B3727

EXTENDED ABSTRACT OF DOCTORAL THESIS

Combinatorial algorithms for the PNS and online scheduling problems

Csanád Imreh

SZEGED
2001

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 develop algorithms which can solve 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 a hard optimization problem, called PNS problem. For solving this problem, we develop and analyse some algorithms based on combinatorial ideas.

Another field where such algorithms are very useful is the online computation. 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 also present some general lower bounds.

2 The PNS problem

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, 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. Naturally, the cost minimization of a process network is indeed essential.

The importance of process network synthesis (for short PNS) arises from the fact that such networks are ubiquitous in the chemical and allied industries. The foundations of PNS and the background of the combinatorial

model studied here can be found in [11], [12].

In the combinatorial approach, the structure of a process can be described by the process graph (see [12]) defined as follows.

Let M be a finite nonempty set, the set of the materials. Furthermore, let $\emptyset \neq O \subseteq \mathcal{P}'(M) \times \mathcal{P}'(M)$ with $M \cap O = \emptyset$, where $\mathcal{P}'(M)$ denotes the set of all nonempty subsets of M. The elements of O are called *operating units*, and for an operating unit, $u = (\alpha, \beta) \in O$, α and β 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 sets. 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

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

Let the process graphs (m, o) and (M, O) be given. (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 a *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 $O \subseteq \mathcal{P}'(M^*) \times \mathcal{P}'(M^*)$ is the set of the available operating units. It is assumed that $P \cap R = \emptyset$ and $M^* \cap O = \emptyset$, moreover, α and β 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 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 [12] 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 $\mathrm{M} = (P, R, O)$ will be denoted by $S(P, R, O)$ or $S(\mathrm{M})$.

PNS problem with weights

Let us consider 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:

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

$$(1) \qquad \min\{\sum_{u \in o} w(u) : (m, o) \in S(P, R, O)\}.$$

In this work by PNS problem we always mean PNS problem with weight, and the solution-structures are also called *feasible solutions*. It is known (see [7]) that this PNS 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 [14] and [15].

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. The first well-solvable classes are presented in [5], [18], and [19].

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 such algorithms can be used in exponential time algorithms, furthermore, there is often not enough time to find an optimal solution or the size of the problem is too large to use an exponential algorithm. In these cases, heuristic algorithms can be useful again. For the PNS problem, heuristic algorithms have not been studied earlier. The first heuristic algorithm for this problem is presented in [6].

2.1 Some well-solvable PNS classes

In this part we present our results on the well-solvable PNS classes. The material of this part is based on [5], [18], and [19]. To present these results, we need the following definitions.

A PNS problem is called *turning back* if for every material, there exists at most one operating unit producing this material. A PNS problem is called *l-hierarchical* if there exist partition $M_0 = R, M_1, \ldots, M_l = P$ of M and partition $O_1, \ldots, O_l$ of O such that O_i consists of operating units having input materials from M_{i-1} and output materials from M_i, for $i = 1, \ldots, l$. A PNS problem is called *k-wide l-hierarchical* if it is an l-hierarchical problem, and $|M_i| \leq k$ is valid for $i = 0, \ldots, l$; furthermore, $|O_i| \leq k$ holds for $i = 1, \ldots, l$. A PNS problem is called (c, l)-*ordered* if there exist partitions $M_0 = R, \ldots, M_l = P$ and $O_1, \ldots, O_l$, of M and O, respectively, such that O_i contains only operating units having input materials from $\cup\{M_j : i - c \leq j < i\}$ and output materials from $\cup\{M_j : i \leq j < i + c\}$, for $i = 1, \ldots, l$. A (c, l)-ordered problem is called *k-wide* if $|M_i| \leq k$ and $|O_j| \leq k$ hold for $i = 0, \ldots, l$, $j = 1, \ldots, l$. A PNS problem is called *integer* if every operating unit has a positive integer weight.

We developed the well-solvable classes included in the following theorems.

Thesis 1

Theorem 2.1.1 ([18]) *If a PNS problem is turning back, then there exists a linear time algorithm which decides whether the problem has a feasible solution, and if it does, then the procedure provides an optimal solution.*

Theorem 2.1.2 ([5]) *If a PNS problem is k-wide l-hierarchical, then there exists an algorithm with time complexity $\mathcal{O}(l)$ which gives an optimal feasible solution of the problem, or it gives that the problem has no feasible solution. The constant in $\mathcal{O}$ may depend on k exponentially.*

Theorem 2.1.3 ([19]) *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 exponentially on $k_1 + k_1^2(c-1)c$.*

2.2 Heuristic algorithms for the PNS problem

In this part we present our results on the heuristic algorithms for the PNS problem. This part is based on [6]. We introduce two heuristic algorithms here. The basic ideas of both algorithms are the same. They can be considered as the generalizations of the well-known Chvatal's algorithm (see [8]) for the set covering problem. They use a cost function c defined in [14]. 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 an 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 Asum_c, this cost is estimated by $\sum_{X \in mat^{in}(v)} c(X)$. In the second algorithm, called Amax_c, this cost is estimated by $\max\{c(X) : X \in mat^{in}(v)\}$.

The algorithms are studied by worst-case analysis. To present our results, we recall the following definitions. For a heuristic algorithm A, C is called a *worst-case bound* of the algorithm on a class $\mathcal{P}$ of instances if for every instance from $\mathcal{P}$, the algorithm gives a feasible solution with at most C times higher cost than the optimal cost. C is called *tight* if it is the smallest possible worst-case bound. We investigated the worst-case bounds for some particular PNS classes. To present the results, we have to define them. A PNS problem is called a *PNS1 problem* if every material is a raw material or a desired product; moreover, every operating unit produces desired products from raw materials. For each fixed positive integer k, a PNS problem belongs to the *class* S_k if every operating unit is of 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 exactly k.

For the worst-case analysis of the algorithms, the following theorems are valid.

Thesis 2

Theorem 2.2.1 ([6]) *There is no polynomial time heuristic algorithm with constant worst-case bound for the class of PNS problems unless P=NP.*

Theorem 2.2.2 ([6]) *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.*

Theorem 2.2.3 ([6]) *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.*

3 Online scheduling

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 for 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 on 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. We will use the competitive analysis, therefore, we give the definition of the competitive ratio below.

An online minimization algorithm is *C-competitive* if the object value of the solution produced by the algorithm is never greater than C times the offline optimal value. The *competitive ratio of an algorithm* is the least C such that the algorithm is C-competitive. The competitive ratio of a problem is the best competitive ratio any online algorithm can achieve.

The problems we investigate are closely related to parallel machine scheduling, therefore, we present its basic model here. In the simplest model, we have a sequence of jobs, each of which has a *processing time*, and we have to process them on the available uniform machines. A *schedule* specifies for each job a machine and a time interval on the machine when the job is processed on the machine. 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 for each machine the time intervals do not overlap. Our goal is to minimize the maximal finishing time. Sometimes, it is allowed to preempt the jobs. In this case, we have to specify for each job a sequence of machines with not overlapping time intervals (one machine can have more time intervals), where the total length of the time intervals must be the processing time.

Probably the most fundamental example of an online machine scheduling problem is where the jobs arrive 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. The starting

time of the job is the completion time of the previous job on the machines. Our goal is to minimize the maximal completion time. By the *load* of a machine we mean the sum of the processing times of all jobs assigned to the machine. It is easy to see that in this case the maximum load is the maximal completion time, this value is often called the *makespan*. The first result is due to Graham [13]. Although the terminology of competitive analysis was not used by him, it was shown that a simple greedy algorithm, the List Scheduling, is $(2 - 1/m)$-competitive.

Another online machine scheduling problem is where the jobs arrive over time. Here again there is 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 a 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 these models, we refer to the survey [23].

We investigate here three different problems which are closely related to online scheduling problems. The first problem, studied in [21], is such a variant where we have to purchase the machines. In the second case we consider some generalized scheduling problems, where the machines have a two-layer multiprocessor architecture. These problems are considered in [16] and [17]. Finally, the third problem, investigated in [20], is a modified strip packing problem, where we use some idea from the area of machine scheduling.

3.1 Online scheduling with machine cost

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 some 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, and the second is a variant of scheduling jobs arriving over time. The differences in both cases 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 the first problem as the *List Model*, and the second problem as the *Time Model*.

We studied a class of online algorithms for these problems. For an increasing sequence $\varrho = (0 = \varrho_1, \varrho_2, \ldots, \varrho_i \ldots)$, we will define an online algorithm A_ϱ. When job j_ℓ is revealed A_ϱ purchases machines (if necessary) so that the current number of machines i satisfies $\varrho_i \leq P_\ell < \varrho_{i+1}$, where P_ℓ is the sum of the processing times of the first ℓ jobs. Algorithm A_ϱ then assigns the job j_ℓ by the following greedy algorithm: In the List Model it assigns the job to the least loaded machine; in the Time Model, 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_ϱ assigns the job with the largest processing time to an idle machine.

Analysing particular algorithms from these classes, and looking for general lower bounds, we obtain the following results.

Thesis 3

Theorem 3.1.1 ([21]) *The competitive ratio of A_ϱ is $(1+\sqrt{5})/2$ for the List Model for $\varrho = (0, 4, 9, 16, \ldots, i^2, \ldots)$. In the Time Model this algorithm is $(6+\sqrt{205})/12 \approx 1.693$-competitive.*

Theorem 3.1.2 ([21]) *No online algorithm can have a competitive ratio smaller than* 4/3 *in the List Model and no online algorithm can have a competitive ratio smaller than* $(\sqrt{33}-1)/4 \approx 1.186$ *in the Time Model.*

3.2 Online scheduling with two-layer multiprocessor structure

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 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 the jobs assigned to $\mathcal{P}$ (respectively, the jobs assigned to $\mathcal{S}$) on the machines of $\mathcal{P}$ (respectively, $\mathcal{S}$) so as to minimize the preemptive makespan. Let $C_{\mathcal{P}}$ (respectively, $C_{\mathcal{S}}$) denote this optimal makespan. In the first problem, which is investigated in [17], and called problem SLS(k,m) (*Sum Layered Scheduling*), the cost of the constructed schedule is the sum $(C_{\mathcal{P}} + C_{\mathcal{S}})$ of the two makespans. In the second problem (cf. [16]), which is called MLS(k,m) (*Maximum Layered Scheduling*), the cost of the constructed schedule is the maximum $(\max\{C_{\mathcal{P}}, C_{\mathcal{S}}\})$ 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 with rejection, jobs arrive one by one, and we have to schedule each job online 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 [4], the preemptive version and some randomized algorithms for it were studied in [22]. In [4], a 2.61-competitive algorithm is presented and it is also proven that this algorithm is optimal. In [22], the problem, where preemption is allowed is investigated and a randomized algorithm is developed. The semi online version is when we have to decide whether we reject or schedule the job in an online fashion, but we do an offline scheduling at the end. One can see immediately *that SLS contains this problem 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 machines scheduling problem with unrelated machines which is investigated in [1], where it is proven that the greedy algorithm is optimal and it is 2-competitive for two machines.

We studied the following online algorithms for these problems. The first algorithm is a simple greedy type algorithm, the second is a more difficult one, which is the generalized version of the reject total penalty algorithm from [4] and [22]. In fact, this algorithm is a class of algorithms since it depends on two parameters $0 < \alpha \leq 1$ and $0 < \gamma \leq 1$.

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

Algorithm $A(\alpha, \gamma)$

- 1. *Initialization.* Let $R := \emptyset$.
- 2. When job j arrives
  - (i) If $p_j/k \leq s_j \cdot \gamma/m$, then assign j to $\mathcal{P}$.
  - (ii) Let r be the cost of the optimal offline preemptive scheduling of the set $R \cup \{j\}$ on $\mathcal{P}$. 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 $\mathcal{S}$.

Analysing these algorithms and looking for general lower bounds we obtained the following results.

Thesis 4

Theorem 3.2.1 ([16], [17]) *The competitive ratio of LG is*

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

on $SLS(k, m)$ *and* $\max\{2, m/k\}$ *on* $MLS(k, m)$.

Theorem 3.2.2 ([16], [17]) *The algorithm $A(\alpha, \gamma)$ has competitive ratio C on problems SLS and MLS, where*

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

Theorem 3.2.3 ([16], [17]) *If an online algorithm is c-competitive for the $MLS(k, m)$ problem, then $c \geq (1 + \sqrt{5})/2 \approx 1.618$. If an online algorithm is c-competitive for the $SLS(k, m)$ problem with $k \geq 2, m \geq 2$, then $c \geq 2$.*

3.3 Online strip packing with modifiable boxes

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 1 by minimizing the total height of the strip. This problem appears in many situations. Usually, scheduling of tasks with shared resources 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 [3]. The best presented algorithms are 7.46 and 6.99 competitive. A lower bound of 2 for the competitive ratio of any online strip packing algorithm is presented in [2]. An improved shelf algorithm and a generalized definition of shelf algorithms are introduced in [9].

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 [10]. The main difference between the two models that in [10] the resource is measured by the number of the used processors and the geometrical structure of the processors network is also considered.

In [20] we examine 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. One basic way of packing into the strip is to define shelves and to 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 have the same heights 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 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 into 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 enough room 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 [3]. We also denote the extended algorithm by NFS_r. This algorithm depends on a parameter $r > 1$, and we can describe it by the following rule.

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 second algorithm uses similar idea like the scheduling algorithm from [24]. It can be defined 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.

Analysing these algorithms and looking for general lower bounds we obtained the following results.

Thesis 5

Theorem 3.3.1 ([20]) *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.*

Theorem 3.3.2 ([20]) *The competitive ratio of algorithm DS is* 4.

Theorem 3.3.3 ([20]) *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* $e^{-(c+1/c-2)} = c - 1$.

References

[1] 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.

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

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

[4] 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.

[5] 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.

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

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

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

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

[10] 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.)

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

[12] 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.

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

[14] 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.

[15] 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.

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

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

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

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

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

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

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

[23] J. Sgall, On-line scheduling, in *Online algorithms: The State of the Art* editors:A. Fiat, and G. J. Woeginger , Vol. 1442 of Lecture Notes in Computer Science, Springer-Verlag Berlin, Heidelberg, pp. 196–231.

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