Annals of Operations Research 41(1993)157–183

Routing and scheduling in a flexible job shop by tabu search

Paolo Brandimarte

Dipartimento di Sistemi di Produzione ed Economia dell'Azienda, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

Abstract

A hierarchical algorithm for the flexible job shop scheduling problem is described, based on the tabu search metaheuristic. Hierarchical strategies have been proposed in the literature for complex scheduling problems, and the tabu search metaheuristic, being able to cope with different memory levels, provides a natural background for the development of a hierarchical algorithm. For the case considered, a two level approach has been devised, based on the decomposition in a routing and a job shop scheduling subproblem, which is obtained by assigning each operation of each job to one among the equivalent machines. Both problems are tackled by tabu search. Coordination issues between the two hierarchical levels are considered. Unlike other hierarchical schemes, which are based on a one-way information flow, the one proposed here is based on a two-way information flow. This characteristic, together with the flexibility of local search strategies like tabu search, allows to adapt the same basic algorithm to different objective functions. Preliminary computational experience is reported.

1. Introduction: the flexible job shop scheduling problem

In the classical Job Shop (JS) scheduling problem [15] the process plan of a part consists of the sequence of the machines the part must visit: there is an a priori assignment of operations to machines. In the Flexible Job Shop (FJS) scheduling problem the assignment of operations to machines is not a priori fixed. For each job a process plan is given consisting of a sequence of operations. For each operation a set of equivalent machines is available with possibly different processing times, and a joint routing and scheduling problem must be solved.

The problem is characterized by the following data:

- a set of jobs $J_i (i = 1, \ldots, N)$;
- a set of machines $m_j (j = 1, \ldots, M)$;
- for each job J_i a sequence $\mathcal{O}_i$ of n_i operations is given forming its process plan; the jth operation $(j = 1, \ldots, n_i)$ of job j_i is denoted by o_{ij};
- for each operation o_{ij} the set $\mathcal{E}_{ij}$ of machines able to perform it is given;

- for each machine $m_l \in \mathcal{C}_{ij}$ able to execute operation o_{ij}, a processing time p_{ijl} is given (it may be the case that the processing time is the same for every machine in $\mathcal{C}_{ij}$ or not).

The problem consists of:

- a routing subproblem, that is, assigning each operation o_{ij} to a machine $m_l \in \mathcal{C}_{ij}$;
- a scheduling subproblem, that is, sequencing the assigned operations on each machine in order to obtain a globally feasible schedule minimizing a given objective function.

A wide class of objective functions can be devised based on the completion times of the jobs. Let C_i be the completion time of J_i; here we will consider the following two objective functions:

- minimum makespan, i.e.
$$\min\left(\max_{i=1,\ldots,N} C_i\right);$$
- minimum (total) weighted tardiness, i.e.
$$\min \sum_{i=1}^{N} v_i T_i,$$
where the tardiness $T_i = \max\{0, C_i - d_i\}$ is the amount by which the completion time exceeds the due date d_i and the weight v_i measures the priority of jobs.

These objective functions have been chosen since they seem to be good representatives of the different evaluation criteria that could be considered: the first one deals with machine utilization issues and is of the minmax type, the second one deals with customer service and is of the minsum type. The focus of the paper is on the minimum makespan, which is the objective function traditionally considered in most literature on JS/FJS scheduling, but the possibility of adapting the proposed method to the minimum weighted tardiness has been a major concern.

1.1. OVERVIEW OF THE LITERATURE

When facing a complex problem, like scheduling a flexible job shop or a flexible manufacturing system (FMS), two basic approaches are available.

(1) A *concurrent* approach, which is based on the idea of solving the routing and scheduling problems together; this approach is followed for example in [22, 28].

(2) A *hierarchical* approach, the more common one, which is based on the idea of decomposing the original problem in order to reduce its complexity. This

approach is followed, for example, in [2,5,11,38]. The decomposition into subproblems may be based on hierarchical control ideas [2], on the statement of a logical sequence of subproblems [11,38] or on the separation of "easy" and "difficult" constraints [5]. In the FJS case the simplest hierarchical approach is based on the observation that when a routing is chosen, the flexible job shop problem turns into the classical job shop problem. Therefore routing and scheduling can be separated. Hierarchical architectures can be further classified according to the information flow among the different levels: in a *one-way* scheme a higher level problem is first solved, and then a lower level problem is solved once; alternatively, in a *two-way* approach, there is an iteration between the two steps, and from the solution of the lower level problem some indications are obtained for the solution of the higher level problem. The architecture described here can be classified as a two-way one; another example of a two-way hierarchical scheme can be found in [9].

It is worth noting that often a one-way scheme is feasible only if the scheduling objective is somewhat surrogated by an objective at the higher hierarchical levels: for example, balancing the workloads and then finding a schedule minimizing the makespan [38] is a sensible approach, since balancing the workloads means minimizing the makespan after having relaxed the precedence constraints among the operations of the process plan of each part. In the context of lot sizing by family aggregation and disaggregation [40], the aggregated objective function is essentially the same as the disaggregated one. However, when dealing with due dates, it is not obvious how to surrogate such an objective at the higher hierarchical level.

Advocates of the different approaches reason in terms of quality of the solution obtained, computational complexity and so on. However, many important practical aspects are usually overlooked. In fact, the Operations Research practitioner must face real-life issues such as the difficulty in collecting the data, the lack of a clear statement of the problem (i.e. of its constraints and of the objective), the difficulty of implementing the devised algorithm in software, and the difficulty of gaining the commitment of some managers.

All the issues lead to the necessity of being able to develop a working prototype quickly and then to refine it without disrupting its structure. The solution approach must therefore be:

- easy to implement;
- simple enough to be understood by the management;
- able to allow some form of interaction with the user;
- sufficiently general to allow low-effort adjustments (e.g. if it is realized that the wrong objectives have been chosen);
- open to future improvements;
- able to yield reasonably good solutions with a reasonable computational effort.

Local search algorithms (see section 2), such as simulated annealing [23,41] and tabu search [16,17], seem very well suited to meet the above requirements. Indeed, the amount of literature devoted to applications of local search to scheduling is growing: applications of simulated annealing are described in [6, 32, 42], whereas tabu search has been adopted in [4, 18, 24–26, 39, 44, 45] (this list of references is far from complete).

The tabu search metaheuristic has been adopted here for many reasons, among which are the following:

- it enjoys the generality and conceptual simplicity of local search based algorithms;
- it can be used as a hierarchical algorithm due to its ability to deal with different memory levels (e.g. short, medium and long term memory); according to the principle of increasing-precision/decreasing-intelligence [36], the hierarchical structure allows to isolate the more specific knowledge in the higher levels: these can be modified or improved without affecting the lower levels;
- it actually encompasses a range of different implementations, which can be rather naive or quite sophisticated, thus allowing relatively smooth enhancements.

The purpose of this paper is to demonstrate these points.

1.2. OVERVIEW OF THE PAPER

Section 2, which is included for the sake of completeness, reviews some heuristic principles which can be used for the FJS problem, including dispatching rules (subsection 2.1) and the disjunctive graph representation (subsection 2.2).

Section 3 is devoted to the description of a hierarchical architecture for the FJS problem, based on the decomposition into a job shop scheduling subproblem (subsection 3.1) and a routing subproblem (subsection 3.2); two interaction schemes between the subproblems are discussed, resulting in a one-way and a two-way architecture. The emphasis is on the minimum makespan problem, but the adaptability of the proposed method to the minimum weighted tardiness problem is the subject of subsection 3.3.

In section 4 limited (although promising) computational results are described; in particular, issues related to the coordination of the two hierarchical levels are discussed in subsection 4.2.

Finally some conclusions are drawn in section 5, where directions for further research are outlined.

2. Heuristic principles for flexible job shop scheduling

The job shop scheduling problem is known as one of the hardest discrete optimization problems. The FJS scheduling problem is even harder. It is, therefore,

natural to look for heuristic algorithms able to provide good solutions for this problem.

Scheduling problems related to a FJS has been studied (among others) in [10, 12, 20, 30, 37]. It is not the purpose of this section to review these approaches in detail: it is rather meant to describe which *general* heuristic principles can be used for the solution of the FJS problem.

Usually general heuristic algorithms are classified as follows [35].

- *Truncated exponential schemes*

They are derived from exact algorithms, when the conditions assuring the optimality of the solutions are relaxed: an example is a heuristic version of the branch and bound methods. In an exact branch and bound scheme for a minimization problem $\mathcal{P}$, a subproblem $\mathcal{P}_k$ of $\mathcal{P}$ can be eliminated from further consideration only if a lower bound $L_B(\mathcal{P}_k)$ for its optimal value is found which is not less than the value of a known upper bound $U_B(\mathcal{P})$ for the optimal solution of $\mathcal{P}$. In a truncated exponential scheme one relaxes the condition

$$L_B(\mathcal{P}_k) \geq U_B(\mathcal{P})$$

by requiring only

$$L_B(\mathcal{P}_k) \geq (1 - \varepsilon)\, U_B(\mathcal{P}),$$

where $0 < \varepsilon < 1$. Unlike other heuristic strategies, truncated exponential schemes provide a guarantee on the quality of the solution found: i.e. one is sure to find a solution within a given $\varepsilon\%$ from the optimal one. However, truncated exponential schemes require good lower bounding procedures like their exact counterpart, which is not always possible with reasonable computational efforts and is usually highly problem dependent. Therefore, they will not be considered here.

- *Greedy algorithms*

In this class of methods a discrete optimization problem is dealt with as a sequential decision problem in which the locally optimal decision is taken at each step, usually at the expense of global optimality. Very similar to greedy algorithms are the well-known dispatching rules [34], which assign a (possibly time-dependent) priority to each job waiting on a machine. Actually some dispatching rules are obtained by applying a priority discipline which is optimal for a single machine. Some rules are treated in subsection 2.1.

In order to reduce the myopy of the greedy algorithms, one could consider at each step not only the best decision, but some among the best ones: this is the idea behind the beam search method [33]. In [10] a beam search strategy is used to minimize the makespan in a FMS similar to the FJS considered here (there, transportation times are considered). The algorithm relies on the concept of the critical path in the graph of operation precedences described in [3] for the job shop

problem. This concept is heavily used in the following, and, for those unfamiliar with it, it is briefly described in subsection 2.2.

- *Local search* algorithms are based on the idea of exploring the set of feasible solutions by perturbing a given solution and comparing the new solution with the old one. The main advantage of this idea is that, at least in principle, it works for any objective function.

The simplest local search algorithm is *local improvement*. Given a current feasible solution whose cost is C_{old}, a simple perturbation is applied to it, obtaining a candidate solution slightly different from the old one (a solution in its neighborhood). Let

$$\Delta C = C_{new} - C_{old}$$

be the difference between the cost of the candidate solution and the current one. When $\Delta C < 0$, i.e. the candidate solution is the better than the current one, the old solution is discarded in favour of candidate solution. The new current solution is perturbed and the cycle repeats until no improving candidate is found and a locally optimal solution is obtained. To avoid getting stuck in a local minimum, schemes like simulated annealing [23,41] and tabu search [16, 17] have been proposed.

2.1. DISPATCHING RULES

Dispatching rules are a distributed sequencing strategy, by which a priority is assigned to each job waiting for service on a machine: when the machine is ready to process a job, the one with maximum priority is selected.

Some dispatching rules are applications of rules which are optimal for the single machine case; others are based on heuristic insights into the scheduling problem.

In the FJS case, dispatching rules can also be used for the routing problem: when the operation o_{ij} of the process plan of J_i is to be executed, J_i is considered as waiting on each machine $m_l \in \mathcal{C}_{ij}$. The operation is assigned to the first machine on which J_i is ranked at top priority and removed from the queues of other machines.

There is a large number of such rules, oriented to different objective functions. In [34] a survey of dispatching rules can be found; their application in manufacturing systems enjoying a certain degree of flexibility is described in [21,29].

Here the following rules will be considered: their purpose will be to find an initial schedule and to provide a comparison for the proposed scheduler.

- The *Shortest Processing Time* (SPT) rule, by which the priority of job J_i waiting on machine m_l for the execution of operation o_{ij} is

$$\frac{1}{p_{ijl}}.$$

When a weight is given to a job (as in the weighted tardiness case), the weighted SPT (WSPT) rule can be used, which assigns a priority

$$\frac{v_i}{p_{ijl}}.$$

- The *Least Work Remaining* (LWKR) rule, which assigns a priority

$$\frac{1}{\sum_{q \in \mathcal{A}_{ij}} \bar{p}_{iq}}$$

where $\mathcal{A}_{ij}$ is the set of operations following o_{ij} in the process plan of J_i and

$$\bar{p}_{iq} = \frac{1}{|\mathcal{C}_{iq}|} \sum_{l \in \mathcal{C}_{iq}} p_{igl}$$

is the mean processing time* for operation o_{iq} ($|\mathcal{C}_{iq}|$ denotes the cardinality of the set $\mathcal{C}_{iq}$).

- The *Most Work Remaining* (MWKR) rule, which assigns a priority

$$\sum_{q \in \mathcal{A}_{ij}} \bar{p}_{iq}.$$

- The *Earliest Due Date* (EDD) rule, which assigns a priority

$$\frac{1}{d_i}.$$

- The *Apparent Tardiness Cost* (ATC) rule, proposed in [43], For the JS case, the priority of J_i on machine m_j at time t is

$$\frac{v_i}{p_{ij}} \exp\left(-\left[\frac{d_i - t - p_{ij} - \sum_{q \in \mathcal{A}_{ij}} (W_{iq} + p_{iq})}{kp} \right]^+ \right),$$

where $X^+ \stackrel{\text{def}}{=} \max\{0, X\}$, p_{ij} is (for the JS case) the processing time of J_i on m_j, k is a given parameter, p is the mean processing time of the jobs waiting on m_j at time t, $\mathcal{A}_{ij}$ is the set of machines on which J_i must be processed

*This rule, like other ones, has been processed for the JS case; to adapt it to the FJS case, a possible choice is to estimate the processing time of next operations, for which no routing has been taken yet, as the mean processing time over the alternative machines.

after m_j, and p_{iq} and W_{iq} are the processing time and the estimated waiting time, respectively, on the next machines.

The numerator of the argument of the exponential is an estimation of the time slack remaining before missing the due date. Note that when the shop is heavily loaded, and the job is late, the slack is negative and the ATC rule turns into the WSPT rule.

To estimate the waiting time, the following formula is suggested in [43]:

$$W_{iq} = bp_{iq},$$

where b is an appropriate parameter.

This rule can be adapted for the FJS problem as follows:

$$\frac{v_i}{p_{ijl}} \exp\left(-\left[\frac{d_i - t - p_{ijl} - \sum_{q \in \mathcal{A}_{ij}} (W_{iq} + \bar{p}_{iq})}{kp} \right]^+ \right),$$

where j refers to the operation o_{ij}, $\mathcal{A}_{ij}$ is the set of operations following o_{ij}, and $\bar{p}_{iq}$ is then the mean processing time for each operation o_{iq}, computed as in the LWKR and MWKR cases.

Again, the waiting time for operation o_{iq} is estimated as

$$W_{iq} = b\bar{p}_{iq}.$$

The last rule, unlike the previous ones, is time dependent. Furthermore it is an example of a rule depending on some parameters, which should be provided by the user. Local search algorithms can be used to learn such parameters by repeatedly simulating the application of the rule, thus yielding a one-way hierarchical scheduler (although the module setting the weights is run iteratively, we cannot speak of a two-way architecture unless the scheduling results are effectively used to guide the learning process). In [13] genetic algorithms (another class of local search algorithms, see [19]) are proposed for this purpose.

2.2. THE DISJUNCTIVE GRAPH REPRESENTATION FOR THE JOB SHOP PROBLEM

The disjunctive graph representation was introduced as a useful representation of operations precedence in the context of minimizing the makespan in a job shop [3].

We will not give a formal definition of the disjunctive graph (it can be found e.g. in [42]), but a simple example will illustrate its role in the optimal or heuristic solution of JS problems.

Given a JS problem, we associate to it a disjunctive graph as follows:

- for each operation of each job a node is created with a weight equal to its processing time;
- two dummy nodes, corresponding to an "initial" and a "final" operation are created with null weight;
- an arc is created from the initial node to the nodes corresponding to the first operation of each job; for each operation of each job an arc is created from the node corresponding to that operation to the node corresponding to the next operation (the last operation of each job is linked to the final node); such arcs represent technological precedence constraints among operations of the same job;
- the nodes corresponding to operations to be executed on the same machine are linked to each other (yielding a complete subgraph for each machine) by "disjunctive" arcs, i.e. arcs whose direction must be chosen in order to represent precedence constraints induced by sequencing decisions on each machine.

In fig. 1 a disjunctive graph is shown (for simplicity the node weights are omitted).

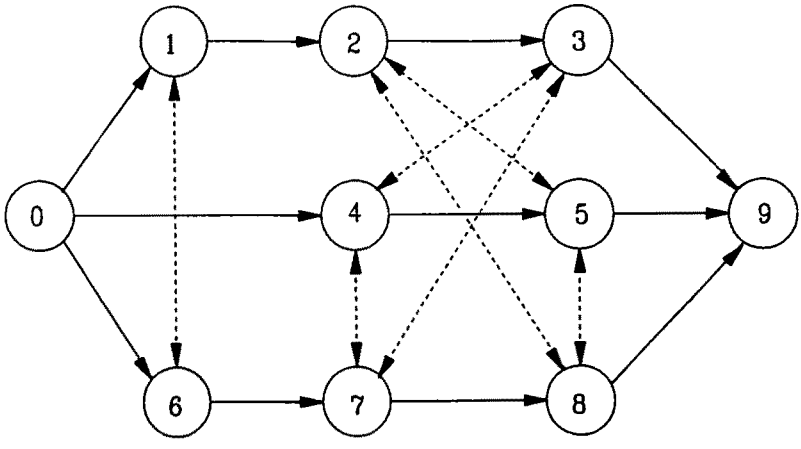

Fig. 1. A disjunctive graph for a 3-jobs 3-machines job shop problem. Nodes 0 and 9 correspond to the initial and final dummy operations. There are three jobs J_1, J_2, J_3 whose process plan consists of operations (1,2,3), (4,5) and (6,7,8), respectively. Operations {1,6} must be executed on the same machine; similarly operations {2,5,8} and {3,4,7}. Dotted lines show the disjunctive arcs corresponding to the sequencing decisions to be taken for each of these operation sets. The processing times are not shown.

When the direction of disjunctive arcs is chosen, a directed graph is obtained: if the directed graph is acyclical, it represents the operation precedences of a feasible schedule. The weight of a path connecting the first and the final nodes is the sum of the weights of the traversed nodes; the maximum-weight path is the critical path and its weight equals the makespan of the corresponding schedule.

One of the possible corresponding precedence graphs obtained for our example is shown in fig. 2.

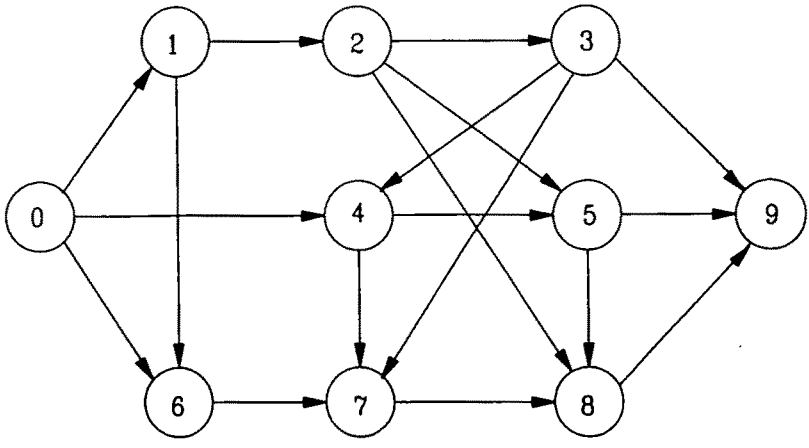

Fig. 2. An operation precedence graph for a job shop problem. The directions of the disjunctive arcs have been chosen. The resulting precedence graph corresponds to the following sequences on the machines: (J_1, J_3), (J_1, J_2, J_3) and (J_1, J_2, J_3).

Actually, the precedence graph shown in fig. 2 is redundant, since it is sufficient to consider, among the precedence constraints between operations on the same machines, only the precedence constraints among adjacent jobs in the sequence; due to an obvious "triangularity" property, the critical path of this reduced graph is the same as the previous one. As stated in [1], considering the reduced graph has a great impact on any algorithm exploiting the disjunctive graph idea. The reduced precedence graph is shown in fig. 3.

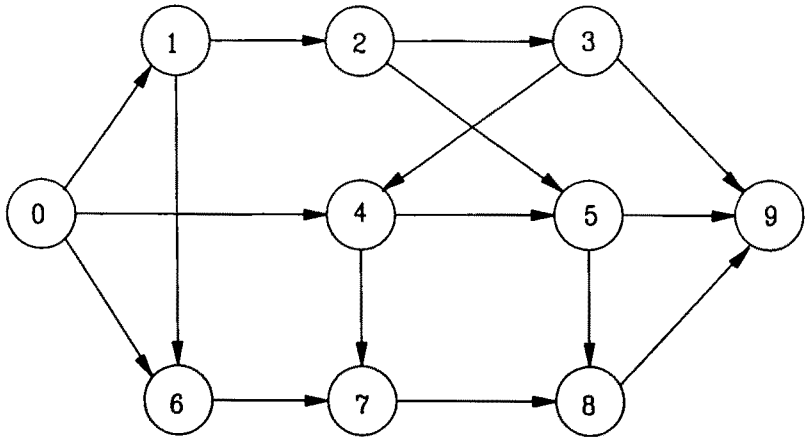

Fig. 3. Reduced operation precedence graph corresponding to the graph shown in fig. 2.

The disjunctive graph has been used both for special purpose heuristic algorithms for the JS problem [1] and for local search based methods [4, 39, 42]. This is due to two reasons:

- in order to reduce the makespan, the operations on the critical path of the precedence graph must be rescheduled;
- reversing an arc on the critical path never results in a cyclical graph (see subsection 3.1)

The disjunctive graph has also been exploited for the FJS problem: the routing can be improved by rerouting critical operations (i.e. operations whose corresponding nodes lie on the critical path) to alternative machines (see subsection 3.2). A beam search based scheduling approach based on this idea has been proposed in [10].

3. A hierarchical tabu search architecture for the flexible job shop problem

A characteristic of tabu search is its ability to cope with different hierarchical memory levels, acting on different time scales (e.g. short, medium and long term memory).

The FJS problem lends itself to a hierarchical scheme since routing and scheduling subproblems can be separated. Once a routing is chosen, a job shop problem remains to be solved.

It is therefore natural to think of a two-level tabu search algorithm, with one level dealing with routing issues and the other one dealing with job shop scheduling. This decomposition approach is also followed in [11, 12, 20, 37] among others. In [20] a branch and bound procedure for the optimal solution of the joint routing and scheduling problem is described, and, by comparing it with the decomposition approach, it is claimed that the deterioration of the solution obtained with the second method is limited.

A tabu search based FJS scheduler can be structured on three layers:

- *long term memory*, which deals with routing selection;
- *medium term memory*, which could specify the neighborhood structure, set some parameters of the tabu navigation algorithm and monitor the search progress; at present, in the prototype scheduler developed, the functionalities of this level have not been exploited; however, as shown in the following, both preliminary computational experience with the makespan problem and the need to extend the algorithm to the weighted tardiness problem call for a full development of this memory level;
- *short term memory*, which deals with low level tabu navigation for the job shop problem given a neighborhood structure.

In order to implement such a scheduler, the following issues must be considered:

(1) how to find an initial routing and an initial schedule: to this aim dispatching rules have been used;

(2) how to solve the JS scheduling subproblem (see subsection 3.1);

(3) how to solve the routing subproblem (see subsection 3.2);

(4) how to coordinate the scheduling and the routing levels (see subsection 4.2).

3.1. SOLVING THE JOB SHOP SCHEDULING SUBPROBLEM

There is a significant amount of literature on solving the JS problem, concerning both exact and heuristic methods: clearly, any heuristic algorithm, like the shifting bottleneck procedure [1], could be adopted; for small problem instances one could even use an exact method. However, a local search approach enjoys the following advantages:

- it can be adapted to different objective functions more easily;
- as shown in subsection 4.2, it is not necessary to find an extremely good solution for the JS subproblem, but it must be done quickly; the tabu search approach, with respect to similar strategies like annealing, is very appropriate to this aim, since it keeps the search process biased towards good solutions.

These reasons justify the selection of a tabu search approach for the JS subproblem.

Having selected the tabu search metaheuristic, the neighborhood structure must be chosen. The most natural neighborhood structure is obtained by exchanging two adjacent jobs in the sequence on a machine. Other types of neighborhood structures are based on exchanging arbitrary jobs and on shifting and inserting operations, as suggested in [24] for a single machine and in [31, 45] for a permutation flow shop case.

When choosing a neighborhood structure one has to pay attention to the following issues.

- The size of the candidate set must be limited. When dealing with a difficult scheduling problem, considering all the possible operations swaps would result in a huge neighborhood to explore. Furthermore many moves in this neighborhood do not affect the objective function: in the JS case, operations not lying on the critical path do not affect the makespan.

  Therefore the neighborhood must be somewhat restricted, without negatively affecting the performance of the search process. In the JS case, when the objective function is the makespan, a suitable neighborhood structure is obtained by considering only the operations lying on the critical path, whose length is the makespan [4, 39, 42]. Similar issues are considered for a parallel machines case in [18, 26].
- Also the feasibility of the candidate solutions is an issue in complex problems. In a single machine case with no precedence constraints, any schedule obtained

by operations swapping is feasible. However, in the job shop case, arbitrary perturbations of the sequence on a machine can yield a globally infeasible schedule, i.e. a schedule whose precedence graph is cyclical, even if the single machine schedules are feasible (see fig. 4).

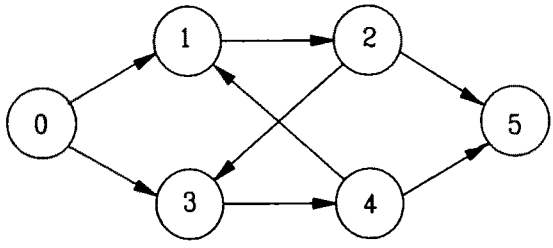

Fig. 4. A cyclical precedence graph. Job J_1 must visit machines m_1, m_2 for the execution of operations 1 and 2. Job J_2 must visit machines m_2, m_1 for the execution of operations 3 and 4. Operations 0 and 5 are dummy. The sequence on machine m_1 is (J_2, J_1). The sequence on machine m_2 is (J_1, J_2). A deadlock situation occurs.

It seems reasonable to avoid scrambling the sequences too much, since many infeasible sequences would result; therefore swapping nonadjacent operations or shifting/inserting will not be considered. Even if adjacent operations swapping is considered, one has to make sure that the resulting precedence graph is acyclical. In the minimum makespan case, the critical path of the precedence graph is again helpful: it can be shown that reversing an arc on the critical path never results in a cyclical graph [42].

The above points suggest the following possible neighborhood structures.

(1) In neighborhood $\mathcal{N}_1$ a random sampling of the potential exchanges is selected; this neighborhood structure is characterized by the size of the sample set.

(2) In neighborhood $\mathcal{N}_2$ a job is randomly selected and it is exchanged on each machine with the adjacent jobs in each machine sequence.

(3) In neighborhood $\mathcal{N}_3$ a machine is randomly selected and the complete set of job exchanges on that machine is considered.

(4) In neighborhood $\mathcal{N}_4$ operations on the critical path are considered for the exchange.

The second and the third structure are based on a sort of focusing strategy in order to restrict the search. The first neighborhood structure tries to limit the neighborhood size while keeping a sufficient degree of exploration capability by randomization.

A very nice feature of the first three neighborhoods is that, at least in principle, they work for every objective function, whereas the fourth one is makespan oriented; on the other hand they require, for each move, a feasibility check: however, if each move is evaluated by exactly computing the resulting makespan, this check is performed concurrently with the makespan evaluation.

Given these neighborhood structures, it is natural to choose as tabu attributes of a move the pair of jobs and the machine on which they are swapped: when a move is applied a tabu record with this information is added to the list of taboos. The tabu status of a move can be overridden if some aspiration criteria are satisfied. A simple choice is that the tabu status of a move is overridden if it yields an improvement over the current optimum.

A pseudocode of the resulting tabu navigation algorithm is given in fig. 5.

```
job(){ /* solution of job shop subproblem */
     init_job();
     /* find an initial solution with a dispatching rule */
     steps = 0;
     /* number of steps */
     lastimpr = 0;
     /* number of steps since last improvement of the optimal solution */
     stop = FALSE;
     while (stop == FALSE){
          make_list();
          /* make candidate list according to the current
               neighborhood structure */
          select(); /* select best non tabu move */
          steps++;
          lastimpr++;
          if (valcur < valopt){
               store(); /* if new optimal solution store it */
               lastimpr = 0; }
          if (steps > MAXSTEPS)
               stop = TRUE;
          if (lastimpr > MAXSTAZ)
               switch_n();
               /* if too many steps without
                    improvement, change neighborhood structure */
     }
}
```

Fig. 5. Pseudocode sketch of the tabu navigation algorithm for the job shop subproblem (short term level).

The pseudocode shown is nothing more than a simple tabu navigation scheme. Only one comment is in order: when too many steps elapse without improving the current

optimum, the neighborhood structure is changed. After some computational experiments with the makespan problem, this strategy has been found rather ineffective (neighborhood $\mathcal{N}_4$ is clearly the best one for the makespan case, so switching to the other ones usually does not give good results); in the present prototype, when no improvement is obtained, the solution of the JS subproblem is simply stopped and the routing is changed. However, in other cases switching the neighborhood structure could be useful.

3.2. SOLVING THE ROUTING SUBPROBLEM

In a hierarchical framework a routing is chosen and the resulting JS problem is solved. Since the solution of the JS problem requires a certain amount of computation, it is advisable to limit routing decisions to the "reasonable" ones. To this aim some knowledge is required, and a degree of dependence on the objective function is introduced.

This is in accordance with the principle of increasing-precision/decreasing-intelligence [36], which states that at the lower hierarchical levels of an intelligent system computationally intensive activities are performed, requiring limited knowledge, whereas at the higher levels more judgement is needed.

As noted in section 1.1, two basic types of hierarchical architecture can be devised:

- a one-way architecture;
- a two-way architecture.

3.2.1. A one-way scheduler

The cheapest way to solve the routing subproblem, concurrently with the scheduling subproblem, is by using dispatching rules. The schedule obtained can then be refined by a tabu search solution of the JS problem obtained by fixing the initial routing. Different rules can be used, depending on the objective function considered, or parametrized rules with different parameters setting (see subsection 2.1).

The resulting architecture is shown in fig. 6.

This scheduler is actually nothing more than a multiple start tabu search approach: it has been considered on one hand for comparison purposes (in section 4), and on the other one because it can be used both for minimizing the makespan and the total tardiness.

A better approach to choose a routing, for the makespan case, would be using a balancing procedure like the one proposed (among others) in [20, 37]: the operations are assigned to the machine in order to evenly distribute the load, i.e. the maximum sum of the processing times of the operations assigned to each machine is minimized. The rationale behind load balancing is that it should avoid the creation of bottleneck machines, which would lower the utilization of other machines, with a corresponding

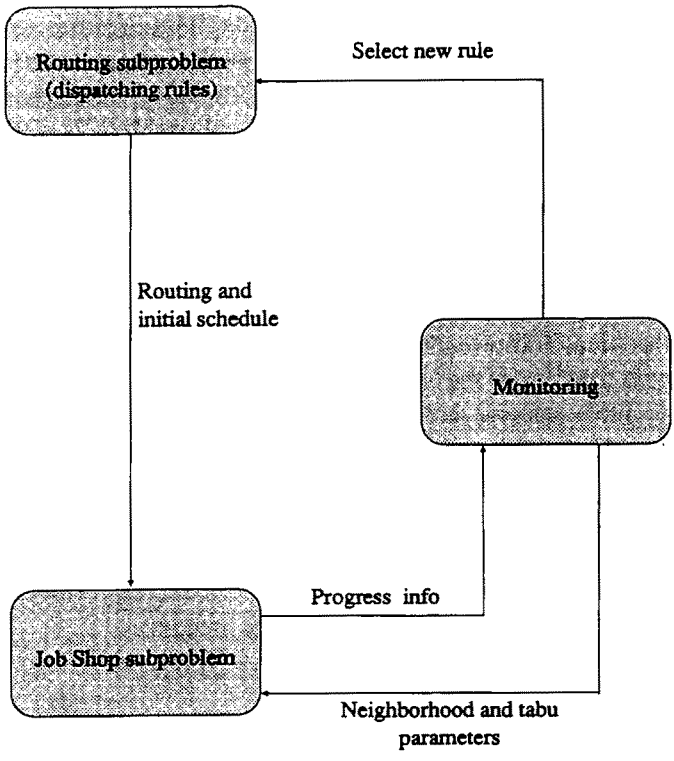

Fig. 6. Overall architecture of the one-way hierarchical tabu search algorithm.

increase in the makespan. However, balancing the machine loads allows to avoid static bottlenecks, whereas bottlenecks are dynamic. Furthermore, this improvement would again result in a one-way hierarchy. In order to cope with dynamic bottlenecks a two-way scheme must be devised.

3.2.2. A two-way scheduler

A two-way scheme for the makespan problem, able to identify and cope with dynamic bottlenecks, can be built by using again the critical path concept. Looking at the precedence graph of the solution of the JS problem, one can try to improve the machine assignment by reallocating operations on the critical path when alternative machines can be used.

Using the same information exploited during the solution of the JS problem minimizes the programming effort. We just add a tabu navigation procedure for the long term level, where the taboos are simply a forbidden operation–machine pair. A pseudocode sketch of the resulting algorithm is given in fig. 7. To find an initial solution, a dispatching rule can be used.

```
fjs(){ /* solution of the flexible job shop problem */
     steps = 0;
     stop = FALSE;
     first = TRUE;
     while (stop == FALSE){
     /* long term loop */
          steps++;
          if (first == TRUE){
               first = FALSE;
               route_prio();}
               /* dispatching rules are used for initialization */
          else{
               make_list();
               /* make candidate list using informations
                    about critical operations of the last
                    optimal job shop solution */
               select();}
               /* select best non tabu reallocation */
          job();
          if (valcur < valopt)
               store();
          if (steps > MAXSTEPS)
               stop = TRUE;
     }
}
```

Fig. 7. Pseudocode sketch of the tabu navigation algorithm for the routing subproblem (long term level).

The reallocation procedure adopted is rather simple. The optimal solution of the last JS subproblem is considered and its critical path is computed. For each operation on this critical path, the effect of assigning it to one of the alternative machines (if any) is considered. For each alternative machine the best possible insertion point is chosen, i.e. the sequence of the new machine is shifted, the operation is inserted in every possible position (among those which do not result in a cyclical precedence graph) and the position yielding the lowest makespan is chosen for each operation–machine pair.

Then the best non tabu operation–machine pair is chosen. When a machine is chosen a new long term memory tabu is added, forbidding the assignment of the reallocated operation to the old machine. As in the JS case the tabu status of a move is overridden if a new optimal solution results.

The overall architecture is shown in fig. 8: the only difference between the architectures of fig. 6 and fig. 8 is in the upward information flow between the JS and the routing subproblems.

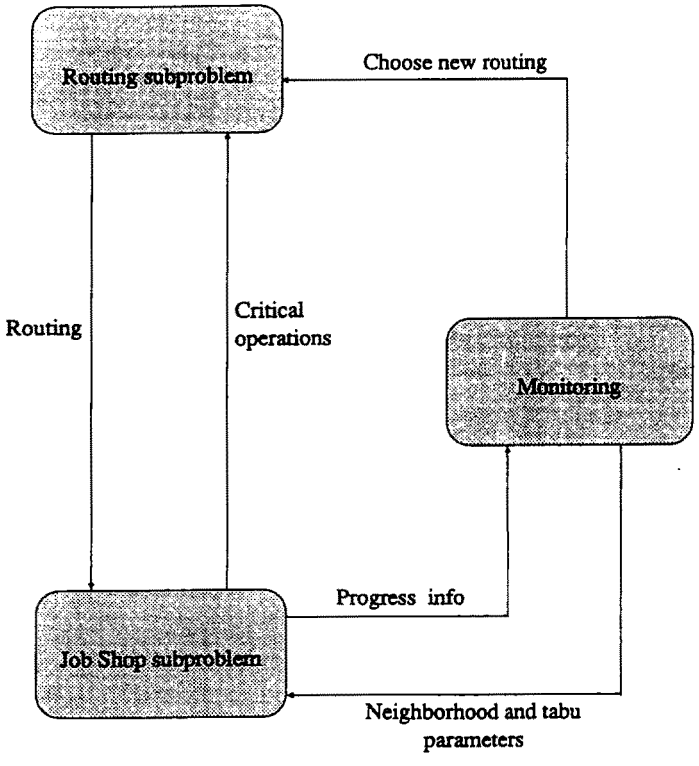

Fig. 8. Overall architecture of the two-way hierarchical tabu search algorithm.

3.3. ADAPTING THE TWO-WAY SCHEDULER TO THE TOTAL WEIGHTED TARDINESS PROBLEM

The two-way scheduler described in the last subsection is makespan oriented, unlike the one-way one. Can it be somehow adapted to total weighted tardiness case?

If the number of tardy jobs is limited, the critical path concept can again be used, both for scheduling and routing: we just need to consider the critical path from the initial dummy operation to the last operation of tardy jobs. This information can be used both for scheduling and routing.

If the shop is heavily loaded, and many jobs are late, it might be difficult to focus on critical operations to be rescheduled (or rerouted): in such a case, a random sampling approach like neighborhood $\mathcal{N}_i$ could be exploited. It should be the task of the medium term memory level to control the choice of the most appropriate neighborhood in each condition.

These ideas are currently under development, and no definitive conclusion can be drawn; however it is apparent that the proposed two-way hierarchical scheme can be adapted to different cases without disrupting its architecture nor its relevant

data structures. The significance of this flexibility should not be underestimated: the foundation of the object oriented programming paradigm is that the data structures, and not the algorithms themselves are the key factors in software development and reusability [27]. Quite different schedulers can be built resting on the same basic data structures [7].

4. Computational results

In order to evaluate the performance of the proposed architecture and to gain some insight into how it could be improved, a relatively restricted number of problem instances have been solved.

Dispatching rules have been used both in order to find initial solutions and to compare the tabu based scheduler with another heuristic strategy. When experimenting with the dispatching rules, a difficulty has been found, which is usually overlooked in the literature: what should be done when two (or more) jobs have the same priority? A natural answer is to choose randomly. The computational experiments have shown that the effect of this random behaviour can be dramatic (the makespan can be almost doubled): this is probably due to the fact that each random choice influences not only the next scheduling but also the next routing decisions. In the following the best result found has been reported for each rule.

4.1. RESULTS FOR THE MINIMUM MAKESPAN CASE

The data were randomly generated using a uniform distribution between given limits; the limits of each example are shown in table 1. The results obtained with the tabu search schedulers, the one-way and the two-way ones, are compared with those obtained by dispatching rules. Only results obtained by applying neighborhood $\mathcal{N}_4$ are reported. The best results obtained by each method are shown in table 2: due to the random outcome of the dispatching rules, the best makespan is reported.

The SPT and MWKR rules have been used to provide the tabu search based schedulers with initial solutions: in table 3 some information is shown about different initial and final solutions obtained by the two-way scheduler. Exact CPU times have not been reported, since the program has been implemented in order to be as interactive as possible (with graphical outputs and asynchronous interrupts from the user to change some parameters on line), with a corresponding increase in computational effort. In any case the elapsed times (on a 386-based PC) were limited, ranging from a few seconds to 12 minutes for the largest problem instances. The only exception has been the case `mk14`, which, when initialized with the SPT rule, required about 25 minutes to reach a solution comparable to that obtained in a few seconds by initializing with the MWKR rule.

In almost all cases the advantage of the proposed architecture is evident (in particular in cases `mk10`, `mk13`, `mk15`, which were the most difficult), but, obviously, a comparison with dispatching rules is not very conclusive.

Table 1

Problem instances for the minimum makespan problem.

	njob	nmac	nop	meq	proc
mk1	10	6	5 - 7	3	1 - 7
mk2	10	6	5 - 7	6	1 - 7
mk3	15	8	10 - 10	5	1 - 20
mk4	15	8	3 - 10	3	1 - 10
mk5	15	4	5 - 10	2	5 - 10
mk6	10	15	15 - 15	5	1 - 10
mk7	20	5	5 - 5	5	1 - 20
mk8	20	10	10 - 5	2	5 - 20
mk9	20	10	10 - 15	5	5 - 20
mk10	20	15	10 - 15	5	5 - 20
mk11	30	5	5 - 8	2	10 - 30
mk12	30	10	5 - 10	2	10 - 30
mk13	30	10	5 - 10	5	10 - 30
mk14	30	15	8 - 12	2	10 - 30
mk15	30	15	8 - 12	5	10 - 30

njob : number of jobs;
nmac : number of machines;
nop : minimum and maximum number of operations per job;
meq : maximum number of equivalent machines per operation;
proc : minimum and maximum processing time per operation.

Table 2

Best results for the minimum makespan problem.

	spt	lwkr	mwkr	tab1	tab2
mk1	65	76	54	49	42
mk2	44	48	41	41	32
mk3	397	506	296	263	211
mk4	109	121	89	89	81
mk5	231	291	188	188	186
mk6	128	143	139	128	86
mk7	188	238	262	188	157
mk8	670	1042	558	523	523
mk9	573	723	536	444	369
mk10	536	627	524	363	296
mk11	760	957	716	716	649
mk12	698	899	640	565	518
mk13	821	887	542	542	478
mk14	1296	2046	715	694	694
mk15	974	985	477	448	383

spt : shortest processing time rule;
mwkr : most work remaining rule;
lwkr : least work remaining rule;
tab1 : one-way tabu search;
tab2 : two-way tabu search.

Table 3

Results of the two-way scheduler for the minimum makespan problem (mean values have been rounded to the nearest integer).

	Initialized by SPT						Initialized by MWKR					
	#	ib	iw	im	rb	rw	rm	#	ib	iw	im	rb	rw	rm
mk1	3	65	91	80	46	51	48	3	54	70	61	42	44	43
mk2	4	49	61	54	32	36	33	4	41	48	44	36	40	38
mk3	5	445	609	511	211	232	224	5	306	391	348	220	225	221
mk4	5	115	157	142	81	102	90	2	100	104	102	97	104	100
mk5	5	258	282	268	187	196	190	5	189	204	194	186	189	188
mk6	5	141	174	152	86	96	91	6	151	191	168	103	120	111
mk7	5	188	280	220	157	179	168	5	271	303	285	229	236	232
mk8	3	713	894	831	523	538	528	3	558	632	595	523	523	523
mk9	5	678	804	724	398	439	417	5	536	602	562	369	392	379
mk10	4	575	709	623	301	348	325	6	527	658	573	296	312	301
mk11	7	760	1104	873	649	692	668	3	716	736	725	597	712	707
mk12	8	729	1090	866	518	579	536	2	640	650	645	534	541	537
mk13	5	1098	1349	1209	502	656	545	5	546	595	570	478	490	485
mk14	3	1620	1743	1699	694	754	726	3	715	778	740	694	694	694
mk15	3	1011	1337	1211	422	446	431	3	477	595	537	383	399	392

: number of trials;
ib : best initial value;
iw : worst initial value;
im : mean initial value;
rb : best result;
rw : worst result;
rm : mean result.

Although limited, the computational tests performed have been sufficient to draw some conclusions about the behaviour of the tabu search schedulers: such conclusions will act as guidelines for the implementation of a more refined tabu search scheduler to be compared with other heuristics proposed in the literature.

- The higher the flexibility (i.e. the number of equivalent machines per operation), the more the increase in complexity required by the two-way architecture is justified.
- The performance of neighborhoods $\mathcal{N}_2$ and $\mathcal{N}_3$ for the makespan problem has been very poor. They are not good at improving a given solution (they do not exploit it well, unlike a clever neighborhood structure based on the critical path of the precedence graph), nor are they able to escape from a local minimum (they do not explore well, unlike a random sampling). Neighborhoods $\mathcal{N}_2$ and $\mathcal{N}_3$ seem "badly" focused.

- When the CPU time must be limited, a good initial solution, or at least not too bad a one, is helpful. However, only in a few cases the best solution has been obtained from the best initial schedule. Sometimes the SPT based schedules were consistently worse than those obtained by the MWKR rule, but the final solutions were systematically better: in such case a very quick and large improvement of the makespan was observed during the solution of the first JS subproblem, and then the rerouting procedure achieved better improvements than in the MWKR case. This suggests the opportunity of finding a good initial routing, rather than a good initial schedule: it is easier to improve the schedule than the routing. Therefore, it seems appropriate, for the makespan problem, finding an initial routing with a load balancing procedure, and then initializing the schedule. Further research will establish whether these conjectures are well founded.
- Sometimes, the rerouting procedure resulted in an increase in the makespan, which led, after a few steps, to an improvement over the current optimum. This shows that often the value of the objective function is not the only criterion to evaluate a move (see [25]). Furthermore, in a non-hierarchical approach, rerouting moves (which are moreover computationally expensive to evaluate) would be seldom selected with respect to rescheduling moves: this shows the need for a hierarchical scheme and a proper interlevel coordination approach.

4.2. INTERLEVEL COORDINATION ISSUES

The coordination of the hierarchical levels is clearly a fundamental issue in a two-way hierarchical architecture.

In particular, a key question is how much time is worth spending in the solution of the JS subproblem, once a routing has been selected: in other words, the JS scheduling problem must be solved to near-optimality (with a corresponding high computational effort) or a few iterations are enough?

The problem can be somewhat clarified by drawing a similarity with unconstrained optimization of a continuous function, i.e.

$$\min_x f(x).$$

In this case, at each iteration a current point x^k, the solution can be improved by selecting a search direction s^k and by minimizing the function along this direction:

$$x^{k+1} = \arg\min_{\alpha} f(x^k + \alpha s^k).$$

In the steepest descent strategy, the search direction is parallel to the gradient of the function at the current solution ($s^k = -\nabla f(x^k)$); the slow convergence, due to

a zigzagging behaviour of the steepest descent method is well known. To avoid zigzagging, a possible strategy is to restrict the step taken along the search direction (trust region methods, see e.g. [14]).

In our case the role of the search direction is played by routing and the step selection corresponds to JS scheduling.

In order to understand whether a sort of restricted step strategy could be useful in the FJS case, another computational test has been performed. The most difficult cases (i.e. mk10, mk13, mk15) have been solved using the two-way architecture, changing the strategy employed for the JS subproblem solution.

The program has always been stopped after 1000 steps, starting from the same initial solution obtained by the MWKR rule. The neighborhood $\mathcal{N}_4$ has been used with different parameters, specifying the maximum number of steps for each JS subproblem and the maximum number of steps without improving the current optimal solution (for the JS subproblem). The neighborhood $\mathcal{N}_1$ has also been experimented with: the size of the sample set was 40, i.e. 40 admissible moves were randomly selected.

These results are shown in table 4. Although the number of experiments is extremely limited, the results are coherent and intuitively justified, leading to the conclusion that a restricted step approach performs better. Even a "bad" neighborhood like $\mathcal{N}_1$ can lead to a better result, in terms of solution quality, than a clever strategy spending too much time solving each JS subproblem: this does not imply that neighborhood $\mathcal{N}_4$ is a useless subtlety, since, in order to obtain good result, the size of the sampled neighborhood must be suitably large, leading to longer CPU times.

Table 4

The influence of the effort in the solution of the JS subproblem for the minimum makespan case.

	$\mathcal{N}_4^{25/25}$	$\mathcal{N}_4^{50/50}$	$\mathcal{N}_4^{25/50}$	$\mathcal{N}_4^{50/100}$	$\mathcal{N}_1^{25/25}$
mk10	305	319	305	318	314
mk13	482	487	486	488	486
mk15	397	406	403	415	407

$\mathcal{N}_4^{25/25}$: 25 steps (max 25 without improvement) with neighborhood $\mathcal{N}_4$;
$\mathcal{N}_4^{50/50}$: 50 steps (max 50 without improvement) with neighborhood $\mathcal{N}_4$;
$\mathcal{N}_4^{25/50}$: 50 steps (max 25 without improvement) with neighborhood $\mathcal{N}_4$;
$\mathcal{N}_4^{50/100}$: 100 steps (max 50 without improvement) with neighborhood $\mathcal{N}_4$;
$\mathcal{N}_1^{25/25}$: 25 steps (max 25 without improvement) with neighborhood $\mathcal{N}_1$.

The results show that the role of solving the JS subproblem is not only to improve the current solution, but also to give some information on how to improve the routing, by finding good candidate operations for a rerouting. The experiment with neighborhood $\mathcal{N}_1$ shows that even a random sampling strategy works well from

this point of view. Using a sophisticated heuristic algorithm or even (for small problem instances) an exact one would be a useless effort.

A proper coordination strategy is therefore fundamental, and more refined approaches must be devised: the medium term level, which has not been developed in the present implementation, will be, therefore, a key topic of further research.

4.3. RESULTS FOR THE MINIMUM TOTAL WEIGHTED TARDINESS CASE

For the weighted tardiness case, only the one-way scheduler has been implemented. The JS scheduler is initialized with a SPT schedule and with different ATC schedules obtained with different parameters: the best final result, which need not correspond to the best initial schedule, is reported. Neighborhood $\mathcal{N}_1$ has been used.

The tardiness weights were uniformly distributed between 1 and 3. The other data limits and the obtained results are shown in tables 5 and 6. Clearly, both the

Table 5

Problem instances for minimum total weighted tardiness problem.

	njob	mac	nop	meq	proc	dd
wt1	10	5	5 – 5	3	6 – 10	20 – 80
wt2	20	5	5 – 5	3	6 – 10	20 – 160
wt3	20	10	10 – 15	3	10 – 30	50 – 1000
wt4	30	10	10 – 15	3	10 – 30	50 – 2000
wt5	30	15	10 – 15	5	10 – 30	50 – 1500

njob : number of jobs;
nmac : number of machines;
nop : minimum and maximum number of operations per job;
meq : maximum number of equivalent machines per operation;
proc : minimum and maximum processing time per operation;
dd : minimum and maximum due date per job.

Table 6

Results for the minimum total weighted tardiness problem.

	edd	atc	tabl
wt1	246	233	137
wt2	1106	683	368
wt3	18924	13990	1666
wt4	766	616	307
wt5	5990	2686	1209

edd : earliest due date rule;
atc : apparent tardiness rule;
tabl : one-way tabu search.

computational experiments and the implemented scheduler are very limited, but the results show that there is a great potential of improvement, with respect to the results of priority rules, which can be obtained by a tabu search based scheduler developed according to the ideas in subsection 3.3.

5. Conclusions and directions for further research

A hierarchical tabu search architecture has been described and evaluated for the problem of routing and scheduling in a flexible job shop.

Tabu search does not yield *one* algorithm, but a range of possible architectures which can exploit other simple or sophisticated heuristic principles (in our case dispatching rules and the critical path in the precedence graph), and can be adapted to different objective functions. The generality of the tabu search concept and its ability to move smoothly on a range of increasingly sophisticated implementations make it an interesting tool for real world application, when the Operations Research practitioner must deal with managers not willing to commit to very sophisticated but rigid algorithms.

Further research is needed in order to assess the performance of tabu search for the FJS problem, since the computational results presented are far from conclusive. Some comparison must be made with other schemes, like for example, the beam search based one proposed in [10] for the minimum makespan problem. The author is not aware of any work concerning minimum total weighted tardiness in a flexible job shop.

Future research will focus on better initialization strategies (e.g. by load balancing procedures for the makespan problem) and on the implementation of a two-way approach for the weighted tardiness problem.

Furthermore, in the scheme proposed, only the basics of tabu search have been exploited: many directions are open to improve the performance.

- At present the whole set of candidate solutions is formed and then they are evaluated. Adaptive sampling strategies could be used by interleaving sampling and evaluations, in order to avoid a smaller or larger sample size than necessary.
- The coordination between the routing and scheduling levels should be carefully considered. The computational tests performed show that it is useless to spend much time optimizing a job shop schedule, and then to change routing: the amount of search should be limited in a clever way, as in restricted step methods for unconstrained optimization [14]. This calls for increased sophistication in the medium term memory level.

  Rerouting one operation considering the best possible insertion at each routing step needs to be the better strategy. More operations could be rerouted at once and one could avoid committing to what seems the best insertion. A possible strategy is to consider a set of possibilities, to allocate to each one

a limited amount of search and then to choose the initial solution for the new JS subproblem.

- Diversification approaches would be useful to avoid long useless chains of moves which sometimes can occur, as reported in [4, 18].

A final remark is in order about the flexibility of the proposed approach with respect to different objective functions: a research is currently in progress on the application of local search to multiobjective problems [8], which has its roots in such flexibility.

References

[1] J. Adams, E. Balas and D. Zawack, The shifting bottleneck procedure for job shop scheduling, Manag. Sci. 34(1988)391–401.

[2] R. Akella, Y. Choong and S.B. Gershwin, Performance of a hierarchical production scheduling policy, IEEE Trans. Components, Hybrids and Manufacturing Technol. CHMT-7(1984)225–248.

[3] E. Balas, Machine sequencing via disjunctive graphs: an implicit enumeration algorithm, Oper. Res. 17(1969)941–957.

[4] J.W. Barnes and J.B. Chambers, Solving the job shop scheduling problem using tabu search, Technical Report ORP91-06, University of Texas at Austin (1991).

[5] B. Bona, P. Brandimarte, C. Greco and G. Menga, Hybrid hierarchical scheduling and control systems in manufacturing, IEEE Trans. Robotics and Automation RA-6(1990)673–686.

[6] P. Brandimarte, R. Conterno and P. Laface, FMS production scheduling by simulated annealing, *Proc. 3rd Conf. on Simulation in Manufacturing*, Torino (Nov. 1987) pp. 235–245.

[7] P. Brandimarte, Using abstract data types in developing search-based schedulers, *Proc. ICARV'90 (Int. Conf. on Automation, Robotics and Computer Vision)*, Singapore (1990) pp. 6–10.

[8] P. Brandimarte, Bicriteria parallel machine scheduling by local search, in preparation.

[9] P. Brandimarte, W. Ukovich and A. Villa, Factory level aggregate scheduling: a basis for a hierarchical approach, *Proc. 1992 IEEE Conf. on CIM*, RPI, Troy, NY.

[10] Y.-L. Chang, H. Matsuo and R.S. Sullivan, A bottleneck-based beam search for job scheduling in a flexible manufacturing system, Int. J. Prod. Res. 27(1989)1949–1961.

[11] L.F. Escudero, A mathematical formulation of a hierarchical approach for production planning in FMS, in: *Modern Production Management Systems*, ed. A. Kusiak (North-Holland, 1987) pp. 231–245.

[12] L.F. Escudero, An inexact algorithm for part input sequencing and scheduling with side constraints in FMS, Int. J. Flexible Manufacturing Syst. 1(1989)143–174.

[13] E. Falkenauer and S. Bouffouix, A genetic algorithm for job shop, *Proc. 1991 IEEE Conf. on Robotics and Automation*, Sacramento, CA, pp. 824–829.

[14] R. Fletcher, *Practical Methods of Optimization*, 2nd ed. (Wiley, 1987).

[15] S. French, *Sequencing and Scheduling: an Introduction to the Mathematics of Job Shop* (Wiley, 1982).

[16] F. Glover, Tabu search, Part I, ORSA J. Comput. 1(1989)190–206.

[17] F. Glover, Tabu search, Part II, ORSA J. Comput. 2(1990)4–32.

[18] F. Glover and R. Hübscher, Bin packing with tabu search, preprint (1991).

[19] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Wiley, 1989).

[20] J. Hutchison, K. Leong, D. Snyder and P. Ward, Scheduling approaches for random job shop flexible manufacturing systems, Int. J. Prod. Res. 29(1991)1053–1067.

[21] Y.-D. Kim, A comparison of dispatching rules for job shops with multiple identical jobs and alternative routings, Int. J. Prod. Res. 28(1990)953–962.
[22] E.J. Lee and P. Mirchandani, Concurrent routing, sequencing and setups for a two-machine flexible manufacturing cell, IEEE J. Robotics and Automation RA-4(1988)256–264.
[23] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing, Science 220(1983)621–680.
[24] M. Laguna, J.W. Barnes and F. Glover, Tabu search methods for a single machine scheduling problem, J. Int. Manufacturing 2(1991)63–74.
[25] M. Laguna and F. Glover, Integrating target analysis and tabu search for improved scheduling systems, Expert Syst. Appl., to appear.
[26] M. Laguna and J.L.G. Velarde, A search heuristic for just-in-time scheduling in parallel machines, J. Int. Manufacturing 2(1991)253–260.
[27] B. Meyer, *Object-Oriented Software Construction* (Prentice–Hall, 1988).
[28] P. Mirchandani, E.J. Lee and A. Vasque, Concurrent scheduling in flexible automation, Technical Report No. 37-88-149, Department of Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, Troy, NY (1988).
[29] M. Montazeri and L.N. Van Wassenhove, Analysis of scheduling rules for an FMS, Int. J. Prod. Res. 28(1990)785–802.
[30] N. Nasr and E.A. Elsayed, Job shop scheduling with alternative machines, Int. J. Prod. Res. 28(1990) 1595–1609.
[31] F.A. Ogbu and D.K. Smith, The application of the simulated annealing algorithm to the solution of the $n/m/C_{max}$ flowshop problem, Comput. Oper.Res. 17(1990)243–253.
[32] I.H. Osman and C.N. Potts, Simulated annealing for permutation flow-shop scheduling, OMEGA Int. J. Manag. Sci. 17(1989)551–557.
[33] P.S. Ow and T.E. Morton, Filtered beam search in scheduling, Int. J. Prod. Res. 26(1988)35–62.
[34] S.S. Panwalkar and W. Iskander, A survey of scheduling rules, Oper. Res. 25(1977)45–61.
[35] R.G. Parker and R.L. Rardin, *Discrete Optimization* (Wiley, 1988).
[36] G.N. Saridis, Analytical formulation of the principle of increasing precision with decreasing intelligence for intelligent machines, Automatica 25(1989)461–467.
[37] T. Sawik, Modelling and scheduling of a flexible manufacturing system, Eur. J. Oper. Res. 45(1990)177–190.
[38] K.E. Stecke, Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems, Manag. Sci. 29(1983)273–288.
[39] E. Taillard, Parallel taboo search technique for the job shop scheduling problem, Research Report ORWP 89/11, Ecole Polytechnique Fédérale de Lausanne (1989).
[40] E. Tocylowski, K.S. Hindi and M.G. Singh, Multi-level production scheduling for a class of flexible machine and assembly systems, Ann. Oper. Res. 17(1989)163–180.
[41] P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated Annealing: Theory and Applications* (Reidel, 1987).
[42] P.J.M. van Laarhoven, E.H.L. Aarts and J.K. Lenstra, Job shop scheduling by simulated annealing, Report OS-R8809, CWI, Amsterdam(1988).
[43] A.P.J. Vepsalainen and T.E. Morton, Priority rules for job shops with weighted tardiness costs, Manag. Sci. 33(1987)1035–1047.
[44] M. Widmer, Job shop scheduling with tooling constraints: a tabu search approach, J. Oper. Res. Soc. 42(1991)75–82.
[45] M. Widmer and A. Hertz, A new heuristic method for the flow shop sequencing problem, Eur. J. Oper. Res. 41(1989)186–193.