# Profile-Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems

**Amedeo Cesta**
IP-CNR
National Research Council
Viale Marx 15
I-00137 Rome, Italy
amedeo@pscs2.irmkant.rm.cnr.it

**Angelo Oddi**
IP-CNR
National Research Council
Viale Marx 15
I-00137 Rome, Italy
oddi@pscs2.irmkant.rm.cnr.it

**Stephen F. Smith**
The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
sfs@isl1.ri.cmu.edu

## Abstract

Though CSP scheduling models have tended to assume fairly general representations of temporal constraints, most work has restricted attention to problems that require allocation of simple, unit-capacity resources. This paper considers an extended class of scheduling problems where resources have capacity to simultaneously support more than one activity, and resource availability at any point in time is consequently a function of whether sufficient unallocated capacity remains. We present a progression of algorithms for solving such multiple-capacitated scheduling problems, and evaluate the performance of each with respect to problem solving ability and quality of solutions generated. A previously reported algorithm, named the Conflict Free Solution Algorithm (CFSA), is first evaluated against a set of problems of increasing dimension and is shown to be of limited effectiveness. Two variations of this algorithm are then introduced which incorporate measures of temporal flexibility as an alternative heuristic basis for directing the search, and the variant making broadest use of these search heuristics is shown to yield significant performance improvement. Observations about the tendency of the CFSA solution approach to produce unnecessarily overconstrained solutions then lead to development of a second heuristic algorithm, named Earliest Start Time Algorithm (ESTA). ESTA is shown to be the most effective of the set, both in terms of its ability to efficiently solve problems of increasing scale and its ability to produce schedules that minimize overall completion time while retaining solution robustness.[1]

## Introduction

Over the past few years, constraint satisfaction problem solving (CSP) techniques have been productively applied to several classes of scheduling problem (e.g., (Cheng & Smith 1994; 1996; Nuijten & Aarts 1994; Oddi & Cesta 1997; Sadeh 1991; Oddi & Smith 1997)). One advantage of CSP approaches to scheduling is the generality of their underlying representational assumptions. Techniques have been proposed, for example, which operate relative to very general temporal constraint models and support resource allocation under complex quantitative time constraints. Thus, in contrast to classical scheduling approaches which tend toward specialized solutions to idealized problem formulations, the representational assumptions underlying CSP scheduling techniques are well matched to the modeling requirements of practical scheduling problems.

However, current CSP scheduling models are not without modeling limitations. Most research has focused on problems that require allocation of simple, "unit-capacity" resources; i.e., resources that must be dedicated exclusively to performing any given activity and, at any point in time, are either "available" or "in-use". In many practical domains, resources in fact have the *capacity* to support some number of activities at any point in time. Indeed, the inability to accommodate capacity resources and to solve multiple-capacitated scheduling problems is perhaps the principal obstacle to widespread application of most CSP scheduling models and heuristics. In this paper, we take some steps toward removing this obstacle.

Let us introduce a very intuitive example of the type of multiple-capacitated scheduling problem we will focus on: suppose that a vacation house has a limited amount of power supply (a generator with capacity $k = 2$) and a family (two parents and a child) rents it for a week-end. They are unaware on the power limitation upon arrival but soon after realize that this will be a serious constraint on their week-end activities. For example Tim, the son, is unable to switch on the hair-drier after taking a shower because the water-heater is on and his mother is simultaneously cooking an apple pie in the electric oven (we assume that any electric device consumes one unit of power supply). It is easy to see that it is not possible to solve these resource availability conflicts by simply postponing one or more of these conflicting activities, because these activities are connected to others by temporal relations: Tim cannot postpone his hair drying for more than 5 minutes after having a shower because it is winter time and he risks catching a cold; mother cannot indefinitely delay the baking of the pie once it has been prepared, because this may harm its taste. In general it is necessary to postpone or interleave blocks of temporally connected activities, and different decisions can have significant impact on how efficiently resources are used and how efficiently activities can be completed.

[1]Copyright (c) 1998 American Association for Artificial Intelligence (www.aaai.org).

The type of scheduling problem illustrated above is distinguished by the need to reason about capacity resources in the presence of quantitative time constraints on the starts, ends and durations of individual activities as well as on the temporal separations between related activities. We call this particular class of scheduling problem the Multiple Capacitated Metric Scheduling Problem (MCM-SP). Variants of the MCM-SP are commonplace in practical domains. Aircraft transportation scheduling applications, for example, generally require management of capacity resources at airports (e.g., onload/offload capacity, aircraft parking space, cargo storage area) while simultaneously enforcing complex temporal constraints on airlift activities (e.g., minimum time on ground, takeoff/landing separation times). Similarly, in most manufacturing environments, production activities must be synchronized to account for the finite processing capacity of various machining centers and operator pools while respecting the ordering, timing and separation constraints on various production steps.

In this paper, we define and evaluate a set of "profile-based" methods for solving MCM-SP. Profile-based solution methods rely centrally on a temporal function called a *demand profile* $D_j(t)$ which describes the quantity of resource $r_j$ required by various activities at time $t$. Generally speaking, profile based solution methods proceed by detecting time instants in which the profile $D_j(t)$ is greater than the available capacity of resource $r_j$ and then "leveling" these utilization peaks by posting additional ordering constraints between those activities competing for $r_j$'s capacity.

Our starting point is a solution procedure based directly on the techniques described in (El-Kholy & Richards 1996). This approach, which emphasizes contention-based criteria for prioritizing the leveling decisions that must be made, is found to be ineffective as problem size is increased. We then propose variants of this procedure which alternatively use the heuristic criteria proposed in (Cheng & Smith 1994) (based on the idea of maintaining sequencing flexibility) to prioritize and make leveling decisions. These variants are shown to yield significant performance improvements. Finally, we propose a new algorithm, also incorporating the heuristic ideas of (Cheng & Smith 1994) but in this case designed specifically to emphasize construction of compact solutions. We show that this approach outperforms all other procedures, both in terms of the number of solved problems and overall solution quality.

The remainder of the paper is organized as follows: first, the MCM-SP is formally defined, and connections and distinctions with related literature are discussed; second, the problem of generating reproducible random instances of MCM-SPs is addressed and performance evaluation metrics are defined; third, a progression of profile-based solution methods is presented and evaluated on the controlled set of problems. Some concluding remarks close the paper.

## Definition of MCM-SP

The Multiple-Capacitated Metric Scheduling Problem (MCM-SP) considered in this work involves synchronizing the use of a set of resources $R = \{r_1 \ldots r_m\}$ to perform a set of jobs $J = \{j_1 \ldots j_n\}$ over time. The processing of a job $j_i$ requires the execution of a sequence of $n_i$ activities $\{a_{i1} \ldots a_{in_i}\}$, a resource $r_j$ can process at most $c_j$ activities at the same time (with $c_j \geq 1$) and execution of each activity $a_{ij}$ is subject to the following constraints:

- *resource availability* - each $a_{ij}$ requires the use of a single resource $r_{a_{ij}}$ for its entire duration.
- *processing time constraints* - each $a_{ij}$ has a minimum and maximum processing time, $proc_{ij}^{min}$ and $proc_{ij}^{max}$, such that $proc_{ij}^{min} \leq e_{ij} - s_{ij} \leq proc_{ij}^{max}$, where the variables $s_{ij}$ and $e_{ij}$ represent the start and end times respectively of $a_{ij}$.
- *separation constraints* - for each pair of successive activities $a_{ij}$ and $a_{i(j+1)}$, $j = 1 \ldots (n_i - 1)$, in job $j_i$, there is a minimum and maximum separation time, $sep_{ik}^{min}$ and $sep_{ik}^{max}$, such that $\{sep_{ik}^{min} \leq s_{i(k+1)} - e_{ik} \leq sep_{ik}^{max} : k = 1 \ldots (n_i - 1)\}$.
- *job release and due dates* - Every job $j_i$ has a release date $rd_i$, which specifies the earliest time that any $a_{ij}$ can be started, and a due date $dd_i$, which designates the time by which all $a_{ij}$ must be completed.

A solution to a MCM-SP is any temporally consistent assignment of start and end times which does not violate resource capacity constraints.

It is worth noting the main features of the problem: (a) the resources are discrete (and not consumable) with capacity $c_j$ greater than or equal to one; (b) all temporal constraints are quantitative and expressed as a bound between a maximal and minimal duration of a distance; (c) an important role is played by the presence of separation constraints between activities; these constraints in fact have a crucial role in increasing the intrinsic difficulty of the problem. They constrain the "fluidity" of a single activity in the solution to be dependent on a set of connected activities. This fact introduces a further level of interdependency between activity start and end time assignments.

**Time and Resource Constraints.** The style of solution investigated here is based on the explicit management of sets of constraints that the solution should satisfy. To sharpen the basis for comparison of alternative algorithms, we will assume a common constraint representation and management infra-structure.

*Time Ontology.* To support the search for a consistent solution, a directed graph $G_d(V, E)$, named *distance graph*, is defined, wherein the set of nodes $V$ represents time-points $tp_i$ and the set of edges $E$ represents temporal distance constraints. The origin, together with the start time $s_{a_k}$ and end time $e_{a_k}$ of each activity $a_k$, comprise the set of represented time points in $G_d$ for any given MCM-SP. Activity processing time constraints, as well as separation constraints and precedence constraints between pairs of activities, are encoded (naturally) as distance constraints. Every constraint in $G_d$ is expressed as a bound on the differences between time-points $a \leq tp_j - tp_i \leq b$ and is represented in $G_d(V, E)$ with two weighted edges:

the first one directed from $tp_i$ to $tp_j$ with weight $b$, the second one from $tp_j$ to $tp_i$ with weight $-a$. The graph $G_d(V, E)$ corresponds directly to a *Simple Temporal Problem* (STP) (Dechter, Meiri, & Pearl 1991) and its consistency can be efficiently determined via shortest path computations (Ausiello *et al.* 1991; Oddi 1997). Thus, the search for a solution to a MCM-SP can proceed by repeatedly adding new precedence constraints into $G_d$ to resolve detected conflicts and recomputing shortest path lengths to confirm that $G_d$ remains consistent (i.e., no negative weight cycles). We let $d(tp_i, tp_j)$ designate the shortest path length in graph $G_d(V, E)$ from node $tp_i$ to node $tp_j$. Note that, given a $G_d$, the distance between any pair of time points satisfies the constraint $tp_j - tp_i = [-d(tp_j, tp_i), d(tp_i, tp_j)]$ (Dechter, Meiri, & Pearl 1991).

*Resource Ontology.* Each problem relies on a set of resources $R$. We denote the required capacity (also called resource demand) of a resource $r_j \in R$ by an activity $a_k$ as $rc_{a_k,j}$ , and the set of activities $a_k$ that demand resource $r_j$ as $A_j$ (i.e., $A_j = \{a_k | \ rc_{a_k,j} \neq 0\}$). Then for each resource $r_j$ a *Demand Profile* $D_j(t)$ can be defined, representing the total capacity requirements of $a_k \in A_j$ at any instant of time. For $r_j$ then, the resource capacity constraint requires $D_j(t) \leq c_j$ for all $t$. For purposes of this paper, we assume that $rc_{a_k,j} = 1$ for all $a_k$.

**Related Literature.** An earlier work that formalizes the scheduling problem as a CSP (Constraint Satisfaction Problem) is (Sadeh 1991). This work focuses on the JSSP (Job-Shop Scheduling Problem) where: (a) temporal constraints represent constant durations of activities; (b) separation between activities are simple qualitative ordering constraints; (c) capacity constraints are binary (a resource is either busy or free).

An extension of JSSP to multi-capacitated problems is developed in (Nuijten & Aarts 1994). The paper introduces and presents a solution procedure for the Multi-Capacitated JSSP (MCJSSP), which retains the temporal constraint model of the classical JSSP but allows resource capacities to be greater than one. In (Smith & Westfold 1996), a similar problem involving a single, multi-capacitated resource is addressed.

An extension of JSSP to include the more general temporal constraint assumptions of an STP (while retaining the assumption of binary capacity constraints) is introduced and solved in (Cheng & Smith 1994), and also subsequently considered in (Oddi & Smith 1997). A variant of this problem where temporal constraints may be relaxed is discussed in (Oddi & Cesta 1997).

Representations and solution procedures that simultaneously account for STP-style temporal constraints and multi-capacitated resource constraints have recently been proposed within the planning literature, principally to provide resource reasoning subcomponents within larger planning architectures (Laborie & Ghallab 1995; El-Kholy & Richards 1996; Cesta & Stella 1997). These efforts will be considered later on in the paper.

## A Controlled Experimental Setting

The scheduling problem introduced here has not been carefully studied from previous research. It is thus important at the outset to consider issues related to establishing a common basis and experimental design for comparison of alternative solution procedures. Below, we consider in turn two critical issues in this regard: (a) generation of benchmark problems, and (b) development of measures that characterize the quality of an algorithm and the solution it produces.

**Generation of Problem Instances.** The first step toward establishing an experimental design is the implementation of a controlled random number generator. We adopt the generator proposed in (Taillard 1993) (pag.179) and obtain the uniform distribution function $U[a, b]$ which generates a random number $n$, where $n$, $a$ and $b$ are positive numbers such that $a \leq n \leq b$ [2]. To generate different problem instances we use the time seeds reported in Figure 1 of (Taillard 1993) (in particular the first 50 seeds in this paper).

Next we define the dimensions along which problem instances will be varied. Using the format for formulating job shop scheduling problems, we call $N_{jobs} \times N_{res}$ the problem with $N_{jobs}$ each of them composed of a sequence of $N_{res}$ activities that must be executed on one of the $N_{res}$ different resources. For purposes of this paper, we create problem sets of 50 instances for each of the following sizes: $5 \times 5$, $10 \times 5$, $15 \times 5$, $20 \times 5$, and $25 \times 5$. To generate random instances of MCM-SP of the above sizes we assign the remaining data as follows:

- Each resource $r_j$ has capacity $c_j$ generated randomly as $U[2, 3]$, with full availability over the horizon.
- The minimum processing time of activities is drawn from a uniform distribution $U[10, 50]$, and the maximum processing time is generated by multiplying the minimum processing time by the value $(1 + p)$, where $p = U[0, 0.4]$.
- The separation constraints $[a, b]$ between every two consecutive activities in a job are generated with $a = U[0, 10]$ and $b = U[40, 50]$.
- Release and due dates for jobs are not considered explicitly in the current experiments so they are fixed to 0 and $H$ respectively for all the jobs.

Finally, the horizon H is computed by the following formula adapted from (Cheng & Smith 1994): $H = (N_{jobs} - 1)p_{bk} + \sum_{i=1}^{N_{res}} p_i$, where $p_{bk}$ is the average minimum processing time of the activity on the bottleneck resource, and $p_i$ is the average minimum processing time of the activity on resource $r_i$. The bottleneck resource is the resource with maximum value of the sum of the minimum processing time of the activities which request the resource.

**Evaluation Criteria.** The identification of criteria for assessing the relative effectiveness of alternative solution methods and solutions is always somewhat con-

[2] If $a$ and $b$ are integers then $n$ is obtained as an integer, if one of them is real $n$ is real.

troversial. For purposes of this paper, we consider the quality of a given solution procedure to be given as

- *number of solved problems* from a fixed set ($N_s$);
- *average CPU time* spent to solve instances of the problem ($CPU$).
- the *number of leveling constraints posted* in the solution $S$ ($N_{lc}$). This number gives a further structural information of the kind of solution created. It indicates the number of sequencing constraints introduced by the algorithm to resolve competing resource capacity requests.
- the average quality of the solutions generated.

We consider two factors as contributing to judgements of solution quality. From an optimization viewpoint, we will measure the *compactness* of the solution and from an executability viewpoint, we will characterize its *robustness*. These notions are formalized below.

One commonly used measure of schedule quality is its overall makespan. Consequently we consider the *makespan* $M_{ks}$ (or *minimum completion time*) of a solution to be one indicator of quality.

Robustness is a trickier notion to formalize. It is commonly believed that robustness is related to the possibilities for obtaining minor variations to a solution (at execution time for example) without incurring in a major disruption to the overall solution. In general, when working with strongly temporalized domains (as is the case in this paper) this capability seems connected to the degree of "fluidity" of the solutions. The kind of measure we are trying to define takes into account the fact that a temporal variation concerning an activity is absorbed by the temporal flexibility of the solution instead of generating a deleterious "domino effect" in the solution itself.

Building on this intuition, we define the *robustness* RB of a solution S to be the average width, relative to the temporal horizon H, of the distance, for all pairs of activities, between the start time of one activity and end time of the other. More precisely:

$$RB(S) = \sum_{a_i \in S, a_j \in S, a_i \neq a_j} \frac{|d(e_{a_i}, s_{a_j}) - d(s_{a_j}, e_{a_i})|}{H \times (N_a \times (N_a - 1))} \times 100$$

where $N_a$ is the number of activities in the solution, $a_k$ is a generic scheduled activity, $d(s_{a_j}, e_{a_i})$ is the length of the shortest path between the start-time of $a_j$ and the end-time of $a_i$, 100 is a scaling factor. With RB(S), we at least obtain a coarse idea of the reciprocal "shifting" potential between pairs of activities in a solution. In fact, we focus on the current average bounds on distances between the end time and start time of any pair of activities in the schedule.

## Profile-Based Algorithms

We can now define and evaluate alternative procedures for solving MCM-SPs. From previous work that has taken into account multiple capacitated resources, we can distinguish at least two general approaches to the identification of resource conflicts [3]:

- *profile-based approaches* (El-Kholy & Richards 1996; Cesta & Stella 1997): these approaches are extensions of a technique quite common in unit-capacity scheduling (e.g. (Sadeh 1991)). Most generally, they consist of characterizing resource utilization demand as a function of time, identifying periods of resource overallocation in this utilization, and incrementally performing "leveling actions" to (hopefully) ensure that resource usage peaks fall below the total capacity of the resource;
- *clique-based approaches* (Laborie & Ghallab 1995): given a current schedule, this approach builds up a "conflicts graph" whose nodes are activities and whose edges represent possible resource conflicts between the connected activities. Fully connected subgraphs (cliques) are identified and if the number of nodes in the clique is greater than resource capacity a conflict is detected.

In this paper we restrict attention to the profile-based solution methods and consider several different algorithms based on this idea. All proposed strategies have in common the basic concept of *demand profile* $D_j(t)$. The goal of a *profile-based* scheduling algorithm is to create a temporally consistent schedule where in every time instant the demand profile of any resource satisfies the resource constraints $D_j(t) \leq c_j$.

In (Cesta & Stella 1997) the complementary aspect of $D_j(t)$ , resource availability $Q_j(t) = c_j - D_j(t)$, is considered, and a set of constraint synthesis techniques is introduced that perform filtering of the search space during resolution. In (El-Kholy & Richards 1996) a more limited form of resource constraint propagation is used. However, a leveling heuristic is also given so that a procedure can be reproduced which forms a complete scheduling algorithm. For this reason, we have chosen this second procedure as the starting point of our investigation and analysis.

## Conflict Free Solution Algorithm (CFSA)

The Conflict Free Solution Algorithm (CFSA) which we first consider is based directly on the procedure defined in (El-Kholy & Richards 1996). The principal point of departure in our formulation has been to eliminate the backtracking search framework that is employed in (El-Kholy & Richards 1996) in favor of a greedy, backtrack-free search model. This more restrictive search model obviously places much stronger importance on the heuristics used to direct the search, and one of our interests in this paper is to improve upon these heuristics. But also, from a pragmatic standpoint, the scale and complexity of the MCM-SP problem that is considered here dwarfs the types of problems considered in (El-Kholy & Richards 1996) and a complete, back-tracking search model is not viable computationally.

Before presenting the initial CFSA algorithm itself, some preliminary definitions are needed.

[3] Any classification of existing works might be partial and a bit arbitrary. For example the current classification is enough for purposes of the paper but does not represent well proposals like the "energetic approach" in (Erschler, Lopez, & Thuriot 1990).

*Demand Profile Bounds.* Due to the temporal flexibility of an intermediate solution, an exact computation of the $D_j(t)$ is not possible but upper and lower bounds for $D_j(t)$ may be defined. Without loss of generality, such bounds are computed taking into account only activity start times. This is because it is in such points that a positive variation of the demand profile happens. Broadly speaking, for each start time $s_{a_i}$, the value of the lower bound profile (upper bound profile) is the sum of the capacity requirements of the activities whose execution interval necessarily (possibly) intersects with the time point $s_{a_i}$.

A more formal definition is based on the temporal distances on the graph $G_d$. The *lower-bound profile* of a resource $r_j$ is defined as:

$$LB_{D_j}(s_{a_i}) = \sum_{a_k \in A_j} P_{ik} \times rc_{a_k,j}$$

where $P_{ik} = 1$ if $d(s_{a_i}, s_{a_k}) \leq 0 \wedge d(e_{a_k}, s_{a_i}) < 0$ and $P_{ik} = 0$ otherwise. Similarly the *upper-bound profile* of a resource $r_j$ is defined as:

$$UB_{D_j}(s_{a_i}) = \sum_{a_k \in A_j} P'_{ik} \times rc_{a_k,j}$$

where $P'_{ik} = 1$ if $d(s_{a_k}, s_{a_i}) > 0 \vee d(s_{a_i}, e_{a_k}) > 0$ and $P'_{ik} = 0$ otherwise.

*Resource Conflicts.* The concept of *peak* formalizes the idea of conflict (or contention) in the use of a resource. A *peak* is the couple $<s_{a_i}, \{a_j\}>$, where $s_{a_i}$ is the start time of an activity $a_i$ and $\{a_j\}$ is a set of activities which produces a violation of the resource constraints. Three types of peaks are introduced:

- *lb-peak*: a lower-bound peak where $LB_{D_j}(s_{a_i}) = c_j$ and $UB_{D_j}(s_{a_i}) > c_j$;
- *unres-peak*: an unresolvable peak where $LB_{D_j}(s_{a_i}) > c_j$;
- *ub-peak*: an upper bound peak where $LB_{D_j}(s_{a_i}) < c_j$ and $UB_{D_j}(s_{a_i}) > c_j$.

*Conflict Resolution.* A solution is always obtainable when $UB_{D_j}(s_{a_i}) \leq c_j$ for each resource $r_j$. In such a case any consistent assignment of values to the solution's time points gives a solution where resource utilizations are consistent. A conflict free solution is built by iteratively posting leveling constraints on the *lb-peaks* and *ub-peaks* until every peak is removed. Given a set of peaks, the leveling action can be formalized in two steps:

1. *conflict selection*: a conflict in this case is a couple $<s_{a_i}, a_j>$ (notice a time-point and an activity) in which the condition $P_{ij} = 0 \wedge P'_{ij} = 1$ holds;
2. *leveling*: one of the two leveling constraints $s_{a_i}\{before\}s_{a_j}$ or $e_{a_j}\{before\}s_{a_i}$ is selected and posted in the partial solution.

Note that it may be the case that one of the two possible leveling constraints cannot be posted, because of its inconsistency in the graph $G_d$. In this case, the choice is uniquely determined. When both constraints are consistent alternatives, a heuristic choice is needed. The strategy followed in (El-Kholy & Richards 1996), according to a least commitment principle, first makes choices in the area of the solution where the most constraints have already been posted. The algorithm first detects the *lb-peaks*, where the lower bound of the demand has reached the maximal resource availability, and tries to satisfy $UB_{D_j}(s_{a_i}) \leq c_j$ by adding leveling constraints. Note that all the *lb-peaks* must be necessarily leveled to ensure a conflict free solution, so the heuristic choice concerns only the order in which the conflicts within a lb-peak are considered and which leveling constraint is applied when both possibilities are consistent. After having leveled the *lb-peaks*, the algorithm starts to consider the *ub-peaks*. In this case the heuristic choice concerns not only the conflict choice and the leveling constraints but also the order in which the *ub-peaks* are considered for leveling.

*The CFSA Algorithm.* The basic CFSA algorithm is sketched in Figure 1. On any given cycle, it first looks for unresolvable peaks (function `Exists-Unresolvable-Peak`). If any are found the algorithm stops. Otherwise the presence of *lb-peaks* is detected (`Exists-Lb-Peak`), the heuristic `Select-Lb-Peak-Lev-Constr` is applied to select a leveling constraint, and it is posted in the underlying temporal constraint graph $G_d$.

```
Conflict-Free-Solver(mcmsp)
1.  loop
2.     if Exists-Unresolvable-Peak(mcmsp)
3.     then return(Failure)
4.     else begin
5.        if Exists-Lb-Peak(mcmsp)
6.        then begin
7.           LC:= Select-Lb-Peak-Lev-Constr(mcmsp)
8.           Post-Lev-Constr(LC)
9.        end
10.       else if Exists-Ub-Peak(mcmsp)
11.            then begin
12.               LC:= Select-Ub-Peak-Lev-Constr(mcmsp)
13.               Post-Lev-Constr(LC)
14.            end
15.            else return(Solution)
16. end-loop
```

Figure 1: Conflict Free Solution Algorithm

The choice is implemented as follows: in the set of all *lb-peaks* $\{<s_{a_i}, \{a_j\}>\}$ the set of *conflicts* which have a unique solution (only one of $s_{a_i}\{before\}s_{a_j}$, $e_{a_j}\{before\}s_{a_i}$ is consistent) is computed. The conflicts in the set are randomly chosen and leveled by posting the uniquely determined ordering constraint. In none of these conflicts exist, the set of conflicts where both leveling constraints are possible is computed and iteratively one is randomly chosen and leveled by randomly posting one of the two ordering constraints. If, in the current solution, the *lb-peaks* have been leveled, the existence of *ub-peaks* is checked (`Exists-Ub-Peak`). If any peaks are found, a differ-

ent leveling heuristic (`Select-Ub-Peak-Lev-Constr`) is applied: in the subset of the *ub-peaks* where the value $LB_{D_j}(s_{a_i})$ is maximal, one of the *ub-peaks* is selected where the value $UB_{D_j}(s_{a_i})$ is minimal. Within this *ub-peak* the same strategy used for *lb-peaks* is applied to level conflicts. When all *ub-peaks* have been leveled a solution is found.

**Experimental Evaluation.** The CSFA algorithm was run on generated problem sets of increasing size and the results are given in Table 1 [4]. We call this version of the algorithm CFSA+PK to stress the composition of a generic conflict free strategy with the particular mechanism that focuses on peaks. Each problem set consists of 50 randomly generated problem instances.

It is interesting to notice that the performance of CSFA, although effective on small problems, degrades rather markedly as problem size increases. At the two largest problem sizes, the algorithm is able to solve very few problems.

Table 1: Experiments with CFSA+PK

| Probl | $N_s$ | $RB$ | $M_{ks}$ | $N_{lc}$ | $CPU$ |
|---|---|---|---|---|---|
| 5×5 | 50 | 5.9 | 241.7 | 29.3 | 0.5 |
| 10×5 | 48 | 2.4 | 424.0 | 114.5 | 7.0 |
| 15×5 | 32 | 1.2 | 606.7 | 212.8 | 25.8 |
| 20×5 | 5 | 0.9 | 785.2 | 365.8 | 67.9 |
| 25×5 | 1 | 0.5 | 884.0 | 497.0 | 133.2 |

## Retaining Temporal Flexibility

The heuristics embedded in the basic CFSA procedure just described attempt to first level a peak where the *lower-bound profile* is maximal and at the same time the *upper bound profile* is minimal. In this way, the leveling action is concentrated in the area of the current solution where it is more probable that an *unres-peak* will appear. However, we observe that the selection of the leveling constraints is randomly done without any evaluation of its temporal commitment on the current solution. On the contrary, as observed in (Cheng & Smith 1994), such analysis can provide crucial guidance for finding solutions to scheduling problems with temporal separation between activities.

Following this observation, we propose an extension of the basic CFSA procedure that utilizes the notion of temporal flexibility as a heuristic basis for prioritizing leveling decisions. At each (previously random) decision point, we propose instead to non randomly make the choice which leaves the maximal degree of temporal flexibility. Intuitively, the notion of temporal flexibility is related to the idea of how many "temporal positions" the time points in a solution may assume with respect to each other (i.e., how many consistent assignments remain). Given two partial solutions $S_1$ and $S_2$ where the degree of temporal flexibility of $S_1$ is much greater than $S_2$ the probability to get a complete solution from $S_1$ is greater because there is more "free temporal space" in it.

We can define the notion of temporal flexibility between any pair of time points in a way similar to (Cheng & Smith 1994). Assume $tp_i$ and $tp_j$ are two time points in a solution's temporal network where the following relation holds: $tp_j - tp_i \in [-d(tp_j, tp_i), d(tp_i, tp_j)]$. We define *temporal flexibility* ($flex(tp_i, tp_j)$) associated to the pair $(tp_i, tp_j)$ the amplitude of the interval of temporal distances between $tp_i$, $tp_j$ with the condition $tp_i \leq tp_j$. For example, in the case that $tp_1 - tp_2 \in [-5, 10]$ the temporal flexibilities are $flex(tp_2, tp_1) = 10$ and $flex(tp_1, tp_2) = 5$. Instead, in the case $tp_1 - tp_2 \in [12, 16]$, $flex(tp_2, tp_1) = 4$ and $flex(tp_1, tp_2) = 0$.

When a temporal constraint is posted between $tp_i$ and $tp_j$ a fixed amount of temporal flexibility is removed from the temporal network. In particular, when solving MCM-SPs, we use only constraints of the type $tp_i\{before\}tp_j$ which removes an amount of temporal flexibility equal to $flex(tp_j, tp_i)$. This simple idea is used in the algorithms proposed in the next paragraphs where the goal of the heuristic is to improve the probability to find a solution by maintaining the value $\sum_{tp_i, tp_j} flex(tp_i, tp_j)$ as high as possible.

## Heuristics Using Temporal Flexibility

To incorporate a bias toward retention of temporal flexibility into the basic CFSA procedure, we need to specify two heuristic estimators. The first estimator is used to select a conflict $<s_{a_i}, a_j>$ from a set of peaks and it is computed as follows:

- *Conflict Selection (CS)* - Given a set of *peaks*, the selection of a single *conflict* $<s_{a_i}, a_j>$ is done on the basis of a measurement of the temporal flexibility associated to $<s_{a_i}, a_j>$. We use the two temporal distances on the $G_d$ graph $d(s_{a_i}, s_{a_j})$ and $d(e_{a_j}, s_{a_i})$. In the case that for each $<s_{a_i}, a_j>$ both the ordering choices $s_{a_i}\{before\}s_{a_j}$ and $e_{a_j}\{before\}s_{a_i}$ are temporally consistent ($d(s_{a_i}, s_{a_j}) > 0$ and $d(e_{a_j}, s_{a_i}) > 0$) then, the conflict $<s_{a_k}, a_m>$ is selected with the minimum value $\omega_{res}(s_{a_k}, a_m)$ [5], where:

$$\omega_{res}(s_{a_k}, a_m) = min\{\frac{d(s_{a_k}, s_{a_m})}{\sqrt{S}}, \frac{d(e_{a_m}, s_{a_k})}{\sqrt{S}}\}$$

with $S = \frac{min\{d(s_{a_k}, s_{a_m}), d(e_{a_m}, s_{a_k})\}}{max\{d(s_{a_k}, s_{a_m}), d(e_{a_m}, s_{a_k})\}}$. Otherwise, in the case there is at least one *conflict* which can be leveled in a unique way (one of the distances $d(s_{a_i}, s_{a_j})$ or $d(e_{a_j}, s_{a_i})$ is $\leq 0$), the conflict is selected with the minimum value $\omega_{res}(s_{a_k}, a_m) = min\{d(s_{a_k}, s_{a_m}), d(e_{a_m}, s_{a_k})\}$—the conflict closer to the resource consistent status.

[4] The CPU times given here and in subsequent tables are in seconds. All procedures described in this paper were implemented in Commonlisp and all experiments were run on a SUN Sparcstation 10.

[5] As suggested in (Cheng & Smith 1994) a balancing factor $\sqrt{S}$ is used. It is possible to see that $S \in [0, 1]$: $S = 1$ when $d(s_{a_k}, s_{a_m}) = d(e_{a_m}, s_{a_k})$ and it is close to 0 when $d(s_{a_k}, s_{a_m}) >> d(e_{a_m}, s_{a_k})$ or $d(e_{a_m}, s_{a_k}) >> d(s_{a_k}, s_{a_m})$. The aim of this balancing factor is to select first *conflicts* with small and similar values of flexibility.

The second estimator is used, given an $<s_{a_i}, a_j>$, to select one of the two leveling constraints $s_{a_i}\{before\}s_{a_j}$ or $e_{a_j}\{before\}s_{a_i}$:

- *Leveling Constraint Selection (LCS)* - The choice is made according to the function $pc$, such that, $pc(s_{a_i}, a_j) = s_{a_i}\{before\}s_{a_j}$ when $d(s_{a_i}, s_{a_j}) > d(e_{a_j}, s_{a_i})$ and $pc(s_{a_i}, a_j) = e_{a_j}\{before\}s_{a_i}$ otherwise.

These two estimators provide a basis for defining two new algorithms for MCM-SP:

- The first variant maintains the CFSA strategy of "peak focusing" and simply utilizes a deterministic choice procedure based on temporal flexibility instead of the previous random choice procedure (we call this version CFSA+PK+TF). The two heuristic functions of CFSA are modified as follows:

  **Select-Lb-Peak-Lev-Constr** - Given the set of all *lb-peaks*: first, apply the $CS$ heuristic to select a *conflict*; next, apply the $LCS$ heuristic to select a leveling constraint.

  **Select-Ub-Peak-Lev-Constr** - Given the set of the *ub-peaks*, the subset of peaks where the value of the lower bound profile is maximal and the value of the upper-bound profile is minimal is calculated. Within this subset of peaks: first, apply the $CS$ heuristic to select a *conflict*; next, apply $LCS$ to select a leveling constraint.

- The second variant is designed to more widely take advantage of the temporal flexibility estimators (we call this algorithm CFSA+TF). In more detail:

  **Select-Lb-Peak-Lev-Constr** - same as above.

  **Select-Ub-Peak-Lev-Constr** - Given the set of all *ub-peaks*: first, apply the $CS$ to select a *conflict*; next, apply the $LCS$ to select a leveling constraint.

**Experimental Evaluation.** The two procedures, CFSA+PK+TF and CFSA+TF, were both run on the same sets of scheduling problems used to evaluate the original CFSA+PK algorithm. Results are shown in Table 2 where the column "Alg" distinguishes between "pktf" for CFSA+PK+TF and "tf" for CFSA+TF.

The results provide an interesting insight. They indicate quite clearly that the use of demand upper and lower bounds as a focusing mechanism is not an effective strategy for this type of problems. Problem solving performance does not improve when the basic procedure is augmented to additionally utilize heuristic estimates of temporal flexibility. Yet when estimations of temporal flexibility are used more broadly as a basis for prioritizing and selecting leveling decisions (variant CFSA+TF), problem solving effectiveness at larger problem sizes increases dramatically. Thus, the power of look-ahead heuristics which emphasize retention of temporal flexibility in the evolving solution is seen to extend directly to MCM-SP. On the contrary, decision prioritization based on the lower and upper bounds on demand profiles appears to be of limited effectiveness.

One additional observation about the performance of CFSA+TF is warranted. It is clear that the time needed to produce a solution is very high at the largest problem size. This fact seems due to the huge number of leveling constraints that the algorithm inserts to produce a solution. This observation motivates the alternative approach described in the next subsection.

Table 2: Experiments with the CFSA variants

| Probl | Alg | $N_s$ | $RB$ | $M_{ks}$ | $N_{lc}$ | $CPU$ |
|---|---|---|---|---|---|---|
| 5×5 | pktf | 50 | 7.0 | 236.1 | 39.5 | 0.6 |
| | tf | 50 | 7.5 | 228.4 | 54.4 | 1.0 |
| 10×5 | pktf | 50 | 2.8 | 422.8 | 144.4 | 7.7 |
| | tf | 50 | 3.6 | 402.8 | 354.9 | 25.8 |
| 15×5 | pktf | 30 | 1.5 | 596.6 | 278.0 | 28.5 |
| | tf | 50 | 2.0 | 580.9 | 884.4 | 137.0 |
| 20×5 | pktf | 6 | 1.1 | 776.2 | 409.0 | 69.2 |
| | tf | 49 | 1.3 | 750.4 | 1630.1 | 430.4 |
| 25×5 | pktf | 2 | 0.6 | 1012.0 | 556.5 | 150.3 |
| | tf | 50 | 0.9 | 911.5 | 2601.6 | 1055.8 |

## Earliest Start Time Algorithm (ESTA)

The rationale behind the CFSA strategy is the idea of maintaining a conflict free solution, forcing the condition $UB_{D_j}(s_{a_i}) \leq c_j$ to be valid for each resource $r_j$ and start time $s_{a_i}$.

*CFSA Leveling Activity.* Observing the solutions produced by all the CFSA algorithms, we have observed that such strategies post large numbers of leveling constraints in the process of determining a feasible solution. As it turns out, the solutions produced are in fact overconstrained solutions. We can demonstrate this fact with a simple example. Consider a simple scheduling problem consisting of a single resource with capacity $c = 2$ and three independent activities $a_1$, $a_2$ and $a_3$. All the activities have a processing time $p_{time} = 1$, they demand one unit of $r$ and must be scheduled in the time window $[0, 10]$. It is easy to verify that to build a conflict free solution it is sufficient to post only one precedence constraint between two of the three activities considered. For example, we can post the constraint $a_1\{before\}a_2$ and we are ensured to have no resource conflicts. However, if after posting this constraint we were to compute the upper-bound profile in $s_{a_3}$, we obtain $UB_D(s_{a_3}) = 3$. In fact both $a_1$ and $a_2$ can overlap $s_{a_3}$ even if separately because of the constraint $a_1\{before\}a_2$. The upper-bound demand is built on the basis of pairwise tests, so following the CFSA strategy a new leveling constraint is posted to get $UB_D(s_{a_3}) = 2$. The problem is due to the fact that a pairwise accounting of possible overlapping gives only partial information [6]. Trying to overcome this kind of limitation of the CFSA approach, we propose another profile-based approach which, instead of focusing on the development of a *conflict free solution* tries instead to find a so-called *earliest start time solution.*

[6] An approach based on cliques detection (Laborie & Ghallab 1995) can avoid this problem because it is global in taking into account overlapping activities. Unfortunately it is more time consuming than profile-based approaches.

*Earliest Start Time Solution.* As is well known from (Dechter, Meiri, & Pearl 1991), the STP temporal model associates with any time-point $tp_i$ (the variables of a temporal CSP) an interval of possible time values $[lb_{tp_i}, ub_{tp_i}]$. It is known that the extremes of the interval, either all the lower bounds $lb_{tp_i}$ or all the upper bounds $ub_{tp_i}$, if chosen as the value for all time variables $tp_i$, identify a consistent solution for the STP. If, in correspondence of these two sets of temporal values, we also have a *resource consistent* solution (i.e., all the resource requirements respect the capacity constraints), then we have obtained a solution for the correspondent MCM-SP. We call these two particular solutions the *Earliest Start Time Solution* and the *Latest Start Time Solution*. Since we have previously identified minimum completion time as an objective criteria it is straightforward to observe that we are more interested in producing *Earliest Start Time Solutions*. It is interesting to notice that an earliest start time solution solves the previous simple example by posting only one leveling constraint.

It should also be noted that if, starting from an earliest start time solution, we consider an arbitrary time value that is consistent for a given time point we can generate a conflicting solution. In other words, while the CFSA strategy guarantees a conflict free solution for any choice of time values for individual time-points, an earliest start time solution guarantees this property only if the values $lb_{tp_i}$ are chosen. Later on we introduce an algorithm to obtain a conflict free solution also in this second case.

*Resource Demand at the Earliest Start Time.* The development of a general algorithm to create earliest start time solutions is based on the following observation: having chosen a single value for the time-points (the $lb_{tp_i}$), we obtain a single value for the resource demand (instead of an approximate bound) and can perform stronger deductions using this information.

A demand profile for a resource $r_j$ is a temporal function $EST_{D_j}(s_{a_i})$ that takes the start time $s_{a_i}$ of activity $a_i$ computes the required resource in the instant $lb(s_{a_i})$. Given a resource $r_j$ and the set of activities $A_j$ which request $r_j$, the *earliest start time demand profile* is defined as follows:

$$EST_{D_j}(s_{a_i}) = \sum_{a_k \in A_j} P_{ik} \times rc_{a_k,j}$$

where $rc_{a_k,j}$ is the required capacity of resource $r_j$ from the activity $a_k \in A_j$, $P_{ij} = 1$ when $lb(s_{a_j}) \leq lb(s_{a_i}) < lb(e_{a_j})$ and $P_{ij} = 0$ otherwise.

*Resource Conflict.* Given an instance of MCM-SP, a *peak* is a tuple $<r_j, s_{a_i}, ca_i>$, where $r_j$ is a resource, $s_{a_i}$ is the start time point of the activity $a_i$ and $ca_i$ is a set of activities $ca_i = \{a_k : P_{ik} = 1\}$, such that $EST_{D_j}(s_{a_i}) > c_j$.

A peak $<r_j, s_{a_i}, ca_i>$ can be leveled by posting precedence constraints between a pair of activities $a_i, a_j \in ca$. It is worth noting that in this algorithm a conflict $<a_i, a_j>$ is composed of two activities. We will refer generally to $<a_i, a_j>$ as a *pairwise conflict*.

On the basis of the shortest paths information contained in the graph $G_d$, as explained in (Cheng & Smith 1994; Oddi & Smith 1997), it is possible to define a set of conditions which identify unconditional decisions and promote early pruning of alternatives. For any pair of activities $a_i$ and $a_j$ that are competing for the same resource, four possible cases of conflict are defined:

1. $d(e_{a_i}, s_{a_j}) < 0 \ \wedge \ d(e_{a_j}, s_{a_i}) < 0$
2. $d(e_{a_i}, s_{a_j}) < 0 \ \wedge \ d(e_{a_j}, s_{a_i}) \geq 0 \ \wedge \ d(s_{a_i}, e_{a_j}) > 0$
3. $d(e_{a_j}, s_{a_i}) < 0 \ \wedge \ d(e_{a_i}, s_{a_j}) \geq 0 \ \wedge \ d(s_{a_j}, e_{a_i}) > 0$
4. $d(e_{a_i}, s_{a_j}) \geq 0 \ \wedge \ d(e_{a_j}, s_{a_i}) \geq 0$

Condition 1 represents a *pairwise unresolvable conflict*. There is no way to sort $a_i$ and $a_j$ without inducing a negative cycle in graph $G_d(V, E)$. Conditions 2, and 3, alternatively, distinguish *pairwise uniquely resolvable conflicts*. Here, there is only one feasible ordering of $a_i$ and $a_j$ and the decision of which constraint to post is thus unconditional. In the case of Condition 2, only $a_j\{before\}a_i$ leaves $G_d(V, E)$ consistent and similarly, only $a_i\{before\}a_j$ is feasible in the case of Condition 3. Condition 4 designates a final class of *pairwise resolvable conflicts*. In this case, both orderings of $a_i$ and $a_j$ remain feasible and it is necessary to make a choice. Now we can give a sufficient condition for detecting an *unresolvable peak*. A peak $<r_j, s_{a_i}, ca_i>$ is unresolvable if for each pairwise conflict $<a_i, a_j>$ with $a_i, a_j \in ca$ and $a_i \neq a_j$ holds $d(e_{a_i}, s_{a_j}) < 0 \ \wedge \ d(e_{a_j}, s_{a_i}) < 0$.

*The ESTA Algorithm.* We now propose a new algorithm for MCM-SP that: (a) finds a solution by leveling the *earliest start time demand* in accordance with the capacity of the resource; (b) extends the heuristic method defined in (Cheng & Smith 1994) to this case. The algorithm, named *Earliest Start Time Algorithm (ESTA)*, is shown in Figure 2. ESTA iteratively selects a *pairwise conflict* by a heuristic method until the demand on all the resources is less than or equal to their respective capacities or until an *unresolvable conflict* is detected. In this last case the procedure stops with failure.

**Earliest-Start-Time-Solver**(*mcmsp*)
1. **loop**
2. **if** Exists-an-Unresolvable-Conflict(*mcmsp*)
3. **then return**(Failure)
4. **else begin**
5. *ResolvableConflict*:=Sel-Res-Conf(*mcmsp*)
6. **if** (*ResolvableConflict* = *NIL*)
7. **then return**(*Solution*)
8. **else begin**
9. *Pc*:= Sel-Prec-Constr(*ResolvableConflict*)
10. Insert-Prec-Constr(*Pc*)
11. **end**
12. **end**
13. **end-loop**

Figure 2: Earliest Start Time Algorithm

*The Leveling Heuristics.* The heuristic methods for selecting the next conflict to resolve and for determining how to resolve it (i.e., which precedence constraints to post between activities $<a_i, a_j>$) are derived from

(Cheng & Smith 1994). Both the selection of pairwise conflicts and precedence constraints are based on the principle of leaving maximum amount of temporal flexibility possible in the solution at each step. We use the values of distances $d(e_{a_i}, s_{a_j})$ (or $d(e_{a_j}, s_{a_i})$) as a quantification (or measurement) of how many time positions the pair of activities $<a_i, a_j>$ may assume with respect to each other while respecting the time constraints.

The proposed heuristic estimators operate as follows:

- *Selection of Pairwise Conflict* - We have to distinguish two cases. When all pairwise conflicts satisfy Condition 4, the conflict $<a_i, a_j>$ with the minimum value $\omega_{res}(a_i, a_j)$ is selected, where:

$$\omega_{res}(a_i, a_j) = min\{\frac{d(e_{a_i}, s_{a_j})}{\sqrt{S}}, \frac{d(e_{a_j}, s_{a_i})}{\sqrt{S}}\}$$

  with $S = \frac{min\{d(e_{a_i}, s_{a_j}), d(e_{a_j}, s_{a_i})\}}{max\{d(e_{a_i}, s_{a_j}), d(e_{a_j}, s_{a_i})\}}$. In the case where a subset of pairwise conflicts satisfying Condition 2 or Condition 3 exists, the heuristic selects the conflict with the minimum (and negative) value $\omega_{res}(a_i, a_j) = min\{d(e_{a_i}, s_{a_j}), d(e_{a_j}, s_{a_i})\}$, that is the pair of activities which are closest to having their ordering decision be forced.
- *Selection of Precedence Constraints* - The choice procedure is defined by the function $pc$, such that, $pc(a_i, a_j) = a_i\{before\}a_j$ when $d(e_{a_i}, s_{a_j}) > d(e_{a_j}, s_{a_i})$ and $pc(a_i, a_j) = a_j\{before\}a_i$ otherwise.

In the case of conflicts $<a_i, a_j>$ with a negative value of $\omega_{res}(a_i, a_j)$, we can observe that since the capacity of a resource is in general greater than one, some activities in a solution can overlap and others have to be sequenced. From the point of view of the heuristic method it is better to resolve pairwise conflicts in the way that minimizes loss of temporal positions.

The time complexity associated with detecting a conflict in the use of a resource $r_j$ which satisfies the heuristic method previously defined, is quadratic. In fact, if $N_{r_j}$ is the number of activities which request the resource $r_j$, then we have to make $O(N_{r_j}^2)$ comparisons for every non sequenced pair of activities $<a_i, a_j>$.

**Chaining Algorithm.** Given an earliest start time solution, one way to generate a conflict free solution is to create a set of $c_j$ chains of activities on the resource $r_j$. That is, we can partition the set of activities which require $r_j$, into a set of $c_j$ linear sequences. This operation can be accomplished by deleting all the leveling constraints in the solution and using the established lower bound solution to post a new set of constraints according to the division into linear sequences. In this situation, which we refer to as *chain-form*, if $N_{r_j}$ is the number of activities which request $r_j$, then the number of precedence constraints posted is at most $N_{r_j} - c_j$. In contrast, we would generally expect that the process of determining an earliest start time solution inserts a greater number of leveling constraints. A solution in *chain-form* is a different way to represent a solution that presents two advantages: (a) the solution is conflict free for *on line* modifications of start or end time of activities; (b) there are always $O(N_a)$ leveling constraints (where $N_a$ is the total number of activities). So, every temporal algorithm whose complexity depends on the number of distance constraints can gain advantages from this new form of the solution.

**Experimental Evaluation.** The ESTA algorithm was also run on our set of scheduling problems and results are shown in Table 3. It should be noted that in this case the parameter $N_{lc}$ refers only to the solution produced by the ESTA without the chaining procedure. On the contrary, the other parameters refer to the production of the conflict free solution (ESTA + Chaining).

The results are particularly encouraging. ESTA solves all the problems presented but more interestingly it is very quick in solving them and the quality of the solution is much better than the CFSA+TF algorithm. The speed in producing a solution is confirmed by the low number of leveling constraints posted. It is also worth noting that with respect to makespan and robustness, ESTA outperforms all CFSA style solution strategy. The result concerning $M_{ks}$ can be justified by the search of the earliest start time solution and by the fact that the CFSA has the pathology of overconstraining the solution by inserting unnecessary leveling constraints (more precisely, these algorithms "over-chain" the solution). The advantage with respect to robustness $RB$ confirms the relevance of techniques which retain solution flexibility in solving MCM-SPs.

Table 3: Experiments with ESTA Algorithm

| Probl | $N_s$ | $RB$ | $M_{ks}$ | $N_{lc}$ | $CPU$ |
|---|---|---|---|---|---|
| 5×5 | 50 | 9.1 | 210.5 | 1.8 | 1.0 |
| 10×5 | 50 | 7.5 | 300.0 | 21.1 | 5.8 |
| 15×5 | 50 | 5.4 | 430.1 | 71.9 | 18.2 |
| 20×5 | 50 | 4.9 | 545.2 | 143.4 | 41.0 |
| 25×5 | 50 | 4.6 | 664.1 | 233.2 | 81.3 |

## Concluding Remarks

In this paper we have introduced and analyzed a progression of algorithms for solving the multi-capacitated metric scheduling problem (MCM-SP): a class that involves both metric time constraints and sharable resources with the capacity to simultaneously support multiple activities. This type of problem is quite commonplace in real applications, but it has received relatively little attention in the planning and scheduling literature.

To support a comparative analysis of solution methods, we began by defining a procedure for generating a controlled set of random benchmark problems. A set of criteria for judging the effectiveness of alternative solution procedures were also identified, including measures relating to both problem solving ability (number of problems solved, average computation time) and quality of the solutions generated. With respect to the latter point, we considered Makespan minimization (a common objective) as one target criterion. We also introduced a definition of robustness. Although initial,

we believe it provides a useful characterization of the "fluidity" of a solution in these "strongly temporal" domains.

Having established a framework and design for experimental evaluation, a number of profile-based scheduling algorithms were then defined and evaluated. Each algorithm that was considered conformed to a general solution schema that entailed (1) monitoring resource demand over time for periods of over-utilization and (2) introducing additional sequencing constraints between pairs of competing activities to level out periods of high demand. The different algorithms considered varied in the heuristic criteria used to prioritize and make sequencing decisions, as well as in assumptions made about the number of sequencing constraints required for a feasible solution.

A first result of the experimentation showed that a greedy version of a previously reported algorithm (El-Kholy & Richards 1996) (called CFSA in this paper) performed rather poorly on larger sized problems. The reason for the limited effectiveness of the algorithm stemmed from the weak look-ahead guidance provided by the computation of upper and lower bound on resource demand profiles.

A second result obtained concerns the extension of ideas of temporal flexibility introduced in (Cheng & Smith 1994) to the case of multiple-capacitated scheduling problems. By substituting the use of temporal flexibility as a basis for prioritizing and making decisions with the basic CFSA procedure, substantial performance improvements were obtained at the larger problem sizes tested. Once again these techniques have been shown to be very effective and relevant for scheduling problems with quantitative separation constraints.

A final result, triggered by the observation that CFSA style procedures tend to insert many more leveling constraints than are necessary to ensure feasibility, concerned the development of an alternative MCM-SP solution approach. In particular, a very quick and effective solution procedure was obtained by focusing on construction of solution for a precise time instant on the time line as opposed to more general construction of a conflict free solution. The ESTA procedure, which embodies this approach and computes an "earliest start time solution", was shown to outperform all variants of CFSA on all dimensions of performance considered. Furthermore, a post-processing algorithm was also summarized for transforming this early-time solution into a conflict-free solution; thus achieving the same level of solution generality as CFSA but without the unnecessary leveling constraints inserted by CFSA style approaches. This fact reproduces, in a different context, the well known effect in scheduling that when reasoning with exact start times it is possible to make stronger deductions than in the case in which fluctuating time-points are reasoned about.

There are a number of directions in which the work reported in this paper might be extended. One path of interest is to attempt to productively expand the level of search performed by the greedy, one-pass solution procedures developed here. Recent work in iterative sampling and re-starting search techniques (e.g., (Oddi & Smith 1997)) would seem to provide a natural basis. Another area of future research concerns comparison of the profile-based solution procedures developed here with other (non profile-based) approaches. We are particularly interested in exploring is the clique-based approach proposed in (Laborie & Ghallab 1995). Due to its global nature, this approach offers an alternative basis for eliminating the posting of extra (over-constraining) sequencing constraints. But the implications remain unclear with respect to how much increase in computation time is required.

## Acknowledgments.

Amedeo Cesta and Angelo Oddi's work is supported by Italian Space Agency, and by CNR Committee 12 on Information Technology (Project SCI*SIA). Angelo Oddi is currently supported by a scholarship from CNR Committee 12 on Information Technology. Stephen F. Smith's work has been sponsored in part by the National Aeronautics and Space Administration under contract NCC 2-976, by the US Department of Defense Advanced Research Projects Agency under contract F30602-97-20227, and by the CMU Robotics Institute.

## References

Ausiello, G.; Italiano, G. F.; Marchetti Spaccamela, A.; and Nanni, U. 1991. Incremental Algorithms for Minimal Length Paths. *Journal of Algorithms* 12:615–638.

Cesta, A., and Stella, C. 1997. A Time and Resource Problem for Planning Architectures. In *Proceedings of the Fourth European Conference on Planning (ECP 97).*

Cheng, C., and Smith, S. F. 1994. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings 12th National Conference on AI (AAAI-94).*

Cheng, C., and Smith, S. F. 1996. A Constraint Satisfaction Approach to Makespan Scheduling. In *Proceedings of the 4th International Conference on AI Planning Systems (AIPS-96).*

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence* 49:61–95.

El-Kholy, A., and Richards, B. 1996. Temporal and Resource Reasoning in Planning: the *parc*PLAN Approach. In *Proc. of the 12th European Conference on Artificial Intelligence (ECAI-96).*

Erschler, J.; Lopez, P.; and Thuriot, C. 1990. Temporal reasoning under resource constraints: Application to task scheduling. In Lasker, G., and Hughes, R., eds., *Advances in Support System Research.* International Institute for Advanced Studies in Systems Research and Cybernetics.

Laborie, P., and Ghallab, M. 1995. Planning with Sharable Resource Constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95).*

Nuijten, W. P. M., and Aarts, E. H. L. 1994. Constraint Satisfaction for Multiple Capacitated Job-Shop Scheduling. In *Proc. of the 11th European Conference on Artificial Intelligence(ECAI-94).*

Oddi, A., and Cesta, A. 1997. A Tabu Search Strategy to Solve Scheduling Problems with Deadlines and Complex Metric Constraints. In *Proc. 4th European Conf. on Planning (ECP 97).*

Oddi, A., and Smith, S. F. 1997. Stochastic Procedures for Generating Feasible Schedules. In *Proceedings 14th National Conference on AI (AAAI-97).*

Oddi, A. 1997. *Sequencing Methods and Temporal Algorithms with Application in the Management of Medical Resources.* Ph.D. Dissertation, Department of Computer and System Science, University of Rome "La Sapienza".

Sadeh, N. 1991. *Look-ahead Techniques for Micro-opportunistic Job-shop Scheduling.* Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Smith, D. R., and Westfold, S. J. 1996. Scheduling an Asynchronously Shared Resource. Working Paper, Kestrel Institute.

Taillard, E. 1993. Benchmarks for Basic Scheduling Problems. *European Journal of Operational Research* 64:278–285.