Sevast'yanov algorithm for the flow-shop scheduling problem

Sevast'yanov's Algorithm for the Flow-Shop Scheduling Problem

Helena Ramalhinho Lourenço*
School of Operations Research and Industrial Enginnering
Cornell University, Ithaca NY 14853, USA

Abstract: We consider the flow-shop scheduling problem. The objective is to schedule the jobs on the machines so that we minimize the time by which all jobs are completed. We studied and implemented different versions of the algorithm of Sevast'yanov based on linear programming to solve this problem. Using CPLEX, we did computational tests with random instances having up to 1000 jobs and 100 machines. If the size of the flow-shop scheduling problem is small or if the running time is not a critical factor, the Nawaz, Enscore and Ham approximation algorithm still performs better. But if the running time is an important factor, Sevast'yanov's algorithm can be a very good alternative especially in presence of very large scale instances with a relatively small number of machines.

Keywords: Scheduling Theory; Heuristics; Computational Analysis.

1 The flow-shop scheduling problem

In the flow-shop scheduling problem we are given a set of m machines $\{M_1, \ldots, M_m\}$ and a set of n jobs $\{J_1, \ldots, J_n\}$; each of the n jobs has to be processed on the m machines $M_1, \ldots, M_m$ in that order, i.e. a job J_j, $j = 1, \ldots, n$ consists of a sequence of m operations $O_{1j}, \ldots, O_{mj}$, where O_{ij} must be processed on machine M_i for a given uninterrupted processing time p_{ij}. Each machine $M_i, i = 1, \ldots, m$, can process at most one job at a time, and each job $J_j, j = 1, \ldots, n$, can be processed by at most at one machine at a time. Let C_{ij} be the completion time of operation O_{ij}. The objective is to produce a schedule that minimizes the maximum completion time $C_{max} = \max_{i,j} C_{ij}$. We will also use the following notation: $p_{max} = \max_{i,j} p_{ij}$, $\Pi_i = \sum_{j=1}^{n} p_{ij}$ and $\Pi_{max} = \max_i \Pi_i$.

In what follows, we just consider permutation schedules, i.e. the sequence in which the jobs are to be processed is the same for each machine. While this restriction is quite standard in the literature, it is important to note that an optimal schedule is not necessarily a permutation one.

The study of the flow-shop scheduling problem started with the publication of the paper by Johnson [10] where he proposed a polynomial-time optimization algorithm for the two-machine flow-shop scheduling problem. In spite of the large number of papers in

*Research supported by Junta Nacional de Investigação Científica e Tecnológica, Lisboa, Portugal. New address: Departamento de Estatística e Investigação Operacional, Faculdade de Ciências, Universidade de Lisboa, Portugal.

the literature dedicated to this problem, very few papers describe applications of the flow-shop scheduling problem to real problems. Recently, there has been a rising interest in the problem as an applicable analytical tool to solve problems in the area of the chemical process industry and the design of flexible manufacturing systems, as mentioned in [4].

The flow-shop scheduling problem has been shown to be $NP-$complete by Garey, Johnson and Sethi [5] and Rinnooy Kan [17]. Therefore, for practical purposes, it is often more appropriate to apply an approximation method which generates an approximate solution in a relatively efficiently time.

After more than 30 years of studying the flow-shop scheduling problem, many approximation methods have been proposed; the reader is referred to the papers of Lawler et al. [11] and Dudeck et al. [4]. Some of these approximation methods belong to the class of constructive methods: an algorithm that builds a sequence of jobs and once a decision is made, it is never changed. This class includes the methods presented by Palmer [16], Campbell, Dudeck and Smith [2], Gupta [6], Dannenbring [3] and by Nawaz, Enscore and Ham [14]. On the other hand, some improvement approximation methods have also been proposed, as are the ones presented by Ho and Chang [9] and Osman and Potts [15], where this last one is based on simulated annealing techniques. Widmer and Hertz [21] presented a two-phase heuristic where in the first phase a initial schedule is constructed and in the second phase, this schedule is improved using tabu-search techniques. Taillard [20] presented a comparative study on methods to solve the flow-shop problem including an improved version of the heuristic by Nawaz, Enscore and Ham [14] and a tabu-search heuristic method.

For most of these approximation methods, no worst-case analysis exists. In other words, there is no upper bound that is guaranteed for the maximum completion time of the schedules produced by these methods when applied to any instance of the problem. Many of them perform well in practice and on a set of randomly generated instances. Of the constructive approximation algorithms, the approximation method presented in [14] is generally considered to perform best in practice.

In 1986, Sevast'yanov proposed a polynomial-time approximation algorithm to solve the flow-shop scheduling problem based on geometric results (see [18] and [12]). The crucial property of this method is its worst-case guarantee: it is guaranteed that the maximum completion time of a schedule produced by this method when applied to any instance of the problem with n jobs and m machines is at most $C^*_{max} + m(m-1)p_{max}$. Note that this bound does not depend on the number of jobs. The purpose of this paper is to understand the computational behavior of the approximation algorithm for the flow shop scheduling problem based on the vector-sum theorem, as proposed by Sevast'yanov [18].

This paper is organized as follows: In Section 2 we introduce the main results without proving them and present Sevast'yanov's algorithm. In Section 3 we take a closer look at the algorithm and derive some results based on the details of the implementation. In Section 4 we present the computational experiments performed and the results obtained. Finally, in Section 5 we make some concluding remarks, discuss possible future work and the application to the job shop problem.

2 Sevast'yanov's Algorithm

The approximation algorithm due to Sevast'yanov is based on the following result, which we call the vector-sum theorem:

Theorem 1 *(Sevast'yanov): Let $V = \{v_1, \ldots, v_n\}$ be a set of d-dimensional vectors such that $\sum_{j=1}^{n} v_j = 0$. Then it is possible to compute, in polynomial time, a permutation π of $\{1, \ldots, n\}$ such that:*

$$\| \sum_{j=1}^{k} v_{\pi(j)} \| \quad \leq \quad d * \max_{j=1,\ldots,n} \| v_j \| \qquad k = 1, \ldots, n.$$

Proof. (see [12]).

From the proof of the Theorem 1 we can extract the Sevast'yanov's algorithm, which we will described in detail at the end of the section. Next, we present the result that links the vector-sum theorem and the flow-shop scheduling problem.

Theorem 2 *(Sevast'yanov): For any instance of the flow-shop scheduling problem, a permutation schedule of length at most $C^*_{max} + m(m-1)p_{max}$ can be found in polynomial time.*

We shall present the main ideas of the proof; for more details the reader is referred to [12]. Suppose that $\Pi_{max} = \Pi_i \ , \forall i$. If for some M_i, $\Pi_{max} > \Pi_i$, we can iterate through the operations on M_i, increasing each p_{ij} to at most p_{max} until the revised total reaches Π_{max}; this will not modify the above bounds. Now, for each job J_j $(j = 1, \ldots, n)$ define the following vector, where p_{ij} is the processing time of the i^{th} operation of job J_j:

$$v_j = (p_{1j} - p_{2j}, p_{2j} - p_{3j}, \ldots, p_{m-1,j} - p_{mj}) \tag{1}$$

Note that $\sum_{j=1}^{n} v_j = 0$, since $\Pi_{max} = \Pi_i \ , \forall i$. Apply the vector-sum theorem and find the permutation. It can be proven that this permutation satisfies the bound in the theorem. □

The bottleneck step of Sevast'yanov's algorithm is the computation of an extreme point of the current linear system. Lawler et al. [12] presents two ways to implement this step: a polynomial-time algorithm based on linear programming and a faster implementation based on projection methods. This last version gives the running time of $O(d^2m^2)$, which is better than for the first version, which involves the use of linear programming routines. However, we will just refer to the first version because, in spite of the theoretical results, the experiments that we did with the faster algorithm appear to be numerically unstable, especially for large-scale instances of the flow-shop scheduling problem.

We now present Sevast'yanov's algorithm:

Input: a set $V = \{v_1, \ldots, v_n\}$ of d-dimensional vectors such that $\sum_{j=1}^{n} v_j = 0$. For the flow-shop scheduling problem, construct v_j, for $j = 1, \ldots, n$ as in (1).

Output: A permutation π of $\{1, \ldots, n\}$ such that:

$$\| \sum_{j=1}^{k} v_{\pi(j)} \| \leq d * \max_{j=1,\ldots,n} \| v_j \| \qquad k = 1, \ldots, n.$$

For the flow-shop scheduling problem, the output is the permutation schedule corresponding by π which satisfies $C_{max} \leq C^*_{max} + m(m-1)p_{max}$.

1. **Case A**: $n \leq d$
   (a) for $j := 1$ to n do write (v_j) in any order.
2. **Case B**: $n \geq d$
   (a) vector-sum routine $\Leftarrow$ get $V \equiv V_n \supseteq \ldots \supseteq V_d$ which are subsets of the vector set $V = \{v_1, \ldots, v_n\}$;
   (b) for $j := 1$ to d do write $(v_j \in V_d)$ in any order.
   (c) for $j := d + 1$ to n do write (v_j s.t. v_j is the unique element in $V_j - V_{j-1}$).

The vector-sum routine:

At each iteration k consider a n-dimensional vector λ_k where each component $\lambda_k(v)$ is associated with vector $v \in V$.

1. $\lambda_n(v) \Leftarrow (n - d)/n, \quad \forall v \in V \equiv V_n$
2. Repeat for $k = n - 1$ downto to d the following: /* V_{k+1} and λ_{k+1} are known */:
   (a) If λ_{k+1} has a zero component, say $\lambda_{k+1}(v^*) = 0$, then $V_k \Leftarrow V_{k+1} - \{v^*\}$ and $\lambda_k(v) \Leftarrow \lambda_{k+1}(v), \quad \forall v \in V_k$;
   (b) If λ_{k+1} does not have a zero component, find one as follows:
      i. Find an extreme point of the following system:

$$\sum_{v \in V_{k+1}} x(v)v = 0,$$

$$\sum_{v \in V_{k+1}} x(v) = k - d,$$

$$0 \leq x(v) \leq 1, \quad v \in V_{k+1}\,.$$

      where there is a variable $x(v)$ for each $v \in V_{k+1}$. Let x^* be an extreme point. (It can be proven that the extreme point has at least one component equal to zero, [12]).
      ii. Let v^* be any vector in V_{k+1} with $x^*(v^*) = 0$, and set:

$$V_k \Leftarrow V_{k+1} - \{v^*\}$$

$$\lambda_k(v) \Leftarrow x^*(v) \quad \forall v \in V_k$$

3 Practical Computational Aspects of the Sevast'yanov's Algorithm

In this section, we give special attention to the practical computational aspects of the Sevast'yanov's algorithm.

For the step of the vector-sum routine where we choose a vector v such that $x^*(v^*) = 0$, where $v^* \in V_{k+1}$ and $v^* \notin V_k$, if there is more than one vector which satisfies the above condition, the choice is arbitrary; therefore we would like to choose the one that gives good solutions. This problem can be stated as: find an ordering of the vectors in the following subset: $S_l = \{v_j : x^*(v_j) = 0\}$, after calling the routine to solve the LP programming problem. Note that the algorithm just calls the LP routine again only after all of the vectors in S_l have been selected, since at step 2(a) of the vector-sum routine, if $\lambda_{k+1}(v) = 0$, then $\lambda_k(v) = 0$. Note that if we sort the vectors in S_l in any order, the upper bounds of Theorems 1 and 2 are still valid, but in terms of the flow-shop scheduling problem, the order of the vectors in the permutation schedule can make a important difference in the length of the schedule.

To obtain the extreme point x^* at Step 2.(b)i. of the vector-sum routine, we can use any routine to solve a linear programming problem. Since we are just looking for an extreme point, any objective function can be used. Note that by using different objective functions we can obtain different solutions; therefore we would like to be able to define an objective function that leads to good solutions.

3.1 Ordering the vectors in the subsets

Suppose that during the execution of the algorithm, q calls of the LP-routine are made. Therefore we have found q sets $S_1, S_2, \ldots, S_q$, where $S_l = \{v_j : x^*(v_j) = 0\}$ for the extreme point x^* found. Note that $S_1, S_2, \ldots, S_q$ are disjoint subsets of the set of the vectors. As mentioned above, as long as the order of the subsets is maintained, any order of the vectors within the subsets maintains the bounds in Theorems 1 and 2. However, the length of the permutation schedule for an instance of the flow-shop scheduling problem can be greatly affected by this order. In other words, using different rules to order the vector can produce different solutions. Therefore, we shall study various rules to order vectors within the subsets and study how a good order for the partial sums related to one for the flow-shop scheduling problem.

Consider one of these subsets, say S_l. We propose the following orders:

- Calculate the $||.||_\infty$ for each vector $v \in S_l$, and sort the vectors in S_l by decreasing order of the norms.
- Calculate the $||.||_2$ for each vector $v \in S_l$, sort and the vectors in S_l by decreasing order of the norms.
- Randomly order the vectors in S_l.
- Apply the Nawaz, Enscore and Ham approximation method to the flow-shop scheduling subproblem where if $v_j \in S_l$, then J_j is a job for this subproblem.
- Complete enumeration; in other words, find all the permutations of the vectors in S_l and choose the best one in terms of the norm of the partial sums, $\| \sum_{j=1}^{k} v_{\pi(j)} \|$.

The motivation for the two first choices is that when we apply the Sevast'yanov's algorithm to the flow shop scheduling problem we would like to schedule first the "big" jobs and then the "small" ones. So, in some empirical sense, we hope to have more flexibility later in the schedule.

To have some rule related to the flow-shop scheduling problem, we choose the Nawaz, Enscore and Ham approximation method which is recognized to perform better in practice than most other approximation methods. The method constructs a permutation schedule by adding, in each iteration, the job not yet selected with highest total processing time and finding the best partial permutation.

Note the difficulty in evaluating a partial permutation at each iteration in terms of the flow-shop scheduling problem, since the order of the jobs corresponding to the vectors in S_l is highly dependent of the order of the jobs to be selected by the Sevast'yanov's algorithm in the following iteration, i.e. vectors in $S_1, S_2, \ldots S_{l-1}$. But, it is relatively easy to order the jobs based on the norm of partial sums $\| \sum_{j=1}^{k} v_{\pi(j)} \|$.

In terms of the norm of the partial sums, the general problem (P) that we are trying to solve can be stated as follows: given the ordered subsets $(S_1, S_2, \ldots, S_q)$, what is the optimal order of the vectors within these sets; that is, if $\mathcal{S}$ denotes the set of permutations consistent with these constraint we wish to:

$$\min_{\pi \in \mathcal{S}} \quad \max_{k=1,\ldots,n} \| \sum_{j=1}^{k} v_{\pi(j)} \| .$$

Define for each set the following problem, call it $P(S_l)$:

$$\min_{\pi_l} \max_{k=1,\ldots,r} \| \sum_{j=1}^{l'} v_{\pi'(j)} + \sum_{j=1}^{k} v_{\pi_l(j)} \|,$$

where π_l represents a permutation of the vectors in S_l and π' represents a permutation of the jobs in the subsets $S_1, \ldots, S_{l-1}$; let $l' = |S_1 \cup \ldots \cup S_{l-1}|$ and $r = |S_l|$.

Now, we present the result that relates the above two problems.

Theorem 3 *The concatenation of the optimal permutations, π_l^* for the sets S_l, $l = 1, \ldots, q$ where each is obtained by solving $P(S_l)$, is optimal for the general problem (P).*

Proof: Let π_{opt} be an optimal permutation to the general problem (P). Suppose now that there is a set S_l^0 such that the permutation of the elements of this set induced by π_{opt} is not optimal for this set. Let $\pi^*(S_l^0)$ be an optimal permutation to $P(S_l^0)$ and construct the following permutation π^* which is equivalent to π_{opt} everywhere except for the elements of S_l^0 which respect the order of $\pi^*(S_l^0)$.

Now define p^* and p^{opt} as follows:

$$\| \sum_{j=1}^{p^*} v_{\pi^*(j)} \| = \max_{k=1,\ldots,n} \| \sum_{j=1}^{k} v_{\pi^*(j)} \| .$$

$$\| \sum_{j=1}^{p_{opt}} v_{\pi_{opt}(j)} \| = \max_{k=1,\ldots,n} \| \sum_{j=1}^{k} v_{\pi_{opt}(j)} \| .$$

As π_{opt} is optimal then:

$$\| \sum_{j=1}^{p^*} v_{\pi^*(j)} \| \geq \| \sum_{j=1}^{p_{opt}} v_{\pi_{opt}(j)} \| .$$

Now assume that $\| \sum_{j=1}^{p^*} v_{\pi^*(j)} \| > \| \sum_{j=1}^{p_{opt}} v_{\pi_{opt}(j)} \|$ and consider the following question: where can p^* occur?

- We know that $p^* > l'$, since if $p^* \leq l'$, then both permutations are equal for the first l' positions and hence

$$\| \sum_{j=1}^{p^*} v_{\pi^*(j)} \| = \| \sum_{j=1}^{p^*} v_{\pi_{opt}(j)} \| \leq \| \sum_{j=1}^{p_{opt}} v_{\pi_{opt}(j)} \| .$$

- We know that $p^* \leq l' + r$, if not, then

$$\begin{aligned} \| \sum_{j=1}^{p^*} v_{\pi^*(j)} \| &= \| \sum_{j=1}^{l'} v_{\pi^*(j)} + \sum_{j=l'+1}^{l'+r} v_{\pi^*(j)} + \sum_{j=l'+r+1}^{p^*} v_{\pi^*(j)} \| \\ &= \| \sum_{j=1}^{l'} v_{\pi_{opt}(j)} + \sum_{j=l'+1}^{l'+r} v_{\pi_{opt}(j)} + \sum_{j=l'+r+1}^{p^*} v_{\pi_{opt}(j)} \| \\ &= \| \sum_{j=1}^{p^*} v_{\pi_{opt}(j)} \| \\ &\leq \| \sum_{j=1}^{p_{opt}} v_{\pi_{opt}(j)} \| . \end{aligned}$$

- Therefore, we can assume that $l' < p^* \leq l' + r$ and consider the problem $P(S_l^0)$. By assumption, $\pi^*(S_l^0)$ is the optimal permutation for this problem. Suppose now that the optimum is obtained at q^*, i.e.

$$\| \sum_{j=1}^{q^*} v_{\pi^*(j)} \| = \min_{\pi_l(S_l)} \max_{k=1,\ldots,r} \| \sum_{j=1}^{l'} v_{\pi(j)} + \sum_{j=1}^{k} v_{\pi_l(j)} \| .$$

  But, since $l' < p^* \leq l' + r$, then $p^* = q^*$. Now, consider the π_{opt} permutation and let q_{opt} be defined as follows:

$$\| \sum_{j=1}^{q_{opt}} v_{\pi_{opt}(j)} \| = \max_{k=l'+1,\ldots,l'+r} \| \sum_{j=1}^{k} v_{\pi_{opt}(j)} \| .$$

  Since π^* is optimal for $P(S_l^0)$, then $\| \sum_{j=1}^{q_{opt}} v_{\pi_{opt}(j)} \| \leq \| \sum_{j=1}^{p_{opt}} v_{\pi_{opt}(j)} \|$, and so,

$$\| \sum_{j=1}^{p_{opt}} v_{\pi_{opt}(j)} \| \geq \| \sum_{j=1}^{q_{opt}} v_{\pi_{opt}(j)} \| \geq \| \sum_{j=1}^{p^*} v_{\pi^*(j)} \| .$$

  This contradicts the assumption that $\| \sum_{j=1}^{p^*} v_{\pi^*(j)} \| > \| \sum_{j=1}^{p_{opt}} v_{\pi_{opt}(j)} \|$.

Hence from the above results we have: $\| \sum_{j=1}^{p^*} v_{\pi^*(j)} \| = \| \sum_{j=1}^{p_{opt}} v_{\pi_{opt}(j)} \|$. Therefore π^* is an optimal permutation to the general problem (P). $\square$

3.2 Complete enumeration

Next, we present a complete enumeration method to solve the problem of ordering the vectors within the subsets: $P(S_l)$. Observe that if the number of vectors within each set S_l is small we can solve the problem $P(S_l)$ by complete enumeration. The problem that we want to solve is:

$$\min_{\pi_l} \max_{k=1,\ldots,r} \parallel \sum_{j=1}^{l'} v_{\pi'(j)} + \sum_{j=1}^{k} v_{\pi_l(j)} \parallel .$$

Let $v_0 = \sum_{j=1}^{l'} v_{\pi(j)}$, which is a constant vector, and for simplicity assume that $S_l \equiv \{v_1, \ldots, v_r\}$. So, the problem can be rewritten as:

$$\min_{\pi} \max_{k=1,\ldots,r} \parallel v_0 + \sum_{j=1}^{k} v_{\pi(j)} \parallel,$$

where π is a permutation of $\{1, \ldots, r\}$.

Let $N_{i\pi}$ be a node where i, $i = 1, \ldots, r$, is the "level" of the node, i.e. the first i vectors are already chosen and π is the sequence of these vectors.

Let $cost(N_{i\pi}) = \max_{k=1,\ldots,i} \parallel v_0 + \sum_{j=1}^{k} v_{\pi(j)} \parallel$. Now, we have for any $v \in S_l$ and $v \notin \{\pi(1), \ldots, \pi(i)\}$, $cost(N_{(i+1),\pi//v}) = \max_{k=1,\ldots,i+1} \parallel v_0 + \sum_{j=1}^{k} v_{\pi//v(j)} \parallel = \max\{cost(N_{i\pi}), \parallel v_0 + \sum_{j=1}^{i} v_{\pi(j)} + v \parallel\}$, where $//$ stands for the concatenation of 2 permutations.

If σ is the sequence of the first i vectors of π, then $cost(N_{n\pi}) \geq cost(N_{i\sigma})$, for $i = 1, \ldots, n$.

If $|S_l| = r$, then complete enumeration takes $O(r!)$. If r is small, then this will not take too long in practice, but it is impractical for large subsets.

We can try to avoid producing all branches. Let $cost^* = \max_{k=1,\ldots,r} \parallel v_0 + \sum_{j=1}^{k} v_{\pi'(j)} \parallel$ for some permutation π' of $\{1, \ldots, r\}$. Hence, if $cost(N_{i\sigma}) \geq cost^*$ then $cost(N_{n\pi}) \geq cost^*$, where π is a permutation where the first i vectors are the subpermutation σ. Therefore, we do not need to subdivide this node further.

3.3 Dynamic Programming

In a similar way as done for the well-known traveling salesman problem, [7], we present next a dynamic programming algorithm to solve the problem that runs in $O(r^2 2^r)$. Note that the complete enumeration method above runs in $0(r!)$, which is significantly worse.

Consider the the problem $P(S_l)$, $S_l = \{v_1, \ldots, v_r\}$, and the following notation:

- $\pi^i(I)$ is a permutation of $I \subseteq \{1, \ldots, r\}$ for which $i \in I$ is the index of the last vector in the permutation.
- $C(I, i) = \min_{\forall \pi^i(I)} \max_{k=1,\ldots,|I|}\{\parallel v_0 + \sum_{j=1}^{k} v_{\pi^i(I)(j)} \parallel\}$.
- $\pi^i(I)^*$ is a permutation in $\pi^i(I)$ that attains the minimum cost: $C(I, i)$.

Note that the optimum value for the problem, say *mincost*, satisfies:

$$mincost = \min_{\forall v_k, k=1,\ldots,r} \{C(\{1, \ldots, r\}, k)\}.$$

Moreover, the $C(I, i)$ satisfies the following relation:

$$C(I,i) = \min_{\forall j \in I-\{i\}} \max\{C(I-\{i\},j), \| v_0 + \sum_{k \in I} v_k \|\}.$$

Since $\| v_0 + \sum_{k\in I} v_k \|$ does not depend on j, we have the following result.

Theorem 4 $C(I,i) = \max\{\min_{\forall j \in I-\{i\}} C(I-\{i\},j), \| v_0 + \sum_{k\in I} v_k \|\}$.

Proof:

At the start, let $I = \{i\}$ and $C(I,i) = \| v_0 + v_i \|$ for $i \in \{1,\dots,r\}$. At the iteration j for each $I \subseteq \{1,\dots,r\}$ with $|I| = t$, we have:

$$C(I,i) = \min_{\forall \pi^i(I)} \max_{k=1,\dots,t} \{\| v_0 + \sum_{j=1}^{k} v_{\pi^i(I)(j)} \|\}.$$

Note that the last vector of $\pi^i(I)$ is fixed, the vector v_i, so the value of $C(I,i)$ only depends on a permutation of the vectors in $I-\{i\}$ and $\| v_0 + \sum_{k\in I} v_k \|$.

$$C(I,i) = \min_{l\in I-\{i\}} \min_{\forall \pi^l(I-\{i\})} \max_{k=1,\dots,t} \{\| v_0 + \sum_{j=1}^{k} v_{\pi^i(I)(j)} \|\}$$

$$= \min_{l\in I-\{i\}} \min_{\forall \pi^l(I-\{i\})} \max\{\| v_0 + \sum_{k\in I} v_k \| , \max_{k=1,\dots,t-1} \{\| v_0 + \sum_{j=1}^{k} v_{\pi^l(I-\{i\})(j)} \|\}$$

$$= \max\{\| v_0 + \sum_{k\in I} v_k \| , \min_{l\in I-\{i\}} \min_{\forall \pi^l(I-\{i\})} \max_{k=1,\dots,t-1} \{\| v_0 + \sum_{j=1}^{k} v_{\pi^l(I-\{i\})(j)} \|\}$$

$$= \max\{ \min_{\forall l\in I-\{i\}} C(I-\{i\},l), \| v_0 + \sum_{k\in I} v_k \|\}. \square$$

Now, we can present the dynamic programming algorithm that is based on the last theorem:

Input: r vectors of dimension d and a vector v_0;

Output: a permutation of $\{1,\dots,r\}$ such that $\min_\pi \max_{k=1,\dots,r} \| v_0 + \sum_{j=1}^{k} v_{\pi(j)} \|$.

The dynamic programming algorithm:

```
1. For $i = 1,\dots,r$ do
    (a) $C(\{i\},i) = \| v_i \|$;
    (b) $\pi^i(\{i\})^* = i$;
2. For $j = 2,3,\dots,r$ do
    (a) For each $I \subseteq \{1,\dots,r\}$ with $|I| = t$ do: for each $i \in I$ do
        i. $C(I,i) = \max\{\min_{\forall l\in I-\{i\}} C(I-\{i\},l), \| v_0 + \sum_{k\in I} v_k \|\}$;
        ii. let $v_u$ be the vector that achieves the minimum;
        iii. $\pi^i(I)^* = \pi^u(S-\{i\})^*//(i)$;
3. $mincost = \min_{\forall v_k, k=1,\dots,r}\{C(\{1,\dots,r\},k)\}$;
4. Let $v_u$ be the vector that achieves the minimum;
5. $bestpermutation = \pi^u(\{1,\dots,r\})^*$;
```

Table 1: Bounds for the Heller's instance.

Bounds:	p_{max}	$d * p_{max}$	$\Pi_{max} + m(m-1)p_{max}$
	8	72	1298

3.4 Definition of the Objective Function:

As mentioned before, Sevast'yanov's algorithm must only find an extreme point. We also studied if there is any difference in the results obtained from using different objective functions:

1. $z(x) = 0$, i.e the zero function;
2. $z(x) = \sum_{v \in V_{k+1}} x(v)$;
3. $z(x) = \sum_{v \in V_{k+1}} r(v) * x(v)$ where $r(v)$ is a random number selected according to the uniform distribution U(0,1).

For the first objective function, we are just interested in finding an extreme point; the second one gives equal weight to each vector. For the random objective function, we consider random weights associated with the vectors.

4 Computational Results

We have implemented Sevast'yanov's algorithm in C and all tests were run on a SPARC-station 1. In our implementation, we used the routines of the Cplex library to solve the linear programming problem.

The test problems were of two types. The first type was a flow shop problem instance from Heller [8]. The second type consists of randomly generated problems. We present detailed results for 10 of these problems. Later, we present overall results for a total of 100 random generated instances.

The basic input data consists of the three parameters: n = number of jobs, $d = m-1$, where m = number of machines, and Π_{max}, with the assumption that $\Pi_{max} = \Pi_i \;, \forall i$. The processing times were generated as follows: for each machine M_i, we randomly generated $n-1$ numbers uniformly distributed over the interval $(0, \Pi_{max})$, say r_{ij}, sorted them by increasing order, set $r_{i0} = 0$, and $r_{in} = \Pi_{max}$, and calculated the processing times as follows: $p_{ij} = r_{ij} - r_{ij-1}$. Unlike many other problems, the literature suggests that the uniform distribution yields difficult problems to solve, see e.g. [2] and [3].

To calculate the length of a permutation schedule, given the permutation π, we use a dynamic programming algorithm that runs in $O(n * m)$ time.

Let $w(i, \pi(j))$ be the completion time of the j^{th} job, $\pi(j)$, to be processed on machine M_i. Assume that $w(0, \pi(1)) = 0$, and let,

$$w(1, \pi(j)) = \sum_{k=1}^{j} p_{1,\pi(k)} \text{ for } j = 1, \ldots, n.$$

Table 2: Results for the Heller's instance.

Program	max-norm sum	length of schedule	Time
0 ob.fn/$\|\|.\|\|_\infty$	15	578	2.200
random ob fn/$\|\|.\|\|_\infty$	18	578	4.333
$\sum$ ob.fn/$\|\|.\|\|_\infty$	15	578	2.233
$\sum$ ob.fn/$\|\|.\|\|_2$	14	579	2.216
$\sum$ ob.fn/random	19	560	2.233
$\sum$ ob.fn/complete enum.	14	566	9.416
random ob fn/random/10	18	578	40.433
random ob fn/random/20	18	578	79.583
random ob fn/NEH	18	584	4.383
$\sum$ ob.fn/NEH	17	572	2.216
Heller	–	577	–
NEH	–	525	31.150
ABZ	–	587	–

$$w(i, \pi(1)) = \sum_{k=1}^{i} p_{k,\pi(1)} = w(i-1, \pi(1)) + p_{i,\pi(1)} \text{ for } i = 2, \ldots, m.$$

Now let, $w(i, \pi(j)) = \max\{w(i, \pi(j-1)), w(i-1, \pi(j))\} + p_{i,\pi(j)}$ for $i = 2, \ldots, m, j = 2, \ldots, n$. Therefore, the length of the permutation schedule for π is equal a $w(m, \pi(n))$.

In Tables 1 and 2 we present the bounds and results for Heller's instance, respectively. All running times are given in seconds. The row NEH presents the results of the Nawaz, Enscore and Ham approximation method, [14]. ABZ is Applegate and Cook implementation of the Adams, Balas and Zawack approximation method for the job shop scheduling problem, [1]. Observe that this last algorithm can produce a non permutation schedule. The other rows are different versions of Sevast'yanov's algorithm, as follows: objective function/order of the vectors/number of repetitions (if greater than 1). For this instance, we can observed that the Nawaz, Enscore and Ham approximation method gave the best permutation schedule, but required a large amount of computation. The second best length was obtained by the version of the Sevast'yanov's algorithm using the sum objective function and the random order of the vectors within the subsets. Note that for this version, the maximum norm of the partial sums was the biggest one. From Table 2, we can also observe that repeating the Sevast'yanov's algorithm did not lead to an improvement in the length of the schedule. The Adams, Balas and Zawack approximation method gave the worst results, over all the methods.

In Tables 3, 4 and 5 we present the results for 10 random generated problem. Analyzing Table 3, it is evident that the Nawaz, Enscore and Ham approximation method performs better than any version of Sevast'yanov algorithm. But, as it happen for Heller's instance, a good result was obtained at a high computational expense, especially for large instances ($n = 1000$).

Among the different versions of Sevast'yanov's algorithm, the version that performs best in most of the cases, 4 out of 6, was the random objective function, in random order and repeated 20 times, but once again this comes at the cost of a high computational

Table 3: Random generated test problem – length of the schedule.

Program	$n/d/\Pi_{max}$	$n/d/\Pi_{max}$	$n/d/\Pi_{max}$
	20/5/200	100/10/1000	100/10/500
0 ob.fn/$\|\|.\|\|_\infty$	328	1424	711
random ob fn/$\|\|.\|\|_\infty$	332	1525	757
$\sum$ ob.fn/$\|\|.\|\|_\infty$	328	1424	711
$\sum$ ob.fn/$\|\|.\|\|_2$	334	1423	727
$\sum$ ob.fn/random	440	1430	705
$\sum$ ob.fn/comp. enum.	372	1505	713
random ob fn/random/10	329	1403	712
random ob fn/random/20	329	1403	710
random ob fn/NEH	372	1468	734
$\sum$ ob.fn/NEH	372	1424	706
NEH	268	1155	573
	100/50/5000	1000/10/5000	1000/100/5000
0 ob.fn/$\|\|.\|\|_\infty$	13919	5430	8228
random ob fn/$\|\|.\|\|_\infty$	13406	5402	8327
$\sum$ ob.fn/$\|\|.\|\|_\infty$	13919	5430	8228
$\sum$ ob.fn/$\|\|.\|\|_2$	13727	5418	8218
$\sum$ ob.fn/random	13566	5400	8153
$\sum$ ob.fn/comp. enum.	13813	5372	8227
random ob fn/random/10	13099	5369	8252
random ob fn/random/20	13099	5366	8201
random ob fn/NEH	13224	5374	8274
$\sum$ ob.fn/NEH	13700	5411	8230
NEH	10081	5049	6806

Table 4: Random generated test problem – maximum norm of partial sums.

Program	$n/d/\Pi_{max}$	$n/d/\Pi_{max}$	$n/d/\Pi_{max}$
	20/5/200	100/10/1000	100/10/500
0 ob.fn/$\|\|.\|\|_\infty$	63	107	52
random ob fn/$\|\|.\|\|_\infty$	63	104	47
$\sum$ ob.fn/$\|\|.\|\|_\infty$	63	107	52
$\sum$ ob.fn/$\|\|.\|\|_2$	63	107	52
$\sum$ ob.fn/random	78	90	45
$\sum$ ob.fn/comp. enum.	40	74	40
random ob fn/random/10	51	69	33
random ob fn/random/20	51	69	41
random ob fn/NEH	73	79	36
$\sum$ ob.fn/NEH	73	107	45
	100/50/5000	1000/10/5000	1000/100/5000
0 ob.fn/$\|\|.\|\|_\infty$	738	63	151
random ob fn/$\|\|.\|\|_\infty$	897	70	148
$\sum$ ob.fn/$\|\|.\|\|_\infty$	738	63	151
$\sum$ ob.fn/$\|\|.\|\|_2$	716	63	151
$\sum$ ob.fn/random	722	61	153
$\sum$ ob.fn/comp. enum.	720	59	146
random ob fn/random/10	842	73	139
random ob fn/random/20	842	65	164
random ob fn/NEH	967	67	144
$\sum$ ob.fn/NEH	777	71	160

expense. Among the remaining ones, there is no clear evidence that one version is better than the others.

Considering the results presented in Tables 3 and 4, we see that a good result for the maximum norm of the partial sums does not reflect a good result with respect to the flow-shop scheduling problem. In 3 cases, 20/5/200, 100/10/100 and 1000/10/5000, the smaller maximum norm of the vectors was obtained by the version with sum objective function and the complete enumeration, but the length of the corresponding schedule was not the best one.

We have noted that the number of calls of the LP-routine is highly dependent of the number of machines, for the same number of jobs. This number was very similar for the 3 different objective functions; for the instances which results are presented in Table 5 in average there was 5, 15 and 16 call of the LP-routine for 20/5/200, 100/10/1000 and 100/10/500 instances respectively; and 4, 146 and 17 call of the LP-routine for 100/50/5000, 1000/10/5000 and 1000/100/5000 instances respectively. For a fixed number of jobs n, if we increase the number of machines, the number of calls of the LP-routine decreases. But, at the same time, whereas there are fewer calls of the routine, there is an increase in the running time. This is due to the fact that the linear programming problem has more constraints, i.e., the number of constraints is $d+1 = m$. And moreover, the size

Table 5: Random generated test problem – running times.

Program	$n/d/\Pi_{max}$	$n/d/\Pi_{max}$	$n/d/\Pi_{max}$
	20/5/200	100/10/1000	100/10/500
0 ob.fn/$\|\|.\|\|_\infty$	$1.333 * 10^{-1}$	$1.983 * 10^{0}$	$2.050 * 10^{0}$
random ob fn/$\|\|.\|\|_\infty$	$1.667 * 10^{-1}$	$4.383 * 10^{0}$	$2.883 * 10^{0}$
$\sum$ ob.fn/$\|\|.\|\|_\infty$	$1.333 * 10^{-1}$	$2.000 * 10^{0}$	$2.033 * 10^{0}$
$\sum$ ob.fn/$\|\|.\|\|_2$	$1.667 * 10^{-1}$	$2.017 * 10^{0}$	$2.067 * 10^{0}$
$\sum$ ob.fn/random	$1.667 * 10^{-1}$	$2.000 * 10^{0}$	$2.116 * 10^{0}$
$\sum$ ob.fn/comp. enum.	$2.000 * 10^{0}$	$2.968 * 10^{1}$	$1.761 * 10^{1}$
random ob fn/random/10	$1.683 * 10^{0}$	$3.980 * 10^{1}$	$3.840 * 10^{1}$
random ob fn/random/20	$3.233 * 10^{0}$	$7.852 * 10^{1}$	$7.725 * 10^{1}$
random ob fn/NEH	$1.833 * 10^{-1}$	$4.567 * 10^{0}$	$3.917 * 10^{0}$
$\sum$ ob.fn/NEH	$1.667 * 10^{-1}$	$2.033 * 10^{0}$	$2.150 * 10^{0}$
NEH	$1.667 * 10^{-1}$	$3.500 * 10^{1}$	$3.447 * 10^{1}$
	100/50/5000	1000/10/5000	1000/100/5000
0 ob.fn/$\|\|.\|\|_\infty$	$8.283 * 10^{0}$	$2.027 * 10^{2}$	$1.429 * 10^{3}$
random ob fn/$\|\|.\|\|_\infty$	$1.647 * 10^{1}$	$3.530 * 10^{2}$	$6.146 * 10^{3}$
$\sum$ ob.fn/$\|\|.\|\|_\infty$	$8.183 * 10^{0}$	$2.041 * 10^{2}$	$1.451 * 10^{3}$
$\sum$ ob.fn/$\|\|.\|\|_2$	$8.267 * 10^{0}$	$2.046 * 10^{2}$	$1.445 * 10^{3}$
$\sum$ ob.fn/random	$8.167 * 10^{0}$	$2.054 * 10^{2}$	$1.449 * 10^{3}$
$\sum$ ob.fn/comp. enum.	$3.359 * 10^{2}$	$4.404 * 10^{2}$	$8.876 * 10^{3}$
random ob fn/random/10	$1.627 * 10^{2}$	$3.696 * 10^{3}$	$6.183 * 10^{4}$
random ob fn/random/20	$3.216 * 10^{2}$	$7.278 * 10^{3}$	$1.229 * 10^{5}$
random ob fn/NEH	$2.268 * 10^{1}$	$3.560 * 10^{2}$	$6.440 * 10^{3}$
$\sum$ ob.fn/NEH	$1.433 * 10^{1}$	$2.053 * 10^{2}$	$1.786 * 10^{3}$
NEH	$1.623 * 10^{2}$	$3.487 * 10^{4}$	$3.272 * 10^{5}$

Table 6: Random generated test problem – average relative difference between the length of the schedule and the reference length obtained by the Nawaz, Enscore and Ham approximation method.

Program	$n/d/\Pi_{max}$		
	100/10/1000	100/25/1000	100/50/1000
0 ob.fn/$\|\|.\|\|_\infty$	0.2917	0.3459	0.3261
$\sum$ ob.fn/$\|\|.\|\|_\infty$	0.2917	0.3459	0.3261
random ob fn/$\|\|.\|\|_\infty$	0.2970	0.3466	0.3255
0 ob.fn/$\|\|.\|\|_2$	0.2863	0.3382	0.3211
random ob.fn/$\|\|.\|\|_2$	0.2901	0.3427	0.3201
random ob.fn/comp. enum.	0.2680	0.3327	0.3146
0 ob.fn/comp. enum.	0.2608	0.3249	0.3150
0 ob.fn/random	0.2876	0.3374	0.3246
random ob fn/random	0.2963	0.3402	0.3249

of the subsets are bigger (there are fewer calls of the routine), and as a consequence it takes more time to order the vectors in the subsets. The version where we use the complete enumeration method to order the vectors within the subsets took a large amount of time on the instances with a large number of machines, as was expected. Therefore we just use the complete enumeration method if $|S_l| \leq 8$; otherwise we split the set S_l in subsets of size 8 or smaller, and apply the enumeration method to each of them.

In comparing the different objective functions, the zero function and the sum function always gave the same results, when the same rule to order the vectors within the subsets was used. Between the random objective function and the sum function, it appears that the second one performed better. Also, no rule for ordering the vectors outperforms the other ones, when the same objective function was used. The rule based on the Nawaz, Enscore and Ham approximation algorithm did not give better results than the other ordering rules.

In Tables 6, 7 and 8, we present the average results over 100 different random generated problems with 100 jobs and 10, 25 and 50 machines, respectively. We can observe that among the different versions of Sevast'yanov's algorithm, there is not one that drastically improves the results in comparison with the other ones.

In Table 7, it is evident that as the number of machines increases, the number of calls of the LP routine is reduced. Note that if d increases, the right-hand-side of the last constraint of the linear programming problem is smaller; this leads to a larger number of variables assigned to zero, and therefore to larger subsets S_l.

From Table 8, we can observe that the complete enumeration to order the vectors within subsets leads to the highest running times, followed by the version where the random objective function was used.

In the above results we observed a linear relation between the number of call of the LP-routine, Y, and the number of jobs n, divide by the number of machines m, X. To better understand this relation we tested the algorithm on 150 randomly generated instances, using the zero objective function and ordered the subsets by decreasing values of the

Table 7: Random generated test problem – average number of calls of the LP-routine.

Program	$n/d/\Pi_{max}$		
	100/10/1000	100/25/1000	100/50/1000
0 ob.fn/$\|\|.\|\|_\infty$	15.58	7.03	3.95
$\sum$ ob.fn/$\|\|.\|\|_\infty$	15.58	7.03	3.95
random ob fn/$\|\|.\|\|_\infty$	16.88	7.19	3.94
0 ob.fn/$\|\|.\|\|_2$	15.58	7.03	3.95
random ob.fn/$\|\|.\|\|_2$	16.88	7.19	3.94
random ob.fn/comp. enum.	16.88	7.19	3.94
0 ob.fn/comp. enum.	15.58	7.03	3.95
0 ob.fn/random	15.58	7.03	3.95
random ob fn/random	16.88	7.19	3.94

Table 8: Random generated test problem – average running times.

Program	$n/d/\Pi_{max}$		
	100/10/1000	100/25/1000	100/50/1000
0 ob.fn/$\|\|.\|\|_\infty$	$2.10 * 10^0$	$3.74 * 10^0$	$8.30 * 10^0$
$\sum$ ob.fn/$\|\|.\|\|_\infty$	$2.11 * 10^0$	$3.76 * 10^0$	$8.34 * 10^0$
random ob fn/$\|\|.\|\|_\infty$	$4.13 * 10^0$	$8.63 * 10^0$	$1.70 * 10^1$
0 ob.fn/$\|\|.\|\|_2$	$2.10 * 10^0$	$3.75 * 10^0$	$8.32 * 10^0$
random ob.fn/$\|\|.\|\|_2$	$4.14 * 10^0$	$8.65 * 10^0$	$1.70 * 10^1$
random ob.fn/comp. enum.	$1.61 * 10^1$	$1.43 * 10^2$	$3.41 * 10^2$
0 ob.fn/comp. enum.	$2.20 * 10^1$	$1.45 * 10^2$	$3.35 * 10^2$
0 ob.fn/random	$2.10 * 10^0$	$3.73 * 10^0$	$8.30 * 10^0$
random ob fn/random	$4.14 * 10^0$	$8.62 * 10^0$	$1.70 * 10^1$

infinity norm. And then, we did a regression analysis study on the results. We conclude that, given the number of jobs n and the number of machines m of an instance of the flow-shop scheduling problem, the number of calls of the LP-routine is well approximated by $1.911663 + 1.384563 * \frac{n}{m}$, [13]. And consequently, we can obtain an approximate running time when the Sevast'yanov algorithm is applied to a given instance, since the LP-routine is the most expensive step in the algorithm. Note that the size of the LP problems to be solved is known, so the running time of one call of the LP-routine can be approximately estimated, and of course, this running time depends on the software used as the LP-routine.

Based on the above results, we can conclude that if the running time is not a critical factor, the Nawaz, Enscore and Ham approximation method still performs best. But, if the time is a important factor, Sevast'yanov's algorithm can be a very good alternative, especially in the presence of very large scale instances with a relatively small number of machines. But, it should be pointed out that Sevast'yanov's algorithm highly depends on the routine to solve the linear programming problem.

5 Conclusions

The Nawaz, Enscore and Ham approximation method performs best in all cases, but took as much time as our most time-consuming version for average size instances, and much longer for the biggest ones. Between the different versions of the Sevast'yanov's algorithm, there is no strong evidence that one outperforms the remaining ones.

A good permutation to the maximum norm of the partial sums does not always corresponds to a good permutation to the flow-shop scheduling problem.

In future work, it would be interesting to implement and test the fast implementation of the Sevast'yanov's algorithm using an efficient and numerically stable method to invert matrices.

The vector sum theorem can also be used to derive an approximation algorithm to solve the job shop problem, see [18] and [12]. For any instance of the job-shop scheduling problem, a schedule of length at most $C^*_{max} + O(m\mu^3_{max}p_{max})$ can be found in polynomial time. In this case we have n vectors that correspond to the n jobs, but now $d = m\mu_{max}$ where m is the number of machines and μ_{max} is the maximum number of operations of a job. This algorithm is more sophisticated than the one for the flow-shop scheduling problem. But, note that for the instances of the job-shop scheduling problem usually reported in the literature with 10 jobs and 10 machines, any permutation achieves the bound.

In terms of theoretical results, it will be important to derive an algorithm for the vector sum problem that run in parallel, or to prove that this is difficult. One reason for this is that the algorithm for the general case of the job shop problem in [19] rely on the algorithm for the job shop problem based on the vector-sum theorem.

Acknowledgment

The author is grateful to David Shmoys for many comments and suggestions which have considerably improved this paper.

References

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for the job-shop scheduling. *Management Science*, 34(3):391–401, March 1988.

[2] H.G. Campbell, R.A. Dudek, and M.L Smith. A heuristic method for the n job, m machine sequencing problem. *Management Science*, 16:630–637, 1970.

[3] D.G. Dannenbring. An evaluation of flowshop sequencing heuristics. *Management Science*, 23:1174–1182, 1977.

[4] R.A. Dudek, S.S. Panwalkar, and M.L. Smith. The lessons of flowshop scheduling research. *Operations Research*, 40(1):7–13, January-February 1992.

[5] M.R. Garey, D.S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1:117–129, 1976.

[6] J.N.D. Gupta. Heuristic algorithms for multistage flowshop problem. *AIIE Trans.*, 4:11–18, 1972.

[7] M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. *SIAM J. Appl. Math*, 10:196–210, 1962.

[8] J. Heller. Some numerical experiments for an $m \times j$ flow shop and its theoretical aspects. *Operations Research*, 8:178–184, 1960.

[9] J.C. Ho and Y.L. Chang. A new heuristic for the n-job, m-machine flow-shop problem. *European Journal Operational Research*, 52(2):194–202, 1991.

[10] S.M. Johnson. Optimal two and three-stage production schedules with setup times included. *Naval Research Logistic Quartely*, 1:61–68, 1954.

[11] E.L. Lawler, J.K.Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: algorithms and complexity. Technical report, Eindhoven University of Technology, Eindhoven, The Netherlands, 1989. Designing Decision Support System Notes.

[12] E.L. Lawler, J.K.Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *Sequencing and scheduling: algorithms and complexity.* to be published, 1993.

[13] H.R. Lourenço. *A computational study of the job-shop and the flow-shop scheduling problems.* PhD thesis, School of OR& IE, Cornell University, Ithaca, NY, USA, August 1993.

[14] M. Nawaz, E.E. Enscore Jr., and I. Ham. A heuristic algorithm for th m-machine, n-job flow-shop sequencing problem. *OMEGA International Journal of Management Science*, 11(1):91–95, 1983.

[15] I.H. Osman and C.N. Potts. Simulated annealing for permutation flow-shop scheduling. Technical report, Faculty of Mathematical Studies, University of Southampton, 1989.

[16] D.S. Palmer. Sequencing jobs through a multi-stage process in the minimum total time - a quick method of obtaining a near optimum. *Operations Research Quartely*, 16:101–107, 1965.

[17] A.H.G. Rinnooy Kan. *Machine scheduling problems: classification, complexity and computations*. Nijhoff, The Hague, 1976.

[18] S.V. Sevast'yanov. Bounding algorithm for the routing problem with arbitrary paths and alternative servers. *Kibernetika*, 22(60):74–79, 1986. Translation in Cybernetics 22, pages 773–780.

[19] D.B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. Technical report, School of OR&IE, Cornell University, Ithaca, NY, USA, 1990.

[20] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, 1990.

[21] M. Widmer and A. Hertz. A new heuristic method for the flow-shop sequencing problem. *European Journal Operational Research*, 41:186–193, 1989.