Università di Pisa
Dipartimento di Informatica

Technical Report : TR-99-22

# Multi-exchange algorithms for the minimum makespan machine

*A. Frangioni, M.G. Scutella', E. Necciari*

December 14, 1999

ADDR: Corso Italia 40,56125 Pisa,Italy. TEL: +39-50-887111. FAX: +39-50-887226

# Multi-exchange algorithms for the minimum makespan machine scheduling problem

Antonio Frangioni Emiliano Necciari Maria Grazia Scutellà *

December 15, 1999

**Abstract**

We address the problem of scheduling independent jobs to identical parallel machines in order to minimize the completion time (makespan). This is a hard problem, for which both constructive and improvement heuristics have been proposed. We describe new local search algorithms, whose neighborhood structure is based on multiple exchanges of jobs among machines. Inspired by the work of Thompson and Orlin (1989) on cyclic transfer neighborhood structures, in these algorithms multiple exchanges are modelled in terms of "disjoint cycles" on suitably defined improvement graphs, and improvement moves are obtained by heuristically constructing special disjoint cycles in such auxiliary graphs. The algorithms have been tested on benchmark instances, characterized by uniformly distributed processing times, and on a new class of "highly non-uniform" instances, generated by the authors, which seem to be, in some cases, more difficult than the "uniform" ones. The algorithms have shown a potential both for generating highly accurate solutions when the running time is not an issue, and for rapidly obtaining good solutions when the running time has to be kept small.

*Subject classifications*: Production/scheduling, Approximations/heuristic: multi-exchange local search. Networks/graphs, Flow algorithms: disjoint cycle computation.

## 1 Introduction

Consider $n$ independent jobs $j_1, j_2, \ldots, j_n$, having positive processing times $p_1, p_2, \ldots, p_n$, respectively. The jobs have to be scheduled, without preemption, to $m$ identical parallel machines $M_1$, $M_2, \ldots, M_m$, where $n \geq m \geq 2$.

Given a job scheduling $S$, i.e., an assignment of each job to one machine, let $C_h(S)$ denote the completion time of machine $M_h$ under the scheduling $S$, $h = 1, \ldots, m$, and $C_{max}(S) = \max_{h=1,\ldots,m}\{C_h(S)\}$ denote the *makespan* of $S$. The *minimum makespan machine scheduling problem* consists of finding a job scheduling $S$ which minimizes the makespan $C_{max}(S)$. Using the standard three field classification scheme, introduced in [Lawler et al., 1993], this problem is usually denoted as $P||C_{max}$.

$P||C_{max}$ is NP-hard in the strong sense when the number of machines is arbitrary, whereas it is solvable in pseudopolynomial time when $m$ is fixed (Garey and Johnson, 1978). That justifies the use of heuristic methods to compute "good" feasible solutions. Heuristics for $P||C_{max}$ can be classified into constructive heuristics and improvement heuristics. The majority of those proposed in the literature fall into the first category, for which a worst-case performance ratio is generally provided. We cite, among the others, the *list scheduling* ($LS$) family of Graham (1966), which includes the well-known *longest processing time* ($LPT$) algorithm, and the *multifit* ($MF$) algorithm

*Dipartimento di Informatica, Corso Italia 40, 56125 Pisa (ITALY), e-mail: [frangio,necciari,scut]@di.unipi.it

by Coffman, Garey and Johnson (1978). See [Lawler et al., 1993] for a complete survey about constructive heuristics for the problem.

As far as improvement heuristics are concerned, few local search methods have been proposed sofar. An $O(n\log m)$ time algorithm is proposed in [Finn and Horowitz, 1979], which reassigns jobs from most loaded to less loaded machines, and performs interchanges of jobs. Such an algorithm, also known as *0/1 interchange* algorithm, has been later improved by Langston (1982). A more sophisticated algorithm is suggested by França et al. (1994). Such an algorithm uses a combined critical reassign and critical swap neighborhood structure. In the same period a tabu search algorithm is proposed by Hübscher and Glover (1994), which uses "influential diversification" to modify the current solution when no improvement is performed for several iterations. Finally, a more recent local search algorithm is described in [Fatemi-Ghomi and Jolai-Ghazvini, 1998], which is based on pairwise interchanges of the jobs.

In this paper, new local search algorithms are proposed, whose neighborhood structure is based on multiple exchanges of jobs among machines. These algorithms model multiple exchanges in terms of "disjoint cycles" on suitably defined improvement graphs.

The plan of the paper is the following. Section 2 contains the presentation of the neighborhood structure. In particular, it contains the definition of the improvement graph, the characterization of the improvement moves in terms of special subgraphs of the improvement graph, and the description of the algorithms used to explore the neighborhood for the discovering of such special subgraphs. Section 3 describes the main issues addressed in implementing the local search algorithms, while Section 4.2 reports the results of a wide computational experimentation aimed to investigate about the computational efficiency and the effectiveness of the proposed local search approach, and the conclusions that can be drawn upon it.

## 2 A multi-exchange neighborhood structure

Given a solution for $P||C_{max}$, i.e., a schedule of the $n$ jobs to the $m$ machines, aim of the proposed neighborhood structure is to try to improve the current solution by allowing multiple exchanges of jobs among the machines, i.e., moving several jobs among several machines. Until now, neighborhood structures based on multiple exchanges, also called *multi-exchanges*, have been proposed only for some Vehicle routing problems (VRP) (Thompson and Psaraftis, 1993), and for the Capacitated minimum spanning tree problem (CMST) (Ahuja, Orlin and Sharma, 1998).

As in [Ahuja, Orlin and Sharma, 1998], two kinds of multi-exchanges are defined, i.e., cyclic and path exchanges.

Let $S$ denote the current (feasible) solution to the problem. Being $S$ a partition of the $n$ jobs to the $m$ machines, it can be represented by the notation $S = \{S_1, S_2, \ldots, S_m\}$, where $S_h$ denotes the set of jobs assigned to machine $M_h$, $h = 1, \ldots, m$. Moreover, given a job $j$, let $M(j)$ denote the machine assigned to job $j$ into the current solution.

Then, a *cyclic exchange* with respect to $S$ (or simply cyclic exchange) is a job sequence $W = j_1, j_2, \ldots, j_r, j_1$ such that $M(j_k) \neq M(j_q)$ for $k \neq q$, with $k, q \in \{1, 2, \ldots, r\}$. The cyclic exchange $W$ represents the following simoultaneous exchange of jobs, currently assigned to different machines: job $j_1$ moves from machine $M(j_1)$ to machine $M(j_2)$, job $j_2$ moves from machine $M(j_2)$ to machine $M(j_3)$, and so on until job $j_r$, which moves from machine $M(j_r)$ to machine $M(j_1)$. If $S'$ denote the obtained (feasible) solution, then the cost of the exchange is, as usual, $C(W) = C_{max}(S') - C_{max}(S)$; $W$ is said to be an *improvement cyclic exchange* if $C(W) < 0$.

Similarly, a *path exchange* is a sequence $P = j_1, j_2, \ldots, j_{r-1}, M_r$, where $j_1, j_2, \ldots, j_{r-1}$ are jobs whereas $M_r$ represents a machine, and $M(j_k) \neq M(j_q) \neq M_r$ for $k \neq q$, with $k, q \in \{1, 2, \ldots, r-1\}$. The meaning is that job $j_1$ moves from machine $M(j_1)$ to machine $M(j_2)$, job $j_2$ moves from machine $M(j_2)$ to machine $M(j_3)$, and so on until job $j_{r-1}$, which moves to machine $M_r$. With respect to the cyclic exchange, the only difference is that no job is assigned to machine $M(j_1)$, and that no job leaves machine $M_r$. The definition of improvement path exchange is analogous to the one provided for the cyclic exchange.

Observe that the definition of cyclic and path exchange could be easily generalized by allowing the movement of subsets of jobs from one machine to another one, simultaneously.

Now, given $S$, define as *neighborhood* of $S$ the set of all the assignements of jobs that can be obtained from $S$ by performing a cyclic or a path exchange. The cardinality of the neighborhood may grow exponentially with respect to the problem size: the idea is then to model and search such a neighborhood by means of a suitably defined graph, called the improvement graph. The graph definition, which extends to the machine scheduling problem concepts already introduced in the context of VRP and CMST, will be the subject of the following section.

## 2.1 The improvement graph

Given a solution $S$, the *improvement graph* relative to $S$ is a directed graph, $G(S) = (N(S), A(S))$, which is associated with $S$ in order to describe, in a compact way, the set of the cyclic and of the path exchanges relative to $S$. The graph contains one node for each job $j_1, j_2, \ldots, j_n$, and one node for each machine in $\{M_1, M_2, \ldots, M_m\}$ which has a completion time less than $C_{max}(S)$; hereafter, these machines will be referred to as *unloaded machines*, whereas the ones having a completion time equal to $C_{max}(S)$ will be referred to as *loaded machines* (with a little abuse of the notation, we will speak indifferently of nodes of the improvement graph, and of jobs and/or machines of the instance under consideration).

The arc set of the improvement graph, $A(S)$, contains an arc $(i, j)$ for each pair of jobs $i$ and $j$ which are currently assigned to different machines, and which satisfy the following property: if we simoultaneously move $i$ to $M(j)$, and remove $j$ from $M(j)$, then machine $M(j)$ is unloaded. Moreover, $A(S)$ contains an arc $(i, M_r)$ for each pair job-(unloaded)machine such that $i$ is currently not assigned to $M_r$, and such that, by moving $i$ to $M_r$, then machine $M_r$ is still unloaded. Clearly, it is $|N(S)| \leq n + m - 1$ and $|A(S)| \leq n(n + m - 2)$.

Now, a directed cycle $W = j_1, j_2, \ldots, j_r, j_1$ in $G(S)$ is said to be a *disjoint cycle* if $M(j_k) \neq M(j_q)$ for $k \neq q$, with $k, q \in \{1, 2, \ldots, r\}$; similarly, a directed path $P = j_1, j_2, \ldots, j_{r-1}, M_r$ in $G(S)$ is a *disjoint path* if $M_r$ represents a machine and $M(j_k) \neq M(j_q) \neq M_r$ for $k \neq q$, with $k, q \in \{1, 2, \ldots, r - 1\}$ (observe that nodes representing machines cannot appear in a cycle or be internal to a path, since they have no outgoing arc in $G(S)$).

From the above definitions the following property can be easily proved, which allows one to map a significant subset of the cyclic and path exchanges relative to the current solution $S$ onto special subgraphs of the associated improvement graph $G(S)$:

**Property 2.1** *Each disjoint cycle (path) in $G(S)$ corresponds to a cyclic (path) exchange relative to $S$.*

Due to the definition of the arc set $A(S)$, the opposite correspondence may not hold, i.e., there may be cyclic or path exchanges which do not correspond to disjoint cycles or paths in $G(S)$. In fact, cyclic (path) exchanges which *increase* the current makespan are not mapped onto cycles (paths) of $G(S)$. Such a quite peculiar property does not hold in the case of VRP and of CMST where, due to the different type of the objective function under consideration (a max function in the case of the machine scheduling problem, a sum function in the case of VRP and of CMST), moves which "locally" increase the objective function cannot be ruled out, and therefore multi-exchanges worsening the objective function have to be taken into account.

Furthermore, also the kind of correspondence existing between improvement multi-exchanges and disjoint cycles (paths) in the case of the minimum makespan machine scheduling problem is quite different from that occurring in the case of VRP and CMST. For those problems, it has been proved that improvement multi-exchanges are in a one-to-one correspondence with negative disjoint cycles and paths in suitably defined improvement graphs, where arc costs are introduced to model the costs of the exchanges represented by the single arcs. In the minimum makespan case, instead, a subset of disjoint cycles and paths in $G(S)$ can be defined which characterizes the improvement multi-exchanges relative to the current solution $S$ only thanks to its special structure, i.e., independently of arc costs. In order to state such a characterization, a further definition is required.

Let $K$ denote the set of all the loaded machines in the current solution $S$: a *$K$-disjoint cycle* is a disjoint cycle $W = j_1, j_2, \ldots, j_r, j_1$ such that the set of the machines involved into the exchange, i.e., $\{M(j_1), M(j_2), \ldots, M(j_r)\}$, includes the set of the loaded machines $K$. A similar definition can be given for *$K$-disjoint paths* in $G(S)$. Then:

**Property 2.2** *There is a one-to-one correspondence between the set of the improvement cyclic (path) exchanges relative to the current solution $S$, and the set of the $K$-disjoint cycles (paths) in $G(S)$.*

***Proof:*** Consider an improvement cyclic exchange relative to $S$; being an improvement move, the makespan associated with $S$ reduces after performing the cyclic exchange. That means that all the loaded machines, i.e., all the machines belonging to the set $K$, are involved; moreover, all the involved machines have a completion time less than $C_{max}(S)$ after the exchange. Such a cyclic exchange thus corresponds to a K-disjoint cycle in $G(S)$; in particular, by definition of $A(S)$, only exchanges that guarantee a strict decrease of the makespan are modelled via the arcs of the improvement graph.

The opposite correspondence and the case of the improvement path exchanges can be proved in a similar way. ◇

Based on Property 2.2, the problem of finding an improvement move in our multi-exchange neighborhood can be reformulated as the problem of computing a $K$-disjoint cycle or path in the improvement graph, $G(S)$, associated with the current solution $S$. Unfortunately this is a difficult problem, as proved by the theorem below.

**Theorem 2.1** *Given a directed graph $G = (V, E)$, whose node set is partitioned in $h$ clusters, $H = (V_1, V_2, \ldots, V_h)$, and given a subset $W$ of the cluster set $H$, the problem of computing a directed cycle in $G$ which includes exactly one node for each cluster in $W$ and at most one node for each cluster in $H \setminus W$ is $NP$-hard.*

***Proof:*** Consider the case $|W| = 1$, and assume that the only selected cluster contains a single node, say $s$; with these assumptions, the problem becomes that of finding a directed cycle in $G$ which includes node $s$, and which contains at most one node for each cluster in $H \setminus W$. This is polynomially equivalent to the problem of verifying the existence of a directed path in $G$, from $s$ to a node $t \neq s$, which contains at most one node for each cluster in $H \setminus W$. This last problem is $NP$-complete since it includes, as its special case, the so-called *path with forbidden pairs* problem, i.e., the problem of deciding about the existence of a directed path, from a given source $s$ to a given destination $t$ in a directed graph $G$, containing at most one node for each pair of a given sequence of (disjoint) pairs of nodes (Garey and Johnson, 1978). ◇

Remarkably, by Theorem 2.1 not only the problem of computing a $K$-disjoint cycle (path) in the improvement graph is $NP$-hard, but also the "relaxed" problem of looking for a disjoint cycle (path) in $G(S)$ which includes *at least one* loaded machine is "difficult" (this is the case $|W| = 1$, with $W$ subset of the nodes assigned to loaded machines). In the machine scheduling terminology, the second problem corresponds to looking for a multi-exchange which does not worse the current makespan, and strictly improves the completion time for at least one loaded machine. In other words, that corresponds to enlarging the set of the improvement moves, by allowing not only multi-exchanges which reduce the current makespan, but also multi-exchanges which "partially" improve the current solution.

In the following sections, both kinds of improvement multi-exchanges will be addressed. Due to Theorem 2.1, heuristic approaches will be investigated both for the problem of discovering a $K$-disjoint cycle (path) in $G(S)$, and for the problem of computing a disjoint cycle (path) in $G(S)$ which includes at least one loaded machine. This is a coherent choice in the present context, being the entire algorithmic paradigm under investigation of the heuristic type.

## 2.2 Heuristic approaches for the $K$-disjoint cycle computation

We propose two different heuristics, which rely on two alternative definitions of the arc costs in $G(S)$. These costs are used to heuristically guide the node selection during the $K$-disjoint cycle (path) computation.

Given an arc $(i,j) \in A(S)$, define the cost of $(i,j)$ as

$$c_{ij} = \begin{cases} p_i - p_j, & \text{if } j \text{ is a job and } M(j) \text{ is loaded;} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

This cost measures the reduction of the completion time of machine $M(j)$ if it is loaded, i.e., it contributes to the current makespan (in this case, it is $c_{ij} < 0$); otherwise, it is $c_{ij} = 0$. An example is provided in figure 1. The example refers to an instance characterized by 3 machines $M_1$, $M_2$ and $M_3$, and by 10 jobs with processing times $\{3,3,4,5,5,6,7,4,2,9\}$, respectively. The figure shows a portion of the improvement graph relative to the solution $S = \{S_1, S_2, S_3\}$, where $S_1 = \{1,4,7\}$, $S_2 = \{2,5,8,10\}$ and $S_3 = \{3,6,9\}$; $M_2$ is the only loaded machine, with makespan 21.

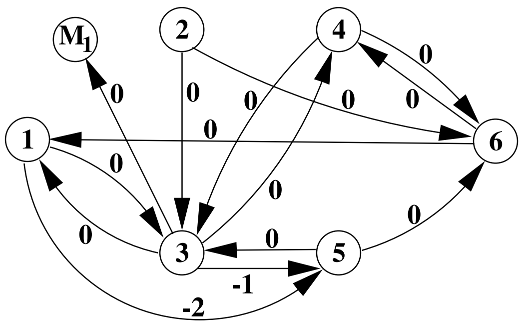


Figure 1: The cost definition (1)

The alternative cost definition measures the reduction of the completion time, with respect to the current makespan, for both loaded and unloaded machines:

$$c_{ij} = \begin{cases} \sum_l p_l + (p_i - p_j) - C_{max}(S), & \text{if } j \text{ is a job;} \\ \sum_l p_l + (p_i) - C_{max}(S), & \text{otherwise,} \end{cases} \tag{2}$$

where $l$ denotes the generic job assigned to machine $M(j)$. Figure 2 shows the same portion of the improvement graph provided in figure 1, where the arc costs are defined accordingly to (2).

Definition (1) suggests to search for a $K$-disjoint cycle or path in the improvement graph by an approach which is analogous to the one proposed in [Ahuja, Orlin and Sharma, 1998] for CMST, i.e., by a modification of the label correcting approach for shortest path computation (for a complete survey about shortest path methods, we refer to [Ahuja, Magnanti and Orlin, 1993]).

On the other hand, the cost definition (2) is appropriate for a different approach, which is based on a modification of the algorithm which computes a bottleneck path tree in a directed graph, and which is ad hoc for the problem under consideration.

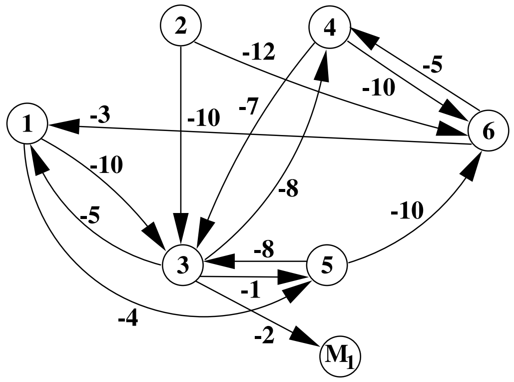


Figure 2: The cost definition (2)

**A label correcting heuristic** Consider the label correcting approach. Given a directed graph $G$ with arc costs $c_{ij}$, and given a root node $r$, a shortest path tree rooted at $r$ is found by maintaing a *distance label* $d(j)$ and a *predecessor* $pred(j)$ for each node $j$. At each iteration, a node $i$ is selected from the set, say $Q$, of the so-called candidate nodes, and the *Bellman conditions* are checked for each arc $(i, j)$ in the forward star of $i$: if

$$d(j) > d(i) + c_{ij}, \tag{3}$$

then the distance label of node $j$ is updated ($d(j) := d(i) + c_{ij}$), the predecessor of $j$ is set equal to $i$ ($pred(j) := i$), and $j$ is added to $Q$ if not already present.

As known from the literature, several algorithms have been derived from the above general framework by suitably implementing the set $Q$; when enriched by a cycle detection strategy, these algorithms allow one to discover negative cycles in the input graph $G$.

Starting from this general algorithmic framework, a heuristic, hereafter referred to as $K - SPT$, has been proposed for finding $K$-disjoint cycles (paths) in $G(S)$. In addition to the labels $d(j)$ and $pred(j)$ associated with each node $j$, a further label $nm(j)$ is maintained, which counts the number of the loaded machines, i.e., the machines belonging to the set $K$, which are involved in the current path from a node $r$, chosen as the root, to the node $j$. When a node $i$ is removed from $Q$, the following typical iteration is performed until either a $K$-disjoint cycle (path) is found, or $Q$ is empty.

*Typical iteration*

The current path from $r$ to $i$ is checked for disjointness: changes in subpaths may in fact have occurred during the algorithm execution, which may have destroyed this property. If the path is not disjoint, then the iteration is terminated (a new node is extracted from $Q$); otherwise, each arc $(i, j)$ in the forward star of $i$ is examined. For each $(i, j)$, if (3) holds, then, in addition to update $d(j)$ and $pred(j)$ as in the standard shortest path computation, label $nm(j)$ is updated, and the following cases are addressed depending on the nature of node $j$:

1. $j$ represents a machine: if the current path from $r$ to $j$ is disjoint, and if $nm(j) = |K|$, then a $K$-disjoint path has been found ($K - SPT$ stops);

2. $j$ represents a job: if the current path from $r$ to $j$ is disjoint, then $j$ is inserted to $Q$;

3. $j$ represents a job, but the current path from $r$ to $j$ is not disjoint: if $j$ is internal to the path from $r$ to $i$, and if $nm(i) - nm(pred(j)) = |K|$, then a $K$-disjoint cycle has been found ($K - SPT$ stops).

Note that, compared with the standard label correcting approach, $K - SPT$ requires an extra $O(m)$ time for each node selection from $Q$, in order to check the disjointness of the path (a disjoint path may involve $m$ nodes at most).

**A bottleneck heuristic** The second heuristic, named $K-BPT$, is based on the cost definition (2), and it is a modification of the algorithm which looks for a bottleneck path tree in a directed graph, where a path is said to be bottleneck if it minimizes the maximum arc cost. It is well–known that the bottleneck path tree paradigm can be derived from the shortest path label correcting framework simply by substituting condition (3) with

$$d(j) > max\{d(i), c_{ij}\} \tag{4}$$

and, in case (4) holds, by modifying the label accordingly, i.e., $d(j) := max\{d(i), c_{ij}\}$. Thus, once selected a candidate node $i$ from the set $Q$, $K-BPT$ performs a typical iteration which coincides with the one described for $K-SPT$, except for the different check of the optimality condition and for the different update of the node labels.

Observe that aim of $K-BPT$ is to heuristically find a $K$-disjoint cycle or path which allows one the maximum reduction of the makespan: this is due to the cost definition (2), and to the use of the bottleneck path tree paradigm. On the other hand, $K-SPT$ looks for a "good" makespan reduction using, as a guide for its search, the label correcting paradigm.

Note also that $K-SPT$ and $K-BPT$ are actually algorithmic paradigms: several heuristics can in fact be derived from them by suitably implementing the set of the candidate nodes $Q$.

### 2.3 Heuristic approaches for the computation of disjoint cycles including a loaded machine

We have developed some heuristics for computing a disjoint cycle (path) in $G(S)$ which includes at least one loaded machine. Our aim in designing these heuristics was twofold: on one hand, we wanted to extend the improvement multi-exchange definition, by considering disjoint cycles which do not necessarily reduce the makespan, but at least decrease the number of the loaded machines; on the other hand, we wanted to reduce the average cost per iteration, by allowing the updating of the current solution $S$ as soon as a "short" disjoint cycle is discovered that improves $S$ in the sense outlined before.

The heuristics derive directly from $K-SPT$ and from $K-BPT$. The first one, referred to as $1-SPT$, performs the typical $K-SPT$ iteration without using the labels $nm(j)$, and stopping whenever a disjoint cycle or path of negative cost is found. Such a stop condition is justified by the property that, due to the cost definition (1), a (disjoint) cycle or path has a negative cost if and only if it includes at least one loaded machine. On the contrary, due to the different cost definition, the second heuristic, called $1-BPT$, maintains more similarity with respect to $K-BPT$. In fact, it still relies on the label $nm(j)$, for each node $j$, to calculate the number of the loaded machines in the paths and cycles it explores: the only difference is that it stops whenever this number is at least one, rather than exactly $|K|$.

We have also developed slightly different versions of $1-SPT$ and $1-BPT$, which exploit some structural properties of the improvement graph associated with the instances of $P||C_{max}$. In particular, we have implemented the heuristics $1D-SPT$ and $1D-BPT$, which are very similar to $1-SPT$ and to $1-BPT$, respectively, but for the fact that they explore a subgraph of the improvement graph $G(S)$, according to the following "monotone" property: if a disjoint cycle (path) exists which includes at least one loaded machine, then a disjoint cycle (path) exists which, with the exception of the nodes belonging to loaded machines, passes through nodes characterized by decreasing processing times. More formally:

**Property 2.3** *If a disjoint cycle exists in $G(S)$ which includes a loaded machine, then $G(S)$ contains a disjoint cycle $(j_1, j_2, \ldots, j_r, j_1)$ such that $j_1$ belongs to a loaded machine, and $p_{j_2} > p_{j_3} > \ldots > p_{j_r}$.*

***Proof:*** Assume that $G(S)$ contains a disjoint cycle, say $C$, which passes through a loaded machine, and rename the nodes in $C$ in such a way that $C = j_1, j_2, \ldots, j_r, j_1$, with job $j_1$ belonging to a loaded machine. We will prove that, if $C$ does not satisfy the "monotone" property, then $C$ can be converted into an alternative disjoint cycle, which satisfies the required relation among the processing times. The proof is by induction on the cardinality of $C$.

First, note that the property trivially holds true if $C$ is composed by only two arcs, i.e., $r = 2$. Therefore, consider the case in which $C$ is composed by at least three arcs. Assume that $C$ does not satisfy the "monotone" property, and consider the first arc of $C$, starting from $j_1$, which violates the property. Let it be $(j_{h-1}, j_h)$, for $3 \leq h \leq r$. By the assumption, it is $p_{j_{h-1}} \leq p_{j_h}$. Consider the successive arc along $C$, i.e., $(j_h, j_{h+1})$ where, with a little abuse of the notation, we set $j_{h+1} = j_1$ if $h = r$. Two situations are possible: either $(j_h, j_{h+1})$ is another arc violating the property, i.e., $p_{j_h} \leq p_{j_{h+1}}$, or not.

In the first case, since $p_{j_{h-1}} \leq p_{j_h}$, then we have $p_{j_{h-1}} \leq p_{j_{h+1}}$ and therefore $(j_{h-1}, j_{h+1})$ is an arc of $G(S)$, too: it is thus possible to construct an alternative disjoint cycle, containing the arc $(j_{h-1}, j_{h+1})$ instead of the pair of arcs $(j_{h-1}, j_h)$ and $(j_h, j_{h+1})$. This new disjoint cycle contains one arc less than $C$, and the proof follows by the induction hypothesis.

In the second case, we have $p_{j_h} > p_{j_{h+1}}$ while $p_{j_{h-1}} \leq p_{j_h}$; since $(j_h, j_{h+1}) \in A(S)$, then $(j_{h-1}, j_{h+1}) \in A(S)$ too (if it is feasible to assign job $j_h$ to $M(j_{h+1})$, then it is feasible to assign it job $j_{h-1}$, whose processing time is smaller than or equal to the one of $j_h$). Hence, once again we can subtitute the pair of arcs $(j_{h-1}, j_h)$ and $(j_h, j_{h+1})$ by $(j_{h-1}, j_{h+1})$, obtaining an alternative shorter disjoint cycle, and the thesis follows. $\diamond$

Observe that the decreasing ordering among the jobs, stated by Property 2.3, does not hold for the last and the first job along the cycle. Indeed, it cannot hold. In fact, by definition of the improvement graph, the arc $(j_r, j_1)$, where $M(j_1)$ is a loaded machine, exists if and only if $p_{j_r} < p_{j_1}$, that is if that move guarantees a completion time less than $C_{max}(S)$ for the machine containing $j_1$. This observation holds true for all the arcs entering loaded machines, therefore:

**Corollary 2.1** *If a disjoint cycle exists in $G(S)$ which includes a loaded machine, then $G(S)$ contains a disjoint cycle $(j_1, j_2, \ldots, j_r, j_1)$ such that $j_1$ belongs to a loaded machine, and such that the jobs from $j_2$ to $j_r$ belong to unloaded machines.*

Similar properties can be stated in the case of disjoint paths including a loaded machine.

Property 2.3 suggests to search for disjoint cycles and paths, containing at least one loaded machine, in the subgraph of $G(S)$ obtained by removing all the arcs $(j_h, j_k)$ such that $M(j_k)$ is unloaded and $p_{j_h} \leq p_{j_k}$. Moreover, if the visit of the improvement graph is started from a root node $r$ which belongs to a loaded machine, then Corollary 2.1 suggests to discard all the nodes of $G(S)$ corresponding to loaded machines.

This "pruning" of the improvement graph, implemented in $1D - SPT$ and $1D - BPT$, has proved to be efficient in practice: for both heuristics, the average running time per iteration proved to be substantially smaller than the one of $1 - SPT$ and $1 - BPT$. Unfortunately, the average reduction of the makespan per iteration was not so effective, maybe due to the restricted portion of the improvement graph visited at each step with respect to $1 - SPT$ and $1 - BPT$.

A final observation. Indeed, Property 2.3 could be stated in the following, slightly more general, form: if the improvement graph $G(S)$ contains a cycle $C$, then it contains a cycle, $C'$, which includes a subset of the nodes of $C$, and which satisfies the "monotone" property. In particular, if $C$ is a disjoint cycle, then $C'$ is disjoint, too. Such a more general formulation could be useful to enlarge the neighborhood definition, e.g., by including cycles or paths which are not disjoint.

# 3 Implementation issues

Several local search algorithms based on the multi-exchange neighborhood structure have been designed and implemented. Here we will report about the most successful variants, shortly discussing those that have not led to interesting results.

The main implementation issue was the choice of the heuristic for computing a disjoint cycle or path at each algorithm iteration (i.e., the choice among $K - SPT$, $K - BPT$, $1 - SPT$, $1 - BPT$,

$1D-SPT$ and $1D-BPT$). In general, our computational investigation did not show any dominance of one heuristic with respect to the others. Exception is the case of $1D-SPT$ and $1D-BPT$: these heuristics proved to be, essentially, pairwise equivalent to $1-SPT$ and to $1-BPT$, respectively, in that they tend to obtain analogous results with comparable computational effort (more iterations performed in less time). Therefore, we will not report results for local search algorithms using $1D-SPT$ and $1D-BPT$.

Other important implementation aspects concern the choice of the initial solution, and the strategies used to efficiently implement the disjoint cycle (path) detection heuristics. All these issues will be discussed in the following.

## 3.1 Starting solution and restarts

A crucial decision has been the selection of the initial solution, i.e., the initial assignment of the jobs to the machines. In a preliminary experimentation (Necciari, 1999), we ran the local search algorithms starting from different solutions, obtained via standard constructive procedures for $P||C_{max}$; among the others, we considered the longest processing time algorithm ($LPT$) and the multifit algorithm ($MF$). In general, $LPT$ showed to be the best choice for generating the starting solution, and therefore it has been chosen to provide the initial job assignment.

To avoid early stopping, always possible within a local search approach, we also implemented a restart procedure, which uses a randomized version of $LPT$ to obtain alternative starting solutions if necessary. The standard $LPT$ examines the jobs according to a non increasing ordering of their processing times, and, at each step, it assigns the currently examined job to the least loaded machine; on the other hand, the randomized version uses a circular list, and it assigns the currently examined job to the least loaded machine with probability 0.5, moving to the next job in the list otherwise, until the list is empty.

In the local search algorithms, the restart device has been implemented by suitably setting two parameters, $MxIter$ and $RstIter$: while the code is limited to perform at most $MxIter$ iterations (i.e., selection of a starting node $r$ in the current improvement graph, and application of the cycle (path) detection heuristic), a restart is performed, anyway, each $RstIter$ iterations at most.

Our preliminary investigation showed that it is preferrable to allow for a small number of restarts in order to compute a good solution. In particular, we discovered that a reasonable setting for $RstIter$ is 50 when $MxIter = 200$: that means that it is generally convenient to allow for about four restarts. These are the values of the parameters that were used in our computational experiments.

## 3.2 Implementation details for disjoint cycle computation

### The root node selection

All the heuristics described in Section 2 assume that some node $r$, which is a job node, is selected as the starting point of the exploration; clearly, different choices of $r$ may lead to different results. Our local search algorithms mantain a list of nodes that have not been used yet as the starting point for the improvement of the current solution $S$, and select one node at random from that list: if one disjoint cycle (path) is found starting from that node, then $S$ is updated and the list is re-initialized; otherwise, a new node is selected from the list. If the list becomes empty, then no improvement move has been found, and the algorithms stop unless an alternative solution $S$ is generated by the restart device previously described.

A number of variants of such a root node selection strategy can be imagined, where some kind of priority is assigned to the nodes in order to select first the most "promising" ones. One natural priority might be to consider the status of the machines; for example, as suggested by Corollary 2.1, one might choose, as a starting node, one job belonging to a loaded machine. Quite surprisingly, in our experience the converse priority proved to be preferrable: our investigation has shown in fact that better results are usually obtained by giving preference to the nodes which are currently assigned to unloaded machines. Thus, the final choice for all the local search algorithms has been to always try to start from nodes belonging to unloaded machines, and resort to nodes belonging to loaded machines only if the first kind of node selection fails.

### Implementation of $Q$

All the disjoint cycle and path detection algorithms use a set of candidate nodes $Q$; the implementation of $Q$, i.e., the rule used for choosing the next node to be extracted from $Q$, typically conditions the kind of cycles (paths) that are explored by the algorithms, and therefore the overall effectiveness of the heuristics. In our preliminary investigation (Necciari, 1999), we have found that the best results are usually obtained by implementing the set of the candidate nodes $Q$ as a queue, i.e., according to a first in-first out policy. Hence, visiting the graph in a breadth-first way seems to be usually the best choice: this may be due to the fact that "short" disjoint cycles, i.e., cycles formed by a low number of arcs, are more likely to be generated by this kind of visit.

### Implicit construction of $G(S)$

If the improvement graph $G(S)$ is "explicitely" constructed, i.e., if data structures implementing the forward star of each node are actually built up, then, when the current solution $S$ changes, these data structures have to be rebuilt essentially from scratch. An alternative is to construct $G(S)$ in an implicit way, i.e., to check for the existence of some arc $(i,j) \in A(S)$ only when, during the visit procedure, the forward star of $i$ has to be scanned. Both choices have advantages and drawbacks. In particular, constructing explicitely the graph allows one to perform the existency test only once for each arc, but it requires more memory and it generates the entire $G(S)$ at each iteration, even parts of the graph which will be never reached by the visit procedure. Indeed, dynamically cheching for the existence of the arcs has shown to be better than constructing the whole improvement graph, especially for the heuristics $1-SPT$ and to $1-BPT$, which typically perform more improvement moves than the other heuristics. Such a strategy has therefore been adopted in our implementation.

### "Dynamic pruning" of $G(S)$

Consider an arc $(i,j)$ in $G(S)$, where $M(j) = M_h$ for $1 \leq h \leq m$. The definition of $G(S)$ states that such an arc $(i,j)$ exists only if

$$C_h(S) - p_j + p_i < C_{max}(S),$$

i.e., if $M(j)$ is an unloaded machine when job $j$ is removed from $M(j)$ and job $i$ is assigned to it. This definition ensures that, if an improvement disjoint cycle (path) is found, then the makespan of the new solution (or the completion time of at least one loaded machine) reduces by at least one unit, being all the data integer valued. Clearly, if the following, stronger condition

$$C_h(S) - p_j + p_i \leq C_{max}(S) - \epsilon$$

would be imposed, for some fixed $\epsilon \geq 1$, then it would be possible to ensure that each improvement disjoint cycle or path reduces the makespan (or some completion time) by at least $\epsilon$.

This stronger requirement has been implemented in our codes, where $\epsilon$ is set to the value $(C_{max}(S) - LB)/4$ whenever a new solution $S$ is generated ($LB$, defined as $\sum_{i=1,\ldots,n} p_i/m$, is a lower bound to the optimum makespan), and it is reduced by a factor of 4 each time the list of the candidate root nodes becomes empty. Thanks to this strategy, all the heuristics visiting $G(S)$ tend to "concentrate" on cycles or paths which result in a "substantial" decrease of the objective function, at least in the initial stages where a larger improvement is has to be acheived; moreover, as a side-effect, a "dynamic pruning" of $G(S)$ can be realized, since a limited number of arcs, i.e., the ones allowing a completion time reduction of at least $\epsilon$, is considered during the implicit construction of $G(S)$.

# 4 Computational results

Four local search algorithms have been selected for our computational experimentation, which differ essentially in the heuristic used to compute a disjoint cycle or path at each algorithm iteration. Our preliminary experimentation showed that these four algorithms are quite representative of the kind of results that can be expected within the multi-exchange approach, even though there is room for adaptation to particular needs. For instance, the variants $1D - SPT$ and $1D - BPT$, which are not included in our experimentation plan, could be useful in the case in which time is the major issue; in fact, they generally provide, with the same maximum number of iterations, slightly less accurate solutions but in a small amount of time.

All the algorithms have been implemented as discussed in the previous paragraph. For the sake of the simplicity, they will be referred to by the same name of the heuristic used to explore the neighborhood, i.e., $K - SPT$, $K - BPT$, $1 - SPT$ and $1 - BPT$.

The algorithms have been coded in C++, compiled with the GNU egcs compiler (ver.2.91.66) with normal optimization options, and run on a PC with a Pentium II/400 processor and 128Mb RAM under the RedHat GNU/Linux 6.0 operating system.

## 4.1 The instances

The algorithms were tested on two different families of instances; in each family, 10 problems were randomly generated for each choice of the number of machines $m$ in $\{5, 10, 25\}$, of the number of jobs $n$ in $\{50, 100, 500, 1000\}$ (for $m = 5$, $n = 10$ is also tested), and by choosing the interval for the (integer) processing times in $\{[1, 100], [1, 1000], [1, 10000]\}$, for a total of 390 problems within each family.

The generator for the first family has been developed by F.M. Muller (França et al., 1994); the processing time of the $i$-th job is chosen at random, with uniform distribution, in the selected time interval.

The generator for the second family of instances, which has been developed for this computational experiment, has been designed with the aim of obtaining "difficult" instances. The main motivation is that, according to the computational results reported in [França et al., 1994], $LPT$ alone seems to produce very good solutions in the case of uniformly distributed processing times; thus, this kind of instances seems to be easily solvable. The idea was then to produce instances for which $LPT$ is not so good; in our preliminary computational experience (Necciari, 1999), we have discovered that instances caractherized by "highly non-uniform" processing times have this property. Thus, given the interval $[a, b]$ of the possible processing times, our generator produces instances where the 98% of the processing times is uniformly distributed in the interval $[(b - a) * 0.9, b]$, whereas the remaining processing times fall into the interval $[a, (b - a) * 0.02]$.

Both generators are available at the URL

$$http://www.di.unipi.it/di/groups/optimize/Data/index.html$$

together with the scripts used for producing the instances.

## 4.2 Plan of the experimentation and computational results

We performed a wide computational experimentation, aimed to study the practical efficiency of the multi-exchange approach from one hand, and to investigate about its capability of discovering accurate solutions when the running time is not an issue. Moreover, experiments were performed to obtain some insight about the improvement capacity related to the special neighborhood structure under investigation.

As far as the practical efficiency is concerned, we compared the multi-exchange algorithms, on the two families of benchmark instances previously introduced, with the improvement heuristic in [França et al., 1994], referred to as $FMHEU$. As reported by the authors, $FMHEU$ proved to be very efficient on the instances of the first family, i.e., in the case of uniformly distributed processing times.

The results are shown in Table 1 to Table 6 below, which report both the CPU time required by each algorithm running (in seconds), and the obtained relative error, computed by estimating the optimum makespan via the lower bound $LB = \sum_{i=1,\dots,n} p_i/m$. In the tables, each entry is the average over ten instances, which differ only in the generator seed. The tables also report the running time and the relative error of $LPT$, in order to provide some insight about the improvement capacity of the multi-exchange approach (recall that all the algorithms start from the solution provided by $LPT$).

| | | **FMHEU** | | **LPT** | | **K-SPT** | | **K-BPT** | | **1-SPT** | | **1-BPT** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **m** | **n** | *gap* | **sec.** | *gap* | **sec.** | *gap* | **sec.** | *gap* | **sec.** | *gap* | **sec.** | *gap* | **sec.** |
| 5 | 10 | 7.52e-03 | 0.00 | 7.52e-03 | 0.00 | 7.52e-03 | 0.00 | 7.52e-03 | 0.00 | 7.52e-03 | 0.00 | 7.52e-03 | 0.00 |
| 5 | 50 | 1.54e-02 | 0.00 | 1.69e-02 | 0.00 | 8.80e-03 | 0.02 | 9.00e-03 | 0.02 | 8.25e-03 | 0.02 | 8.04e-03 | 0.02 |
| 5 | 100 | 6.05e-03 | 0.00 | 8.28e-03 | 0.00 | 2.12e-04 | 0.03 | 1.11e-03 | 0.07 | 0 | 0.01 | 0 | 0.01 |
| 5 | 500 | 2.03e-03 | 0.01 | 6.06e-04 | 0.00 | 0 | 0.26 | 0 | 0.38 | 0 | 0.04 | 0 | 0.06 |
| 5 | 1000 | 1.25e-03 | 0.01 | 1.38e-04 | 0.00 | 0 | 0.06 | 0 | 0.49 | 0 | 0.06 | 0 | 0.07 |
| 10 | 50 | 1.42e-02 | 0.01 | 1.76e-02 | 0.00 | 1.61e-02 | 0.03 | 1.65e-02 | 0.03 | 1.56e-02 | 0.03 | 1.56e-02 | 0.02 |
| 10 | 100 | 7.31e-03 | 0.01 | 8.48e-03 | 0.00 | 6.25e-03 | 0.15 | 6.04e-03 | 0.15 | 4.34e-03 | 0.08 | 4.24e-03 | 0.07 |
| 10 | 500 | 6.63e-03 | 0.01 | 8.22e-03 | 0.00 | 1.89e-03 | 2.00 | 8.22e-03 | 3.07 | 1.72e-03 | 0.63 | 1.72e-03 | 0.91 |
| 10 | 1000 | 2.74e-03 | 0.01 | 7.86e-04 | 0.00 | 6.36e-05 | 13.70 | 5.30e-05 | 9.84 | 0 | 0.32 | 0 | 0.41 |
| 25 | 50 | 5.28e-03 | 0.00 | 5.28e-03 | 0.00 | 5.28e-03 | 0.09 | 5.28e-03 | 0.05 | 5.28e-03 | 0.02 | 5.28e-03 | 0.02 |
| 25 | 100 | 7.44e-03 | 0.00 | 9.30e-03 | 0.00 | 9.30e-03 | 0.43 | 9.30e-03 | 0.25 | 7.97e-03 | 0.09 | 8.24e-03 | 0.09 |
| 25 | 500 | 6.43e-03 | 0.01 | 8.13e-03 | 0.00 | 3.66e-03 | 13.50 | 8.13e-03 | 5.06 | 0 | 1.06 | 1.11e-03 | 1.38 |
| 25 | 1000 | 6.16e-03 | 0.01 | 8.31e-03 | 0.00 | 5.04e-04 | 44.90 | 8.31e-03 | 14.60 | 0 | 4.09 | 1.25e-03 | 9.89 |

Table 1: Results for "non-uniform" instances with processing times in $[1, 100]$

| | | **FMHEU** | | **LPT** | | **K-SPT** | | **K-BPT** | | **1-SPT** | | **1-BPT** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **m** | **n** | *gap* | **sec.** | *gap* | **sec.** | *gap* | **sec.** | *gap* | **sec.** | *gap* | **sec.** | *gap* | **sec.** |
| 5 | 10 | 6.13e-03 | 0.01 | 6.13e-03 | 0.00 | 6.13e-03 | 0.00 | 6.13e-03 | 0.00 | 6.13e-03 | 0.00 | 6.13e-03 | 0.00 |
| 5 | 50 | 1.59e-02 | 0.01 | 1.75e-02 | 0.00 | 1.01e-02 | 0.02 | 1.02e-02 | 0.01 | 9.89e-03 | 0.02 | 9.87e-03 | 0.02 |
| 5 | 100 | 6.52e-03 | 0.01 | 8.48e-03 | 0.00 | 3.71e-05 | 0.03 | 4.78e-05 | 0.05 | 3.71e-05 | 0.03 | 4.24e-05 | 0.04 |
| 5 | 500 | 0 | 0.01 | 6.74e-04 | 0.00 | 4.24e-06 | 0.91 | 0 | 0.42 | 0 | 0.15 | 0 | 0.21 |
| 5 | 1000 | 1.88e-04 | 0.01 | 1.39e-04 | 0.00 | 3.18e-06 | 3.19 | 0 | 0.51 | 0 | 0.26 | 0 | 0.46 |
| 10 | 50 | 1.42e-02 | 0.01 | 1.82e-02 | 0.00 | 1.61e-02 | 0.03 | 1.63e-02 | 0.02 | 1.64e-02 | 0.03 | 1.62e-02 | 0.02 |
| 10 | 100 | 7.18e-03 | 0.01 | 8.87e-03 | 0.00 | 5.14e-03 | 0.10 | 5.26e-03 | 0.10 | 5.16e-03 | 0.09 | 5.26e-03 | 0.08 |
| 10 | 500 | 6.46e-03 | 0.01 | 8.38e-03 | 0.00 | 6.75e-03 | 2.14 | 8.37e-03 | 3.40 | 5.11e-03 | 1.66 | 5.12e-03 | 3.18 |
| 10 | 1000 | 4.53e-04 | 0.01 | 8.60e-04 | 0.00 | 5.74e-05 | 10.30 | 3.18e-06 | 6.63 | 0 | 1.50 | 0 | 1.43 |
| 25 | 50 | 4.03e-03 | 0.01 | 4.03e-03 | 0.00 | 4.03e-03 | 0.03 | 4.03e-03 | 0.03 | 4.03e-03 | 0.02 | 4.03e-03 | 0.03 |
| 25 | 100 | 7.50e-03 | 0.00 | 9.57e-03 | 0.00 | 8.88e-03 | 0.15 | 8.72e-03 | 0.14 | 8.71e-03 | 0.10 | 8.53e-03 | 0.09 |
| 25 | 500 | 6.59e-03 | 0.01 | 8.32e-03 | 0.00 | 8.49e-04 | 4.11 | 3.63e-03 | 3.23 | 8.12e-04 | 1.74 | 2.34e-03 | 2.67 |
| 25 | 1000 | 6.18e-03 | 0.01 | 8.53e-03 | 0.00 | 4.03e-04 | 16.20 | 1.54e-03 | 15.70 | 7.97e-06 | 8.68 | 1.50e-03 | 13.50 |

Table 2: Results for "non-uniform" instances with processing times in $[1, 1000]$

In the second part of the experimentation, we tested the effectiveness of the proposed local search algorithms, i.e., their capability of providing accurate solutions when the running time is not an issue. To do that, we run the algorithms $K - SPT$ and $1 - SPT$ by setting $MxIter = 2 * n * 100$ and $RstIter = 2 * n$, thus allowing around 100 restarts and leaving some room for exploring the neighborhood of each solution. Anyway, we limited the maximum running time for each instance to 5 minutes. The results are shown in Table 7 and in Table 8 below: it appears that the algorithms

| m | n | FMHEU gap | FMHEU sec. | LPT gap | LPT sec. | K-SPT gap | K-SPT sec. | K-BPT gap | K-BPT sec. | 1-SPT gap | 1-SPT sec. | 1-BPT gap | 1-BPT sec. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 6.18e-03 | 0.01 | 6.18e-03 | 0.00 | 6.18e-03 | 0.00 | 6.18e-03 | 0.00 | 6.18e-03 | 0.00 | 6.18e-03 | 0.00 |
| 5 | 50 | 1.58e-02 | 0.01 | 1.76e-02 | 0.00 | 1.03e-02 | 0.02 | 1.03e-02 | 0.02 | 1.03e-02 | 0.03 | 1.02e-02 | 0.01 |
| 5 | 100 | 6.64e-03 | 0.00 | 8.51e-03 | 0.00 | 4.87e-05 | 0.04 | 4.55e-05 | 0.05 | 5.57e-05 | 0.06 | 6.94e-05 | 0.07 |
| 5 | 500 | 0 | 0.01 | 6.78e-04 | 0.00 | 1.27e-06 | 0.78 | 0 | 1.01 | 9.54e-07 | 1.01 | 4.24e-07 | 1.15 |
| 5 | 1000 | 0 | 0.01 | 1.39e-04 | 0.00 | 1.59e-07 | 1.76 | 5.31e-08 | 1.75 | 2.12e-07 | 2.80 | 5.31e-08 | 1.50 |
| 10 | 50 | 1.42e-02 | 0.01 | 1.82e-02 | 0.00 | 1.63e-02 | 0.03 | 1.61e-02 | 0.02 | 1.65e-02 | 0.03 | 1.60e-02 | 0.03 |
| 10 | 100 | 7.25e-03 | 0.00 | 8.92e-03 | 0.00 | 5.24e-03 | 0.09 | 5.24e-03 | 0.08 | 5.24e-03 | 0.10 | 5.23e-03 | 0.08 |
| 10 | 500 | 6.55e-03 | 0.01 | 8.40e-03 | 0.00 | 6.78e-03 | 1.99 | 7.59e-03 | 3.46 | 6.77e-03 | 1.98 | 6.80e-03 | 3.51 |
| 10 | 1000 | 0 | 0.01 | 8.69e-04 | 0.00 | 2.01e-05 | 2.63 | 1.80e-06 | 7.22 | 7.43e-07 | 2.42 | 1.06e-06 | 7.00 |
| 25 | 50 | 4.09e-03 | 0.01 | 4.09e-03 | 0.00 | 4.09e-03 | 0.01 | 4.09e-03 | 0.03 | 4.09e-03 | 0.02 | 4.09e-03 | 0.03 |
| 25 | 100 | 7.58e-03 | 0.01 | 9.72e-03 | 0.00 | 8.83e-03 | 0.11 | 8.72e-03 | 0.10 | 9.02e-03 | 0.10 | 8.82e-03 | 0.10 |
| 25 | 500 | 6.58e-03 | 0.01 | 8.36e-03 | 0.00 | 3.86e-04 | 1.43 | 1.14e-03 | 2.74 | 1.52e-04 | 1.51 | 1.97e-03 | 2.76 |
| 25 | 1000 | 6.20e-03 | 0.01 | 8.54e-03 | 0.00 | 1.88e-04 | 6.53 | 1.32e-04 | 14.40 | 2.76e-05 | 5.78 | 8.65e-04 | 14.40 |

Table 3: Results for "non-uniform" instances with processing times in $[1, 10000]$

| m | n | FMHEU gap | FMHEU sec. | LPT gap | LPT sec. | K-SPT gap | K-SPT sec. | K-BPT gap | K-BPT sec. | 1-SPT gap | 1-SPT sec. | 1-BPT gap | 1-BPT sec. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 3.26e-02 | 0.00 | 3.76e-02 | 0.00 | 3.26e-02 | 0.00 | 3.26e-02 | 0.00 | 3.26e-02 | 0.00 | 3.26e-02 | 0.00 |
| 5 | 50 | 1.96e-04 | 0.01 | 4.23e-03 | 0.00 | 1.96e-04 | 0.01 | 3.92e-04 | 0.01 | 0 | 0.00 | 0 | 0.00 |
| 5 | 100 | 0 | 0.01 | 1.46e-03 | 0.00 | 0 | 0.00 | 0 | 0.01 | 0 | 0.00 | 0 | 0.00 |
| 5 | 500 | 0 | 0.01 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | 1000 | 0 | 0.01 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 10 | 50 | 2.35e-03 | 0.00 | 1.60e-02 | 0.00 | 1.86e-03 | 0.02 | 3.80e-03 | 0.01 | 7.41e-04 | 0.01 | 1.51e-03 | 0.01 |
| 10 | 100 | 0 | 0.01 | 4.93e-03 | 0.00 | 5.97e-04 | 0.05 | 9.60e-04 | 0.05 | 0 | 0.00 | 0 | 0.01 |
| 10 | 500 | 0 | 0.00 | 4.10e-05 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 10 | 1000 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 25 | 50 | 3.42e-02 | 0.01 | 3.35e-02 | 0.00 | 3.35e-02 | 0.04 | 3.35e-02 | 0.04 | 3.35e-02 | 0.03 | 3.35e-02 | 0.03 |
| 25 | 100 | 3.62e-03 | 0.01 | 2.52e-02 | 0.00 | 1.15e-02 | 0.24 | 1.56e-02 | 0.16 | 4.41e-03 | 0.06 | 6.02e-03 | 0.07 |
| 25 | 500 | 0 | 0.01 | 5.04e-04 | 0.00 | 4.07e-04 | 2.61 | 5.04e-04 | 1.84 | 0 | 0.07 | 1.03e-04 | 0.23 |
| 25 | 1000 | 0 | 0.01 | 1.01e-04 | 0.00 | 0 | 0.02 | 0 | 0.03 | 0 | 0.01 | 0 | 0.03 |

Table 4: Results for "uniform" instances with processing times in $[1, 100]$

are capable of finding very good solutions, most often provably optimal ones, in reasonable running times, especially in the case of the variant $1-SPT$.

Finally, we focused on the improvement capacity of our algorithms. By considering both kinds of instances, we run the algorithms $K-SPT$ and $1-SPT$ without providing an upper bound to the maximum number of iterations, and without performing restarts (but keeping the maximum time limit of 5 minutes); therefore, the obtained gaps are only due to the "quality" of the proposed neighborhood. The results are shown in Table 9 and in Table 10 below. Gaps are obtained which are comparable with those of the heuristic $FMHEU$ on the "uniform" instances, and better on the "non-uniform" ones: this seems to indicate that the multi-exchange neighborhood is actually capable of providing better solutions than the two-exchange neighborhood used by $FMHEU$. In particular, the variant $1-SPT$ obtains better results than $K-SPT$, probably thanks to the larger set of improvement moves taken into consideration.

By the comparison of the results in Tables 9 and 10 with those in Tables 7 and 8, it is possible to observe that, in general, allowing restarts is better than insisting on the initial solution, i.e., the one provided by $LPT$.

## 4.3 Conclusions

The computational results show that the multi-exchange approach can be successfully applied to the minimum makespan machine scheduling problem. When only a limited number of iterations is allowed, the multi-exchange local search algorithms obtain, on the "non-uniform" instances, solutions that are generally more accurate than the ones provided by the improvement heuristic $FMHEU$ in a

| m | n | FMHEU gap | FMHEU sec. | LPT gap | LPT sec. | K-SPT gap | K-SPT sec. | K-BPT gap | K-BPT sec. | 1-SPT gap | 1-SPT sec. | 1-BPT gap | 1-BPT sec. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 4.03e-02 | 0.01 | 4.03e-02 | 0.00 | 3.86e-02 | 0.00 | 3.86e-02 | 0.00 | 4.03e-02 | 0.00 | 4.03e-02 | 0.00 |
| 5 | 50 | 2.48e-04 | 0.01 | 4.26e-03 | 0.00 | 2.28e-04 | 0.02 | 3.62e-04 | 0.01 | 5.09e-04 | 0.03 | 4.93e-04 | 0.02 |
| 5 | 100 | 6.03e-05 | 0.00 | 1.22e-03 | 0.00 | 7.16e-05 | 0.04 | 9.10e-05 | 0.05 | 9.92e-05 | 0.06 | 1.32e-04 | 0.05 |
| 5 | 500 | 0 | 0.01 | 2.05e-05 | 0.00 | 0 | 0.04 | 2.06e-06 | 0.17 | 0 | 0.05 | 2.06e-06 | 0.18 |
| 5 | 1000 | 0 | 0.01 | 5.99e-06 | 0.00 | 0 | 0.01 | 0 | 0.11 | 0 | 0.11 | 0 | 0.12 |
| 10 | 50 | 4.43e-03 | 0.00 | 2.80e-02 | 0.00 | 4.30e-03 | 0.03 | 5.00e-03 | 0.02 | 4.78e-03 | 0.03 | 4.94e-03 | 0.03 |
| 10 | 100 | 3.43e-04 | 0.01 | 4.76e-03 | 0.00 | 7.27e-04 | 0.06 | 1.18e-03 | 0.07 | 1.23e-03 | 0.08 | 1.05e-03 | 0.08 |
| 10 | 500 | 0 | 0.01 | 1.61e-04 | 0.00 | 2.87e-05 | 0.51 | 3.68e-05 | 1.38 | 1.56e-05 | 0.52 | 4.19e-06 | 0.31 |
| 10 | 1000 | 0 | 0.01 | 4.01e-05 | 0.00 | 1.02e-05 | 1.54 | 1.22e-05 | 2.83 | 2.01e-06 | 0.87 | 2.01e-06 | 0.78 |
| 25 | 50 | 2.33e-02 | 0.01 | 2.46e-02 | 0.00 | 1.99e-02 | 0.02 | 1.99e-02 | 0.01 | 2.12e-02 | 0.02 | 1.99e-02 | 0.02 |
| 25 | 100 | 4.04e-03 | 0.01 | 2.24e-02 | 0.00 | 8.82e-03 | 0.08 | 1.12e-02 | 0.09 | 8.69e-03 | 0.08 | 1.06e-02 | 0.09 |
| 25 | 500 | 0 | 0.01 | 1.06e-03 | 0.00 | 3.41e-04 | 1.53 | 5.97e-04 | 2.33 | 1.40e-04 | 1.02 | 1.91e-04 | 2.17 |
| 25 | 1000 | 0 | 0.01 | 2.81e-04 | 0.00 | 7.53e-05 | 6.07 | 1.35e-04 | 9.24 | 2.02e-05 | 1.92 | 4.52e-05 | 5.07 |

Table 5: Results for "uniform" instances with processing times in $[1, 1000]$

| m | n | FMHEU gap | FMHEU sec. | LPT gap | LPT sec. | K-SPT gap | K-SPT sec. | K-BPT gap | K-BPT sec. | 1-SPT gap | 1-SPT sec. | 1-BPT gap | 1-BPT sec. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 6.78e-03 | 0.00 | 6.78e-03 | 0.00 | 6.78e-03 | 0.00 | 6.78e-03 | 0.00 | 6.78e-03 | 0.00 | 6.78e-03 | 0.00 |
| 5 | 50 | 5.44e-04 | 0.01 | 5.43e-03 | 0.00 | 2.60e-04 | 0.02 | 4.13e-04 | 0.02 | 6.79e-04 | 0.02 | 7.12e-04 | 0.02 |
| 5 | 100 | 7.78e-05 | 0.01 | 9.84e-04 | 0.00 | 4.19e-05 | 0.05 | 1.28e-04 | 0.05 | 1.34e-04 | 0.08 | 1.60e-04 | 0.06 |
| 5 | 500 | 3.96e-07 | 0.00 | 4.49e-05 | 0.00 | 3.96e-06 | 0.79 | 5.73e-06 | 1.48 | 7.72e-06 | 1.48 | 6.33e-06 | 1.49 |
| 5 | 1000 | 0 | 0.01 | 1.03e-05 | 0.00 | 1.40e-06 | 3.17 | 1.60e-06 | 4.69 | 1.50e-06 | 3.64 | 1.80e-06 | 5.73 |
| 10 | 50 | 4.81e-03 | 0.00 | 2.07e-02 | 0.00 | 2.84e-03 | 0.02 | 4.12e-03 | 0.02 | 5.45e-03 | 0.03 | 4.63e-03 | 0.02 |
| 10 | 100 | 2.48e-04 | 0.00 | 5.63e-03 | 0.00 | 5.77e-04 | 0.05 | 9.24e-04 | 0.08 | 1.19e-03 | 0.08 | 1.17e-03 | 0.07 |
| 10 | 500 | 2.38e-06 | 0.01 | 1.59e-04 | 0.00 | 2.57e-05 | 0.74 | 4.34e-05 | 1.58 | 3.72e-05 | 1.26 | 3.70e-05 | 1.65 |
| 10 | 1000 | 0 | 0.01 | 4.81e-05 | 0.00 | 8.81e-06 | 2.59 | 1.10e-05 | 5.49 | 9.63e-06 | 5.31 | 1.18e-05 | 6.61 |
| 25 | 50 | 2.08e-02 | 0.01 | 2.72e-02 | 0.00 | 2.19e-02 | 0.03 | 2.38e-02 | 0.03 | 2.42e-02 | 0.03 | 2.26e-02 | 0.03 |
| 25 | 100 | 4.33e-03 | 0.01 | 3.26e-02 | 0.00 | 6.67e-03 | 0.07 | 8.48e-03 | 0.08 | 1.05e-02 | 0.07 | 9.44e-03 | 0.09 |
| 25 | 500 | 2.28e-05 | 0.01 | 1.31e-03 | 0.00 | 2.06e-04 | 0.77 | 5.16e-04 | 2.00 | 2.74e-04 | 1.45 | 3.05e-04 | 2.12 |
| 25 | 1000 | 3.52e-06 | 0.01 | 3.55e-04 | 0.00 | 6.10e-05 | 3.14 | 1.23e-04 | 6.51 | 6.65e-05 | 4.31 | 7.95e-05 | 7.11 |

Table 6: Results for "uniform" instances with processing times in $[1, 10000]$

comparable running time, except for the largest instances where the running time is much larger (but still reasonable). The results for "uniform" instances are not so good, since the heuristic $FMHEU$ generally obtains better bounds; however, given time enough, our algorithms obtain more accurate solutions also in the "uniform" case, thereby showing that the multi-exchange neighborhood can be better than simples 2-exchange neighborhoods, as the one used in $FMHEU$.

Thus, according to the computational results presented in [Ahuja, Orlin and Sharma, 1998] for CMST, multi-exchange neighborhoods seem to be an interesting framework for developing local search approaches to hard combinatorial optimization problems.

# References


ANDERSON, J.E., C.A. GLASS AND C.N. POTTS.1997. Machine scheduling, in Local Search in Combinatorial Optimization. Aarts and Lenstra Eds., Wiley, 361-414.

AHUJA, R.K., T.L. MAGNANTI AND J.B. ORLIN. 1993. Network Flows: theory, algorithms and applications. Prentice Hall, New Jersey.

AHUJA, R.K., J.B. ORLIN AND D. SHARMA. 1998. New neighborhood search structures for the capacitated minimum spanning tree problem. submitted to Mathematical Programming.

COFFMAN Jr., E.G., M.R. GAREY AND D.S. JOHNSON. 1978. An application of bin packing to multiprocessor scheduling. SIAM Journal on Computing 7, 1-17.

FATEMI-GHOMI, S.M. AND F. JOLAI-GHAZVINI. 1998. A pairwise interchange algorithm for parallel machine scheduling. Production Planning and Control 9(7), 685-689.

| m | n | K-SPT [1,100] | 1-SPT [1,100] | K-SPT [1,1000] | 1-SPT [1,1000] | K-SPT [1,10000] | 1-SPT [1,10000] |
|---|---|---|---|---|---|---|---|
| 5 | 10 | 7.52e-03 | 7.52e-03 | 6.13e-03 | 6.13e-03 | 6.18e-03 | 6.18e-03 |
| 5 | 50 | 7.50e-03 | 7.50e-03 | 8.62e-03 | 8.63e-03 | 8.79e-03 | 8.78e-03 |
| 5 | 100 | 0 | 0 | 1.59e-05 | 1.59e-05 | 2.38e-05 | 2.38e-05 |
| 5 | 500 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1000 | 0 | 0 | 7.95e-06 | 0 | 0 | 0 |
| 10 | 50 | 1.35e-02 | 1.35e-02 | 1.44e-02 | 1.44e-02 | 1.45e-02 | 1.46e-02 |
| 10 | 100 | 4.13e-03 | 3.92e-03 | 4.51e-03 | 4.53e-03 | 4.61e-03 | 4.64e-03 |
| 10 | 500 | 2.12e-05 | 0 | 4.67e-05 | 8.35e-04 | 8.71e-04 | 8.49e-04 |
| 10 | 1000 | 4.24e-05 | 0 | 6.06e-05 | 0 | 7.42e-07 | 0 |
| 25 | 50 | 5.28e-03 | 5.28e-03 | 4.03e-03 | 4.03e-03 | 4.09e-03 | 4.09e-03 |
| 25 | 100 | 7.97e-03 | 7.44e-03 | 7.87e-03 | 7.90e-03 | 8.16e-03 | 8.11e-03 |
| 25 | 500 | 1.27e-03 | 0 | 2.02e-04 | 0 | 1.65e-05 | 0 |
| 25 | 1000 | 7.48e-03 | 0 | 4.36e-03 | 0 | 9.02e-06 | 0 |

Table 7: Best results on the "non-uniform" instances

| m | n | K-SPT [1,100] | 1-SPT [1,100] | K-SPT [1,1000] | 1-SPT [1,1000] | K-SPT [1,10000] | 1-SPT [1,10000] |
|---|---|---|---|---|---|---|---|
| 5 | 10 | 3.26e-02 | 3.26e-02 | 3.86e-02 | 3.86e-02 | 6.78e-03 | 6.78e-03 |
| 5 | 50 | 0 | 0 | 0 | 0 | 7.81e-05 | 9.48e-05 |
| 5 | 100 | 0 | 0 | 0 | 0 | 1.07e-05 | 7.73e-06 |
| 5 | 500 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1000 | 0 | 0 | 0 | 0 | 2.02e-07 | 0 |
| 10 | 50 | 1.15e-03 | 0 | 1.49e-03 | 1.18e-03 | 1.04e-03 | 1.17e-03 |
| 10 | 100 | 0 | 0 | 3.03e-04 | 6.05e-05 | 1.10e-04 | 1.39e-04 |
| 10 | 500 | 0 | 0 | 4.19e-06 | 0 | 5.22e-06 | 0 |
| 10 | 1000 | 0 | 0 | 8.16e-06 | 0 | 2.60e-06 | 0 |
| 25 | 50 | 3.35e-02 | 2.96e-02 | 1.65e-02 | 1.65e-02 | 1.89e-02 | 1.75e-02 |
| 25 | 100 | 8.47e-03 | 1.06e-03 | 3.54e-03 | 2.00e-03 | 1.84e-03 | 2.18e-03 |
| 25 | 500 | 4.07e-04 | 0 | 2.71e-04 | 0 | 5.49e-05 | 1.19e-05 |
| 25 | 1000 | 0 | 0 | 7.53e-05 | 0 | 2.81e-05 | 2.01e-06 |

Table 8: Best results on the "uniform" instances


FINN, G. AND E. HOROWITZ. 1979. A linear time approximation algorithm for multiprocessor scheduling. BIT 19, 312-320.

FRANÇA, P.M., M. GENDREAU, G. LAPORTE AND F.M. MULLER. 1994. A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective. Computers Ops. Res. 21(2), 205-210.

GAREY, M.R. AND D.S. JOHNSON. 1978. Strong NP-completeness results: motivation, examples and implications. J. Ass. Comp. Mach. 25, 499-508.

GRAHAM, R.L. 1966. Bounds for certain multiprocessing anomalies. Bell System Tech. J. 45, 1563-1581.

HÜBSCHER,R. AND F. GLOVER. 1994. Applying tabu search with influential diversification to multiprocessor scheduling. Computers Ops. Res. 21(8), 877-884.

LANGSTON, M.A. 1982. Improved 0/1 interchange scheduling. BIT 22, 282-290.

LAWLER, E.L., J.K. LENSTRA, A.H.G. RINNOY-KAN AND D.B. SHMOYS. 1993. Sequencing and Scheduling: Algorithms and Complexity, in Logistics of production and inventory, Handbooks in Operations Research and Management Science, Volume 4. Graves, Rinnoy Kan and Zipkin, Eds., Elsevier Science Publishers, 445-522.

NECCIARI, E. 1999. Algoritmi di ricerca locale basati su grafi di miglioramento per il problema di assegnamento di lavori a macchine. Master Thesis, Dipartimento di Informatica, Università di Pisa.

THOMPSON, P.M. AND J.B. ORLIN. 1989. Theory of cyclic transfers. Working paper, Operations Research Center, MIT, Cambridge, Mass.

THOMPSON, P.M. AND H.N. PSARAFTIS. 1993. Cyclic transfer algorithms for multivehicle routing and scheduling problems. Operations Research 41(5), 935-946.

| m | n | K-SPT [1,100] | 1-SPT [1,100] | K-SPT [1,1000] | 1-SPT [1,1000] | K-SPT [1,10000] | 1-SPT [1,10000] |
|---|---|---|---|---|---|---|---|
| 5 | 10 | 7.52e-03 | 7.52e-03 | 6.13e-03 | 6.13e-03 | 6.18e-03 | 6.18e-03 |
| 5 | 50 | 1.55e-02 | 7.61e-03 | 9.54e-03 | 8.63e-03 | 8.78e-03 | 8.73e-03 |
| 5 | 100 | 6.74e-03 | 0 | 3.43e-03 | 1.59e-05 | 4.40e-05 | 3.23e-05 |
| 5 | 500 | 7.43e-05 | 0 | 4.14e-05 | 0 | 7.02e-06 | 0 |
| 5 | 1000 | 1.06e-05 | 0 | 2.07e-05 | 0 | 3.18e-07 | 0 |
| 10 | 50 | 1.67e-02 | 1.40e-02 | 1.58e-02 | 1.42e-02 | 1.43e-02 | 1.42e-02 |
| 10 | 100 | 8.27e-03 | 3.92e-03 | 4.55e-03 | 4.42e-03 | 4.58e-03 | 4.50e-03 |
| 10 | 500 | 1.91e-04 | 0 | 9.34e-05 | 0 | 3.40e-06 | 2.13e-07 |
| 10 | 1000 | 1.59e-04 | 0 | 7.01e-05 | 0 | 1.38e-06 | 0 |
| 25 | 50 | 5.28e-03 | 5.28e-03 | 4.03e-03 | 4.03e-03 | 4.09e-03 | 4.09e-03 |
| 25 | 100 | 9.03e-03 | 7.44e-03 | 8.22e-03 | 7.50e-03 | 7.64e-03 | 7.56e-03 |
| 25 | 500 | 8.02e-03 | 0 | 6.64e-03 | 0 | 2.76e-05 | 0 |
| 25 | 1000 | 7.51e-03 | 0 | 4.37e-03 | 0 | 7.97e-06 | 0 |

Table 9: Best results on the “non-uniform” instances without restarts

| m | n | K-SPT [1,100] | 1-SPT [1,100] | K-SPT [1,1000] | 1-SPT [1,1000] | K-SPT [1,10000] | 1-SPT [1,10000] |
|---|---|---|---|---|---|---|---|
| 5 | 10 | 3.26e-02 | 3.26e-02 | 4.03e-02 | 4.03e-02 | 6.78e-03 | 6.78e-03 |
| 5 | 50 | 5.81e-04 | 1.96e-04 | 6.82e-04 | 4.55e-04 | 4.87e-04 | 6.09e-04 |
| 5 | 100 | 3.00e-04 | 0 | 1.31e-04 | 4.00e-05 | 7.89e-05 | 8.06e-05 |
| 5 | 500 | 0 | 0 | 0 | 0 | 2.18e-06 | 5.93e-07 |
| 5 | 1000 | 0 | 0 | 0 | 0 | 1.10e-06 | 0 |
| 10 | 50 | 4.21e-03 | 2.27e-03 | 4.45e-03 | 3.62e-03 | 3.17e-03 | 3.30e-03 |
| 10 | 100 | 9.99e-04 | 0 | 7.06e-04 | 3.60e-04 | 5.60e-04 | 3.57e-04 |
| 10 | 500 | 0 | 0 | 2.87e-05 | 0 | 1.82e-05 | 3.15e-06 |
| 10 | 1000 | 0 | 0 | 1.21e-05 | 0 | 3.59e-06 | 2.02e-07 |
| 25 | 50 | 3.35e-02 | 3.35e-02 | 2.46e-02 | 2.46e-02 | 2.72e-02 | 2.72e-02 |
| 25 | 100 | 1.25e-02 | 3.56e-03 | 5.93e-03 | 3.59e-03 | 4.80e-03 | 4.63e-03 |
| 25 | 500 | 4.07e-04 | 0 | 2.72e-04 | 1.96e-05 | 9.13e-05 | 2.29e-05 |
| 25 | 1000 | 0 | 0 | 7.53e-05 | 0 | 2.70e-05 | 4.01e-06 |

Table 10: Best results on the “uniform” instances without restarts