Comparing Iterative Improvement Heuristics for Multi-Capacity Scheduling Problems

Angelo Oddi[1], Nicola Policella[2], Amedeo Cesta[1], and Stephen F. Smith[3]

[1] ISTC-CNR, Rome, Italy
{name.surname}@istc.cnr.it
[2] European Space Agency, Darmstadt, Germany
nicola.policella@esa.int
[3] The Robotics Institute, Carnegie Mellon University,USA
sfs@cs.cmu.edu

Abstract. Iterative Flattening search is a local search schema introduced for solving scheduling problems with a makespan minimization objective. It is an iterative two-step procedure, where on each cycle of the search a subset of ordering decisions on the critical path in the current solution are randomly retracted and then recomputed to produce a new solution. Since its introduction, other variations have been explored and shown to yield substantial performance improvement over the original formulation. In this spirit, we propose and experimentally evaluate further improvements to this basic local search schema. Specifically, we examine the utility (1) of operating with a more flexible solution representation, (2) of adopting a more focused decision retraction strategy and (3) of integrating iterative-flattening search with a complementary tabu search procedure. We evaluate these extensions on large benchmark instances of the Multi-Capacity Job-Shop Scheduling Problem (MCJSSP) which have been used in previous studies of iterative flattening search procedures.

1 Introduction

The integration of local search and heuristic procedures has produced interesting and efficient approaches to several complex scheduling problems. One such example is the *iterative flattening* (or IFLAT) algorithm proposed in [1] for solving scheduling problems with a makespan minimization objective. IFLAT consists of an iterative, two-step local search schema. Within each cycle of the search, a *relaxation step* is first applied to remove some search decisions from the current solution and create a partial solution. In this context, search decisions correspond to precedence constraints that must be added between pairs of activities to resolve resource conflicts, and the relaxation step limits its attention to those constraints residing on the solution's *critical path*. Starting from this new partial solution, the second *flattening step* then incrementally adds back new precedence constraints to regain a feasible solution. In the original work, IFLAT was shown to produce high-quality solutions in reasonable time on a challenging set of large multi-capacitated job-shop scheduling problem MCJSSP benchmarks.

More recently two works [2, 3] have extended the results of the original paper through refinement of the basic IFLAT search schema. [2] identified an anomaly in

IFLAT search and proposed a simple extension, which dramatically improved the quality of its schedules while preserving its computational efficiency. Their key idea was to iterate the relaxation step multiple times, hence the name IFLATRELAX used in what follows. The resulting algorithm found many new upper bounds and produced solutions within 1% of the best upper bounds on average. The work of [3] presented an approach which follows the same two step schema of IFLAT but used different engines for both the flattening and the relaxation steps. This procedure was able to find additional optimal solutions and to further improve known upper-bounds for MCJSSP benchmarks.

In the same spirit, this paper proposes and evaluates further extensions to the IFLAT family of search procedures and two meta-heuristic schemas based on iterative flattening search. We focus specifically on three potential shortcomings of the basic approach: (1) the lack of temporal flexibility in the solutions that are manipulated by IFLAT, (2) the tendency to re-explore the same solution subspaces over time and (3) the inability to perform a fine-grained neighborhood search in the vicinity of near-optimal solutions. To cope with the first two issues, we explore the use of partial order schedules [4] as an underlying solution representation and add a *tabu-list* to better control the relaxation step. To address the third issue, we introduce a complementary *tabu-seach* procedure to refine solutions found by IFLAT and propose two meta-heuristic search schemas for integrating the two procedures to achieve an appropriate interplay of diversification (exploration) and intensification (exploitation) in the overall search process.

The paper is organized as follows. We first review the iterative flattening search schema that has evolved from previous work. The central part of the paper then introduces the proposed extensions, the two meta-heuristic methods, and indicates how they are integrated (individually and in combination) within the basic search schema. Next we recall the MCJSSP problem domain and benchmark problem sets used in our evaluation. Performance results are then given that demonstrate the leverage provided by the extended search procedure. We conclude by briefly discussing further opportunities to extend and enhance the basic iterative flattening search concept.

2 The Iterative Flattening Schema

Before describing the iterative flattening approach it is necessary to introduce the modeling perspective on which this schema is based. Underlying the approach, a solution Sol is represented as a directed graph $G_S(A, E)$. A is the set of activities specified in MCJSSP, plus a fictitious a_{source} activity temporally constrained to occur before all others and a fictitious a_{sink} activity temporally constrained to occur after all others. E is the set of precedence constraints defined between activities in A. Following a Precedence Constraint Posting (PCP) approach, the set E can be partitioned in two subsets, $E = E_{prob} \cup E_{post}$, where E_{prob} is the set of precedence constraints originating from the problem definition, and E_{post} is the set of precedence constraints posted to resolve resource conflicts. In general the directed graph $G_S(A, E)$ represents a set of temporal solutions. The set E_{post} is added in order to guarantee that at least one of those temporal solutions is also resource feasible.

Given this description of the solution model, lets turn attention to the Iterative Flattening search procedure itself [1]. This heuristic procedure iterates two main steps:

```
iFlat($Sol, P_{rem}, MaxFail, MaxRelaxations$)
begin
1.  S$_{best}$ ← $Sol$
2.  $counter$ ← 0
3.  while ($counter \leq MaxFail$) do
4.      Relax($Sol, P_{rem}, MaxRelaxations$)
5.      $Sol$ ←Flatten($Sol$)
6.      if Mk($Sol$) < Mk(S$_{best}$) then
7.          S$_{best}$ ← $Sol$
8.          counter ← 0
9.      else
10.     counter ← counter + 1
11. return S$_{best}$
end
```

Fig. 1. The *Iterative Flattening* Algorithm

Relaxation step: first, a feasible schedule is relaxed into a possibly resource infeasible, but precedence feasible, schedule by removing some search decisions represented as precedence constraints between pair of activities;

Flattening step: second, a sufficient set of new precedence constraints is posted to re-establish a feasible schedule.

These two steps are executed for some number of iterations or until no better feasible schedule has been found for some number of iterations.

Figure 1 shows the *iterative flattening* algorithm in detail. iFlat takes as input four elements: (1) a starting solution Sol; (2) a value P$_{rem}$ $\in [0, 1]$ designating the percentage of precedence constraints $pc_i \in E_{post}$ on the critical path to be removed; (3) a positive integer $MaxFail$ which specifies the maximum number of non-makespan-improving moves that the algorithm will tolerate before terminating; and (4) a positive integer $MaxRelaxations$ which specifies the maximum number of relax iterations to be performed in the relaxation step. Note that it is the value of this last parameter that distinguishes between the approaches taken in [1] and [2] respectively (see below). Returning to the algorithm description in Figure 1, after initialization (Steps 1-2), a solution is repeatedly modified within the while loop (Steps 3-10) by the application of the Relax and Flatten procedures. In the case that a better makespan solution is found (at Step 6), the new solution is stored in S_{best} and the counter is reset to 0. Otherwise, if no improvement is found in $MaxFail$ moves, the algorithm terminates and returns the best solution found. The rest of this section describes the relaxation and the flattening steps in more detail.

Relaxation. The relaxation step is based on the concept of *critical path*. As known in the scheduling literature, information about *critical paths* can provide a strong heuristic basis for makespan minimization. A *path* in $G_S(A, E)$ is a sequence of activities $a_1 \ldots a_k$, such that, $(a_i, a_{i+1}) \in E$ with $i = 1 \ldots (k - 1)$. The length of a path is the

```
RELAX($Sol, P_{rem}, MaxRelaxations$)
begin
1.  for 1 to $MaxRelaxations$
2.      forall $(a_i, a_j) \in$ CriticalPath($Sol$)$\cap E_{post}$
3.          if random(0,1)$< P_{rem}$
4.              $Sol \leftarrow Sol \backslash (a_i, a_j)$
end
```

Fig. 2. The RELAX procedure

sum of the activities processing times and a *critical path* is a path from a_{source} to a_{sink} which determines the solution's makespan.

Any improvement in makespan will necessarily require change to some subset of precedence constraints situated on the *critical path*, since these constraints collectively determine the solution's current makespan. Following this observation, the relaxation step introduced in [1] is designed to retract some number of posted precedence constraints on the solution's critical path.

Figure 2 shows the RELAX procedure. Steps 2-4 consider the set of posted precedence constraints ($pc_i \in E_{post}$) which belong to the current critical path. A subset of these constraints is randomly selected and then removed from the current solution. Step 1 represents the crucial difference between the approaches of [1] and [2]. In the former approach the steps 2-4 are repeated only once (i.e. $MaxRelaxations = 1$) whereas in [2] these steps are iterated several times (from 2 to 6). In practice the new critical path of Sol is computed at each iteration. Notice that this path can be completely different from the previous one. This allows the relaxation step to also take into account those paths that have a criticality very close to the overall critical path.

Flattening. The flattening step used in [1] is inspired by prior work on the Earliest Start Time Algorithm (ESTA) from [5]. ESTA was designed to address more general, multi-capacity scheduling problems with generalized precedence relations between activities (i.e., corresponding to metric separation constraints with minimum and maximum time lags). This algorithm is a variant of a class of PCP scheduling procedures, characterized by a two-phase, solution generation process:

Constructing an infinite capacity solution: the current problem is formulated as an STP [6] temporal constraint network.[4] In this initial problem representation, temporal constraints are modeled and satisfied (via constraint propagation) but resource constraints are ignored, yielding a time feasible solution that assumes infinite resource capacity.

[4] In a STP (Simple Temporal Problem) network we make the following representational assumptions: temporal variables (nodes or time-points) represent the start and end of each activity, and the beginning and end of the overall temporal horizon; distance constraints (edges) represent the duration of each activity and separation constraints between activities including simple precedences.

```
FLATTEN(Problem)
begin
1.  TCSP ← CreateCSP(Problem)
2.  loop
3.      Propagate(TCSP)
4.      ConflictSet ← ComputeResourceConflicts(TCSP)
5.      if ConflictSet= $\emptyset$ then
6.          return ESS(TCSP)
7.      else
8.          if Unsolvable(ConflictSet) then
9.              return $\emptyset$
10.         else
11.             Conflict ← SelectConflict(ConflictSet)
12.             $pc_i$ ← SelectPrecedence(Conflict)
13.             TCSP ← TCSP$\cup\{pc_i\}$
end
```

Fig. 3. The FLATTEN implementation

Leveling resource demand by posting precedence: Resource constraints are superimposed by projecting "resource demand profiles" over time. Detected resource conflicts are then resolved by iteratively posting simple precedence constraints between pairs of competing activities.

The constraint posting process of ESTA is based on the Earliest Start Solution (ESS) consistent with currently imposed temporal constraints (computed in step 3). It then proceeds to compute the set of resource conflicts (step 4). If this set is empty the ESS is also resource feasible and a solution is found; otherwise this set of conflicts is considered. If a conflict exists that can be solved, a new precedence constraint is posted to do so (step 11-13); otherwise the process fails and returns the empty set (step 9). For further details on the functions ComputeResourceConflict(), SelectConflict(), and SelectPrecedence() the reader should refer to the original references.

3 Basic Extensions to Iterative Flattening Search

The concept of Iterative Flattening introduced in [1] is quite general and provided an interesting new basis for designing more sophisticated and effective local search procedures for scheduling optimization. The IFLATRELAX procedure proposed in [2] is a nice example of an IFLAT extension which obtains substantial improvements over its original version. This section describes further extensions to the iterative flattening search schema based on two possible *drawbacks*.

A first potential shortcoming is the lack of temporal flexibility in the initial solution provided to seed IFLAT. Previous work has used ESTA as an initial solution generator and, as indicated earlier, ESTA only guarantees that the earliest start time solution (ESS) is resource feasible. In fact, the effectiveness of the IFLAT procedure relies on finding new orderings of activities such that an increasingly more compact solution is found on each cycle. The greater the flexibility, the higher the probability that new start times for relaxed activities can be found on a given *relax-and-flatten* cycle that reduce the overall makespan.

A second possible drawback stems instead from the uninformed manner in which precedence constraints are selected for retraction. In fact, constraints are selected for removal without regard to when and in which order they were originally introduced. This can lead to repeated selection of the same constraints and repeated solving of the same (or very similar) partial solution.

In the following subsections we propose two extensions to address each of these two potential limitations.

3.1 Introducing Partial Order Schedules

The first extension of the search schema, aimed at increasing the temporal flexibility of solutions generated during the flattening step, is to substitute the use of partial order schedules (POS) for the solutions produced by ESTA. Both types of the solutions are based on a graph representation. The difference is that while ESTA solutions guarantee that at least one of the temporal solutions they represent is also resource feasible, a POS guarantees that *all* delineated temporal solutions are also resource feasible. The use of a POS in general increases the possibilities for rearranging relaxed activities.

The common thread underlying a POS is the characteristic that activities which require the same resource units are linked via precedence constraints into precedence *chains*. Given this structure, each constraint becomes more than just a simple precedence. It also represents a producer-consumer relation, allowing each activity to know the precise set of predecessors that will supply the units of resource it requires for execution. In this way, the resulting network of chains can be interpreted as a flow of resource units through the schedule; each time an activity terminates its execution, it passes its resource unit(s) on to its successors. It is clear that this representation is flexible if and only if there is temporal slack that allows chained activities to move "back and forth".

Polynomial methods for producing a POS from an input solution represented as a precedence graph (or equivalently as a set of start times) have been introduced in [5, 4]. Given an input solution, a transformation method, named *chaining*, is defined that proceeds to create sets of chains of activities. This operation is accomplished over three steps: (1) all the previously posted leveling constraints are removed form the input partial order; (2) the activities are sorted by increasing activity earliest start times; (3) for each resource and for each activity a_i (according to the increasing order of start times), one or more predecessors a_j are chosen, which supplies the units of resource required by a_i – a precedence constraint (a_i, a_j) is posted for each predecessor a_j. The last step is iterated until all the activities are linked by precedence chains.

Having a temporal flexible solution is not the only benefit in considering the use of partial order schedules. A second property that appears to be relevant is the reduction in the number of additional precedence constraints that must be posted to obtain a solution: Given a problem with n activities to be scheduled, the number of constraints appearing in the solution is always $O(n)$. This because the chaining procedure creates POSs with only the "necessary" precedence constraints, and eliminates all "redundant" constraints. The removal of redundant precedence constraints tends to intensify the effect of the IFLAT relaxation step in the IFLAT procedure. More solutions are accessible at each flattening cycle, because the removal of redundant constraints increases possibilities for rearranging relaxed activities. This intuition is confirmed in the experimental section of the paper, where we see how the use of partial order schedules as input allows the iterative flattening search to find better quality solutions in comparison to the procedure proposed in [2]. It is worth noting that although [3] uses the concept of POS, it uses them solely to overcome the natural lack of flexibility of fixed time solutions and to apply a large neighborhood search schema.

3.2 Using a Tabu List

The IFLAT search procedure can be viewed as a kind of *random walk* in the space of feasible solutions which is driven by a simple and effective heuristic criterion: only precedence constraints in the solution's critical path are selected for random removal, as they represent the most probable candidate decisions which can be changed for improving the solution's makespan. An unfortunate side-effect though is that the same constraints can be selected repeatedly.

To overcome this problem we introduce, as a second extension of IFLATRELAX, the use of a *tabu-list* mechanism to dampen the probability of quickly turning back to previously explored solutions. A tabu list is a special memory structure commonly used in tabu search, a type of local search procedure. The tabu list contains the solutions that have been visited in the recent past in order to prevent cycling in the sequence of explored solutions. In our context, the idea is to record (and hence mark as non-retractable) the set of constraints that have been most recently posted by IFLAT.

The introduction of the tabu list mechanism obviously requires an adjustment of the procedure described in Fig. 1. Two are the main differences:

1. the *Relax* procedure takes into account a *tabu-list* of precedence constraints, which cannot be removed from the current solution;
2. the *Flatten* procedure updates the tabu-list at each cycle with a subset of the new precedence constraints posted.

In particular, the tabu list is a first-in-first-out data structure with maximal length l, which is updated at each flattening cycle. We use an additional parameter δ to modulate the use of the tabu-list, such that, the procedure *Flatten* inserts at each cycle at most the last δ precedence constraints generated during each leveling step.

As indicated above, the tabu list is a mechanism that focuses the search toward unexplored regions of the search space. In fact, by avoiding retraction of the most recently inserted precedence constraints (contained in the tabu list), it is possible to increase the number of distinct precedence constraints appearing in two consecutive solutions during the iterative flattening search. In other words, this mechanism tends to maintain a certain distance between the two consecutive solutions. This works as a diversification strategy, with the idea that the more time the search spends visiting new region of the search space, the higher the probability of finding good quality solutions. The experimental section confirms this intuition: iterative flattening search with tabu list improves its performance on the reference strategy.

4 A Meta-heuristics for fine-grained exploration

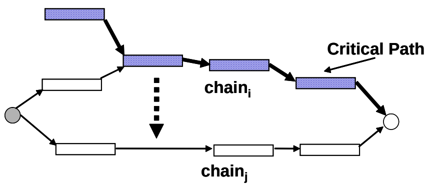
A third potential drawback of the basic iFLATRELAX procedure is its lack of an ability to conduct a *fine-grained* search when a near-optimal solution is generated. In fact, due to its random behavior, the procedure is actually unlikely to to explore *close* neighbors of a near-optimal solution, and hence will miss opportunities to further improve it before moving on to another region of the space.

In this section we describe a complementary extension to designed to overcome this structural drawback of iFLATRELAX. Inspired by the well-known principle in local search that advocates interleaved *diversification* and *intensification* of the search over time (e.g., [7]), we propose a meta-heuristic search strategy that couples the iFLATRELAX procedure with a complementary *tabu-search* procedure. By its nature, iFLATRELAX tends to promote diversification, emphasizing exploration of different subspaces of the search on successive cycles. Introduction of the tabu search procedure, alternatively, provides a mechanism for exploitation of good solutions found by iFLATRELAX, intensifying the search in the neighborhood of such solutions and maximizing the chances of reaching local optima in this particular subspace. To enable integration the tabu search procedure is designed to operate, like iFLATRELAX, with a POS solution representation.

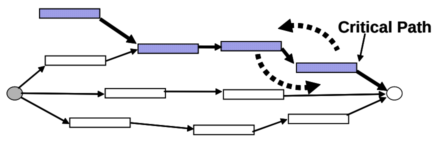
We refer to this meta-heuristic search schema for coupling iFLATRELAX with tabu search as METAFLAT. In the subsections below, we first describe the tabu search procedure that we have developed for this purpose and then summarize the ways in which these two components have been integrated.

4.1 A Tabu Search Procedure on Partial Order Schedules

To complement iFLATRELAX's randomized search behavior, we combine it, in a larger meta-schema, with a tabu search algorithm to enable fine-grained exploration in the neighborhood of near-optimal solutions discovered by iFLAT. In brief, tabu search is a local search procedure that proceeds by iteratively moving from a given current solution S to a new solution S' in the neighborhood of S, until some stopping criterion has been satisfied. For any current solution S, a set of moves m are applied to S to define the neighborhood of S of interest, and then the neighbor S_i with the best objective value (in our case the S_i with the smallest makespan) is selected as the new solution S'. The move leading to the best neighbor is then performed, and a new neighborhood is then calculated and searched to find a new best neighbor S_{i+1}. To hedge against getting trapped in local optima, the procedure modifies its neighborhood structure as the search progresses. This is accomplished by maintaining the set of most recently visited solutions in a *tabu list*, and forbidding selection of a neighbor that has been visited within the past $MaxSt$ moves (where $MaxSt$ is the length of the tabu-list).



(a) Vertical move.



(b) Horizontal move.

Fig. 4. Tabu search moves

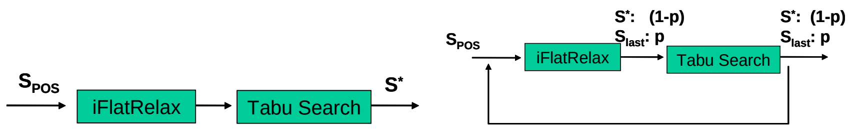
The tabu search procedure we have developed can be seen as an extension of the algorithm first proposed in [8] for Job Shop Scheduling Problems (JSSP). The tabu search procedure is designed to operate on the directed graph $G(A, E)$ representation of a solution (schedule), with analogous definitions of *path*, *length* of a path and *critical path* assumed in the previous sections. In particular, we consider a POS form for the input solutions, such that for each resource r_k with capacity c_k, we assume the activities partitioned in a set of c_k different chains $chain_1, chain_2, \ldots, chain_k$.

Our *tabu search* algorithm interleaves two types of moves in searching for a minimum makespan solution: *vertical moves* and *horizontal moves* (see Fig. 4). A *vertical move* on a resource r_k is defined as the movement of an activity a_m on a *critical path* from one $chain_i$ to another $chain_j$ (Fig. 4(a)). We use a heuristic criterion to determine where in $chain_j$ to insert a_m. For each pair of consecutive activities (a_i, a_{i+1}) in $chain_j$, we compute a penalty function which estimates the increase in the solution's makespan if a_m is inserted between a_i and a_{i+1}, and the insertion point with the minimum penalty is chosen. Alternatively, a *horizontal move* on a resource r_k is defined as the swap of the execution order of a pair of consecutive activities (a_i, a_j) belonging to the same chain and *critical path* (Fig. 4(b)). Our implementation of horizontal moves directly exploits the results introduced in [8] concerning neighborhood definition in the case of unit capacity problems. It is worth noting that by applying the two types of move to a POS we still obtain a partial order schedule. The presence of POS is then crucial to preserve the solution feasibility after a move.

The tabu search algorithm takes as input the initial type of move from which to start (*init-move*), the *tabu-list* length $MaxSt$ and two integer parameters: max_{intrlv} and max_{iter}. These parameters respectively designate the maximum number of interleaving steps between moves of each type without makespan improvements and the maximum number of move steps of a given move type without further makespan improvement. In practice, the algorithm alternates the use of the two types of moves and each time it changes type, restarts from the best solution found with the previous type of move.

4.2 Interleaving Iterative Flattening and Tabu Search

We have defined two possible alternatives for interleaving the IFLATRELAX method and the tabu search procedure (a graphic representation of the two alternatives is shown in Fig. 5). A first approach, called *serial integration*, simply executes IFLATRELAX and tabu search algorithms in sequence, submitting the best solution found in the first step



(a) Serial Integration. (b) Loop Integration.

Fig. 5. Combining Iterative Flattening and Tabu Search. S^* is the best solution found by the component strategy, S_{last} is the last one, and p is the noise parameter.

to a tabu-search session for further improvement. In this case the tabu search procedure works as a classical *intensification* step within a meta-heuristic schema by focusing more intently on a region close to the best solution found by the previous IFLATRELAX step.

A second approach, called *loop integration*, iteratively applies the following two steps until a termination condition is met:

1. the IFLATRELAX procedure takes as input a solution S (the initial solution or the output of the tabu search): with probability p it returns the last solution found in the search process S_{last}; with probability $(1-p)$ the best solution found S^*;
2. the output of the previous step becomes the input of the tabu search procedure: with probability p it returns the last solution found in the local search procedure; with probability $(1-p)$ the best solution found.

The value of probability p works as a *noise* mechanism, to avoid the circumstance where a near optimal solution *circulates* in the loop without any improvement. In fact, it is possible that a solution S cannot be improved either by the IFLAT algorithm or by the tabu search algorithm. When we compose the two basic strategies (IFLAT and Tabu Search) in loop integration, by default each returns the best solution found each time it is invoked. As the makespan decreases, the probability that a component strategy returns the input solution, as the best one, becomes increasingly higher. The noise p promotes restarting from different solutions and thus controls the interplay between search intensification and diversification. In fact, when both component strategies return the best solution found, the search tends to be more localized in a subregion of the search space (intensification). However, in this case where one of the two component procedures returns last solution found - a solution which can be seen as randomly generated - the search tends to move toward another region of the search space. This step can be seen as a diversification step.

5 The MCJSSP Scheduling Problem and Test Sets

As in previous research we use the Multi-Capacity Job-Shop Scheduling Problem (in short MCJSSP) as a basis for evaluating the performance of our search procedures. This problem involves synchronizing the use of a set of resources $R = \{r_1 \dots r_m\}$ to perform a set of jobs $J = \{j_1 \dots j_n\}$ over time. The processing of a job j_i requires the execution of a sequence of m activities $\{a_{i_1} \dots a_{i_m}\}$, each a_{ij} has a constant processing time p_{ij} and requires the use of a single unit of resource $r_{a_{ij}}$ for its entire duration. Each resource r_j is required only once in a job and can process at most c_j activities at the same time ($c_j \geq 1$).

A *feasible solution* to a MCJSSP is any temporally consistent assignment to the activities' start times which does not violate resource capacity constraints. An *optimal solution* is a feasible solution with minimal overall duration or makespan. Generally speaking, MCJSSP has the same structure as JSSP but involves multi-capacitated resources instead of unit-capacity resources.

Benchmarks. For our analysis, we use the benchmarks introduced in [9]. They consist of four sets of problems which are derived from the Lawrence job-shop scheduling problems [10] by increasing the number of activities and the capacity of the resources. In particular we distinguish:

Set A: LA1-10x2x3 (Lawrence's problems numbered 1 to 10, with resource capacity duplicated and triplicated). Using the notation #jobs $\times$ #resources (resource capacity), this set consists of 5 problems each of sizes 20x5(2), 30x5(3), 30x5(2), 45x5(3).

Set B: LA11-20x2x3. 5 problems each of sizes 40x5(2), 60x5(3), 20x10(2), 30x10(3).

Set C: LA21-30x2x3. 5 problems each of sizes 30x10(2), 45x10(3), 40x10(2), 60x10(3).

Set D: LA31-40x2x3. 5 problems each of sizes 60x10(2), 90x10(3), 30x15(2), 45x15(3).

We observe that the proposed benchmark set still represents a challenging and an effective benchmark for comparing algorithms. In fact, (a) in relatively few instances they cover a wide range of problem sizes; (b) they also provide a direct basis for comparative evaluation. In fact, as noted in [9], one consequence of the problem generation method is that the optimal makespan for the original JSSP is also a tight upper bound for the corresponding MCJSSP (Lawrence upper bounds). Hence, even if for many instances there are known better solutions, distance from these upper-bound solutions can provide a useful measure of solution quality.

6 Experimental Results

The experimental analysis has been performed in two phases. A first, explorative, phase evaluates the effect of the set of the *component* modifications described in the previous sections. In this phase we work only with the Set C benchmark. This set is a representative sub-set of instances ranging from 300 to 600 activities. A second phase then compares the best performing procedures from the first phase with current best MCJSSP benchmark results in a more CPU intensive test. All algorithms are implemented in Allegro Common Lisp, and all experiments were run on a P4 processor 1.8 GHz under Windows XP.

Table 1. Comparative performance on Set C

Algorithm	$\Delta LWU_{\%}$	$\Delta iFlat_{\%}$
IFLATRELAX	4.01	0.0
IFLATRELAX-pos	3.61	-0.37
IFLATRELAX-pos-tl	2.93	-1.03
METAFLAT-serial	2.99	-0.97
METAFLAT-loop	2.80	-1.16

Explorative comparison. We have defined a progression of extended strategies with respect to the previous best IFLATRELAX from [2]. In particular they are defined as follows:

1. IFLATRELAX-pos: this variant augments IFLATRELAX with a POS representation of the input solution;
2. IFLATRELAX-pos-tl: in this variant, the previous algorithm is integrated with the *tabu-list* mechanism;
3. METAFLAT-serial: in this configuration, the best solution found by IFLATRELAX-pos is serialized with the tabu search algorithm (Fig. 5(a));
4. METAFLAT-loop: this variant interleaves IFLATRELAX-pos-tl with the tabu search procedure in a loop integration mode (Fig. 5(b)).

In running the first phase experiments, the following settings were used for IFLATRELAX: $P_{rem} = 0.2$, $MaxFail = 400$, $MaxRelaxations = 6$. When a tabu list is used, we adopt the following parameters: length $l = 16$, $\delta = 16$. The Tabu Search parameters were set as follows: *tabu-list*'s length $MaxSt = 9$, *init-move= 'vertical'*, $max_{intrlv} = 1$ and $max_{iter} = 50$. The noise value for the strategy METAFLAT-loop was set to $p = 0.2$. We finally imposed a timeout of 1000 seconds for each instance of the Set C and for each composite strategy. It is worth noting how we have implemented the previous four strategies in order to met the imposed CPU bounds. In the case of the strategy METAFLAT-loop, the procedure executes the loop until the time bound is reached. For the other three strategies, we adopt the same restarting schema used in previous works [1, 2]. In the case a first run finishes before the imposed time limit, the random procedure restarts from the initial solution until the time bound is reached. At the end, the best solution found is returned.

Table 1 compares on the Set C the basic IFLATRELAX with the alternative strategies described above. The column *Algorithm* represents the algorithmic variant, the column $\Delta LWU_{\%}$ represents the average percentage deviation from the Lawrence upper bound [10], and finally, the column $\Delta iFlat_{\%}$, represents the percentage deviation from the performance of the original IFLATRELAX algorithm.

The results shown in Table 1 give a first empirical evidence to the hypotheses described in the previous sections. In fact, the results show that all methods improve on the base IFLATRELAX algorithm. In particular, the simple use of a partial order schedule (POS) as input solution for IFLATRELAX, improves the previous approach described in [2] from 4.01% to 3.61%. This because a partial order schedule entails a reduced number of redundant precedence constraints in the input precedence graph with respect to

Table 2. $\Delta LWU_{\%}$ values on the complete benchmark

Algorithm	**Set A**	**Set B**	**Set C**	**Set D**	**All**
IFLATRELAX-pos	-0.06	-1.38	1.19	0.52	0.07
METAFLAT-loop	-0.07	-1.68	0.69	0.26	-0.2
STRand	-0.28	-2.04	-0.71	-0.22	-0.81

the one produced by ESTA. This different solution's shape tends to intensify the effect of the relaxation step within the IFLATRELAX procedure and increases the degrees of freedom in rearranging the relaxed activities.

Note also that the introduction of the tabu list mechanism, which could appear relatively simple, also produces a rather interesting further improvement to 2.93%. In fact, as described before, this mechanism contrasts the possibility of falling in local optima and allows to explore regions of the search space that would be left unexplored. Moreover it is worth noting that the tabu list mechanism results more efficient than the use, in serial, of the tabu search procedure (in this case we have a $\Delta LWU_{\%}$ value of 2.99%). Finally, as it can be expected, the best performance (2.80%) is obtained with the METAFLAT-loop. This result confirms the general principle adopted in meta-heuristic search to interleave diversification and intensification. For this reason this variant was selected for the intensive experiments of the next phase.

Intensive evaluation. In the second phase of the experiments, we have compared the three following procedures: IFLATRELAX-pos, METAFLAT-loop, and the *STRand* algorithm introduced in [3] which has obtained the best current results.

In this phase, the following settings have been adopted for the IFLATRELAX procedure: $P_{rem} = 0.2$, $MaxFail = 1000$, $MaxRelaxations = 6$. For the tabu list and the tabu search we preserved the same parameters as before. It is worth remarking that the selection of the values of these parameters was not random, but rather the result of a set of preliminary exploratory runs. Finally, for the intensive evaluation we set a timeout of 8000 seconds for each of the 80 instances of the MCJSSP benchmark.

Table 2 shows the figures of the intensive evaluation. For each algorithm we present the $\Delta LUW_{\%}$ values obtained for all the four benchmarks and the cumulative results. The results in the table show that both extended methods introduced, IFLATRELAX-pos and METAFLAT-loop, outperform the original procedure. In fact, although in [2] the authors use a more intensive search, the average $\Delta LWU_{\%}$ is only about 1%, which is significantly higher than the -0.2% obtained with our extensions.

In comparison to the STrand procedure, neither IFLATRELAX-pos and METAFLAT-loop achieve the level of performance of the STRand procedure [3], which currently maintains the best performance on the MCJSSP benchmarks. However in this case it is worth again noting that while this approach shares the same search schema of IFLATRELAX it uses different components to implement the flattening and the relaxation steps. Additional analysis is needed to assess the real difference between METAFLAT-loop and STRand, given that the results of the latter are from the original paper and take advantage of an implementation built on top of the ILOG suite. One of our next steps will be the implementation of the STRand algorithm within our algorithmic framework (written in Common Lisp), in particular of the algorithm SetTimes – a description of this algorithm can be found in [11] – used in place of ESTA within STRrand. Moreover we expect that the STRand procedure may benefit from one or more of the same extensions that we have introduced into METAFLAT-loop. In fact, it suffers of some of the same limitations of iterative flattening: the potential for repeatedly searching the same solution subspace and the inability to explore the close neighborhood of a near-optimal solution. The empirical experimentation demonstrates the effectiveness of both the tabu-list mechanism and the use of a companion tabu search strategy in conjunction with the iterative flattening search schema.

We conclude with one final remark: a possible criticism of our approach is the need to tune several input parameters. However, such tuning of parameters is "endemic" to every meta-heuristic approach, particularly when composing several basic strategies. At the same time, we consider all extensions we have introduced in the meta-heuristic strategy to be quite natural, each compensating for some basic limitation of the basic IFLATRELAX strategy. In fact, the results confirm the earlier observation that the composite strategy METAFLAT-loop improves over the performance of IFLATRELAX. Additionally, the introduction into the iterative flattening schema of partial order schedules, the tabu list and the tabu search further boosts the performance of the approach described in [2].

7 Conclusions and Future Work

In this paper we have explored a set of extensions to the IFLAT family of search procedures and two meta-heuristic procedures based on iterative flattening search. This is a local search procedure for solving large-scale scheduling problems with a makespan minimization objective criterion [1, 2]. The works presented were motivated by three potential limitations in the IFLAT algorithm: (1) the lack of flexibility in the initial seed solution, (2) the potential for repeatedly searching the same solution subspace due to the nature of the relaxation (i.e., constraint retraction) step, and (3) the inability of IFLAT procedure to explore the close neighborhood of a near-optimal solution for purposes of further improvement.

To address these issues the IFLAT search schema was enhanced in several ways, including use of a partial order schedule as the initial seed solution, the addition of a *tabu-list* to better control the relaxation step, and integration with a complementary *tabu-seach* procedure to refine solutions found by means of IFLAT.

The proposed extensions were found to significantly improve the performance of the reference strategies on benchmark multi-capacitated job-shop scheduling problems, and these results give first experimental evidence of the effectiveness of coupling tabu search concepts and procedures with iterative flattening search. Further study will be necessary to clearly understand the effectiveness of the algorithms proposed, especially with regard to the best results available in the current literature and given by STRand [3]. However, we believe that the proposed extensions are quite general and can be also usefully used within the STRand algorithm. For example, the tabu list mechanism can also be useful in the relaxation phase of STRand where constraints are removed in a random way.

There are further directions for future research. One particular interest is investigation of a more sophisticated tabu list mechanism, which biases the tenure value according to the estimated quality of a given constraint. Another general focus will be exploration of alternative approaches to integrating iterative flattening and tabu search. In this regard, we believe a Back Jumping Tracking schema [8], where search is restarted from promising solutions accumulated during the search, holds particular promise.

References

1. Cesta, A., Oddi, A., Smith, S.F.: Iterative flattening: A scalable method for solving multi-capacity scheduling problems. In: AAAI/IAAI, Seventeenth National Conference on Artificial Intelligence. (2000) 742–747
2. Michel, L., Van Hentenryck, P.: Iterative relaxations for iterative flattening in cumulative scheduling. In: ICAPS. (2004) 200–208
3. Godard, D., Laborie, P., Nuitjen, W.: Randomized Large Neighborhood Search for Cumulative Scheduling. In: Proceedings of the International Conference on Automated Planning & Scheduling (ICAPS 2005). (2005)
4. Policella, N., Smith, S.F., Cesta, A., Oddi, A.: Generating robust schedules through temporal flexibility. In: ICAPS. (2004) 209–218
5. Cesta, A., Oddi, A., Smith, S.: Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In: Proceedings of the Fourth Int. Conf. on Artificial Intelligence Planning Systems (AIPS-98). (1998)
6. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence **49** (1991) 61–95
7. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput. Surv. **35** (2003) 268–308
8. Nowicki, E., Smutnicki, C.: A Fast Taboo Search Algorithm for the Job Shop Problem. Management Science **42** (1996) 797–813
9. Nuijten, W., Aarts, E.: A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling. European Journal of Operational Research **90** (1996) 269–284
10. Lawrence, S.: Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement). Technical report, Graduate School of Industrial Administration, Carnegie Mellon University (1984)
11. Junker, U.: Preference-Based Search for Scheduling. In: Proceedings of the 17^{th} National Conference on Artificial Intelligence. (2000) 904–909