax}_{Z \cup U \cup V \cup W} + \text{Smax}_{X \cup Y}$. In this case, if $\text{Smax}_{X \cup Y} = \text{Smax}_X$, then it is upper bounded by $o_2$. Furthermore, as we have seen in the earlier cases

$$\text{Pmax}_{Z \cup U \cup V \cup W} \leq \max\{o_1, \frac{k \cdot \gamma}{m} o_2, \alpha \cdot o_2, \}.$$

This yields that $A(L) \leq 2 \cdot o_1 + 2 \cdot o_2$. Therefore, it is enough to consider the case $\text{Smax}_{X \cup Y} = \text{Smax}_Y$. Then, we have an upper bound on $\text{Smax}_Y$ from (3). If $\text{Smax}_Y \leq \frac{\text{Pmax}_{Z \cup W \cup Y}}{\alpha}$, then we have (*cf.* Case 2) that

$$\text{Smax}_Y \leq \max\{\frac{o_1}{\alpha}, o_2\}.$$

This inequality and the above bound on $\text{Pmax}_{Z \cup U \cup V \cup W}$ imply the upper bound in this case. Therefore, the only remaining case is that

$$\text{Smax}_Y \leq \frac{P_Z + P_W + P_Y}{k \cdot \alpha}.$$

Then, by (1) and the definition of $o_1$ and $o_2$, we obtain that $\text{Smax}_Y \leq \frac{o_1}{\alpha} + o_2$. This bound together with the bound on $\text{Pmax}_{Z \cup U \cup V \cup W}$ imply immediately the desired statement.

Since we considered all the possible cases, we proved that our algorithm is $C$-competitive.

Now, we prove that the upper bound $C$ is tight for the algorithm. We show this statement by the following examples. First let $\varepsilon > 0$ be arbitrary, and choose a value of $k$ and $m$ for which $\alpha \cdot (\frac{m}{k \cdot \gamma} - \varepsilon) > 1$ and $\frac{\gamma}{m \cdot \alpha - \gamma} + \frac{\varepsilon}{k} < 1$ are valid. Consider the following sequence of jobs. The first part contains $k$ jobs of size $(1, \frac{m}{k \cdot \gamma} - \varepsilon)$, and the last job is the size of $(\frac{k \cdot \gamma}{m \cdot \alpha - \gamma} + \varepsilon, \frac{m}{m \cdot \alpha - \gamma})$. Algorithm $A(\alpha, \gamma)$ assigns the first $k$ jobs in Step (ii) to $\mathcal{P}$. Then the $k+1$th job arrives. Since

$$\frac{k + \frac{k \cdot \gamma}{m \cdot \alpha - \gamma} + \varepsilon}{k} = 1 + \frac{\varepsilon}{k} + \frac{\gamma}{m \cdot \alpha - \gamma} > \frac{\alpha \cdot m}{m \cdot \alpha - \gamma},$$

this job is assigned to $\mathcal{S}$. This yields that the schedule, produced by the algorithm has $1+\frac{m}{m\cdot\alpha-\gamma}$ cost. If we assign each job to $\mathcal{P}$, we obtain a schedule with cost

$$\frac{k+\frac{k\cdot\gamma}{m\cdot\alpha-\gamma}+\varepsilon}{k}=\frac{m\cdot\alpha}{m\cdot\alpha-\gamma}+\frac{\varepsilon}{k}.$$

This yields that by choosing a sufficiently small $\varepsilon$, the competitive ratio on the considered sequence can be arbitrarily close to

$$1+(1-\frac{\gamma}{m})\frac{1}{\alpha}.$$

As $m$ increases to infinity, we obtain that the first bound is tight.

To prove the tightness of the second bound, consider the following sequence of jobs. The first part contains $m\cdot k$ jobs of size $(\frac{\gamma(m-1)}{m},\frac{m-1}{k})$, the second part contains $k$ jobs of size $(\alpha\cdot m,m)$, finally, the last job has size $(\infty,m)$. Then, $A(\alpha,\gamma)$ assigns the first $m\cdot k$ jobs to $\mathcal{P}$ in Step (i), and the next $k$ jobs in Step (ii). Finally, it assigns the last job to $\mathcal{S}$. It has a cost $(1+\alpha)m+\gamma(m-1)$. If we assign each job to $\mathcal{S}$, then we have the cost

$$\frac{m(m-1)+m+k\cdot m}{m}=m+k.$$

Hence, considering this sequence, the algorithm has the competitive ratio

$$\frac{(1+\alpha)m+\gamma(m-1)}{m+k}.$$

Fixing $k$ and increasing $m$ to infinity, we obtain that the second bound is tight.

To prove the tightness of the third bound, choose again a small $\varepsilon>0$ and a great integer $M$, and consider the following sequence of jobs. The first job is $(p_1,s_1)$, where

$$p_1=\alpha\cdot\frac{m}{k\cdot\gamma}+\varepsilon,\quad s_1=\alpha\cdot\frac{m^2}{k^2\cdot\gamma^2}.$$

The second part contains $M$ jobs of size $(1+\varepsilon,\frac{m}{k\cdot\gamma})$ and the last job is $(\frac{(1+\varepsilon)M}{k-1},\frac{(1+\varepsilon)M}{k-1}\frac{m}{k\cdot\gamma})$. One can easily see that we can choose such $k$ and $m$ for which $A(\alpha,\gamma)$ assigns the first job in Step (ii) to $\mathcal{P}$. Then, it will assign the

next $M$ jobs to $S$, and finally, the last job to $\mathcal{P}$. This yields that its cost is at least

$$\frac{(1+\varepsilon)M}{k-1} + \frac{M}{k \cdot \gamma}.$$

If we assign each job to $\mathcal{P}$, then the cost is $\frac{(1+\varepsilon)M}{k-1} + \frac{p_1}{k}$. As $M$ increases to infinity and $\varepsilon$ decreases to 0, the competitive ratio of the algorithm tends to

$$1 + \frac{k-1}{k} \cdot \frac{1}{\gamma},$$

and the tightness of the last bound follows.

□

To optimize the competitive ratio over the possibilities of $\alpha, \gamma$, we have to choose $\alpha = \gamma = 1/\sqrt{2}$. Substituting these values into Theorem 7.3.1, we obtain the following result.

**Corollary 7.3.3** ([28]) *The algorithm $A(1/\sqrt{2}, 1/\sqrt{2})$ is $1+\sqrt{2}$ competitive on SLS.*

For the *SOSR* problem, the algorithm has a better competitive ratio. It is worth noting that in this case the algorithm is reduced to the total rejection scheme from [38].

**Theorem 7.3.4** ([28]) *Algorithm $A(\alpha, \gamma)$ has a competitive ratio $C$ on the SOSR problem where*

$$C = \max\{1 + \frac{1}{\alpha}, 1 + \alpha + \gamma, \frac{1}{\gamma}\}.$$

*Proof:* To prove that the algorithm is $C$-competitive, we just have to simplify the upper bound proof of Theorem 7.3.1. Since $k = 1$, the makespan on $\mathcal{P}$ is the sum of the jobs. This yields that we just have to consider Case 1 and Case 2. Moreover, in this case we can give a sharper upper bound for $\text{Smax}_Y$: it is not greater than $(P_Z + P_W + R + P_Y)/\alpha$, and this yields that in Case 2 we have to consider only Case 2.a. We can apply the first two examples from the proof of Theorem 7.3.1 to show the tightness of the first two bounds. Moreover, we can show the tightness of the last bound as follows. (This example was considered in [38]). Let $0 < \varepsilon \leq \alpha - \gamma/m$, and

consider $M$ jobs of size $(\gamma/m+\varepsilon, 1)$. From this sequence the first $i$ jobs are assigned to $\mathcal{P}$, where $i$ is the greatest integer with $i(\gamma/m+\varepsilon) \leq \alpha$, the others are assigned to $\mathcal{S}$. Thus, by choosing a sufficiently great $M$, the cost of the assignment produced by the algorithm is $i(\gamma/m+\varepsilon) + (M-i)/m$. If we assign each job to $\mathcal{P}$, then we have $M(\gamma/m+\varepsilon)$ cost, hence, fixing $m$ and increasing $M$ to infinity, the tightness of the third bound follows.

□

To minimize the competitive ratio over $(\alpha, \gamma)$, we have to choose the values $\bar{\alpha} = 0.802$ and $\bar{\gamma} = 0.445$. By substituting to Theorem 7.3.4, we obtain the following result.

**Corollary 7.3.5** ([28]) *Algorithm* $A(0.802, 0.445)$ *is* 2.247*-competitive on problem SOSR.*

In case of the problem $MLS$, we can do a similar analysis. We can prove the following theorem.

**Theorem 7.3.6** ([27]) *The competitive ratio of algorithm* $A(\alpha, \gamma)$ *is* $C$ *on the problem* $MLS$*, where*

$$C = \max\{1+\frac{1}{\alpha}, 1+\alpha+\gamma, 1+\frac{1}{\gamma}\}.$$

*Proof:* First we prove that the algorithm is $C$-competitive. Let us consider an arbitrary sequence of jobs, and denote the list of the jobs by $L$. Fix an optimal schedule of the jobs. Define the sets $P_{opt}, P_0, X, Y, Z, U, V, W$ in the same way as in the proof of Theorem 7.3.1.
Then the algorithm gives the following cost on $L$:

$$A(L) = \max\{\frac{P_Z + P_U + P_V + P_W}{k},\ \mathrm{Pmax}_Z,\ \mathrm{Pmax}_U,$$

$$\mathrm{Pmax}_V,\ \mathrm{Pmax}_W, \frac{S_X + S_Y}{m},\ \mathrm{Smax}_X,\ \mathrm{Smax}_Y\}.$$

The optimal cost is

$$OPT(L) = \max\{\frac{P_Y + P_Z + P_U}{k},\ \mathrm{Pmax}_Y,\ \mathrm{Pmax}_Z,$$

$$\text{Pmax}_U, \frac{S_X + S_V + S_W}{m}, \text{Smax}_X, \text{Smax}_V, \text{Smax}_W\}.$$

Let us observe that the proof of Lemma 7.3.2 depends only on the definition of the algorithm, therefore, this lemma is also valid for the $MLS$ problem.

Using this lemma, we can prove the desired upper bound. For this reason, let us consider the following cases.

*Case 1:* Suppose that $A(L) = \max\{\text{Pmax}_Z, \text{Pmax}_U, \text{Smax}_X\}$. In this case $A(L) \leq OPT(L)$, thus the algorithm results in an optimal solution.

*Case 2:* Suppose that $A(L) = \frac{P_Z+P_U+P_V+P_W}{k}$. Then, the definition of the set $V$ yields that $\frac{P_V}{k} \leq \frac{\gamma}{m} \cdot S_V \leq \gamma OPT(L)$. On the other hand $\frac{P_Z+P_U}{k} \leq OPT(L)$. Furthermore, by Lemma 7.3.2, we have that $P_W/k \leq \alpha\ \text{Smax}_W \leq \alpha OPT(L)$. Therefore, $A(L) \leq (1+\alpha+\gamma)OPT(L)$ in this case.

*Case 3:* Suppose that $A(L) = \text{Pmax}_V$. Then, by the definition of the set $V$, $\text{Pmax}_V/k \leq \gamma\ \text{Smax}_V/m$. Since $k \leq m$, this yields, that $\text{Pmax}_V \leq \gamma\ \text{Smax}_V \leq \gamma OPT(L)$. This is possible only in the case when $\gamma = 1$, and in this case the algorithm results in an optimal solution.

*Case 4:* Suppose that $A(L) = \text{Pmax}_W$. Then, by Lemma 7.3.2, $\text{Pmax}_W \leq \alpha\ \text{Smax}_W \leq \alpha \cdot OPT(L)$, therefore, this case is possible only if $\alpha = 1$ and the algorithm gives an optimal solution.

*Case 5:* Suppose that $A(L) = \frac{S_X+S_Y}{m}$. Now, by the definition of the set $Y$, we have that $\frac{S_Y}{m} \leq \frac{P_Y}{\gamma k} \leq OPT(L)/\gamma$. Thus, we obtain that $A(L) \leq (1+1/\gamma)OPT(L)$.

*Case 6:* Suppose that $A(L) = \text{Smax}_Y$. By Lemma 7.3.2, this yields that

$$A(L) \leq \frac{1}{\alpha}\max\{\frac{P_Z + P_W + P_Y}{k}, \text{Pmax}_{Z\cup W\cup Y}\}.$$

Consider three subcases. If $A(L) \leq \frac{P_Z+P_W+P_Y}{\alpha k}$, then since $\frac{P_W}{\alpha k} \leq \text{Smax}_W \leq OPT(L)$ (by Lemma 7.3.2), we obtain that $A(L) \leq (1+1/\alpha)OPT(L)$. If $A(L) \leq 1/\alpha\ \text{Pmax}_{Z\cup Y}$, then we obtain immediately that $A(L) \leq OPT(L)/\alpha$. Finally, if $A(L) \leq 1/\alpha\ \text{Pmax}_W$, then by Case 4, we obtain that $A(L) \leq OPT(L)$.

Since we considered all the possible cases, we proved that the algorithm is $C$-competitive. We can prove that the bound $C$ is tight by the following examples.

First assume that $k = 1$ and $m > \gamma \cdot (1 + \alpha)/\alpha$. Consider the following sequence, which contains two jobs. Let the first job be $(\alpha \cdot M, M)$, the second one be $(M + \varepsilon, M(1+\alpha)/\alpha)$ for some great $M$ and small $\varepsilon$. By the definition of $m$ we have that $\alpha M > \gamma M/m$, thus the first job is assigned to $\mathcal{P}$ in Step (ii). The second job is assigned to $\mathcal{S}$. Therefore, the cost of the algorithm is $M(1 + \alpha)/\alpha$ on this sequence. The optimal cost is $M + \varepsilon$, we assign the first job to $\mathcal{S}$ and the second to $\mathcal{P}$. As $M$ tends to $\infty$, the ratio of these costs tends to $1 + 1/\alpha$, hence we proved that the first bound is tight.

To prove the tightness of the second bound, fix the value of $k$ and let $m$ be much greater than $k$. Consider the following sequence of jobs. First consider $M(m - k)$ jobs of size $(\gamma \cdot k/m, 1)$. The second part of the sequence contains $k$ jobs of size $(M, \infty)$, finally the third part contains $k$ jobs of size $(\alpha \cdot M, M)$. Then the first and second part is assigned to $\mathcal{P}$ in Step (i), the third part is also assigned to $\mathcal{P}$ in Step (i) or in Step (ii). Therefore, the cost of the algorithm is

$$\frac{M(m-k)\gamma k/m + Mk + \alpha Mk}{k}.$$

The optimal solution assigns the first and the third part to $\mathcal{S}$, and the second part to $\mathcal{P}$ and its cost is $M$. As $m$ tends to $\infty$, the ratio of these costs tends to $1 + \alpha + \gamma$, hence,.we proved that the second bound is tight.

To prove that the third bound is tight, consider such $k$ and $m$ that satisfy the inequality $\alpha/k > \gamma/m$ and the following sequence of jobs. The first part of the sequence is one job of size $(\alpha(m/(\gamma k)+2\varepsilon), (m/(\gamma k)+2\varepsilon))$. The second part contains $Mk$ jobs of size $(1, m/(\gamma k) + \varepsilon)$, and the third part contains $m$ jobs of size $(\infty, M)$. Then, the algorithm assigns the first job to $\mathcal{P}$ in Step (ii), and assigns the other jobs to $\mathcal{S}$. Therefore, its cost is

$$\frac{Mk(m/(\gamma k) + \varepsilon) + mM}{m}.$$

There is a feasible schedule which assigns the second part of the jobs to $\mathcal{P}$ and the third part to $\mathcal{S}$, therefore, the optimal cost is no more than

$M + m/(\gamma k) + 2\varepsilon$. As $M$ tends to $\infty$ and $\varepsilon$ tends to 0, the ratio of these costs tends to $1 + 1/\gamma$, hence, we proved that the third bound is tight.

□

To find the best values of $\alpha, \gamma$, we have to choose $\alpha = \gamma = 1/\sqrt{2}$. By substituting these values into Theorem 7.3.6, we obtain the following result.

**Corollary 7.3.7** ([27]) *The algorithm $A(1/\sqrt{2}, 1/\sqrt{2})$ is $1+\sqrt{2}$-competitive on MLS.*

## 7.4 Lower bounds

In this section we present some lower bounds, one for the $SOSR$ problem, one for the $SLS(k, m)$ problem with $k \geq 2$, and one for the $MLS$ problem. The first lower bound is presented in [5] for the online scheduling with rejection, but it is also valid with the same proof for $SOSR$. Therefore, we only recall here this result and we omit the proof of the statement.

**Theorem 7.4.1** ([5], [27]) *Assume that an online algorithm is c-competitive on the problem $SOSR(m)$. Then c satisfies the following inequality:*

$$c^{m-1} + c^{m-2} + \ldots + 1 \leq c^m.$$

Theorem 7.4.1 yields immediately the following statement for the $SOSR$ problem.

**Corollary 7.4.2** ([27]) *If an online algorithm is c-competitive on the problem $SOSR$, then $c \geq 2$.*

For the case $k \geq 2$, we can obtain a lower bound as follows.

**Theorem 7.4.3** ([28]) *If an online algorithm is c-competitive on the problem $SLS(k, m)$ with $k \geq 2$, then $c \geq 2$.*

*Proof:* Consider a job which is $(1, 1)$. If the algorithm assigns it to $\mathcal{P}$, then the second job is $(2, 1)$, otherwise it is $(1, 2)$. This yields that the cost of the algorithm is 2. The optimal algorithm assigns the two jobs to the same set and it has cost 1.

□

By this observation and Theorem 7.2.1, we obtain immediately the following result.

**Corollary 7.4.4** *Algorithm LG is optimal for SLS(k,k).*

We can prove the following statement for the problem $MLS$.

**Theorem 7.4.5** ([27]) *If an online algorithm is c-competitive on problem* $MLS(k,m)$, *then* $c \geq (1+\sqrt{5})/2 \approx 1.618$.

*Proof:* We prove this statement by contradiction. Suppose that there is a pair $(k,m)$ and an algorithm $A$ which is $c < (1+\sqrt{5})/2$-competitive for the problem $MLS(k,m)$. Consider the following list $L$ of jobs. Let the first $k$ jobs be $((\sqrt{5}-1)/2, 1)$ and the next $k$ jobs $(1,\infty)$. In this case, the optimal offline algorithm assigns the first $k$ jobs to $\mathcal{S}$, and the next $k$ jobs to $\mathcal{P}$, hence $OPT(L) = 1$. On the other hand, since $A$ is $c$-competitive it must assign the first $k$ jobs to $\mathcal{P}$, otherwise we omit the next $k$ jobs, and the offline optimum is $(\sqrt{5}-1)/2$, while the cost of the algorithm is 1. Therefore, the online algorithm must assign every job to $\mathcal{P}$, thus it has a makespan $(1+\sqrt{5})/2$. This yields that $A(L)/OPT(L) = (1+\sqrt{5})/2$, which is a contradiction.

□

We can obtain a sharper, general lower bound as follows.

**Theorem 7.4.6** ([27]) *Let k be a fixed constant. If an online algorithm is c-competitive for every m on the problem* $MLS(k,m)$, *then* $c \geq (3+\sqrt{17})/4 \approx 1.781$.

*Proof:* We prove this statement by contradiction. Suppose that there exist such $k$ and an online algorithm $A$ that is $c$-competitive for every $m$ on the problem $MLS(m,k)$, where $c < (3+\sqrt{17})/4$. For the sake of simplicity, in the rest of the proof we denote the number $(3+\sqrt{17})/4$ by $b$. Let $m$ be greater than $5k$ and consider the following sequence of jobs. Let the first $k$ jobs be $(1/b, 1)$ and the following $m-k$ jobs be $(\frac{1}{m-k}, 1)$. Finally, depending on the decisions made by $A$, we finish the sequence with $k$ jobs which are $(1,\infty)$, this is list $L_1$, or we finish the list with $m-k$ jobs which are $(\infty, 1)$, this is list $L_2$.

Consider first the offline optimum. In the first case we can assign the first $m$ jobs to $\mathcal{S}$ and the last $k$ jobs to $\mathcal{P}$, which assignment gives makespan 1. In the second case, we can assign the first $k$ jobs and the last $m-k$ jobs to $\mathcal{S}$, and the other $m-k$ jobs to $\mathcal{P}$, thus we can obtain makespan 1. Therefore, $OPT(L_1) \leq 1$, and $OPT(L_2) \leq 1$.

Consider the algorithm $A$. Since $A$ is online algorithm, it cannot see any difference between $L_1$ and $L_2$ before it gets the $m+1$-th job, thus it has the same behaviour in both cases. Furthermore, $A$ is $c$-competitive with $c < b$, therefore it must assign the first $k$ jobs to $\mathcal{P}$, otherwise we get a contradiction in the same way as in the proof of Theorem 7.4.5. From the next $m-k$ jobs $A$ can assign $x$ to $\mathcal{P}$ and $m-k-x$ to $\mathcal{S}$. Therefore, in case of list $L_1$ we have that $A(L_1) \geq \frac{1}{b} + \frac{x}{m-k} + 1$ and in case of list $L_2$ we get that $A(L_2) \geq \frac{2m-2k-x}{m}$. The algorithm can choose $x$ to be any integer between 1 and $m-k$, and we can choose the list which yields the larger makespan. Therefore, since the offline optimum is at most 1 for both lists, and $A$ is $c$-competitive, we have that

$$c \geq \min_{1 \leq x \leq m-k} \max\{\frac{1}{b} + \frac{x}{m-k} + 1, \frac{2m-2k-x}{m}\}.$$

Here we omitted the condition that $x$ is an integer. This does not cause any problem since it decreases the right side of the inequality. It can be easily seen that the function of $x$, which is on the right side of the inequality is minimal for

$$x = \frac{m(m-k)}{2m-k}(\frac{m-2k}{m} - \frac{1}{b}).$$

If we substitute this value into the bound we have for $c$, we obtain that

$$c \geq \frac{1}{b} + \frac{3m-3k}{2m-k} - \frac{m}{(2m-k)b}.$$

Since this inequality is valid for arbitrary $m$, it is also valid if we let $m$ to tend to infinity. Therefore,

$$c \geq \frac{3}{2} + \frac{1}{2b}.$$

On the other hand, $b = \frac{3}{2} + \frac{1}{2b}$, and thus, we obtain that $c \geq b$, which is a contradiction.

□

# Chapter 8

# Online strip packing with modifiable boxes

In this chapter a modified strip packing model is introduced, which describes the problem of scheduling with shared resource. We establish and analyse two online algorithms for the solution of this problem. Finally, a general lower bound is presented.

## 8.1 Problem definition

In the strip packing problem there is a set of two dimensional boxes, defined by their widths and heights, and the task is to pack them without rotation into a vertical strip of width $W$ by minimizing the total height of the strip. This problem appears in many situations. Usually, scheduling of tasks with shared resource involves two dimensions, the resource and the time. We can consider the widths as the resource and the heights as the time. Our goal is to minimize the total amount of time used. Some applications can be found in computer scheduling problems.

Only few online algorithms are presented for this problem, the first algorithms, the shelf algorithms are developed in [4]. The best presented algorithms are 7.46 and 6.99-competitive. A lower bound 2 for the competitive ratio of any online strip packing algorithm is presented in [3]. An improved shelf algorithm and a generalized definition of shelf algorithms are defined in [15].

Let us suppose that the two dimensions of the boxes are the required resource and time. The given parameters, however, show only one possible configuration: one can also satisfy the task by using less resource. Of course, using less resource means that it takes more time to satisfy the task. We can give a mathematical model for this extended problem by slightly changing the strip packing model. In the modified model the width of the box gives the maximal resource which can be used to satisfy the task and the height of the box gives the time which is necessary using this amount of resource. The fact that we can use less resource for more time means that we can lengthen the box, keeping the area fixed. A similar question with different model is investigated in [16]. The main difference between the two models that in [16] the resource is measured by the number of the used processors and the geometrical structure of the processors' network is also considered.

In [31] we study the online version of this problem where the boxes arrive from a list and we have to lengthen and pack each box without any knowledge on further boxes. The mathematical model of the problem can be given as follows. There is a list of rectangles $p_i = (w_i, h_i)$, $i = 1, \ldots,$ where each rectangle is defined by two parameters: by its width and its height. Our goal is to pack the rectangles into a vertical strip so as to minimize the total height of the strip needed. Before packing the rectangles, we can lengthen them, we can increase the height and decrease the width, keeping the area fixed. After lengthening the rectangle, we have to pack it into the strip, the rectangles cannot be rotated and they cannot overlap. We call this problem *strip packing with modifiable boxes.* In the following part of this chapter we will assume that the width of the strip is 1.

For the analysis of the online algorithms, the following observation on the offline optimal solution will be useful.

**Proposition 8.1.1** *The optimal solution of the offline strip packing problem with modifiable boxes has*

$$OPT(L) = \max\{S, H\}$$

*total height, where* $S = \sum_{i \in L} w_i \cdot h_i$ *and* $H = \max_{i \in L} h_i$.

*Proof:* First consider the case when $S > H$. Then, it is possible to lengthen each rectangle so as to have $S$ height. Packing the lengthened

rectangles into the bottom of the strip, we get a packing with $S$ total height. On the other hand, the sum of the areas of the rectangles does not change, thus it is not possible to pack the rectangles so that they have smaller total height than $S$. The other case is similar, the only difference is that we have to lengthen each rectangle to the same height as the highest one.

□

## 8.2 Online algorithms

One basic way of packing into the strip is to define shelves and pack the rectangles into the shelves. By *shelf* we mean a rectangular part of the strip with the width of 1. Shelf packing algorithms place each rectangle into one of the shelves. If the algorithm decides which shelf will contain the rectangle, then first the rectangle is lengthened to the same height as the shelf has. The lengthened rectangle is placed into the shelf as much to the left as it is possible without overlapping the other rectangles have already placed earlier into the shelf considered. Therefore, after the arrival of a rectangle, the algorithm has to make two decisions. The first decision is whether to create a new shelf or not. If the algorithm creates a new shelf, it also has to decide the height of the new shelf. The created shelves always start from the top of the previous shelf. The first shelf is placed to the bottom of the strip. The algorithm also has to choose the shelf to which it puts the rectangle. In what follows, we will say that it is possible to pack a rectangle into a shelf if there is room enough for the lengthened rectangle in the shelf.

We consider two algorithms for this problem. The first algorithm is an extended version of the $NFS_r$ (*next fit shelf* $r$) *algorithm*, which is presented in [4]. We also denote the extended version by $NFS_r$. This algorithm depends on a parameter $r > 1$ and it works as follows.

### Algorithm $NFS_r$

When a rectangle $p_i = (w_i, h_i)$ arrives, choose a value for $k$ that satisfies $r^k < h_i \leq r^{k+1}$. Lengthen the rectangle to the form $(w_i \cdot h_i/r^{k+1}, r^{k+1})$. If there is an active shelf with height $r^{k+1}$ and it is possible to pack the rectangle into it, then pack it there. If there is no active shelf with height $r^{k+1}$, or it is not possible to pack the rectangle into the active shelf with height $r^{k+1}$,

then create a new shelf with height $r^{k+1}$, put the rectangle into it, and let this new shelf be the active shelf with height $r^{k+1}$.

The following theorem gives the competitive ratio of our algorithm.

**Theorem 8.2.1** ([31]) *The competitive ratio of algorithm $NFS_r$ is $2+\frac{r^2}{r-1}$. By choosing an optimal $r$, we obtain a 6-competitive algorithm.*

*Proof:* First we prove that the algorithm is $2+\frac{r^2}{r-1}$-competitive. Consider an arbitrary list $L$ of rectangles. Let $H_A$ denote the sum of the heights of the shelves which are active at the end of the packing, and let $H_D$ be the sum of the heights of the other shelves. Let $h$ be the height of the highest active shelf, and let $H$ be the height of the highest rectangle. Since the algorithm created a shelf with height $h$, we have $H > h/r$. As there is at most 1 active shelf for each height

$$H_A \leq h\sum_{i=0}^{\infty} r^{-i} = \frac{hr}{r-1} \leq \frac{Hr^2}{r-1}.$$

Consider the shelves which are not active at the end. Let $S$ be one of these shelves with height $r^k$. The next shelf $S'$ with height $r^k$ contains one rectangle which (after the lengthening) would not fit into $S$. Therefore, the total sum of the rectangles' areas packed into $S$ and $S'$ is at least $r^k$. If for each $k$ we pair the shelves of height $r^k$ in this way, using the active shelf if the number of the shelves of the considered height is odd, we obtain that $H_D$ is not greater than twice the sum of the rectangles' areas. This yields that $H_D \leq 2OPT(L)$. Using this inequality and the fact that $H \leq OPT(L)$, we obtain the desired statement.

Now, we prove that the upper bound on the competitive ratio is tight. Let $\varepsilon > 0$ be arbitrary, and let $n$ be an integer with the property $r^{-n} < \varepsilon$. Consider the following list $L_\varepsilon$ of rectangles. Let the first $2 \cdot \lfloor r^n \rfloor$ rectangles be $(1/2+\varepsilon/r^n, r^{-n})$, and let the second part of the list be $p_i = (\varepsilon, r^{-i}+\varepsilon)$, $i = n-1, \ldots, 0$. Using these lists, a simple calculation shows that the ratio $NFS_r(L_\varepsilon)/OPT(L_\varepsilon)$ tends to $2+\frac{r^2}{r-1}$ as $\varepsilon$ tends to 0.

□

Choosing an optimal value for $r$ we obtain the following corollary:

**Corollary 8.2.2** *The competitive ratio of algorithm $NFS_2$ is 6.*

The $NFS_r$ algorithm is called next fit shelf algorithm since it can be interpreted as an algorithm which considers the shelves like bins and uses the well-known bin packing algorithm to pack them. There are similar algorithms for the original strip packing problem with the difference that they are using more efficient bin packing algorithms to pack the shelves. Many of these algorithms can be modified for this problem. On the other hand, we present a more efficient algorithm below so we do not deal with the improved versions of this algorithm.

The second algorithm uses a similar idea like the scheduling algorithm from [39]. It can be given as follows.

### Algorithm DS

After the arrival of the first rectangle create a shelf with height $h_1$, and let this shelf be the active shelf. Later, when a rectangle arrives, use the following iteration to pack it.

*Step 1.* If it is possible to pack the rectangle into the active shelf, pack it (first lengthen it), otherwise go to Step 2.

*Step 2.* Create a new shelf which is twice higher than the active shelf, let the new shelf be the active shelf and go to Step 1.

The competitive ratio of the algorithm is given by the following theorem.

**Theorem 8.2.3** ([31]) *The competitive ratio of algorithm DS is* 4.

*Proof:* First we show that the algorithm is 4-competitive. Consider an arbitrary list of rectangles. Denote the list by $L$. Let $H$ denote the height of the shelf which is active at the end. When the algorithm created this shelf, it was not possible to pack the rectangle into the previous active shelf, hence, we obtain that the height of the previous active shelf is smaller than $OPT(L)$. Therefore, $H \leq 2 \cdot OPT(L)$. On the other hand, the heights of the other shelves are $H/2, H/4, \ldots, H/2^i$. Thus, the total height of the packing is smaller than $2H$, and the result follows. To see that the upper bound is tight, consider the following list $L_i$ of rectangles. Let the first $i$ rectangles be $(1/4, 2^j)$, $j = 1, \ldots, i$, and let the last rectangle be $(1/4, 2^i + 1)$. A simple calculation shows that the ratio $DS(L_i)/OPT(L_i)$ tends to 4 as $i$ tends to infinity, which yields the tightness of the upper bound.

Algorithm DS has a better competitive ratio than algorithm $NFS_r$. On the other hand, $NFS_r$ has some advantage, it does not create much higher shelves, than the maximal height of the rectangles. Therefore, it can be also useful in some more restricted models.

## 8.3 Lower Bounds

First we introduce the following definitions. For any rectangle $p_i$ we denote the lengthened rectangle by $p_i'$. Furthermore, two rectangle $p_i$ and $p_j$ are called to be *colateral in a packing* if there is a horizontal line which intersects both $p_i$ and $p_j$. Using these definitions, we can prove the following lower bound.

**Theorem 8.3.1** ([31]) *There is no online algorithm for the strip packing with modifiable boxes that has smaller competitive ratio than $C$, where $C \approx 1.73$ is the solution of the equation $e^{-(c+1/c-2)} = c - 1$.*

*Proof:* Contrary, let us suppose that we have an algorithm with smaller competitive ratio than $c$. Consider such an algorithm and denote it by $A$, further denote its competitive ratio by $d$. Investigate the following list $L_n$ of rectangles. Let the first rectangle be $p_1 = (1, 1)$. This rectangle is lengthened to $p_1' = (x, 1/x)$, with $1 \leq x \leq d$. Furthermore, $x > 1$ since in the opposite case a second rectangle $(\varepsilon, 1)$ with a small $\varepsilon$ implies that the algorithm is not better than 2-competitive. Let the second rectangle be $p_2 = (1 - 1/x, x)$, and the $i + 2$-th rectangle be $(1/n, x(\frac{n}{n-1})^i)$ for $i = 1, 2 \ldots$. The sequence of rectangles is stopped when the first rectangle is packed which is not colateral with the others. Suppose that the sequence is finished after the $j + 2$-th rectangle. Then by Proposition 8.1.1, we obtain that the optimal offline cost is

$$x(1 + \sum_{i=1}^{j} 1/n \cdot (\frac{n}{n-1})^i) = x(\frac{n}{n-1})^j.$$

Since the first rectangle is lengthened to $x$ and all the others are higher than $x$, the algorithms cost is at least

$$x + x(\frac{n}{n-1})^j.$$

Therefore,

$$A(L_n)/OPT(L_n) \geq \frac{x + x(\frac{n}{n-1})^j}{x(\frac{n}{n-1})^j} = 1 + (\frac{n-1}{n})^j.$$

On the other hand, it follows by Proposition 8.1.1, that the optimal offline cost after the $i+2$-th rectangle, $i = 0, \ldots, j-1$ is $x(\frac{n}{n-1})^j$, thus the $d$-competitive algorithm can lengthen these rectangles at most $d$ times higher. Therefore, the sum of the widths of the first $j+1$ lengthened rectangles is at least

$$1/x + 1/d(1 - 1/x + \sum_{i=1}^{j-1} 1/n).$$

Since these rectangles are packed colateral, this total width is not greater than 1. Consequently, $j \leq n(1-1/x)(d-1)+1$. Using this bound, we obtain that

$$A(L_n)/OPT(L_n) \geq 1 + (\frac{n-1}{n})^{n(1-1/x)(d-1)+1}.$$

The function on the right side of the inequality is decreasing in $x$ on the interval $[1, d]$, therefore it is minimal when $x = d$. Furthermore, $(\frac{n-1}{n})^n$ tends to $e^{-1}$ as $n$ goes to infinity, and $d \geq A(L_n)/OPT(L_n)$ for each $n$. This yields the following inequality

$$d \geq 1 + e^{-(1-1/d)(d-1)} = 1 + e^{-(d+1/d-2)}.$$

On the other hand, taking the derivative, it can be easily seen that the function $d-1-e^{-(d+1/d-2)}$ is strictly monotone increasing in $d$ on the interval $d \geq 1$ and it is 0 if $d = c$, therefore the above inequality cannot be true for any $d < c$. This means that we obtained a contradiction, which ends the proof of the theorem.

# Acknowledgements

I would like to express my gratitude toward my thesis supervisor János Csirik. He was already my supervisor during my work on the PNS problem, later he taught me the principles of the competitive analysis, and thanking to him my interest was turned to the area of online algorithms. He introduced me to the specialists working in Graz, and that made it possible to visit Graz and work with them. Moreover, he helped me a lot during the time I wrote the thesis, his useful suggestions improved the presentation of the results.

Thanks to the financial support of the START program Y43-MAT of the Austrian Ministry of Science the author could visit Gerhard Woeginger at the Technical University of Graz. He also taught me a lot about the methods and ideas which can be used in the competitive analysis, and he showed me many interesting open problems. I am very grateful to him for that. I am also grateful to the Institute for Mathematics B. of TU Graz for the kind hospitality.

# Summary of the doctoral thesis

For many optimization problems their structures make it possible to develop fast algorithms for solving them by using combinatorial ideas. On the other hand, often the original problem is too difficult to find efficient algorithms. Then some combinatorial ideas can be used to establish algorithms for solving some particular cases of these problems. Combinatorial ideas can also be used to develop algorithms which do not solve the problem, but give an approximate solution which is close to the optimal one in some sense. In the first part of this work we study a hard optimization problem, called PNS problem and develop some combinatorial algorithms for its solution.

In a manufacturing system, materials of different properties are consumed through various mechanical, physical and chemical transformations to yield desired products. Devices in which these transformations are carried out are called operating units, e.g., a lathe or a chemical reactor. Thus, a manufacturing system can be considered as a network of operating units which is called process network. A process design problem in general means to construct a manufacturing system. A design problem is defined from a structural point of view by the raw materials, the desired products, and the available operating units, which determine the structure of the problem as a process graph containing the corresponding interconnections among the operating units. Thus, the appropriate process networks can be described by some subgraphs of the process graph belonging to the design problem under consideration. Our goal is to find an appropriate process network with minimal cost. This minimization yields a combinatorial optimization problem. It is known that this optimization problem is NP-hard.

In general there are three basic approaches to attack NP-hard problems. If a problem is NP-hard, then we do not expect to find a polynomial algorithm which solves the problem, thus the first approach is to develop exponential time algorithms for solving the problem. For the solution of the PNS problem, more exponential algorithms were developed, most of them are based on Branch and Bound technique.

Another approach is to investigate specially structured instances which can be solved efficiently. These classes are called well-solvable classes. In the case of PNS problem well-solvable classes have not been developed earlier. In this thesis we present the first well-solvable PNS classes. First we consider

the turning back problems and establish a linear time algorithm for their solution. Then, we introduce the hierarchical PNS problems. For some restricted cases we also present a linear time algorithm. Finally, by proving a reduction theorem, we show that a further class, called the class of ordered PNS problems, is also well-solvable.

The third approach to attack NP-hard problems is to establish fast, (polynomial time) algorithms which do not guarantee an optimal solution in general, but always result in a feasible solution which is close to the optimal solution in some sense. Such algorithms, called heuristic algorithms or heuristics, are important for several reasons. The feasible solutions found by heuristics can be used in exponential time procedures. Moreover, in practical problems often there is no time enough to find an optimal solution by an exponential algorithm, or the size of the problem is too large to use an exponential algorithm. Then heuristic algorithms can be useful again. For the PNS problem, heuristic algorithms have not been studied earlier. The first heuristic algorithms are presented in this thesis. We analyse these algorithms by worst-case analysis, and establish the tight worst-case bounds for some particular PNS classes. Furthermore, we prove that there is no heuristic algorithm with constant worst-case bound for the PNS problem while $P \neq NP$.

Another field, where algorithms based on combinatorial ideas are very useful, is the area of online computation. The theory of online algorithms and competitive analysis is a new, rapidly developing area. In the second part of the thesis we investigate three different online problems which are closely related to online machine scheduling. We develop some online algorithms for the solution of these problems. These algorithms are analysed by competitive analysis. Some general lower bounds are also presented.

The first problem considered is a particular scheduling problem. Usually in scheduling problems, the number of the available machines is a fixed parameter of the problem. We study the problem of scheduling with machine cost. Here the number of the machines is not given, we also have to purchase the machines and the total cost which we want to minimize is the sum of the cost of purchasing the machines and the cost of the produced schedule. We consider two models for this online problem. In the first one the jobs arrive one by one, and after the arrival of a job the decision maker must purchase the new machines (if he wants to purchase some) and schedule the

job without any information about the further jobs. This model is called List Model. In the second model the jobs arrive over time, each job has a release time, and the jobs cannot be scheduled before their release time. This model is called Time Model. We establish two online algorithms for solving these scheduling problems. The algorithms are investigated by competitive analysis. The first algorithm has a competitive ratio of $(1+\sqrt{5})/2$ for the List Model, the second algorithm is 1.693-competitive for the Time Model. We also prove some general lower bounds. It is proven that there exists no online algorithm which has a smaller competitive ratio than 4/3 for the List Model. Moreover, we prove that there exists no online algorithm which has a smaller competitive ratio than 1.186 for the Time Model.

The second problem considered is a scheduling problem where a two-layer multiprocessor architecture is given. In this problem there are two sets of machines $\mathcal{P}$ and $\mathcal{S}$, containing $k$ and $m$ machines with $k \leq m$. The jobs arriving one by one, and each job has two processing times, one for the machines in set $\mathcal{P}$ and one for the machines in set $\mathcal{S}$. The decision maker has to make an online assignment of jobs to one of the machine sets. The jobs are scheduled in an optimal offline preemptive way within a set. We studied two models. In the first one the goal is to minimize the sum of the makespans of the machine sets, this model is called SLS(k,m). In the second model we want to minimize the maximum of these makespans, this model is called MLS(k,m). We consider two algorithms for these problems. First we investigate a greedy type algorithm and establish the competitive ratio of this algorithm for both problems. The competitive ratios are linear in $m/k$, therefore, the greedy algorithm is effective for these problems only in the cases when $m/k$ is small. We also consider a more difficult algorithm, its competitive ratio is also determined for both problems. This algorithm has constant competitive ratio for both problems, even in the general cases without fixing the numbers $k$ and $m$. We also present some general lower bounds. It is proven that there exists no online algorithm which has smaller competitive ratio than 2 for the SLS(k,m) problem, and there exists no online algorithms which has smaller competitive ratio than $(1+\sqrt{5})/2$ for the MLS(k,m) problem.

The third problem which we consider is a strip packing problem with modifiable items. We investigate the online strip packing problem, where the sizes of the items are not fixed, we can lengthen them. This mathematical problem is interesting since it models the problem of scheduling with

shared resources. We establish two online algorithms called $NFS_r$ and $DS$ for the solution of this problem. We determine the competitive ratios of both algorithms. It is proven that by choosing an optimal $r$, the competitive ratio of $NFS_r$ is 6, moreover, the competitive ratio of $DS$ is 4. Furthermore, we present a general lower bound. It is shown that there exists no online algorithm having smaller competitive ratio than 1.73.

# A disszertáció összefoglalása

Optimalizálási problémák esetén a feladatok struktúrája gyakran lehetővé teszi, hogy kombinatorikus tulajdonságaik alapján gyors megoldó eljárásokat fejlesszünk ki. Másrészt sokszor a probléma túlságosan nehéz ahhoz, hogy hatékony megoldó eljárást találjunk. Ezekben az esetekben kombinatorikus meggondolások hozzájárulhatnak olyan eljárások kifejlesztéséhez, amelyek hatékonyan megoldják a probléma néhány speciális esetét. Szintén ilyen meggondolások alapján gyakran kifejleszthetők olyan gyors eljárások, amelyek nem minden esetben adnak optimális megoldást, de olyan közelítő megoldást eredményeznek, amely az optimális célfüggvényértékhez közeli értéket ad. A disszertáció első részében egy nehéz optimalizálási feladatot mutatunk be, a hálózati folyamatok szintézisének problémáját, a probléma megoldására kombinatorikus tulajdonságokra épülő eljárásokat fejlesztünk ki és analizáljuk ezeket.

Különböző ipari, főképp vegyipari alkalmazásokban sokszor fordul elő, hogy nyersanyagok adott halmazából bizonyos végtermékeket kell előállítani. A gyártás során végrehajtható elemi lépéseket olyan adott költségű gépeknek tekinthetjük, amelyek mindegyike anyagok egy halmazát - ezek az illető gép input anyagai - anyagok egy másik halmazába - ezek az illető gép output anyagai - alakítja át. Ezen interpretáció mellett a probléma az, hogy a rendelkezésre álló gépekből egy olyan kollekciót kell kiválasztani, amely az adott nyersanyagokból legyártja a kívánt végtermékeket. Ezt a feladatot kiegészítve azzal a gyakorlati szempontból fontos céllal, hogy a választott kollekció együttes költsége minimális legyen, egy optimalizálási feladathoz jutunk. A vázolt problémát *Hálózati folyamatok szintézise* (röviden PNS) problémának nevezzük. A probléma az NP-nehéz feladatok csoportjába tartozik.

Általában az NP-nehéz feladatokat három alapvető módszerrel szokás vizsgálni.

Mivel NP-nehéz problémákra nem várható, hogy gyors polinomiális idejű megoldó algoritmust találjunk, ezért egy lehetséges megközelítés exponenciális időigényű eljárások kidolgozása a feladat megoldására. A PNS problémára több ilyen eljárás is kifejlesztésre került, ezek többsége a Korlátozás és szétválasztás elvén működik.

Egy másik megközelítés az, hogy igyekszünk olyan speciális részosztályait meghatározni a problémaosztálynak, amely részosztályba eső feladatok már megoldhatóak polinomiális időben. Ezeket a feladatosztályokat jól megoldható osztályoknak nevezzük. A PNS problémára korábban nem dolgoztak ki jól megoldható osztályokat. A disszertációban mutatjuk be az első jól megoldható PNS osztályokat. Elsőként a visszafejthető problémák osztályát vizsgáljuk, és egy lineáris idejű megoldó algoritmust adunk meg ezen problémák megoldására. Ezt követően a szintezett problémákkal foglalkozunk, és ezen problémák egyes részosztályaira is egy lineáris idejű megoldó eljárást prezentálunk. Végül egy redukciós tétel alapján meghatározunk egy további jól megoldható osztályt, a rendezett PNS problémák osztályát.

Az NP-nehéz problémák harmadik megközelítése olyan gyors polinomiális idejű algoritmusok kifejlesztése, amelyek nem garantálják az optimális megoldást, de egy olyan lehetséges megoldást szolgáltatnak, amelyre a célfüggvény értéke nem sokkal nagyobb, mint az optimális célfüggvényérték. Ezek az eljárások, amelyeket heurisztikus algoritmusnak vagy heurisztikáknak nevezünk több szempontból is rendkívül fontosak. A heurisztikus algoritmusok által előállított megoldások jól használhatók exponenciális idejű eljárásokban. Továbbá gyakorlati alkalmazásokban sokszor nincs elegendő idő az optimális megoldást meghatározni, ebben az esetben is jól használhatók a heurisztikus algoritmusok. A PNS problémára korábban heurisztikus algoritmusok nem készültek. Az első heurisztikus algoritmusokat a disszertációban mutatjuk be. Ezeket az eljárásokat a legrosszabb-eset analízissel vizsgáljuk meg. Nevezetesen meghatározzuk az algoritmusok éles legrosszabb-eset korlátját bizonyos speciális PNS osztályokra, továbbá igazoljuk, hogy amennyiben $P \neq NP$, akkor nincs konstans legrosszabb-eset korláttal rendelkező polinomiális heurisztikus eljárás a PNS problémára.

Egy másik jelentős terület, ahol kombinatorikus algoritmusok igen jól használhatók az online algoritmusok témaköre. A disszertáció második részében három olyan online problémát vizsgálunk, amelyek szorosan kapcsolódnak az online ütemezés témaköréhez. Ezen problémák megoldására különböző eljárásokat dolgozunk ki és vizsgálunk kompetitív analízissel, továbbá igazolunk néhány általános alsó korlátot.

Az első probléma egy olyan ütemezési feladat, amelyben a gépek száma nem adott paraméter, hanem a gépeket is meg kell vásárolnunk. Ebben a problémában a minimalizálandó célfüggvény a gépekre összesen költött

összeg plusz a kapott ütemezésben a maximális befejezési idő. Ezt a problémát két különböző modellben vizsgáljuk. Az első modellben a munkák egy listáról érkeznek, és a munka érkezése után kell új gépeket venni és ütemezni a munkát, a további munkákra vonatkozó információk nélkül. Ezt a modellt Lista modellnek nevezzük. A második modellben, amit Idő modellnek nevezünk, a munkáknak egy érkezési ideje van, és a munkákat csak az érkezési idejük után lehet elkezdeni végrehajtani. Egy - egy online algoritmust fejlesztünk ki az egyes modellekre. Az algoritmusokat a kompetitív analízissel elemezzük. Az első algoritmus kompetitív hányadosa $(1 + \sqrt{5})/2$ a Lista modellre, a másik algoritmus 1.693-kompetitív az Idő modellre. Szintén igazolunk néhány általános alsó korlátot. Nevezetesen megmutatjuk, hogy a Lista modellben nincs olyan online algoritmus, amelynek kisebb a kompetitív hányadosa, mint 4/3, továbbá az Idő modellre nincs olyan online algoritmus, amelynek kisebb a kompetitív hányadosa, mint 1.186.

A második vizsgált probléma egy olyan ütemezési feladat, ahol a gépek egy két csoportból álló géprendszert alkotnak. A problémában adott két géphalmaz $\mathcal{P}$ és $\mathcal{S}$. A halmazok $k$ és $m$ gépet tartalmaznak, a továbbiakban feltesszük, hogy $k \leq m$. A munkák egy listán érkeznek. Minden munkához két ütemezési idő, $p_j$ és $s_j$ tartozik, ezek azt adják meg, hogy mennyi ideig tart végrehajtani a munkát a $\mathcal{P}$ illetve $\mathcal{S}$ halmazokba eső gépeken. Az egyes munkák érkezése után online módon, a további munkákra vonatkozó ismeretek nélkül meg kell határoznunk, hogy melyik géphalmazon hajtjuk végre a munkát. Végül, amikor a munkasorozat véget ér, ütemezzük az egyes géphalmazokhoz rendelt munkákat a megfelelő géphalmazokon offline módon, minimalizálva a maximális befejezési időt, megengedve a munkák megszakítását. Az első vizsgált modellben, amit SLS(k,m)-el jelölünk, a konstruált ütemezés költsége a két maximális befejezési idő összege, és a célunk ezen összeg minimalizálása. A második vizsgált modellben a minimalizálandó érték a két befejezési idő maximuma, ezt a problémát MLS(k,m) jelöli. Két eljárást vizsgálunk meg ezekre a problémákra. Az első egy mohó algoritmus. Erre az eljárásra meghatározzuk a kompetitív hányadost mindkét modellre. Mindkét kompetitív hányados lineáris $m/k$-ban, tehát a mohó eljárás csak azon esetekben hatékony, amelyekben ez a hányados kicsi. Szintén bevezetünk és megvizsgálunk egy bonyolultabb eljárást. Ezen eljárásra is meghatározzuk a kompetitív hányadost mindkét modellre. A kompetitív hányados konstans mindkét modellre, mindkét általános problémára is, ahol

nem rögzítjük a $k$ és $m$ számokat. Szintén bizonyítunk alsó korlátokat. Megmutatjuk, hogy nincs olyan online algoritmus, amelynek kisebb a kompetitív hányodosa, mint 2 az SLS(k,m) problémára, és nincs olyan online algoritmus, amelynek kisebb a kompetitív hányadosa, mint $(1 + \sqrt{5})/2$ az MLS(k,m) problémára.

A harmadik megvizsgált probléma egy olyan sávpakolási probléma, amelyben nyújthatóak a sávba elhelyezendő téglalapok. Ez a matematikai kérdés azért érdekes, mert modellezi a megosztott erőforrások melletti ütemezés problémáját. Két online algoritmust fejlesztünk ki ezen probléma megoldására. Ezeket az eljárásokat $NFS_r$ és $DS$ jelöli. Meghatározzuk az eljárások kompetitív hányadosát. Igazoljuk, hogy az $r$ érték optimális megválasztása mellett az $NFS_r$ algoritmus kompetitív hányadosa 6, és megmutatjuk, hogy a $DS$ algoritmus kompetitív hányadosa 4. Továbbá igazolunk egy általános alsó korlátot, nevezetesen belátjuk, hogy nincs olyan online algoritmus, amelynek a kompetitív hányadosa kisebb, mint 1.73.

# Bibliography


[1] S. Albers, Better Bounds for Online Scheduling, *Proc. 29th Symp. Theory of Computing*, 1997, 130-139.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, O. Waarts, On-line load balancing with applications to machine scheduling and virtual circuit routing, *J. ACM*, **44(3)**, 1997, 486-504.

[3] B. S. Baker, D. J. Brown, H. P. Katseff, Lower bounds for two dimensional packing algorithms, *Acta Informatica*, **8**, 1982, 207-225.

[4] B. S. Baker, J. S. Schwartz, Shelf algorithms for two dimensional packing problems, *SIAM J. Computing*, **12**, 1983, 508-525.

[5] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, L. Stougie, Multiprocessor scheduling with rejection, *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, 1996, 95–103.

[6] M. Bellore, S. Goldwasser, C. Lund, A. Russel, Efficient probabilistically checkable proofs and applications to approximation, *Proc. of 25-th STOC*, 1993, 294-304.

[7] Z. Blázsik, Cs. Holló, B. Imreh, Cs. Imreh, Z. Kovács, On a well-solvable class of the PNS problem, *Novi Sad Journal of Mathematics*, **30**, 2000, 21-30.

[8] Z. Blázsik, Cs. Holló, Cs. Imreh, Z. Kovács, Heuristics for the PNS Problem, *Mátraháza Optimization Days*, ed: F. Gianessi, P. Pardalos, T. Rapcsák, Kluwer Academic Publisher, to appear.

[9] Z. Blázsik, B. Imreh, A note on connection between PNS and set covering problems, *Acta Cybernetica*, **12**, 1996, 309-312.

[10] A. Borodin, R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.

[11] R. E. Burkard, Efficiently solvable cases of hard combinatorial problems, Mathematical Programming Series *B 79*, 1997, 55-71.

[12] B. Chen, A van Vliet, G. Woeginger, New Lower and Upper Bounds for On-line Scheduling, *Operations Research Letters*, **16**, 1994, 221-230.

[13] C. Chekuri, R. Motwani, B. Natarajan, C. Stein, Approximation techniques for average completion time scheduling, *Proc. 8th Symp. on Discrete Algorithms*, 1997, 609-618.

[14] V. Chvatal, A Greedy Heuristic for the Set-Covering Problem, *Math. Oper. Res.*, **4**, 1979, 233-235.

[15] J. Csirik, G. Woeginger, Shelf algorithms for on-line strip packing, *Information Processing Letters*, **63**, 1997, 171-175.

[16] A. Feldman, J. Sgall, S. H. Teng, Dynamic scheduling on parallel machines, *Theoretical Comput. Sci.*, **130(1)**, 1994, 49-72. (*Also in Proc. FOCS'91*, 111-120.)

[17] A. Fiat, G. J. Woeginger (eds), *Online algorithms: The State of the Art*, Vol. **1442** of Lecture Notes in Computer Science, Springer-Verlag Berlin, Heidelberg,

[18] F. Friedler, L. T. Fan, B. Imreh, Process Network Synthesis: Problem Definition, *Networks*, **28**, 1998, 119-124.

[19] F. Friedler, K. Tarján, Y. W. Huang, L. T. Fan, Graph-Theoretic Approach to Process Synthesis: Axioms and Theorems, *Chem. Eng. Sci.*, **47**(8), 1992, 1973-1988.

[20] F. Friedler, K. Tarján, Y. W. Huang, L. T. Fan, Combinatorial Algorithms for Process Synthesis, *Computer Chem. Engng.*, **16**, 1992, 313-320.

[21] Friedler, F., Tarján, K., Huang, Y. W., and Fan, L. T.: Graph-Theoretic Approach to Process Synthesis: Polynomial Algorithm for Maximal Structure Generation, *Computers Chem. Engng.* **17**(9), 1993, 929-942.

[22] G. Galambos, G. Woeginger, An On-Line Scheduling Heuristic with Better Worst Case Ratio than Graham's List Scheduling, *SIAM Journal on Computing*, **22**, 1993, 349-355.

[23] M. R. Garey, D. S. Johnson *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman & Co, San Francisco, 1979.

[24] L. R. Graham, Bounds for certain multiprocessing anomalies, *Bell System Technical Journal*, **45**, 1966, 1563-1581.

[25] B. Imreh, F. Friedler, L. T. Fan, An Algorithm for Improving the Bounding Procedure in Solving Process Network Synthesis by a Branch-and-Bound Method, *Developments in Global Optimization*, editors: I. M. Bonze, T. Csendes, R. Horst, P. M. Pardalos, Kluwer Academic Publisher, Dordrecht, Boston, London, 1996, 301-348.

[26] B. Imreh, G. Magyar, Empirical Analysis of Some Procedures for Solving Process Network Synthesis Problem, *Journal of Computing and Information Technology*, **6**, 1998, 373-382.

[27] Cs. Imreh, An online scheduling algorithm for a two-layer multiprocessor architecture, submitted to *Acta Cybernetica.*

[28] Cs. Imreh, Online classification with offline scheduling, submitted to *Algorithmica.*

[29] Cs. Imreh, Some well-solvable PNS classes (In Hungarian), *New lines in the Hungarian Operations Research*, ed. by Komlósi, S. and Szántai T., Dialóg Campus Kiadó, Budapest-Pécs, 1999, 168-181.

[30] Cs. Imreh, A new well-solvable class of PNS problems, *Computing*, to appear.

[31] Cs. Imreh, Online strip packing with modifiable boxes, submitted to *Operations Research Letters.*

[32] Cs. Imreh, J. Noga, Scheduling with Machine Cost, In *Randomization Approximation and Combinatorial Optimization Algorithms and Techniques*, ed.: D. Hochbaum and K. Jansen, 1999, 168-176.

[33] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, (1976).

[34] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys, Sequencing and scheduling: algorithms and complexity, in *Handbooks in Operations Research and Management Science, Volume 4, Logistics of Production and Inventory* ed.: S. Graves, A. H. G. Rinnooy Kan, P. Zipkin, North Holland, Amsterdam, 1993, 445-522.

[35] J. Noga, S. Seiden, An Optimal Online Algorithm for Scheduling Two Machines with Release Times, *Theoretical Comput. Sci.*, to appear.

[36] C. H. Papadimitriou, K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, 1982.

[37] S. A. Roosta, *Parallel Processing and Parallel Algorithms*, Springer-Verlag, New York,1999.

[38] S. Seiden, Preemptive Multiprocessor Scheduling with Rejection, *Theoretical Comput. Sci.*, to appear.

[39] D. B. Shmoys, J. Wein, D. P. Williamson, Scheduling parallel machines online. *SIAM J. Computing*, **24**, 1995, 1313-1331.

[40] W. S. Tanaev, Y. N. Sotskov, V. A. Strusevich, *Scheduling Theory: Multi-Stage Systems*, Kluwer Academic Publishers, Dordrecht, 1994.

# Index