Università di Pisa
Dipartimento di Informatica

Technical Report: TR-00-17

A multi-exchange neighborhood for minimum makespan machine scheduling problems

Antonio Frangioni Emiliano Necciari Maria Grazia Scutellà

December 5, 2000

ADDRESS: Corso Italia 40, 56125 Pisa, Italy. TEL: +39 050 887111 FAX: +39 050 887226

A multi-exchange neighborhood for minimum makespan machine scheduling problems

Antonio Frangioni Emiliano Necciari Maria Grazia Scutellà

December 5, 2000

Abstract

We propose new local search algorithms for minimum makespan machine scheduling problems, which perform multiple exchanges of jobs among machines. Inspired by the work of Thompson and Orlin (1989) on cyclic transfer neighborhood structures, we model multiple exchanges of jobs as special disjoint cycles and paths in a suitably defined improvement graph, by extending definitions and properties introduced in the context of VRP (Thompson and Psaraftis, 1993) and of CMST (Ahuja, Orlin and Sharma, 1998). Several algorithms for searching the neighborhood are suggested.

We report the results of a wide computational experimentation, on different families of benchmark instances, performed for the case of identical machines. The minimum makespan machine scheduling problem with identical machines has been selected as a case study to perform a comparison among the alternative algorithms, and to discover families of instances for which the proposed neighborhood may be promising in practice. The obtained results are very interesting. On some families of instances, which are very hard to solve exactly, the most promising multi-exchange algorithms proved to dominate, in gap and in time, competitive benchmark heuristics.

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

1 Introduction

Let $J = \{1, 2, \ldots, n\}$ be a set of n independent jobs. The jobs have to be assigned, without preemption, to m parallel machines M_1, M_2, ..., M_m, where $n \geq m \geq 2$, thereby creating a partition of J into m subsets. Let p_{jh} be the (positive) processing time of job j when assigned to machine M_h, $j = 1, \ldots, n$, $h = 1, \ldots, m$.

Given a job assignment (or job partition) S, denote by $C_h(S)$ the completion time of machine M_h under the assignment S, i.e., the latest job finishing time of M_h, $h = 1, \ldots, m$. Moreover, denote by $C_{max}(S) = \max_{h=1,\ldots,m}\{C_h(S)\}$ the maximum completion time, or *makespan*, associated with S.

The *minimum makespan machine scheduling problem* consists in finding a job assignment which minimizes the makespan. Using the standard three field classification scheme, introduced in (Graham et al., 1979), this problem is usually denoted as $R||C_{max}$. It includes, as special cases, the problem $P||C_{max}$ (case of identical machines), where the job processing times do not depend on the machines, and the problem $Q||C_{max}$ (case of uniform machines), where the job processing times p_{jh} have the form $p_{jh} = p_j \alpha_h$.

$R||C_{max}$ is NP-hard in the strong sense (Garey and Johnson, 1978). Few exact methods have been proposed for its solution. We mention the branch-and-bound algorithm of (Van de Velde, 1993), which solves instances with up to 200 jobs and 4 machines, and the algorithm of (Martello et al., 1992). These algorithms may require a significant computational effort, mainly on large instances. This justifies the important role of heuristic methods to compute "good" feasible solutions.

Heuristics are generally classified into constructive heuristics, and improvement heuristics. The majority of those proposed in the literature for minimum makespan machine scheduling problems fall within the first category, for which a worst-case performance ratio is generally provided. We cite, among the others, the *list scheduling* family of (Graham, 1966), which includes the well-known *longest processing time* (LPT) algorithm. See (Lawler et al., 1993) for a survey about constructive heuristics for these problems.

As far as improvement heuristics are concerned, few methods have been proposed (Anderson et al., 1997). One motivation might be that, as observed in (Hübscher and Glover, 1994) for the case of identical machines, "Minimizing the makespan on multiprocessors seems to be hard for local optimization techniques, since neighboring solutions generally differ widely in quality". A descent algorithm which uses a critical reassign and a critical swap neighborhood has been proposed in (Hariri and Potts, 1991); it appears to generate better quality solutions than two-phases heuristics, as the one of Lenstra, Shmoys and (Tardos, 1990). Simulated annealing and tabu search algorithms, which use critical reassign and critical swap neighborhoods, as well as genetic algorithms are described in (Glass et al., 1994).

Recent advances have been performed in local search methods, with the aim of designing more powerful neighborhood structures for hard combinatorial optimization problems. Among them, the so-called *ejection chain methods* (Glover, 1996; Rego, 1998) focus on the development of compound neighborhood structures which encompass succession of dependent moves, rather than simple moves or sequences of independent moves. Other sophisticated local search algorithms are the *network flow based improvement algorithms*, which also perform sequences of dependent moves, called multiple exhanges or *multi-exchanges*. The distinction between these two classes of algorithms is not clear, as outlined in (Ahuja et al., 1999). However, a distinguishing property of network flow based improvement algorithms is that they use network flow techniques to identify improvement moves. For example, some on these algorithms characterize improvement moves in terms of negative cost *disjoint cycles* in an associated graph, called the *improvement graph*. Network flow based improvement algorithms have been successfully applied to several combinatorial optimization problems, such as vehicle routing problems (VRP) (Thompson and Psaraftis, 1993), dynamic vehicle dispatching problems (Gendreau et al., 1999), and the capacitated minimum spanning tree problem (CMST) (Ahuja et al., 1998).

In this paper, we propose network flow based improvement algorithms for minimum makespan machine scheduling problems, which perform multiple exchanges of jobs among machines. Multi-exchanges are modelled as special disjoint cycles and paths in a suitably defined improvement graph, by extending definitions introduced in the context of VRP and CMST. Both the improvement graph and the improvement moves have some peculiar properties, which are formally investigated. These properties are then used to design several algorithms for searching the neighborhood, for the discover of improvement moves.

The proposed neighborhood structure, and the corresponding searching algorithms, have to be intended as a first step towards the design of new local search algorithms for minimum makespan machine scheduling problems, which go beyond the classical ones proposed in the literature, generally based on exchanges of two jobs (*2-exchanges*). This work is in fact essentially methodological, aimed to study new techniques to design neighborhood structures for minimum makespan machine scheduling problems, exploiting peculiar graph properties, and to investigate in what cases, and under what algorithmic choices, these new structures may produce "good" feasible solutions.

The ideas contained in this paper have also to be considered as a first step towards the design of multi-exchange neighborhood structures for a more general class of partition problems, characterized by a "bottleneck-type" objective function, as is the case of $R||C_{max}$. Such a more general class, which includes bottleneck knapsack problems and some location problems, will be the subject of a future work.

In order to perform a comparison among the alternative algorithms, and to try to characterize properties of the instances for which the proposed algorithms are expected to provide good solutions in practice, we have selected, as a case study, the problem $P||C_{max}$. The case of identical machines has been considered here since it is "easier", and therefore the practical behavior of the heuristics in the case of identical machines may reasonably provide a worst-case estimate of their behavior for more general cases. Moreover, a larger number of (both theoretical and experimental) papers are available in the literature for $P||C_{max}$. Several heuristics, especially constructive ones, have been proposed for the solution of $P||C_{max}$, and tested on some benchmark instances. Furthermore, a branch-and-bound algorithm (Dell'Amico and Martello, 1995) is available to compute the optimum solution in some tractable instances and to discover, on the other hand, "difficult" instances on which to study the behavior of the multi-exchange heuristics.

Four families of instances have been selected for our computational experience, which have a very different structure: the uniform instances proposed in (França et al., 1994), some non-uniform instances proposed in (Necciari, 1999) and two families deriving from bin packing instances, available at the OR Library of Beasley, which are reputed very difficult.

The obtained results are very interesting. In general, some trends emerged from our computational experimentation, which suggest what multi-exchange algorithms are expected to provide good results, depending on the properties of the instances under consideration. Moreover, on some families of instances, which are very hard to solve with exact approaches, the most promising multi-exchange algorithms were able to provide solutions which dominate, in gap and in time, the solutions returned by benchmark algorithms.

The plan of the paper is the following. Section 2 contains the presentation of the multi-exchange neighborhood. In particular, it contains the definition of the improvement graph, the statement of its properties, and the characterization of the improvement moves in terms of peculiar subgraphs of the improvement graph. Section 3 contains the description of the algorithms used to explore the neighborhood. Finally, Section 4 describes the results of the computational investigation concerning $P||C_{max}$.

2 The multi-exchange neighborhood

Let S denote the current (feasible) solution of $R||C_{max}$. Moreover, given a job j, let $M(j)$ denote the machine assigned to j in the current solution.

As in (Ahuja et. al, 1998), two kinds of multi-exchanges are defined, i.e., cyclic and path exchanges.

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 simultaneous 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 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 is 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 exchanges could be easily generalized, by allowing the movement of subsets of jobs from one machine to another one, simultaneously. This will be the subject of future investigation.

Given S, the *neighborhood* of S is the set of all the assignments 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; thus, the idea is to model and search such a neighborhood by means of a suitable graph, called the improvement graph. The definition of the improvement graph, which is based on the theory of cyclic transfers introduced in (Thompson and Orlin, 1989), and which extends concepts introduced in the context of VRP and of CMST, will be the subject of the following section.

2.1 The improvement graph and its properties

Given a solution S, the *improvement graph* related 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 corresponding to S. The graph contains one node for each job $1, 2, \ldots, n$, 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*. On the other hand, the machines having a completion time equal to $C_{max}(S)$ will be referred to as *loaded machines*, and the set of the loaded machines will be denoted by K (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: adding job i to $M(j)$ and removing job j from $M(j)$, $M(j)$ is unloaded. Moreover, $A(S)$ contains an arc (j, M_r) for each pair job-(unloaded) machine such that j is currently not assigned to M_r, and such that, by moving j to M_r, then machine M_r is still unloaded. Clearly, $|N(S)| \leq n + m - 1$ and $|A(S)| \leq n(n + m - 2)$; the size of the improvement graph is thus polynomial with respect to the input size.

The improvement graph $G(S)$ has a special structure, which is now formally investigated.

Since S is 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, $h = 1, \ldots, m$. Accordingly, the set of the nodes of $G(S)$ is partitioned in clusters as follows: there is one cluster for each subset S_h, $h = 1, \ldots, m$, plus one cluster, composed by a single node, for each unloaded machine.

By definition, no arc exists between nodes belonging to the same cluster, that is $G(S)$ is a $(2m - |K|)$-partite graph. Moreover, strong *convexity properties* are revealed by ordering the jobs in S_h, $h = 1, \ldots, m$, in a non-decreasing way with respect to their processing times when assigned to machine M_h: for any pair of clusters S_q and S_h, the subgraph of $G(S)$ formed by the sets S_q and S_h, and by the arcs going from S_q to S_h, is a convex graph with respect to S_h. In fact, by definition, if $i \in S_q$ and $(i, j) \in A(S)$ with $j \in S_h$, then $(i, v) \in A(S)$ for each job v following j in S_h with respect to the introduced ordering of the jobs. The same property trivially holds for the arcs entering a node M_r corresponding to an unloaded machine, since the corresponding cluster is a singleton. As a consequence, for each node $i \in S_q$, $q = 1, \ldots, m$, and for each set S_h, $h \neq q$, the set of the nodes of S_h which are heads of arcs outgoing from i, when not empty, forms an interval $[i_1(h), i_2(h)]$, where $i_2(h)$ is the job of S_h which is maximal with respect to the considered ordering of the jobs. Clustered directed graphs satisfying the properties of $G(S)$ will be referred to as *F-convex graphs* (with respect to a given ordering of the nodes within the clusters).

The structure of the improvement graph is illustrated in figure 1. The example refers to an instance characterized by 3 identical machines M_1, M_2 and M_3, and by 6 jobs with processing times $\{2, 4, 3, 5, 6, 4\}$, respectively. The figure shows the improvement graph associated with the solution

$S = \{S_1, S_2, S_3\}$, where $S_1 = \{1, 4\}$, $S_2 = \{2, 5\}$ and $S_3 = \{3, 6\}$; M_2 is the only loaded machine, with makespan 10. To make compact the representation, each couple of arcs (i, j) and (j, i) has been depicted as a bidirectional arc.

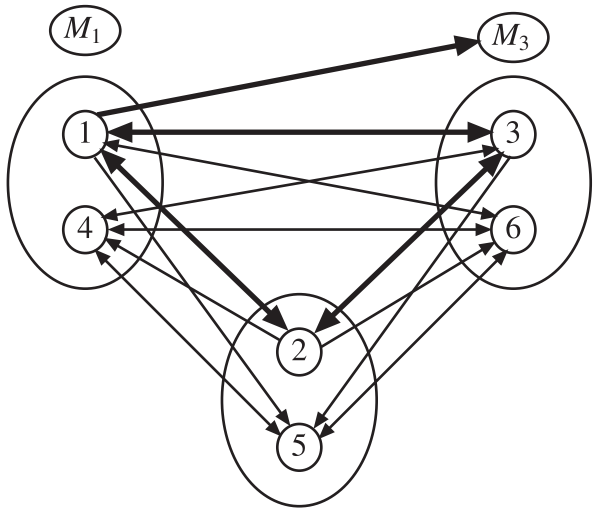

Figure 1: The improvement graph

In the figure, there is one cluster for each subset S_h, $h = 1, 2, 3$, plus two clusters corresponding to the unloaded machines M_1 and M_3. The convexity property is revealed by looking, for example, at the subsets S_1 and S_2: the set of the nodes of S_2 which are heads of arcs outgoing from node 1 forms the interval $[2, 5]$, that is $i_1(h) = 2$ and $i_2(h) = 5$ for $i = 1$ and $h = 2$.

Observe that the convexity property is more general in the case of identical or uniform machines (problems $P||C_{max}$ and $Q||C_{max}$, respectively). There, for any pair of clusters S_q and S_h, the subgraph of $G(S)$ formed by the sets S_q and S_h, and by the arcs going from S_q to S_h, is also convex with respect to S_q. More precisely, for each node $j \in S_h$ the set of the nodes of S_q which are tails of arcs entering j, when not empty, forms an interval, whose first end is the job of S_q which is minimal with respect to the ordering of S_q described before. The same holds true for the arcs entering a node M_r corresponding to an unloaded machine.

2.2 The improvement subgraphs

Let us define now subgraphs of $G(S)$ which model improvement cyclic or path exchanges, and discuss the problem of finding these subgraphs.

A directed cycle $W = (j_1, j_2, \ldots, j_r, j_1)$ in $G(S)$, where $j_1, j_2, \ldots, j_r$ represent jobs, is 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 $j_1, j_2, \ldots, j_{r-1}$ represent jobs, M_r represents a machine, and if $M(j_k) \neq M(j_q) \neq M_r$ for $k \neq q$, with $k, q \in \{1, 2, \ldots, r-1\}$.

From the above definitions the following property can be easily proved:

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

The relationship between cyclic (path) exchanges and disjoint cycles (paths) is illustrated in figure 1. There, the disjoint cycle (1, 2, 3) corresponds to the cyclic exchange consisting in moving job 1 from M_1 to M_2, job 2 from M_2 to M_3, and job 3 from M_3 to M_1. The disjoint path (1, M_3) describes the assignment of job 1 to the unloaded machine M_3.

A (significant, as shown below) subset of cyclic (path) exchanges for the current solution S can thus be modelled in terms of peculiar subgraphs of the improvement graph $G(S)$, that is disjoint cycles and paths. Due to the definition of the arc set $A(S)$, the opposite correspondence does not hold, that is, there may be cyclic or path exchanges which are not mapped onto disjoint cycles or paths in $G(S)$. These are the cyclic (path) exchanges which worsen the current makespan. In fact, each arc of type (i, j) in $G(S)$ models a movement of job which guarantees a completion time less than $C_{max}(S)$ for machine $M(j)$; similarly, each arc of type (j, M_r) describes a movement of job which maintains M_r unloaded.

Such a quite peculiar property does not hold in the case of the multi-exchange neighborhood structures introduced in the literature for VRP and for CMST. There, due to the different type of the objective function under consideration (which is a sum function), moves which "locally" worsen the objective function cannot be ruled out. Therefore, multi-exchanges worsening the objective function can not be excluded from the neighborhood, and they are modelled into the associated improvement graph.

Also the characterization of the improvement multi-exchanges in terms of disjoint cycles and paths is quite different from that occurring in the case of VRP and CMST. For those problems, improvement multi-exchanges are in a one-to-one correspondence with the negative disjoint cycles and paths in the associated improvement graph, 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 characterize the improvement multi-exchanges for the current solution S only thanks to their peculiar structure, that is, independently of arc costs. In order to state such a characterization, a further definition is required.

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

In figure 1, the two disjoint cycles (1,2,3) and (1,3,2) are examples of K-disjoint cycles, since $K = \{M_2\}$ and job 2 belongs to S_2. Observe that the exchange (1,2,3) reduces the makespan from 10 to 8, whereas the alternative K-disjoint cycle (1,3,2) generates an improved solution where M_1 and M_2 are the loaded machines, and the makespan is 9.

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

Proof: Consider an improvement cyclic exchange. Since it is an improvement move, the makespan associated with the current solution S reduces after performing the cyclic exchange; hence all the loaded machines, i.e., all the machines belonging to K, must be involved. Such a cyclic exchange thus corresponds to a K-disjoint cycle in $G(S)$. The opposite correspondence and the case of the improvement path exchanges can be proved in a similar way. $\diamond$

Based on Property 2.2, the problem of finding an improvement move in the proposed 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. In (Thompson, 1988) and in (Thompson and Orlin, 1989) it was proved that, given a generic clustered graph, that is a graph whose set of nodes is partitioned in a set of clusters H, the problem of determining whether there is a cycle which includes at most one node for each cluster of H (the so-called *disjoint cycle*) is NP-complete (observe that this is the same definition of disjoint cycle given for the improvement graph $G(S)$; in fact, nodes representing a machine cannot appear in a cycle or be internal to a path, since they have no outgoing arcs in $G(S)$). Also the problem of deciding, in a generic clustered graph, about the existence of a disjoint cycle which includes exactly one node for each cluster in a given subset of clusters, W, is NP-complete. This is true, in particular, when $|W| = 1$.

We will show that these computations remain difficult in the special case of the F-convex graphs, which is the case of the improvement graphs associated with the instances of $R||C_{max}$. Hereafter, $G = (V, E)$ will be used to denote a F-convex graph, that is a directed graph whose node set V is partitioned in a cluster set $H = (V_1, V_2, \ldots, V_h)$, and which satisfies the following convexity property with respect to a given ordering of the nodes within the clusters: for any pair of clusters V_q and V_p, if E contains the arc (i, j), with $i \in V_q$ and $j \in V_p$, then E contains all the arcs (i, v), with v following node j with respect to the given ordering of the nodes in V_p. Given a non-empty subset W of the cluster set H, the term W-disjoint cycle (path) will be used to denote a disjoint cycle (path) which includes exactly one node for each cluster in W.

The proof relies on a peculiar property of the F-convex graphs, which will be formally proved in the following. This property states that, in order to verify the existence in G of a W-disjoint cycle or path, we can restrict our attention to a special subset of the arcs of G. More precisely, for any pair of clusters V_q and V_p, and for each node $j \in V_p$ which has an entering arc (i, j) with $i \in V_q$, we can consider only the arc (w, j), where w is the maximal node in V_q (with respect to the ordering of the nodes of V_q) for which there exists an arc entering j. The arc (w, j) will be referred to as the *maximal arc from* V_q *to* j. An example is provided in figure 2(a), where the ordering of the nodes within the clusters is given by their height (top nodes preceed bottom nodes in the ordering); (i, w) is the maximal arc from V_q to w, whereas (v, z) is the maximal arc from V_q to z.

Denote by $\bar{E}$ the set of all the *maximal arcs* of G, and by $\bar{G} = (V, \bar{E})$ the partial graph of G formed by the arcs in $\bar{E}$. Then the following property holds true.

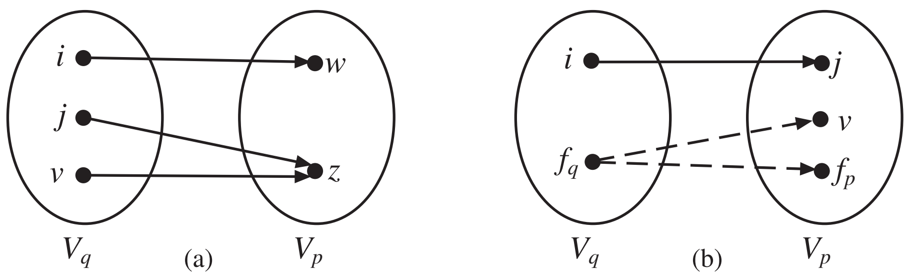

Figure 2: The maximal arcs

Property 2.3 *Let $G = (V, E)$ be a F-convex graph, and W be a subset of its set of clusters H. G contains a W-disjoint cycle (path) if and only if $\bar{G}$ contains a W-disjoint cycle (path).*

Proof: The "if" implication is obvious, since $\bar{G}$ is a partial graph of G.

In order to prove the "only if" implication, assume that G contains a W-disjoint cycle C. Consider a pair of consecutive arcs, (j, i) and (i, v), along this cycle, with $j \in V_r$, $i \in V_q$ and $v \in V_k$ ($V_k = V_r$ if the cardinality of the cycle is 2). If (i, v) is not the maximal arc from V_q to v, we can substitute (i, v) with the maximal arc, say (w, v). Then, since G is a F-convex graph and $(j, i) \in E$, (j, w) is also an arc of E; thus we can substitute (j, i) by (j, w). In this way we obtain an alternative cycle C' where an arc which is not maximal has been substituted by a maximal arc. C' includes the same subset of clusters as C, and therefore it is a W-disjoint cycle. However, as a side effect, it may happen that the introduced arc (j, w) is not maximal. The substitution process has therefore to be iterated until the cycle contains only maximal arcs.

This process terminates in a finite number of steps. In fact, whenever an arc (i, v) is substituted, the introduced arc is of type (w, v), where w follows i in the ordering of the nodes within the cluster of i. Therefore, for each cluster V_r included by C, the unique node of the cycle belonging to V_r can only "move" forward with respect to the ordering of the nodes in V_r, and this can be performed at most $|V_r| - 1$ times.

The same process can be applied if we start from a W-disjoint path of G. The thesis follows. $\diamond$

Corollary 2.1 *It is polynomially equivalent to decide about the existence of a W-disjoint cycle (path) in a F-convex graph G or in any partial graph of G which contains all the maximal arcs of G.*

We have now the ingredients to prove the NP-completeness result.

Theorem 2.1 *Given a F-convex graph $G = (V, E)$, and given a subset W of its cluster set H, the problem of deciding about the existence of a W-disjoint cycle in G is NP-complete.*

Proof: Due to Corollary 2.1, it is polynomially equivalent to prove the result for G or for a partial graph of G, provided that this partial graph contains all the maximal arcs of G. In constructing the proof we will exploit this equivalence.

Consider the special case $|W| = 1$. In this case, the problem is polynomially equivalent to the problem of verifying the existence of a directed path, containing at most one node for each cluster in H, from some node s belonging to the unique cluster in W, to some node t belonging to a cluster in $H \setminus W$, such that the arc (t, s) exists. We will therefore prove the NP-completeness of the following problem defined on a partial graph of a F-convex graph, which contains all its maximal arcs: given a source s and a destination t, verify the existence of a directed path from s to t which contains at most one node for each cluster of the graph. The proof is by reduction from the *path with forbidden pairs* problem. This is 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 $G' = (V', E')$, which must contain at most one node for each pair of a given sequence $\{(i_1, j_1), \ldots, (i_k, j_k)\}$ of disjoint pairs of nodes (Garey and Johnson, 1978).

Consider an instance of the path with forbidden pairs problem. Define a directed graph $\bar{G} = (\bar{V}, \bar{E})$ as follows. Initially, add to $\bar{V}$ all the nodes of V', and partition $\bar{V}$ in a cluster set H as follows: there is one cluster V_h for each pair of "forbidden" nodes, containing i_h and j_h, and there is one cluster for each remaining node p of G', containing p. Furthermore, add to each cluster, say V_q, an additional, fictitious, node f_q, with the exception of the cluster containing the destination node t'.

In order to define the arc set $\bar{E}$, let us impose an arbitrary ordering among the nodes of each cluster, with the only constraint that each fictitious node is always the last node of its cluster.

Initially, add to $\bar{E}$ all the arcs of G', with the exception of those entering s' and of those leaving t'. Note that, at this step of the construction, the fictitious nodes have no entering or leaving arcs. Now, consider each pair of clusters, V_q and V_p, which does not satisfy the F-convexity property. There are two possible cases for that. The first situation is when $V_p = \{j, v, f_p\}$, with j preceeding v in the considered ordering; there is an arc (i, j) with $i \in V_q$, but the arc (i, v) does not belong to E. In this case, let us add the fictitious arcs (f_q, v) and (f_q, f_p) to $\bar{E}$. An example of this construction is illustrated in figure 2(b). The second case verifies when $V_p = \{j, f_p\}$ and there is an arc (i, j), with $i \in V_q$. In this case let us add the fictitious arc (f_q, f_p).

Finally, set $s = s'$ and $t = t'$.

By the above construction, there is a directed path from s' to t' in G', containing at most one node for each pair $\{i_h, j_h\}$, $h = 1, \ldots, k$, if and only if there is a directed path in $\bar{G}$ from s to t which contains at most one node for each cluster in H. The "only if" part is trivial, since each path of G' corresponds to a path of $\bar{G}$. The "if" part comes from the property that no fictitious node can be reached from s (no fictitious arc leaves s), and therefore no fictitious arc belongs to a directed path of $\bar{G}$ from s to t.

$\bar{G}$ is not necessarily a F-convex graph. However, if we consider the F-convex graph, G, obtained from $\bar{G}$ by adding to $\bar{E}$ the minimal set of arcs which are required to convert $\bar{G}$ into a F-convex graph (the arcs (i, v) and (i, f_p) in figure 2(b)), then it is easy to verify that none of these additional arcs is maximal. Thus, $\bar{G}$ is a partial graph of G which contains all its maximal arcs. The thesis follows. $\diamond$

When the F-convex graph is an improvement graph $G(S)$ and $W = K$, then a consequence of Theorem 2.1 is that the problem of computing a K-disjoint cycle (path) in $G(S)$ is NP-complete. Theorem 2.1 shows another interesting result. If W contains a single cluster, corresponding to a loaded machine, then also the "relaxed" problem of looking for a disjoint cycle (path) which includes *at least one* loaded machine is a "difficult" problem. In the machine scheduling terminology, this second problem corresponds to looking for a multi-exchange which does not worsen the current makespan, and strictly improves the completion time for at least one loaded machine. In other words, that corresponds to enlarge the set of the improvement moves, by allowing not only multi-exchanges which reduce the current makespan, but also multi-exchanges which reduce the completion time for at least one loaded machine. These relaxed moves will be referred to as *1-disjoint cycles and paths.*

In the following sections, both kinds of multi-exchanges will be addressed. Due to Theorem 2.1, heuristic approaches will be investigated both for the problem of computing K-disjoint cycles and paths in $G(S)$, and for computing 1-disjoint cycles and paths. This is a coherent choice, since the entire algorithmic paradigm under investigation is of the heuristic type.

3 Heuristic approaches for disjoint cycle and path computation

We propose three families of heuristics, aimed at computing either K-disjoint or 1-disjoint cycles and paths in the improvement graph $G(S)$.

Two families, called Label correcting and Bottleneck path heuristics, are inspired by classical algorithmic paradigms for the shortest path tree and for the bottleneck path tree computations. They use two alternative definitions of arc costs, which are introduced to heuristically guide the selection of the nodes during the disjoint cycle and path computation. The third family is formed by search heuristics, and no arc costs are used to guide the computation.

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

$$c_{ij} = \begin{cases} p_{ih} - p_{jh}, & \text{if } j \text{ is a job assigned to machine } M_h, \text{ and } M_h \text{ is loaded;} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

This cost measures the reduction of the completion time of M_h if it is loaded, i.e., it contributes to the current makespan (in this case it is $c_{ij} < 0$); otherwise, the cost is 0. An example is provided in figure 3. The example refers to the instance described in figure 1, by restricting the attention to the subgraph formed by the clusters S_1, S_2, M_1 and M_3.

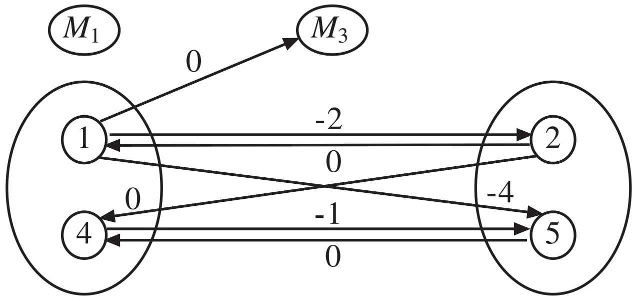

Figure 3: The cost definition (1)

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

$$c_{ij} = \begin{cases} \sum_l p_{lh} + (p_{ih} - p_{jh}) - C_{max}(S), & \text{if } j \text{ is a job assigned to machine } M_h; \\ \sum_l p_{lh} + (p_{ih}) - C_{max}(S), & \text{if } j \text{ represents the (unloaded) machine } M_h, \end{cases} \tag{2}$$

where l denotes the generic job assigned to M_h. Figure 4 shows the same portion of the improvement graph provided in figure 3, where the arc costs are defined accordingly to (2).

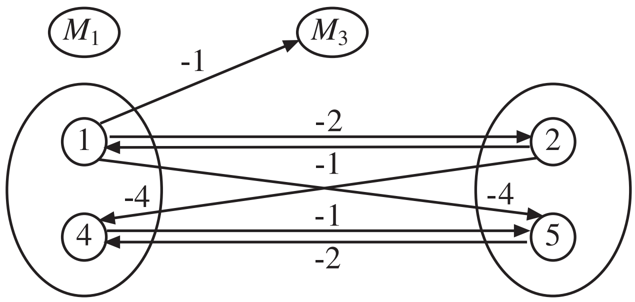

Figure 4: The cost definition (2)

Definition (1) suggests to search for a disjoint cycle or path in the improvement graph $G(S)$ by an approach similar to the one proposed in (Ahuja et al., 1998) for CMST, that is by a modification of the label correcting approach for the shortest path tree computation (for a complete survey about shortest path methods, we refer to (Ahuja et al., 1993)).

On the other hand, definition (2) is more 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.

Now let us describe the three families in more detail.

Label correcting heuristics

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 can be derived from this general framework by suitably implementing the set Q. When enriched with a cycle detection strategy, these algorithms can be used to identify, in polynomial time, (unrestricted) negative cycles in the input graph G (Cherkassky and Goldberg, 1999).

Starting from the general label correcting framework, we propose two heuristic approaches, $K-SPT$ and $1-SPT$, which use the cost definition (1) for finding K-disjoint cycles (paths) and 1-disjoint cycles (paths), respectively, 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 disjoint path from a node r, chosen as the root, to the node j.

Consider $K - SPT$. It performs the following typical iteration until either a K-disjoint cycle (path) is found or Q is empty; i denotes the current node extracted from Q.

Typical iteration of $K - SPT$ The current path from r to i is checked for disjointness (changes in subpaths may have occurred during the algorithm execution, which may have destroyed this property). If the path is not disjoint, then the iteration is terminated, and 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 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, and the current path from r to j is not disjoint: if j is internal to the disjoint path from r to i, and if $nm(i) - nm(pred(j)) = |K|$, then a K-disjoint cycle has been found ($K - SPT$ stops);

3. j represents a job, and the current path from r to j is disjoint: in this case j is inserted to Q, if not already present; $d(j)$ and $pred(j)$ are updated as in the standard shortest path computation; moreover, label $nm(j)$ is suitably updated.

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).

The heuristic $1-SPT$ is similar to $K-SPT$. It 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.

This stopping 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.

Bottleneck path heuristics

The heuristics $K-BPT$ and $1-BPT$ are based on the cost definition (2), and they are 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. 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, that is $d(j) := max\{d(i), c_{ij}\}$.

Therefore, 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 updating of the labels of the nodes. $1-BPT$ maintains a strong similarity with respect to $K-BPT$. In particular, 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 $|K|$.

Search heuristics

The search heuristics, $K-search$ and $1-search$, do not rely on arc costs to guide the computation of K-disjoint and 1-disjoint cycles and paths, respectively. They are the "degenerate" versions of $K-SPT$ and $K-BPT$ from one hand, and of $1-SPT$ and $1-BPT$ on the other hand, when considering all the arc costs equal to zero.

In these heuristics, the set of the candidate nodes Q is implemented either as a *stack*, which corresponds to visit the improvement graph in a *depth first* way from a given root node, or as a *queue*, which corresponds to visit the improvement graph in a *breadth first* way. Clearly, this choice may affect the kind of cycles and paths found by the algorithms.

By looking at the three families of heuristics, we can observe that aim of $K-BPT$ and $1-BPT$ is to heuristically compute a disjoint cycle or path which allows for the maximum reduction of the completion time for all the involved machines: this is due to the cost definition (2), and to the use of the bottleneck path tree paradigm (however, the algorithms do not ensure the maximum reduction of the makespan, due to the unloaded machines which are not involved in the computed cycle or path). On the other hand, $K-SPT$ and $1-SPT$ look for a "good" makespan reduction using, as a guide for their search, the label correcting paradigm. $K-search$ and $1-search$, on the contrary, simply try to find any improvement disjoint cycle or path, without any attempt of discriminating the "quality" of the cycles.

We can also observe that all the proposed heuristics are actually algorithmic paradigms: several heuristics can be derived from them by suitably implementing the set of the candidate nodes Q. In addition to the *stack* and to the *queue*, used for implementing $K-search$ and $1-search$, the Label correcting and the Bottleneck path heuristics have been also specialized by implementing the set Q as a *priority queue* (the node j extracted from Q has the smallest distance label $d(j)$), and as a *deque* (the nodes are inserted either at the front or at the rear of the list of the candidate nodes).

4 $P||C_{max}$: a case study oriented to computational investigation

In order to evaluate the potential of the multi-exchange heuristics, we have selected $P||C_{max}$ as a case study. One motivation for this choice is that the majority of scientific papers about minimum makespan machine scheduling problems are devoted to the case of identical machines. If we look at the constructive heuristics, in addition to the *list scheduling* family of (Graham, 1966), which includes the *longest processing time* (*LPT*) algorithm, we can cite the *multifit* algorithm of (Coffman et al., 1978). See (Lawler et al., 1993) for a survey about the classical constructive heuristics for the problem. As far as improvement heuristics are concerned, an $O(n \log m)$ time algorithm has been proposed in (Finn and Horowitz, 1979), which reassigns jobs from most loaded to less loaded machines, and performs interchanges of jobs. This algorithm, also known as *0/1 interchange* algorithm, has been later improved in (Langston, 1982). A more sophisticated algorithm is proposed in (França et al., 1994); it uses a combined critical reassign and critical swap neighborhood structure, and essentially consists in exchanging two jobs between two machines. There is also a tabu search algorithm, proposed in (Hübscher and Glover, 1994), which performs 2-exchanges, and 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 also based on 2-exchanges of jobs.

Another motivation for the choice of $P||C_{max}$ is that a branch-and-bound algorithm is available for its exact solution (Dell'Amico and Martello, 1995). This algorithm receives in input the maximum number of allowed backtracks; when the backtracking limit is not sufficient to compute the optimum solution, the algorithm returns the best computed solution, and therefore it turns into a heuristic method. The branch-and-bound algorithm has been used for discriminating "easy" instances, which can be efficiently solved, from more "difficult" ones, for which heuristic approaches may be appropriate. Also, the branch-and-bound algorithm finds the optimum solution in several cases, thus providing a better measure of the quality of the solutions constructed by the heuristics, whereas it is a competitive heuristic in the "difficult" cases.

Finally, there is a more philosophical motivation: since the case of identical machines is a special and easier case of both the uniform and the unrelated cases, we think that the results of a computational experimentation performed for $P||C_{max}$ may provide useful "worst-case" indications for the more general cases.

4.1 Plan of the computational investigation

The computational investigation consists of two phases, having a different purpose.

In the first phase we tested a large number of multi-exchange heuristics, which differ for the following alternative options:

- computation of K-cycles (paths) or 1-cycles (paths);
- use of a cost definition ((1) or (2)) to guide the computation, or no costs at all (case of the search heuristics);
- implementation of the set of the candidate nodes Q: deque, queue, stack, priority queue (queue e stack for the search heuristics);
- use of all the arcs of the improvement graph, or use of the subset of the maximal arcs (according to the results stated by Property 2.3).

The aim of this phase was mainly methodological: to compare the alternative heuristics, in order to establish guidelines for the implementation of the multi-exchange algorithms. Indeed, we will show that some general trends exist, which can help in choosing the proper implementation. We believe that some of the obtained results are still valid for more general minimum makespan machine scheduling problems, as well as for other combinatorial optimization problems with a bottleneck-type objective function. Due to the large number of variants tested in this phase, we will avoid to report tables of results, describing, whenever possible, the trends that emerged from the experimentation.

In the second phase, we investigated about the potential of the multi-exchange neighborhood, by considering the most promising heuristics selected during the first phase. We studied the behavior of the algorithms on four families of benchmark instances, having a very different structure: the uniform instances proposed in (França et al., 1994), the non-uniform instances proposed in (Necciari, 1999), and two families deriving from bin packing instances, available at the OR Library of Beasley, which qualify as very difficult instances. The aim was to trace out properties of the instances which may be suitable for local search (and, in particular, for the multi-exchange heuristics), and to perform a comparison with other heuristics from the literature. We used the branch-and-bound algorithm of (Dell'Amico and Martello, 1995) to classify the instances in "easy" (for which the branch-and-bound algorithm computes the optimum solution very rapidly) and "not easy" ones, in order to focus our analysis on the "not easy" instances. Then we analyzed the behavior of the best multi-exchange heuristics by estimating the relative error, and reporting the computational time, the average number of the computed cycles and paths, and their average cardinality. We also tested the improvement algorithm of (França et al., 1994), as an example of local search algorithm essentially based on 2-exchanges, and the algorithm of (Dell'Amico and Martello, 1995) with a limit on the number of backtracks, in order to provide a comparison with a different type of heuristic from the literature.

All the multi-exchange algorithms were 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 256Mb RAM under the RedHat GNU/Linux 6.0 operating system. The benchmark algorithms were also compiled and run in the same environment. As far as the code of (Dell'Amico and Martello, 1995) is concerned, referred to as $MTB\&B$, we experimented with different settings of the maximum number of allowed backtracks, starting from the value 4000, which is suggested by the authors in their work. The code of (França et al., 1994), referred to as $FMHEU$, could not be run on all the instances due to problems related with the maximum allowed dimension of some data structures (these are the two families of instances deriving from bin packing instances).

Again, we want to emphasize that it was not a goal of this research to deal with all the aspects which would be necessary to develop efficient heuristics for the problem. Thus, no effort has been performed to embed the pure multi-exchange local search, as presented here, into a more general metaheuristic framework, with devices, like tabu-lists, which could enhance the computational efficiency and provide a guidance to overcome local optimality.

4.2 Some implementation issues

In this section we will describe some issues that arise when implementing a multi-exchange heuristic, and how they were solved in our implementation. Our choices were mainly motivated by a preliminary experimentation described in [Necciari, 1999].

Implicit construction of $G(S)$

The improvement graph $G(S)$ is dynamic, as (portions of) it changes each time an improvement move is performed. There are two possibilities for implementing a visit of $G(S)$: either the data structures describing the graph (basically, the forward star of each node) are computed and stored, or the existence of an arc $(i, j) \in A(S)$ is checked each time, during the visit, the forward star of i has to be scanned.

With the first solution, the data structures describing $G(S)$ should be rebuilt each time the current solution S changes. Moreover, an explicit representation requires quite a lot of memory and generates, at each change of S, even parts of the graph which are never reached during the visit. On the contrary, the second solution avoids all these problems, but the checks which are necessary to decide about the existence of an arc (i, j) are potentially repeated many times, i.e., each time i is selected from Q. Our experience has shown that the implicit construction of $G(S)$ is usually more efficient than constructing the whole improvement graph. This strategy has therefore been adopted in our implementation.

The root selection and the stop criterion

All the heuristics visit the improvement graph starting from a given (job) node r. Clearly, different choices of r may lead to different results. In our implementation, the multi-exchange algorithms mantain a list of the nodes that have not been used yet as a starting point for the improvement of the current solution S, and select one node at random from that list: if a disjoint cycle or path is found starting from the selected 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 multi-exchange has been found, and the algorithm stops.

More sophisticated selection strategies can be imagined to select "promising" root nodes. In our preliminary experiments we tested some of these strategies, where the probability of selecting a node in the above list is biased by considering factors such as the processing time of the job and the completion time of the machine it is currently assigned to. The results showed that these strategies can have a positive impact on the performance, but the gains were unclear; therefore we decided to adopt the simplest strategy.

The initial solution

In local search algorithms, especially when, as in our case, no restarting mechanism is available, a crucial decision is the selection of the initial solution, i.e., the initial assignment of the jobs to the machines. In the preliminary experimentation we ran the multi-exchange algorithms starting from different solutions, obtained via standard constructive procedures for $P||C_{max}$. Among the others, we considered LPT and the multifit algorithm. In general, LPT showed to be the best choice, and therefore it has been chosen for generating the starting solution.

4.3 The instances

In almost all the previous computational experiences concerning $P||C_{max}$, the benchmark instances were randomly generated with uniform distribution of the processing times. During our preliminary experiences, we quickly realized that these instances tend to be "easy", up to the point that LPT alone produces very good solutions, which often are optimum ones. Thus we decided to develop and collect other, more challenging, classes of instances. As a result, the algorithms have been tested on four different families of instances.

The first two families are composed, respectively, by uniform and by "non-uniform" instances. In these families, the number of machines m varies in $\{5, 10, 25\}$, the number of jobs n varies in $\{50, 100, 500, 1000\}$ (for $m = 5$, $n = 10$ is also tested), and the interval for the (integer) processing times varies in $\{[1, 100], [1, 1000], [1, 10000]\}$. 10 problems were randomly generated for each choice of m, n and of interval of the processing times, for a total of 390 problems within each family.

The two families differ for the shape of distribution of the processing times. The generator of the first family, which has been presented in (França et al., 1994), generates the processing times at random, with uniform distribution, from the considered time interval. This family will be denoted UNIFORM. Given an interval $[a, b]$, the generator of the second family, which has been developed for this computational experimentation, 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 within the interval $[a, (b - a) * 0.02]$. This second family will be denoted NON-UNIF. Our preliminary computational experience showed that instances characterized by such "highly non-uniform" processing times are considerably more "difficult", at least for simple heuristics such as

LPT. 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.

The last two families of instances derive from some difficult bin packing instances, which are available at the OR-Library of J.E. Beasley, at the URL

$$http://mscmga.ms.ic.ac.uk/jeb/orlib/binpackinfo.html$$

In the first family, denoted BINPACK, the processing times are uniformly distributed in (20,100) (files binpack1 to binpack4, problem identifiers beginning with 'u'). The second family, denoted TRIPLET, consists of 'triplets' of elements with processing times from (25,50) (files binpack5 to binpack8, problem identifiers beginning with 't'). The triplet instances are such that the optimum solution has exactly three jobs per machine. To convert bin packing instances into $P||C_{max}$ instances, the number of machines m has to be specified; in our case, m is the number of bins in the best known solution of the bin packing instances.

4.4 Computational results: the first phase

In the first phase we compared a large number of alternative multi-exchange heuristics on the four families of instances, in order to verify whether some algorithmic choices are dominant. Fortunately some clear trends emerged, which allowed us to restrict the testing of the second phase to a limited set of alternatives. We compared each algorithm using a given option (say, visiting the whole improvement graph $G(S)$) with the identical algorithm but for the selected option (say, visiting the partial graph $\bar{G}(S)$ containing the maximal arcs of $G(S)$). When dominance was detected (for all the possible choices of the other options), then we concluded about the dominance of one option with respect to the alternative choices. The main criterion of dominance, in this phase, was the accuracy of the returned solution, measured as the relative error with respect to the best known lower bound. We could rule out a number of algorithmic options which most often produced poor results. Only when the solutions were of comparable quality we also took into account the running time.

The results of this phase can be summarized as follows:

1) when looking for both K-disjoint and 1-disjoint cyles (paths), it is preferable to visit the whole improvement graph $G(S)$ rather than the partial graph $\bar{G}(S)$, containing only the maximal arcs of $G(S)$;

2) it is preferable to use arc costs to guide the discover of K-disjoint and 1-disjoint cycles and paths; in fact, the algorithms $K-search$ and $1-search$ were always consistently outperformed by the corresponding algorithms using costs;

3) the alternative implementations of Q are essentially equivalent, i.e., no appreciable difference was found among variants that only differ in the implementation of the set Q.

Result 1) was somewhat surprising, in view of Property 2.3 which guarantees that an improving move exists in $G(S)$ if and only if it exists in the partial graph $\bar{G}(S)$. However, it is reasonable when combined with result 2), which suggests that, even when the whole $G(S)$ is considered, a simple search visit is in general not capable of finding improvement moves. Together, results 1) and 2) suggest that finding improvement moves is not an easy task also in practice, so that any choice which reduces the number of the available cycles and paths (e.g., by reducing the number of the available arcs) severely cripples the performance of the algorithms. These results also suggest that algorithms which are able to visit a larger number of paths (e.g., by maintaining multiple labels for each node) could be promising.

		$1-SPT$								1-Search		$1-SPT$	
		deque		queue		stack		priority				$\bar{G}$	
m	n	r. gap	sec	r. gap	sec	r. gap	sec	r. gap	sec	r. gap	sec	r. gap	sec
5	10	0.00e00	0.00	0.00e00	0.00	0.00e00	0.00	0.00e00	0.00	0.00e00	0.00	0.00e00	0.00
5	50	8.58e-03	0.01	8.90e-03	0.01	8.80e-03	0.01	8.80e-03	0.00	1.65e-02	0.00	1.80e-02	0.00
5	100	5.31e-05	0.02	5.31e-05	0.03	1.06e-04	0.03	5.31e-05	0.05	5.15e-03	0.01	8.33e-03	0.00
5	500	0.00e00	1.44	0.00e00	1.52	0.00e00	1.36	0.00e00	1.84	1.06e-05	0.04	6.06e-04	0.01
5	1000	0.00e00	12.01	0.00e00	11.12	0.00e00	11.41	0.00e00	14.73	5.31e-06	0.22	1.38e-04	0.04
10	50	1.57e-02	0.00	1.59e-02	0.00	1.55e-02	0.01	1.59e-02	0.00	1.85e-02	0.00	1.95e-02	0.00
10	100	5.09e-03	0.04	5.20e-03	0.04	5.09e-03	0.03	5.31e-03	0.05	8.49e-03	0.01	9.55e-03	0.00
10	500	2.13e-05	3.26	2.13e-05	3.28	2.13e-05	3.23	2.13e-05	3.58	4.18e-03	0.07	8.24e-03	0.02
10	1000	0.00e00	20.21	0.00e00	22.99	0.00e00	20.61	0.00e00	23.71	6.37e-05	0.32	7.86e-04	0.09
25	50	0.00e00	0.01	0.00e00	0.01	0.00e00	0.01	0.00e00	0.01	0.00e00	0.01	0.00e00	0.00
25	100	9.85e-03	0.04	9.85e-03	0.03	9.85e-03	0.04	1.01e-02	0.04	1.09e-02	0.01	1.60e-03	0.01
25	500	2.12e-04	5.78	2.12e-04	5.44	2.12e-04	5.97	2.12e-04	4.06	8.23e-03	0.03	8.34e-03	0.08
25	1000	7.97e-05	52.99	7.97e-05	45.33	7.97e-05	51.79	7.97e-05	40.63	7.57e-03	0.22	8.39e-03	0.19

Table 1: The first phase

Result 3) was also quite surprising. However, it may be partly explained by the fact that the improvement graphs tend to be quite dense, so that the strategy used to explore them probably has a poor influence on the performance of the algorithms.

Although it is impossible to show the results for all the tested algorithms, on all the families of instances, the above findings can be easily confirmed by looking at Table 1, which refers to the instances NON-UNIF with processing times in [1,100]. The Table shows the results (in terms of relative error and running time) of $1-SPT$ with the alternative implementations of the set Q, together with the results di $1-Search$ and the results of $1-SPT$ on the partial graph $\bar{G}(S)$ ($\bar{G}$ in the table), with Q implemented as a deque.

In our experience, no dominance was detected, on the other hand, between the algorithms looking for K-disjoint cycles (paths) and those looking for 1-disjoint cycles (paths), and between the variants based on the shortest path or on the bottleneck path tree paradigm. However, a form of dominance can be established for some particular classes of instances, as will be shown later on.

Due to the above results, for the second phase of the computational investigation we selected the versions of the heuristics $1-SPT$, $1-BPT$, $K-SPT$ and $K-BPT$ which visit the entire graph $G(S)$, and which implement the set Q using the *deque* data structure.

4.5 Computational results: the second phase

The aim of the second phase was to assess the potential of the multi-exchange algorithms. This was done by comparing the best versions of the multi-exchange heuristics, as emerged from the first phase, with other algorithms in the literature. For the comparison, we considered both the quality of the obtained solution and the time required to compute it.

An interesting by-product of our computational investigation is the awareness that different classes of instances have different "difficulty". In particular the UNIFORM instances, often used as benchmark instances in the literature, are generally very easy (most authors have developed their own set of benchmark instances, but only the generator of (França et al.,1994) was available). Because of that, we will present the results separately for each family of instances.

In the tables, columns $K-SPT$, $1-SPT$, $K-BPT$and $1-BPT$describe the results of the corresponding multi-exchange heuristics (visiting the entire graph $G(S)$ and implementing Q as a deque). Results are averaged for a group of 10 instances, and are given both in terms of the *relative* *error* with respect to the best lower bound known to date (column *r. gap*) and in terms of the total running time, included the generation of the initial solution (column *sec*). We also report the number of the computed cycles (column *cy.*) and their average cardinality (column *mlc*).

Columns LPT, $MTB\&B$ and $FMHEU$describe the results of the constructive heuristic LPT, of the algorithm of (Dell'Amico and Martello, 1995) and of the code of (França et al., 1994), respectively, both in time and in gap. In some cases we also report the number of instances that have been solved to optimality (column *o.*) For the algorithm $MTB\&B$, without further notice a limit of 4000 backtracks is intended. This is because the instances that the branch-and-bound method could not solve within 4000 backtracks could not be solved even with 40000 bactracks, and the improvement in the gap was negligible, while the running time grew considerably. Larger numbers of allowed backtracks resulted in an analogous situation, i.e., no significant improvements and longer running times.

UNIFORM instances

The UNIFORM instances are pretty "easy", in the sense that most of them can be solved to optimality in a very little time. This is shown in Table 2, where the results obtained by LPT, by $FMHEU$and by $MTB\&B$ are reported. Three sets of results are reported for each algorithm, which corresponds, from left to right, to UNIFORM instances with processing times in the range $[1-100]$, $[1-1000]$ and $[1-10000]$, respectively.

The results clearly show that these instances can be efficiently approached with these algorithms. LPT usually obtains reasonably low gaps, and solves to optimality a fair number of instances. $FMHEU$obtains results comparable with, and often better than, LPT. $MTB\&B$ solves all the $[1-100]$ instances, all but one of the $[1-1000]$ instances, and all but 24 of the $[1-10000]$ ones. Due to these results, the uniform instances seem not suited for sophisticated local search approaches, like multi-exchange. In particular, the multi-exchange heuristics improved sensibly the solutions computed by LPT, but at a high computational cost.

It is interesting to discuss some trends which emerge from the pool of the obtained results. First of all, counting the instances solved to optimality, we can see that the instances seem to become more "difficult" as the range of their processing times grows. Moreover, the instances with processing times in $[1-10000]$ which are not solved to optimality by the branch-and-bound method are those for which LPT obtains the worst gaps: these instances seem to be characterized by an n/m ratio which is neither too small nor too large, i.e., larger than 2 but smaller or equal to 10.

Thus, the UNIFORM instances are generally "easy"; however, there are some "difficult" instances, characterized by an intermediate n/m ratio. This is an interesting finding, and it will be confirmed by the next set of results.

NON-UNIF instances

The results for the NON-UNIF instances are shown in Table 3, Table 4 and Table 5 for the three subsets of instances with processing times in $[1-100]$, $[1-1000]$ and $[1-10000]$, respectively. The tables report the results obtained by LPT, $FMHEU$and $MTB\&B$. Moreover, they show the results obtained by $1-SPT$, $1-BPT$ and $K-SPT$. The results of $K-BPT$ are very similar to those of $K-SPT$, and are not reported here.

The first observation is that the results show a trend which is analogous to the one observed for the uniform instances, i.e., the most "difficult" instances seem to be those with $2 < n/m \leq 10$. However, for large values of n, also instances with $n/m \leq 40$ can be "difficult", although the difficulty decreases when n/m increases. In all the cases, these instances are generally more difficult than the UNIFORM instances, as both LPT and $MTB\&B$ obtain less good gaps, and solve less instances. Now, the trend about the processing times is inverse, with more $[1-10000]$ instances solved to

		LPT			*MTB&B*			*FMHEU*		*LPT*			*MTB&B*			*FMHEU*		*LPT*			*MTB&B*			*FMHEU*	
m	n	r. gap	o.	sec	r. gap	o.	sec	r. gap	sec	r. gap	o.	sec	r. gap	o.	sec	r. gap	sec	r. gap	o.	sec	r. gap	o.	sec	r. gap	sec
5	10	4.90e-03	9	0.00	0.00e00	10	0.00	3.26e-02	0.02	1.59e-03	9	0.00	0.00e00	10	0.00	4.03e-02	0.02	0.00e00	10	0.00	0.00e00	10	0.00	6.78e-03	0.02
5	50	4.23e-03	0	0.00	0.00e00	10	0.00	1.96e-04	0.02	4.27e-03	0	0.00	0.00e00	10	0.00	2.48e-04	0.02	5.44e-03	0	0.00	6.55e-06	8	0.07	5.44e-04	0.01
5	100	1.46e-03	3	0.00	0.00e00	10	0.00	0.00e00	0.02	1.22e-03	0	0.00	0.00e00	10	0.00	6.03e-05	0.02	9.84e-04	0	0.00	0.00e00	10	0.00	7.78e-05	0.02
5	500	0.00e00	10	0.00	0.00e00	10	0.00	0.00e00	0.02	2.05e-05	4	0.00	0.00e00	10	0.00	0.00e00	0.02	4.49e-05	0	0.00	0.00e00	10	0.00	3.96e-07	0.03
5	1000	0.00e00	10	0.00	0.00e00	10	0.00	0.00e00	0.02	6.00e-06	4	0.00	0.00e00	10	0.00	0.00e00	0.02	1.03e-05	0	0.00	0.00e00	10	0.00	0.00e00	0.03
10	50	1.60e-02	1	0.00	0.00e00	10	0.00	2.35e-03	0.02	2.80e-02	0	0.00	6.96e-05	9	6.30	4.43e-03	0.02	2.07e-02	0	0.00	9.18e-04	0	1.44	4.81e-03	0.02
10	100	4.93e-03	2	0.00	0.00e00	10	0.00	0.00e00	0.02	4.76e-03	0	0.00	0.00e00	10	0.00	3.43e-04	0.02	5.63e-03	0	0.00	3.98e-06	8	0.15	2.48e-04	0.02
10	500	4.10e-05	9	0.00	0.00e00	10	0.00	0.00e00	0.03	1.61e-04	0	0.00	0.00e00	10	0.00	0.00e00	0.02	1.59e-04	0	0.00	0.00e00	10	0.00	2.38e-06	0.02
10	1000	0.00e00	10	0.00	0.00e00	10	0.00	0.00e00	0.02	4.01e-05	0	0.00	0.00e00	10	0.00	0.00e00	0.03	4.81e-05	0	0.00	0.00e00	10	0.01	0.00e00	0.03
25	50	3.89e-03	7	0.00	0.00e00	10	0.00	3.42e-02	0.02	9.29e-03	8	0.00	0.00e00	10	0.00	2.33e-02	0.02	9.98e-03	7	0.00	0.00e00	10	0.00	2.08e-02	0.02
25	100	2.52e-02	0	0.00	0.00e00	10	0.00	3.62e-03	0.02	2.23e-02	0	0.00	0.00e00	10	0.01	4.04e-03	0.03	3.27e-02	0	0.00	4.11e-03	0	6.39	4.33e-03	0.02
25	500	5.04e-04	5	0.00	0.00e00	10	0.00	0.00e00	0.02	1.06e-03	0	0.00	0.00e00	10	0.00	0.00e00	0.03	1.31e-03	0	0.00	0.00e00	10	0.01	2.28e-05	0.03
25	1000	1.01e-04	8	0.00	0.00e00	10	0.00	0.00e00	0.02	2.81e-04	0	0.00	0.00e00	10	0.00	0.00e00	0.02	3.55e-04	0	0.00	0.00e00	10	0.01	3.52e-06	0.03

Table 2: The UNIFORM instances

		LPT			$MTB\&B$			$FMHEU$		$1-SPT$				$1-BPT$				$K-SPT$			
m	n	r. gap	o.	sec	r. gap	o.	sec	r. gap	sec	r. gap	sec	cy.	mlc	r. gap	sec	cy.	mlc	r. gap	sec	cy.	mlc
5	10	0.00e00	10	0.00	0.00e00	10	0.00	7.52e-03	0.00	0.00e00	0.00	0	0.0	0.00e00	0.00	0	0.0	0.00e00	0.00	0	0.0
5	50	1.80e-02	0	0.00	1.46e-02	0	30.44	1.54e-02	0.01	8.58e-03	0.01	19	2.5	8.80e-03	0.01	23	2.2	1.50e-02	0.01	6	3.4
5	100	8.33e-03	0	0.00	5.31e-04	9	3.99	6.05e-03	0.01	5.31e-05	0.02	34	2.4	1.59e -04	0.03	44	2.2	5.68e-03	0.11	7	3.2
5	500	6.06e-04	0	0.00	0.00e00	10	0.01	2.03e-03	0.00	0.00e00	1.44	10	2.3	0.00 e00	1.01	11	2.1	3.19e-05	4.83	4	3.1
5	1000	1.38e-04	2	0.00	0.00e00	10	0.00	1.25e-03	0.01	0.00e00	12.01	5	2.4	0.0 0e00	9.71	5	2.2	0.00e00	12.69	3	2.6
10	50	1.95e-02	0	0.00	1.65e-02	0	148.67	1.42e-02	0.01	1.57e-02	0.00	10	2.4	1.63e-02	0.00	11	2.2	1.87e-02	0.02	1	2.7
10	100	9.55e-03	0	0.00	8.70e-03	0	141.19	7.31e-03	0.01	5.09e-03	0.04	28	2.4	5.52e-03	0.02	30	2.1	9.02e-03	0.20	1	3.6
10	500	8.24e-03	0	0.00	4.26e-05	9	11.19	6.63e-03	0.01	2.13e-05	3.26	183	2.2	2. 13e-05	3.39	207	2.1	5.52e-04	30.05	34	5.2
10	1000	7.86e-04	0	0.00	0.00e00	10	0.11	2.74e-03	0.01	0.00e00	20.21	28	2.1	0 .00e00	17.14	31	2.1	1.70e-04	197.51	6	5.3
25	50	0.00e00	10	0.00	0.00e00	10	0.00	5.28e-03	0.00	0.00e00	0.01	0	0.0	0.00e00	0.01	0	0.0	0.00e00	0.01	0	0.0
25	100	1.17e-02	0	0.00	1.04e-02	0	1508.43	7.44e-03	0.00	9.85e-03	0.04	10	2.4	1.01e-02	0.01	9	2.3	1.12e-02	0.13	1	8.5
25	500	8.34e-03	0	0.00	5.84e-04	6	252.18	6.43e-03	0.01	2.12e-04	5.78	267	2.4	2 .12e-04	3.29	299	2.1	8.23e-03	218.07	1	3.5
25	1000	8.39e-03	0	0.00	1.86e-04	7	276.32	6.16e-03	0.01	7.97e-05	52.99	461	2.9	7.97e-05	36.59	530	2.1	7.59e-03	2437.87	29	11.6

Table 3: The NON-UNIF instances [1-100]

		LPT			$MTB\&B$			$FMHEU$		$1-SPT$				$1-BPT$				$K-SPT$			
m	n	r. gap	o.	sec	r. gap	o.	sec	r. gap	sec	r. gap	sec	cy.	mlc	r. gap	sec	cy.	mlc	r. gap	sec	cy.	mlc
5	10	0.00e00	10	0.00	0.00e00	10	0.00	6.13e-03	0.00	0.00e00	0.00	0	0.0	0.00e00	0.00	0	0.0	0.00e00	0.00	0	0.0
5	50	1.76e-02	0	0.00	1.61e-02	0	30.19	1.59e-02	0.00	8.92e-03	0.01	40	2.4	9.13e-03	0.01	53	2.9	1.17e-02	0.01	32	2.4
5	100	8.48e-03	0	0.00	1.38e-04	9	7.74	6.52e-03	0.01	7.43e-05	0.06	100	2.4	9.55e-05	0.07	144	2.9	7.86e-03	0.06	72	2.5
5	500	6.74e-04	0	0.00	0.00e00	10	0.00	0.00e00	0.01	0.00e00	1.87	63	2.2	0.00 e00	1.39	67	2.1	2.35e-04	7.80	29	2.4
5	1000	1.39e-04	0	0.00	0.00e00	10	0.01	1.88e-04	0.01	0.00e00	18.31	33	2.2	0.00e00	14.80	38	2.0	2.44e-05	25.88	16	2.4
10	50	1.83e-02	0	0.00	1.58e-02	0	177.34	1.42e-02	0.01	1.46e-02	0.01	33	2.6	1.48e-02	0.01	32	2.3	1.74e-02	0.01	11	2.5
10	100	8.98e-03	0	0.00	8.45e-03	0	123.39	7.18e-03	0.01	4.64e-03	0.07	99	2.6	4. 92e-03	0.05	103	2.3	8.70e-03	0.09	25	3.4
10	500	8.38e-03	0	0.00	0.00e00	10	0.10	6.46e-03	0.01	0.00e00	5.61	789	2.4	0. 00e00	6.24	984	2.2	8.37e-03	9.96	224	2.0
10	1000	8.61e-04	0	0.00	0.00e00	10	0.61	4.53e-04	0.01	0.00e00	21.81	212	2.2	0.00e00	14.61	221	2.1	5.78e-04	89.23	51	2.6
25	50	0.00e00	10	0.00	0.00e00	10	0.00	4.03e-03	0.01	0.00e00	0.01	0	0.0	0.00e00	0.01	0	0.0	0.00e00	0.01	0	0.0
25	100	9.84e-03	0	0.00	8.88e-03	0	1115.34	7.50e-03	0.00	7.93e-03	0.08	54	2.8	8.27e-03	0.02	39	2.6	9.47e-03	0.08	3	3 .4
25	500	8.37e-03	0	0.00	9.23e-04	2	1971.29	6.59e-03	0.01	4.25e-05	15.39	1794	2.5	2.18e-04	10.88	2156	2.4	8.36e -03	10.84	51	6.0
25	1000	8.53e-03	0	0.00	7.18e-05	7	658.17	6.18e-03	0.01	7.97e-06	138.21	3335	2 .4	7.97e-06	88.55	3957	2.3	8.53e-03	91.52	208	2.7

Table 4: The NON-UNIF instances [1-1000]

		LPT			$MTB\&B$			$FMHEU$		$1-SPT$				$1-BPT$				$K-SPT$			
m	n	r. gap	o.	sec	r. gap	o.	sec	r. gap	sec	r. gap	sec	cy.	mlc	r. gap	sec	cy.	mlc	r. gap	sec	cy.	mlc
5	10	0.00e00	10	0.00	0.00e00	10	0.00	6.18e-03	0.00	0.00e00	0.00	0	0.0	0.00e00	0.00	0	0.0	0.00e00	0.00	0	0.0
5	50	1.76e-02	0	0.00	1.63e-02	0	26.35	1.58e-02	0.01	8.96e-03	0.01	44	2.4	8.95e-03	0.01	56	2.2	8.84e-03	0.01	49	2.2
5	100	8.51e-03	0	0.00	4.26e-04	5	48.78	6.64e-03	0.01	5.78e-05	0.09	116	2.4	6.7 8e-05	0.09	176	2.2	5.78e-05	0.05	136	2.2
5	500	6.78e-04	0	0.00	0.00e00	10	0.02	0.00e00	0.01	0.00e00	1.97	110	2.2	0.00e00	1.59	136	2.1	6.69e-06	5.0 8	81	2.3
5	1000	1.39e-04	0	0.00	0.00e00	10	0.01	0.00e00	0.01	0.00e00	19.19	94	2.3	0. 00e00	13.88	111	2.1	9.77e-06	67.91	51	2.5
10	50	1.82e-02	0	0.00	1.60e-02	0	145.27	1.42e-02	0.01	1.46e-02	0.02	41	2.7	1.46e-02	0.01	41	2.3	1.47e-02	0.01	39	2.3
10	100	8.93e-03	0	0.00	8.54e-03	0	42.05	7.25e-03	0.01	4.63e-03	0.15	130	2.7	4.77e-03	0.09	123	2.3	4.68e-03	0.1 0	98	2.6
10	500	8.40e-03	0	0.00	0.00e00	10	0.17	6.55e-03	0.01	1.91e-06	7.15	1078	2.5	1 .06e-06	7.99	1312	2.2	4.89e-06	12.90	1026	2.7
10	1000	8.68e-04	0	0.00	0.00e00	10	0.23	0.00e00	0.01	0.00e00	15.57	388	2.4	2.13e-07	11.41	399	2.2	2.02e-06	9 3.07	303	3.4
25	50	0.00e00	10	0.00	0.00e00	10	0.00	4.09e-03	0.01	0.00e00	0.01	0	0.0	0.00e00	0.01	0	0.0	0.00e00	0.01	0	0.0
25	100	9.75e-03	0	0.00	9.12e-03	0	659.75	7.58e-03	0.01	7.76e-03	0.14	85	3.1	7.90e-03	0.06	57	2.6	8.02e-03	0.24	40	4.3
25	500	8.36e-03	0	0.00	2.71e-04	1	8185.42	6.58e-03	0.01	2.07e-05	23.20	3138	3	1.91e-05	20.37	4211	2.6	6.16e-05	51.98	1184	6.6
25	1000	8.54e-03	0	0.00	0.00e00	10	6.86	6.20e-03	0.02	5.31e-07	195.88	7656	2.8	7.97e-07	141.18	9204	2.5	2.03e-04	656.47	2473	6.2

Table 5: The NON-UNIF instances [1-10000]

optimality than the corresponding $[1-1000]$ instances, which in turn appear to be "easier" than the corresponding $[1-100]$ instances.

On the "difficult" cases, $1-SPT$ and $1-BPT$ often dominate $MTB\&B$, providing substantially better gaps in much less running time. We notice however that, when the number of jobs, n, increases, the running time of the multi-exchange heuristics grows faster than the one of $MTB\&B$, up to the point that, for large values of n, the latter is comparable to, or even better than, the former (yet, for large values of n the multi-exchange heuristics are sometimes outperformed also in terms of gap). This is probably due to the fact that, in those cases, the improvement graph $G(S)$ is almost dense, and therefore the number of the arcs grows about quadratically with respect to n. As far as the comparison with LPT and $FMHEU$is concerned, the multi-exchange algorithms usually obtain consistently better gaps, sometimes at a higher computational cost.

Finally, no strict dominance can be established between $1-SPT$ and $1-BPT$: $1-SPT$ usually provides a better gap, but it is generally slower than $1-BPT$. $K-SPT$ is dominated, sometimes consistently, by $1-SPT$ and $1-BPT$, both in time and in gap. This phenomenon can be explained by looking at the average cardinality of the improving cycles found by $K-SPT$, which is typically around 3 for $m=10$ and, with few exceptions, around 6 or less for $m=25$. Thus, relatively few machines are loaded in each solution, and therefore it is probably convenient to look for moves which improve the completion time for at least one loaded machine at a time. Yet, K-disjoint cycles are harder to find that 1-disjoint cycles, thus it is possible that $K-SPT$ stops since it is not able to detect improvement moves.

In conclusion, the results on this class of instances show that $1-SPT$ and $1-BPT$ are competitive on the "difficult" instances. $K-SPT$ seems to be less competitive on the instances with relatively few (loaded) machines, since in those cases it seems to be advantageous to compute 1-disjoint cycles, which are easier to find.

BINPACK instances

The results obtained for the BINPACK instances are shown in Table 6. These instances present a curious mix of "difficult" and "easy" instances, in the sense that roughly half of them were solved very quickly to optimality by $MTB\&B$, whereas the others were not solved even by allowing much more than 4000 backtracks. Yet, difficult and easy instances are about evenly distributed, almost irrespective of their size (although it seems that the larger instances tend to be slightly easier than the smaller ones), so that we could not find any property to explain this huge difference in the behaviour of the code. We can only note that the n/m ratio is always greater than 2 but strictly smaller than 3. Thus, also these instances seem to be critical ones.

The behaviour of the multi-exchange algorithms is quite interesting. First of all, again, $1-SPT$ and $1-BPT$ generally outperform $K-SPT$ and $K-BPT$. This is especially true for $K-BPT$, which seems to be uncapable of finding improvement cycles, and in all the cases finds very few cycles, obtaining a final gap which is pretty close to the (bad) one of the initial solution provided by LPT. On the contray, $K-SPT$ finds a fair number of K-disjoint cycles, and it computes a gap which is competitive with the one of $1-SPT$ and $1-BPT$, but at a higher computational time.

The interesting observation is that the average cardinality of the computed K-disjoint cycles is very large, which probably means that very many machines are loaded in the solutions that are generated. Thus, finding these long K-disjoint cycles is probably "difficult", which explains why $K-SPT$ is so much slower than $1-SPT$ and $1-BPT$ (where the average cardinality of the computed cycles is always less than 3). As far as $1-BPT$ and $1-SPT$ are concerned, the former usually obtains slightly worse gaps, but it is usually faster. Thus, for these instances the cost definition behind $1-SPT$ seems to provide a remarkably more efficient guide for computing improvement moves, especially the K-disjoint cycles which are more difficult to find.

$MTB\&B$ and the best multi-exchange heuristics obtain, on average, quite comparable gaps in

			LPT		$MTB\&B$			$1-SPT$				$1-BPT$				$K-SPT$				$K-BPT$			
	m	n	r. gap	sec	r. gap	o.	sec	r. gap	sec	cy.	mlc	r. gap	sec	cy.	mlc	r. gap	sec	cy.	mlc	r. gap	sec	cy.	mlc
u120	48	120	1.22e-01	0.00	2.04e-02	3	6.40	2.30e-02	0.25	168	2.95	2.71e-02	0.11	139	2.5	4.06e-02	1.83	12	20.9	1.16 e-01	0.19	1	3.3
u120	50	120	1.38e-01	0.00	1.65e-02	3	3.02	1.90e-02	0.31	185	2.94	2.65e-02	0.13	146	2.5	4.07e-02	1.89	13	22.0	1.29 e-01	0.23	2	3.5
u250	102	250	1.35e-01	0.00	1.93e-02	4	8.03	2.28e-02	3.00	462	2.85	1.54e-02	1.38	469	2.6	4.84e-02	28.54	13	47.2	1.3 2e-01	1.75	1	2.8
u250	102	250	1.33e-01	0.00	1.68e-02	8	5.01	1.94e-02	2.48	460	2.87	1.74e-02	1.21	436	2.6	4.75e-02	26.83	13	43.9	1.2 7e-01	1.83	1	3.1
u500	203	500	1.43e-01	0.00	2.68e-02	4	49.10	2.48e-02	21.73	1131	2.75	5.55e-02	5.33	705	2.6	5.69e-02	440.82	12	91.0	1.40e-01	13.43	1	2.8
u500	200	500	1.29e-01	0.00	3.36e-02	9	15.09	1.33e-02	25.92	1097	2.74	8.87e-02	2.20	194	2.5	4.47e-02	447.17	12	98.4	1.27e-01	14.14	1	5.5
u1000	402	1000	1.40e-01	0.00	2.68e-02	7	222.00	1.80e-02	220.90	2776	2.69	2.60e-02	72.78	2451	2.6	6.01e-02	2611.98	11	192.2	1.39e-01	107.72	1	2.0
u1000	399	1000	1.33e-01	0.01	2.68e-02	6	266.76	2.67e-02	191.95	2305	2.65	5.15e-02	54.97	1720	2.6	6.62e-02	2246.92	9	190.6	1.32e-01	103.86	1	2.0

Table 6: The BINPACK instances

roughly the same time. However, no clear winner emerges in this context. In fact, $MTB\&B$ solves about half of the problems to optimality, in a very quick way, and therefore it is preferable in those cases. However, on the "difficult" instances the multi-exchange heuristics are generally competitive, both in gap and in time, up to the point that they almost balance the worse behavior on the "easy" instances.

TRIPLET instances

The results obtained on the TRIPLET instances are shown in Table 7. These instances are almost all "difficult", i.e., with very few exceptions, they were not solved by $MTB\&B$. Observe that these instances have, by construction, an n/m ratio which is exactly 3: this further confirms the findings of all the previous cases.

On these instances, the multi-exchange heuristics are preferable, in terms of gap, with respect to $MTB\&B$, although they require sometimes a longer running time. In this case, moreover, $K-SPT$ outperforms all the other heuristics, included $1-BPT$ and $1-SPT$. As in the previous case, instead, $K-BPT$ seems to be not capable of finding enough improvement cycles, and it quickly terminates with a gap which is not consistently smaller than the one returned by LPT. Curiously enough, the reverse happens for the algorithms based on 1-disjoint cycle computations: $1-BPT$ outperforms $1-SPT$ in terms of gap, slightly but uniformly.

The dominance of $K-SPT$ can be explained in terms of the average cardinality of the K-disjoint cycles found by the algorithm: as the number of machines, m, increases, longer and longer cycles are found, but their cardinality is relatively small compared to the total number of the machines, unlike what happens in the case of BINPACK (this implies that the number of loaded machines is usually small). Thus, for the TRIPLET instances finding K-disjoint cycles appears to be "easier", and therefore it seems to be preferable to improve the current makespan by considering all the loaded machines, rather than improving the completion time for at least one loaded machine. Finally, it is interesting to observe that the difference in gap between $K-SPT$ and $1-BPT$ (the best algorithm for computing 1-disjoint cycles) generally grows with m: $K-SPT$ dominates $1-BPT$ as the K-disjoint cycles become more and more longer than the 1-disjoint cycles.

To summarize the computational results reported in this section, even "pure" local search heuristics based on a multi-exchange neighborhood outperform existing approaches on "difficult" instances of $P||C_{max}$. Moreover, as far as the alternative multi-exchange heuristics are concerned, some interesting guidelines emerged. One of them is that finding "global" improvement moves (i.e., K-disjoint cycles and paths rather than 1-disjoint cycles and paths) is preferable only if the cardinality of the cycles that have to be found is neither too small (since otherwise the two kinds of moves are basically coincident), nor too large (since otherwise finding K-disjoint cycles may be too difficult). Also, the combination of the K-disjoint cycles computation with a bottleneck-type searching strategy does not appear to be promising, whereas both label correcting and bottleneck path heuristics are worth testing when 1-disjoint cycles are looked for.

5 Conclusions

In this paper, we have proposed a large family of new local search algorithms based on a multi-exchange neighborhood for minimum makespan machine scheduling problems. The proposed neighborhood has quite different properties with respect to the other multi-exchange neighborhoods previously proposed in the literature, and some interesting theoretical results have been obtained.

By means of an extensive computational investigation, where a large number of algorithms have been tested on a large data set, we have obtained a set of guidelines for the implementation of this kind of algorithms. Although the results have been obtained for the case of identical machines, we

			LPT		$MTB\&B$			$1-SPT$				$1-BPT$				$K-SPT$				$K-BPT$			
	m	n	r. gap	sec	r. gap	o.	sec	r. gap	sec	cy.	mlc	r. gap	sec	cy.	mlc	r. gap	sec	cy.	mlc	r. gap	sec	cy.	mlc
t60	20	60	2.27e-02	0.00	5.70e-03	3	1.30	5.80e-03	12.41	60019	2.4	7.00e-03	3.81	20017	2.1	6.00e-03	15.78	20017	2.7	9.50e-03	1.83	25	2.7
t60	20	60	2.08e-02	0.00	6.20e-03	0	1.59	8.10e-03	15.04	40025	2.8	7.00e-03	5.70	40019	2.3	6.00e-03	1.83	20014	2.8	1.30e-02	0.03	11	2.8
t120	40	120	2.79e-02	0.00	1.96e-02	1	3.95	8.70e-03	81.04	100085	2.8	7.00e-03	44.03	60107	2.7	6.00e-03	19.03	20068	6.1	2.53e-02	0.25	4	5.6
t120	40	120	2.37e-02	0.00	1.72e-02	0	4.46	9.60e-03	79.26	140044	2.5	9.00e-03	110.67	140024	2.6	6.00e-03	0.96	20066	6.5	2.09e-02	0.32	5	6.0
t249	83	249	2.52e-02	0.00	2.32e-02	0	10.89	1.63e-02	270.40	132513	2.4	1.00e-02	259.31	133991	2.5	8.00e-03	79.60	73185	7.7	2.26e-02	1.99	3	2.4
t249	83	249	2.51e-02	0.00	2.16e-02	0	11.34	1.64e-02	277.56	132769	2.5	2.00e-02	253.32	133824	2.6	8.00e-03	80.04	60064	9.5	2.13e-02	2.28	4	2.8
t501	167	501	2.77e-02	0.00	2.58e-02	0	51.20	2.31e-02	273.50	106015	2.1	2.00e-02	243.73	86035	2.8	1.00e-02	330.25	68566	9.6	2.69e-02	17.76	2	3.4
t501	167	501	2.65e-02	0.00	2.49e-02	0	46.75	2.08e-02	296.64	107391	2.1	2.00e-02	289.54	60721	2.6	1.00e-02	343.47	56212	13.3	2.57e-02	16.61	2	4.5

Table 7: The TRIPLET instances

believe that the obtained insights can be useful for other minimum makespan machine scheduling problems, and even for different problems with a similar structure.

Testing the multi-exchange algorithms against other algorithms from the literature, we have shown that some algorithms based on the multi-exchange neighborhood are promising, in terms of quality and time, for "difficult" $P||C_{max}$ instances. Moreover, we have collected an interesting set of $P||C_{max}$ instances and analyzed their characteristics. This may be of help in the development of new and more efficient algorithms for "difficult" $P||C_{max}$ instances.

In conclusion, we believe that local search algorithms based on multi-exchanges can be successfully applied to minimum makespan machine scheduling problems. Relatively simple variants, lacking any metaheuristic guidance, are capable of producing better quality solution than other algorithms from the literature for some classes of instances. Our work has hopefully set the ground for more efficient implementations of multi-exchange heuristics for minimum makespan machine scheduling problems, comprised the cases of the uniform and the unrelated machines. Also, we believe that our research may provide good insights for the construction of multi-exchange heuristics for other problems with a bottleneck-type objective function; research in this field is ongoing, and it will be the subject of a future paper.

Acknowledgements We thank F.M. Muller for having provided the code $FMHEU$ and the generator for the family of the uniform instances. We also thank M. Dell'Amico for having provided the code of the exact algorithm used during the computational testing. Finally, many thanks to R.K. Ahuja and to S. Pallottino for their helpful comments and suggestions.

References

J.E. Anderson, C.A. Glass and C.N. Potts, "Machine scheduling", in Local Search in Combinatorial Optimization, Aarts and Lenstra Eds., Wiley, pp. 361-414, 1997.

R.K. Ahuja, O. Ergun, J.B. Orlin and A.P. Punnen, "A survey of very large-scale neighborhood search techniques", Working Paper, Center for Applied Optimization, University of Florida, 1999.

R.K. Ahuja, T.L. Magnanti and J.B. Orlin, "Network Flows: theory, algorithms and applications", Prentice Hall, New Jersey, 1993.

R.K. Ahuja, J.B. Orlin and D. Sharma, "New neighborhood search structures for the capacitated minimum spanning tree problem", Technical Report, Center for Applied Optimization, University of Florida, 1998.

B.V. Cherkassky and A.V. Goldberg "Negative cycle detection algorithms", Mathematical Programming 85, pp. 277-311, 1999.

E.G. Coffman Jr., M.R. Garey and D.S. Johnson, "An application of bin packing to multiprocessor scheduling", SIAM Journal on Computing 7, pp. 1-17, 1978.

M. Dell'Amico and S. Martello, "Optimal scheduling of tasks on identical parallel processors", ORSA Journal on Computing 7 (2), pp. 181-200, 1995.

S.M. Fatemi-Ghomi and F. Jolai-Ghazvini, "A pairwise interchange algorithm for parallel machine scheduling", Production Planning and Control 9(7), pp. 685-689, 1998.

G. Finn and E. Horowitz, "A linear time approximation algorithm for multiprocessor scheduling", BIT 19, pp. 312-320, 1979.

P.M. França, M. Gendrau, G. Laporte and F.M. Muller, "A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective", Computers Ops. Res. 21(2), pp. 205-210, 1994.

M.R. Garey and D.S. Johnson, "Strong NP-completeness results: motivation, examples and implications", J. Ass. Comp. Mach. 25, pp. 499-508, 1978.

M. Gendrau., F. Guertin, J. Potvin and R. Seguin, "Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries", Publication CRT-98-10, Centre de Recherche sur les Transports, Université de Montreal, 1999.

C.A. Glass, C.N. Potts and P. Shade, "Unrelated parallel machine scheduling using local search", Mathematical and Computer Modelling 20, pp. 41-52, 1994.

F. Glover, "Ejection chains, reference structures and alternating path methods for traveling salesman problems", Discrete Applied Mathematics 65, pp. 223-253, 1996.

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

R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnoy-Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey", Annals of Dicrete Mathematics 5, pp. 287-326, 1979.

A.M.A. Hariri and C.N. Potts, "Heuristics for scheduling unrelated parallel machines", Computers Ops. Res. 18, pp. 323-331, 1991.

R. Hübscher and F. Glover, "Applying tabu search with influential diversification to multiprocessor scheduling", Computers Ops. Res. 21(8), pp. 877-884, 1994.

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

E.L. Lawler., J.K. Lenstra, A.H.G. Rinnoy-Kan and D.B. Shmoys, "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, pp. 445-522, 1993.

J.K. Lenstra, D.B. Shmoys and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines", Mathematical Programming 46, pp. 259-271, 1990.

S. Martello, F. Soumis and P. Toth. "An Exact Algorithm for Makespan Minimization on Unrelated Parallel Machines", Integer Programming and Combinatorial Optimization, (proc. of the Second IPCO Conference), Carnegie-Mellon University, Pittsburgh, E. Balas, G. Cornuejols and R. Kannan, Eds., pp. 181-200, 1992.

E. Necciari, "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, 1999.

C. Rego, "Relaxed tours and path ejections for the traveling salesman problem", European Journal of Operational Research 106, pp. 522-538, 1998.

P.M. Thompson, "Local search algorithms for vehicle routing and other combinatorial problems", Ph.D. Thesis, Operations Research Center, MIT, Cambridge, 1988.

P.M. Thompson and J.B. Orlin, "Theory of cyclic transfers", Working paper, Operations Research Center, MIT, 1989.

P.M. Thompson and H.N. Psaraftis, "Cyclic transfer algorithms for multivehicle routing and scheduling problems", Operations Research 41(5), pp. 935-946, 1993.

S.L. Van De Velde, "Duality-based algorithms for schedling unrelated parallel machines", Orsa Journal on Computing 5, pp. 192-205, 1993.