Scheduling Multi-Capacitated Resources under Complex Temporal Constraints

CMU-RI-TR-98-17

Amedeo Cesta[1], Angelo Oddi[1], Stephen F. Smith[2]

[1] IP-CNR, National Research Council of Italy
Viale Marx 15, I-00137 Rome, Italy, {*amedeo, oddi*}*@pscs2.irmkant.rm.cnr.it*
Phone: +39-6-86090-209

[2] The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA, *sfs@isl1.ri.cmu.edu*
Phone: +1-412-268-8811

June 30th, 1998

Abstract

Most CSP scheduling models make the restrictive assumption that a resource can only support a single activity at a time (i.e., it is either available or in-use). However, in many practical domains, resources in fact have the capability to simultaneously support multiple activities, and hence availability at any point is a function of unallocated *capacity*. In this paper, we develop and evaluate algorithms for solving multi-capacitated scheduling problems. We first define a basic CSP model for this extended problem class, which provides a basic framework for formulating alternative solution procedures. Using this model, we then develop variants of two different solution approaches that have been recently proposed in the literature: (1) a profile-based procedure - which relies on local analysis of potential resource conflicts to heuristically direct the problem solving process, and (2) a clique-based procedure - which exploits a global analysis of resource conflicts at greater computational cost. In each case, improvements are made to previously proposed techniques. Performance results are given on a series of problems of increasing scale and constrainedness, indicating the relative strengths of each procedure.

Contents

1 Introduction

Constraint satisfaction problem solving (CSP) techniques have been productively applied to several classes of scheduling problems in recent years (e.g., [3, 9, 11, 10]). Some of these techniques assume very general temporal constraint models and support resource allocation under complex qualitative and quantitative time constraints. However, this same level of modeling generality has not been achieved with resource representations. With very few exceptions, CSP scheduling research has restricted attention to 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, this model of resources is either insufficient or impractical. Resources (at the appropriate level of modeling detail) in fact have the capability to simultaneously support multiple activities, and availability must be determined as a function of unallocated capacity.

Early work on formalizing and solving the scheduling problem as a CSP (Constraint Satisfaction Problem) (e.g., [11, 12]) focused on the classical Job-Shop Scheduling Problem (JSSP), where: (a) temporal constraints represent constant durations of activities, (b) temporal separation constraints between activities are simple qualitative ordering constraints, and (c) capacity constraints are binary (a resource is either busy or free). More recent work has considered various extended JSSP models. A Multi-Capacitated JSSP (MCJSSP) is defined and solved in [9]. This problem class retains the temporal constraint assumptions of the classical JSSP but allows resource capacities to be greater than one. In [3] and subsequently in [10] a complementary extension of the JSSP is addressed. In this case a more general, Simple Temporal Problem (STP) [4] representation of temporal constraints is incorporated, but the unit-capacity resource assumption of the JSSP is retained.

In this paper, we focus on development and evaluation of procedures that simultaneously accommodate both extended resource capacity and extended temporal constraint models; i.e., procedures that schedule multi-capacitated resources in the presence of STP-style temporal constraints on the activities that require resource capacity. We call this class of problems the *Multiple Capacitated Metric Scheduling Problem* (MCM-SP). Summarizing the principal features of the MCM-SP: (a) resources are discrete (and not consumable) with capacity $c_j \geq 1$, (b) there are quantitative time constraints (i.e., lower, upper bounds) on the starts, ends and durations of activities that require resource capac-

ity, and (c) sets of activities corresponding to independent "jobs" are temporally related (e.g., precedence) with the possibility of metric constraints (lower, upper bounds) on temporal separation. 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 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.

Representations and solution procedures that are capable of simultaneously accounting for STP-style temporal constraints and multi-capacitated resource constraints have recently been proposed within the planning literature[7, 5]. These works follow two different solution approaches, distinguishable by the basic mechanism used to identify resource capacity conflicts:

- *profile-based approaches* (e.g., [5]): these approaches are extensions of a technique quite common in unit-capacity scheduling (e.g., [11, 8]). Most generally, they consist of characterizing resource demand as a function of time, identifying periods of over-allocation in this demand profile, and incrementally performing "leveling actions" to (hopefully) ensure that resource usage peaks fall below the total capacity of the resource.

- *clique-based approaches* [7]: given a current schedule, this approach builds up a "conflicts graph" whose nodes are activities and whose edges represent overlapping resource capacity requests of 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.

Though each of these approaches offers the generality to formulate and solve the general MCM-SP, the work reported in both [5] and [7] has been concerned principally with providing resource reasoning subcomponents within larger planning architectures and attention has been limited to quite restrictive special cases of the MCM-SP (e.g., small, single resource problems). In [2], the application of profile-based procedures to the general MCM-SP is directly considered. Several derivatives of the approach originally proposed

in [5] are defined and evaluated in this context, and it is shown that by extending and incorporating CSP heuristics previously developed for unit-capacity scheduling problems, a much higher-performance, profile-based procedure for solving the MCM-SP is obtained.

In the sections below, we build on these results with profile-based solution procedures and consider a more general analysis of profile-based and clique-based approaches to solving the MCM-SP. Profile-based methods rely on local, pairwise analysis of resource capacity conflicts and promote computational efficiency; clique-based approaches perform more global analysis and offer greater accuracy in conflict detection but at potentially much higher computational cost. We begin by defining the MCM-SP to be addressed and specifying a common CSP scheduling framework for formulating alternative solution procedures. We then specify profile-based and clique-based procedures for solving the MCM-SP. Our approach in both cases is *least-commitment*; i.e., our goal is to produce solutions which temporally constrain the execution of activities as opposed to fixing their execution times. We present a set a experiments which evaluate cost/performance tradeoffs on problem instances of increasing scale and difficulty, and indicate the respective performance characteristics of each approach.

2 Definition of MCM-SP

The Multiple-Capacitated Metric Scheduling Problem (MCM-SP) we will consider 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 a set of 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 *feasible solution* to a MCM-SP is any temporally consistent assignment of start and end times which does not violate resource capacity constraints.

The required capacity (also called resource demand) of resource r_j by activity a_k is denoted $rc_{a_k,j}$. The set of activities a_k that demand resource r_j is called A_j ($A_j = \{a_k |\ rc_{a_k,j} \neq 0\}$). In this paper we consider the case $rc_{a_k,j} = 1$.

3 CSP Algorithms for MCM-SP

In the case of simple, unit-capacity resources, the scheduling problem can be formulated as a CSP with decision variables $O_{h,l,r}$ corresponding to each pair of activities a_h and a_l that requires a given resource r, and each capable of be "assigned" a value of either $a_h\{before\}a_l$ or $a_l\{before\}a_h$. Here, each decision variable represents a potential resource conflict, and an assignment (or posting) of either other precedence constraint, if it can be done consistently, eliminates this possibility. Our approach to the MCM-SP simply generalizes this formulation to the case where potential resource conflicts in the use of resource r_j involve (minimally) $c_j + 1$ activities. Decision variables continue to designate potential conflicts, and possible values correspond to the (now larger) set of ordering constraints that might be posted to eliminate contention.

To provide an infra-structure for constructing search procedures that exploit this formulation, we assume an underlying temporal representation as a Simple Temporal Problem (STP) [4]. More precisely, a directed graph $G_d(V, E)$, named *distance graph*, is defined for, wherein the set of vertexes 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 with two weighted edges [1]: the first

[1] We are aware that other researchers in this topic are in favor of reversing the distance graph of [4] so that an arc of length 0 between x and y corresponds to "x before y" and an arc of length a between

one directed from tp_i to tp_j with weight b, the second one from tp_j to tp_i with weight $-a$. The STP consistency can be determined efficiently by shortest path computations.

Each time point tp_i has associated an interval $[lb_i, ub_i]$ of the possible time instants, or temporal values, where the event associated to the time-point may happen. A particular time-point tp_1, called the origin point, has associated the constant interval $[0, 0]$. We denote with $d(tp_i, tp_j)$ the shortest path length from node tp_i to node tp_j in G_d. As shown in [4], the interval $[lb_i, ub_i]$ of time values associated to the generic time variable tp_i is computed on the graph G_d as the interval $[-d(i, 1), d(1, i)]$.

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). What is needed to configure a complete search procedure are mechanisms and heuristics for recognizing, prioritizing and resolving resource conflicts.

Figure 1 gives an algorithmic template that operationalizes this approach and will be used below to specify alternative MCM-SP solution procedures. The template identifies three basic steps that require instantiation: `Exists-Unresolvable-Conflict` which detects an infeasible state, `Select-Conflict-Set` which identifies the set of activities included in the resource conflict to be resolved next, and `Select-Leveling-Constraint` which chooses a temporal ordering constraint to solve the conflict by reducing (leveling) resource requirements in conflict.

3.1 A Profile-Based Algorithm

We first consider the specification of a *profile-based* approach to the MCM-SP. For each resource r_j the *Demand Profile* $D_j(t)$ represents 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.

Profile based solution methods proceed generally by detecting time instants where the profile $D_j(t)$ of a given r_j is greater than the available capacity c_j and then sequencing selected activities that are contributing to these detected usage peaks.

In a previous paper [2], several profile-based solution procedures were developed and evaluated, each descendant from the approach to multiple-capacity planning originally proposed in [5]. The procedure developed in [5], owing to its orientation toward solution

x and y means $x + a \leq y$. We use here the convention of [4] for consistency with previous work of ours.

```
MCM-SP-Solver(Mcmsp)
1.  CSolution ← Mcmsp
2.  if Exists-Unresolvable-Conflict(CSolution)
3.       then return(NIL)
4.       else begin
5.         Cset ← Select-Conflict-Set(CSolution)
6.         if (Cset = NIL)
7.            then return(CSolution)
8.            else begin
9.               Lc ← Select-Leveling-Constraint(Cset)
10.              Post-Leveling-Constraint(Lc,CSolution)
11.              MCM-SP-Solver(CSolution)
12.           end
13.      end
```

Figure 1: MCM-SP Solver

of fairly simple scheduling problems, paid little attention to the heuristics employed for choosing resource conflicts and ordering constraints. The derivative procedures developed in [2], in contrast, were predicated on the hypothesis that incorporation of stronger heuristics would result in higher-performance solution procedures, and they were designed to exploit extended versions of heuristics originally developed for unit-capacity resource scheduling. Experimental results confirmed the superior performance of these derivative procedures in solving the MCM-SP, but, at the same time, indicated that all tested procedures tended to produce solutions that contained a large number of unnecessary ordering constraints. This deficiency motivated the design of a different style of profile-based procedure, referred to as ESTA (*Earliest Start Time Algorithm*), and from a performance standpoint, this procedure was found to dominate all others. Accordingly, we choose ESTA as a representative example of the profile-based approach for our purposes in this paper, and subject it to a broader experimental analysis. In the rest of this section we first present some preliminary motivations that justify ESTA and then specialize the template in Figure 1 to the ESTA case.

Preliminaries. We pursue development of a two step procedure where (a) a first step finds a solution which is conflict-free only in a particular set of time instants, and (b) a second, post-processing step transforms the solution into one that is *conflict free* for all sets of times implied by the final set of temporal constraints.

As just mentioned, an interval of possible time values $[lb_i, ub_i]$ is associated with any time-point tp_i in a STP representation. It is known [4] that the extremes of such intervals, either the lower bounds lb_i or the upper bounds ub_i of all time variables tp_i identify a consistent solution to the STP. If in correspondence with either of these two sets of temporal values also the resource capacity constraints are satisfied, then we have constructed a solution for the correspondent MCM-SP. We call these two particular solutions the *earliest start time solution* and the *latest start time solution*; and both are logical candidates to be the target solution for step (a).

We focus attention on the development of the earliest start time solution because of its potential for better makespan. The demand profile for a resource r_j is thus a temporal function $EST_{D_j}(s_{a_i})$ that takes the start time s_{a_i} of activity a_i computes the resource utilization in the instant $lb_{s_{a_i}}$. Without loss of generality, such resource demand is 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. 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 $P_{ik} = 1$ when $lb_{s_{a_k}} \leq lb_{s_{a_i}} < lb_{e_{a_k}}$ and $P_{ik} = 0$ otherwise.

Given a MCM-SP, a resource conflict, or *peak*, is a tuple $< r_j, s_{a_i}, ca_i >$, where ca_i is a set of activities $ca_i = \{a_k \mid P_{ik} = 1\}$, 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 any pair of activities $a_h, a_l \in ca_i$. It is worth noting that in this algorithm a conflict $< a_h, a_l >$ is composed of two activities. We refer generally to $< a_h, a_l >$ as a *pairwise conflict set*.

Using shortest path length information contained in the graph G_d, it is possible to define a set of *dominance conditions* which identify unconditional decisions and promote early pruning of alternatives [3, 10]. For any pair of activities a_h and a_l competing for the same resource, four possible cases of conflict are defined:

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

Condition 1 represents a *pairwise unresolvable conflict*. There is no way to sort a_h and a_l 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_h and a_l and the decision of which constraint to post is thus unconditional. In the case of Condition 2, only $a_l\{before\}a_h$ leaves $G_d(V, E)$ consistent and similarly, only $a_h\{before\}a_l$ 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_h and a_l remain feasible and it is necessary to make a choice.

The ESTA Algorithm. We are now in a position to specify an algorithm which finds a solution by leveling the *earliest start time demand*, which we will refer to as ESTA (*Earliest Start Time Algorithm*). We make this precise by describing the three basic steps of the general schema in Figure 1.

The predicate `Exists-Unresolvable-Conflict` is implemented straightforwardly, by identifying peaks $< r_j, s_{a_i}, ca_i >$ where, for each pairwise conflict set $< a_h, a_l >$ with $a_h, a_l \in ca$ and $a_h \neq a_l$, the dominance condition 1 holds.

The heuristic methods for selecting a pairwise conflict set to resolve and for deciding which leveling constraint to post are derived from [3]. The general least-commitment principle behind both methods is to retain the maximum amount of temporal flexibility possible in the solution at each step; and shortest path values $d(e_{a_h}, s_{a_l})$ (or $d(e_{a_l}, s_{a_h})$) are used to quantify how many time values a given pair of activities $< a_h, a_l >$ may assume with respect to each other while respecting the time constraints.

The function `Select-Conflict-Set` is concerned with selection of a pairwise conflict $< a_h, a_l >$ inside a resolvable peak. Two cases are distinguished. When all pairwise conflicts in the current solution satisfy Condition 4, then the conflict set $< a_h, a_l >$ with the minimum value $\omega_{res}(a_h, a_l)$ is selected [2], where:

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

with $S = \frac{min\{d(e_{a_h}, s_{a_l}), d(e_{a_l}, s_{a_h})\}}{max\{d(e_{a_h}, s_{a_l}), d(e_{a_l}, s_{a_h})\}}$. Instead when a subset of pairwise conflicts set satisfy Conditions 2 or 3, the heuristic selects the conflict with the minimum (and negative)

[2] As suggested in [3] a balancing factor $\sqrt{S}$ is used. It is possible to see that $S \in [0, 1]$: $S = 1$ when $d(s_{a_h}, s_{a_l}) = d(e_{a_l}, s_{a_h})$ and it is close to 0 when $d(s_{a_h}, s_{a_l}) >> d(e_{a_l}, s_{a_h})$ or $d(e_{a_l}, s_{a_h}) >> d(s_{a_h}, s_{a_l})$. The aim of this balancing factor is to select first *conflicts* in which both choices are strongly constrained and close to the failure state (dominance condition 1).

value $\omega_{res}(a_h, a_l) = min\{d(e_{a_h}, s_{a_l}), d(e_{a_l}, s_{a_h})\}$, i.e., the pair of activities that are closest to having their ordering decision forced.

`Select-Leveling-Constraint` within ESTA simply returns the constraint which leaves the most temporal flexibility. If $d(e_{a_h}, s_{a_l}) > d(e_{a_l}, s_{a_h})$ the leveling constraint chosen is $a_h\{before\}a_l$, otherwise $a_l\{before\}a_h$.

Postprocessing Step. 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 of activities. This operation can be accomplished by deleting all of the leveling solution's constraints and posting a new set of leveling constraints according to the division in 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 the step 1 process of determining an earliest start time solution to insert 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.

3.2 A Clique-Based Algorithm

We now turn attention to specification of a second type of solution procedure, where conflict graphs are substituted for demand profiles and clique detection drives conflict analysis. Let $G(V, E)$ be a finite undirected graph with no parallel edges and no self-loops. A clique C is a completely connected subset of V ($C \subseteq V$). The size of C is the number of vertexes in C. The clique in G with maximum size is called the maximum clique. Given a vertex v_i, we denote with J_i the set of vertexes v_j connected with v_i ($J_i = \{v_j \in V \mid (v_i, v_j) \in E\}$). We denote with d_i the cardinality of J_i (or degree of v_i, $d_i = \mid J_i \mid$).

The use of cliques for managing multiple-capacity constraints was first proposed in [7]. The basic idea follows from the observation that, given a current solution, it is possible to establish if any two activities may reciprocally overlap from the temporal information in the corresponding distance graph G_d. We can represent this "overlapping information" for

each resource r_j in a graph, named *Possible Intersection Graph* (PIG_j), whose vertexes are the activities requiring the resource r_j and such that an edge represents the fact that the execution intervals of its two vertexes (activities) may overlap/intersect in the current solution. According to our previous formalization, if the temporal bounds of any two activities (both $\in A_j$) verify one of the 4 dominance conditions introduced in the ESTA preliminaries, then they are connected by an undirected edge in the graph PIG_j. Note that if two activities in the current solution are connected by an explicit or implicit ordering constraint, then they are not connected in the PIG_j. Given r_j with capacity c_j , a clique of size at least c_j +1 in the graph PIG_j (called a *critical clique*) represents a *possible resource conflict* in the current solution. It can be immediately observed that this is an alternative representation of resource conflicts with respect to peaks in the demand profile $D_j(t)$. Note also that posting an ordering (leveling) constraint between a pair of activities whose vertexes are in a critical clique may cause elimination of the possible resource conflict (i.e., if the size of such a critical clique is exactly c_j +1, one leveling constraint eliminates the potential conflict). A current solution is conflict free with respect to resource constraints when no PIG_j contains any critical cliques.

To support the implementation described below, we introduce, for each resource r_j a further graph named the *Definite Intersection Graph* (DIG_j), whose vertexes are the activities in A_j and such that an edge exists between a pair of vertexes if the execution intervals of the corresponding activities verify dominance condition 1 (pairwise unresolvable conflict) in G_d. A clique in DIG_j of size at least c_j +1 represents an *unresolvable resource conflict* in the current solution. In this case, it is not possible to post an ordering constraint between elements of this clique to decrease its size. It should be clear that $DIG_j \subseteq PIG_j$, and that both evolve during solution development. So we assume the existence of a dynamic update function that at each step of the solver maintains these graphs for any resource r_j.

Searching Conflicts. A problem solving algorithm based on clique-detection requires a basic algorithm finding the *critical clique sets* in a conflict graph. For reasons elaborated in [6] we are interested in identifying so-called *minimal critical sets* (MCSs), which represent the minimally sized sets of activities that may potentially conflict. In a MCM-SP, where the resource demand of the activities is one ($rc_{a_k,j}$= 1), we are interested in detecting cliques of size $c_j + 1$ in PIG_j and DIG_j.

Figure 2 shows an algorithm, named `Clique-Tree`, which systematically traverses the

Clique-Tree(C,I_Δ)
1. $mcsSet \leftarrow \emptyset$
2. **loop**
3. **if** ($I_\Delta = \emptyset$) **then return** $mcsSet$
4. $v_i \leftarrow$ SelectVertex(I_Δ)
5. $I_\Delta \leftarrow I_\Delta - \{v_i\}$
6. $I_{\Delta new} \leftarrow I_\Delta \cap J_i$
7. **if** ($|C| + |I_{\Delta new}| + 1 \leq c_j$)
8. **then return** $\emptyset$
9. **else if** ($|C \cup \{v_i\}| > c_j$)
10. **then** $mcsSet \leftarrow mcsSet \cup \{C \cup \{v_i\}\}$
11. **else** $mcsSet \leftarrow mcsSet \cup$Clique-Tree($C \cup \{v_i\}$,$I_{\Delta new}$)
12. **end-loop**

Figure 2: To find a set of critical cliques (MCSs)

edges of a undirected graph, incrementally detecting each MCS and collecting it in $mcsSet$. Our algorithm relies on the current assumption $rc_{a_k,j} = 1$ and takes into account the basic idea used in [7].

`Clique-Tree` takes as input two parameters: a current clique C and a set of vertexes I_Δ used to enlarge the current clique C as search progresses. Given an undirected graph $G(V, E)$ the algorithm starts with $C = \emptyset$ and $I_\Delta = V$ (this situation corresponds to the search level $i = 0$). At any level i of the search tree, the set C is a clique with i vertexes $C = \{v_1, v_2, \ldots, v_i\}$ and the set I_Δ is obtained by the incremental intersection $I_\Delta = V \cap J_1 \cap J_2 \cap \cdots \cap J_i$, where the sets $J_1 \ldots J_i$ have been defined above.

At each step of the search process one of the three following cases may hold: (1) The current clique C has size less or equal to the resources capacity c_j and it is not possible to enlarge it over the treshold c_j (i.e., $|C| + |I_{\Delta new}| + 1 \leq c_j$), in this case the search process stops with failure on that path; (2) the set $C \cup \{v_i\}$ is a clique with size greater than the resources capacity c_j ($|C \cup \{v_i\}| > c_j$), in this case the clique with size $c_j + 1$ is collected in $mcsSet$ (note that the vertex v_i is selected from the set $I_\Delta = V \cap J_1 \cap J_2 \cap \cdots \cap J_i$, so it is surely connected to all the vertexes in C, i.e., $C \cup \{v_i\}$ is a clique with size $|C| + 1$; (3) in all the other cases the procedure is recursively invoked on the parameters $C \cup \{v_i\}$ and $I_{\Delta new}$ to check for larger cliques.

The CCSA Algorithm. We can now define the clique-based algorithm for the MCM-SP called CCSA (*Critical Clique Selection Algorithm*). As before, we do so, by specifying the

three basic steps of the generic MCM-SP-Solver.

The predicate `Exists-Unresolvable-Conflict` is realized by invoking for each resource r_j the function Clique-Tree($\emptyset$,V_{DIG_j}) where V_{DIG_j} are the vertexes of the Definite Intersection Graph DIG_j. If all calls return nil, all existing conflicts are solvable.

The function `Select-Conflict-Set` is instantiated for CCSA as shown in Figure 3. It is designed to take into account both the concept of retaining temporal flexibility and the need for an efficient implementation of the clique procedure. To select a MCS we use the heuristic estimator K suggested in [7]. The function K calculates a numerical value such that, the closer a MCS is to a clique in the DIG_j graph (which represents an unresolvable conflict), the higher is the evaluation function K(MCS). This idea implements a *least commitment* strategy and looks to the set of activities that is either closer to an unresolvable state or has the smallest number of precedence constraints that can still be posted to fix the conflict. The evaluator K(MCS) uses a function *commit* which estimates the loss of temporal flexibility in the current solution as a result of posting a given precedence constraint $a_i\{before\}a_j$ between two activities.

Given a candidate MCS and a set $\{pc_i\}$ of *precedence constraints* that can be posted between pairs of activities in the MCS, K is defined as follows

$$\frac{1}{K(\text{MCS})} = \sum_{i=1}^{k} \frac{1}{1 + commit(pc_i) - commit(pc_{min})}$$

where pc_{min} is the precedence constraint with the minimum value of $commit(pc)$.

To complete the specification of CCSA we define the `Select-Leveling-Constraint` function. According to the least-commitment strategy, after selecting an MCS to be leveled, the leveling constraint $a_i\{before\}a_j$ correspondent to pc_{min} is chosen.

The algorithm described so far follows the spirit of a complete planning algorithm proposed in [7], but such an approach is simply not practical for the kind of problems addressed here. The function `Clique-Tree`, in particular, is the critical point, given the frequency that the function is invoked and the exhaustive nature of the search it performs. Following some preliminary experimentation, the need for a more efficient counterpart was clear, and the present implementation of CCSA thus contains a number of distinctive features.

There are two different situations where we need to search for cliques: in the detection of an unresolvable conflict and in the collection of the minimal critical sets in the PIG

Select-Conflict-Set(CSolution)
1. $mcs_{max} \leftarrow \emptyset$
2. **foreach** $r_j \in R$ **do begin**
3. $CV \leftarrow V_{PIG_j}$
4. **while** ($\mid CV \mid > c_j$) **do begin**
5. $v_i \leftarrow$ Select-Vertex(CV)
6. $CV \leftarrow CV - \{v_i\}$
7. $mcsSet \leftarrow$ Clique-Tree($\{v_i\}$,$CV \cap J_i$)
8. $mcs \leftarrow max\{K(mcs_i) \mid mcs_i \in mcsSet\}$
9. **if** K(mcs) > K(mcs_{max}) **then** $mcs_{max} \leftarrow mcs$
10. **end**
11. **end**
12. **return**(mcs_{max})

Figure 3: To select a set of conflicting activities

graphs for applying the heuristic strategy. In the first case, it is sufficient to demonstrate that at least one MCS exists on the DIG graph. This task is not too critical because generally DIGs are very sparse. In the second case, however, we need to potentially detect all the MCSs associated with a given PIG. This task is more critical since, when the solution method starts, PIG graphs are close to the complete status and the number of MCSs can reach $\binom{|A_j|}{c_j+1}$.

To provide a more tractable procedure for this latter case, we introduce the notion of a *sampling strategy* - aimed at only partially computing the $mcsSet$ for each PIG_j. Given a PIG_j we sample a subset of MCSs while paying attention to "cover" the entire graph. The present *sampling strategy* works as follows: the `Clique-Tree` is iteratively applied starting from each activity $a_i \in A_j$ For each call, when the first MCS is found, the detection of further MCSs continues with two restrictions: (1) only a limited number of cliques is collected (at present 5) and (2) only a limited number of nodes is explored (at present 100). For example, in the case of a resource r_j with 25 activities and capacity $c_j = 5$, the number of MCS cliques can reach $\binom{25}{6} = 177,100$. Employing the sampling strategy, the maximum number of MCSs sampled is $25 \times 5 = 125$. The experimental results presented later clearly indicate the utility and power of this "bounded search strategy" within CCSA.

A second heuristic adjustment to `Clique-Tree` concerns the function `Select-Vertex`. At present it first selects the vertex v_i with the maximum value of d_i, and in this way, the

probability of finding a clique as soon as possible is heuristically maximized. The same `Select-Vertex` function is used at Step 5 in the algorithm `Select-Coflict-Set`. This intuition has been confirmed by empirical comparative analysis of computation times. For example, when the minimum value of d_i is chosen first instead of the maximum, the CPU time lengthens by up to the 30%.

4 Experimental Evaluation

In this section we establish a common basis for experimental comparison of alternative solution procedures to the MCM-SP. We first specify relevant evaluation criteria. Second, we describe a reproducible procedure for generating test problems.

Evaluation Criteria. 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_{ps});
- *average CPU time*, in seconds, 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;
- the average quality of the solutions generated. Two factors contribute to judgements of this aspect. From an optimization viewpoint, we will measure the *compactness* of the solution and from an executability viewpoint, we will characterize its *robustness*.

As a measure for compactness we use the solution's overall *makespan* (M_{ks}), a standard and well-known measure of schedule quality.

Robustness is a trickier notion to formalize. It is generally believed to relate to the likelihood of being able to absorb minor variations to a solution without incurring a major disruption to the overall solution. When considering strongly temporalized domains (such as the problem of interest in this paper) this capability seems connected to the degree of "fluidity" of solutions, as defined by the temporal flexibilities in the execution intervals of scheduled activities. Building on this intuition, we define as the *robustness* RB of a solution S the average width, relative to the temporal horizon H, of the distance, for all pairs of activities, between of start time of one activity and end time of the other. More precisely,

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

where N_a is the number of activities in the solution, a_h and a_l are generic scheduled activities, $d(s_{a_l}, e_{a_h})$ is the length of the shortest path between the start-time of a_l and the end-time of a_h, and 100 is a scaling factor.

Experimental Design. The first step toward establishing a reproducible experimental setting is the implementation of a controlled random number generator. We adopt the generator proposed in [13] (pag.179). In this way we obtain the uniform distribution function $U[a, b]$ which generates a random number n, where n, a and b are positive numbers such that $a \leq n \leq b$ (if a and b are integers then n is obtained as an integer, if one of them is real n is real). To generate different problem instances we use the time seeds reported in Figure 1 of [13] (in particular we use the first 50 seeds).

Next, we define the dimensions along which problem instances will be varied. According to the usual format for formulating job shop scheduling problems, we use the terminology $N_{jobs} \times N_{res}$ and define problems 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 our purposes in this paper, we create problem sets of 50 instances at each of the following sizes: 5×5, 10×5, 15×5, 20×5 and 25×5.

The remaining data to generate the instances of MCM-SP are assigned as follows: (a) every resource r_j has a capacity c_j, generated randomly as either $U[2, 3]$ or $U[2, 5]$, and a full availability on the horizon; (b) 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 $p = U[0, 0.4]$; (c) the separation constraints $[a, b]$ between every two consecutive activities in a job are generated with $a = U[0, 10]$ and $b = U[40, 50]$; (d) 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 as $H = m_v H_0$ where H_0 is adapted from [3]: $H_0 = (N_{jobs} - 1)p_{bk} + \sum_{i=1}^{N_{res}} p_i$, where p_{bk} is the average minimum processing time of the activities on the bottleneck resource, and p_i is the average minimum processing time of the activities 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. The parameter m_v is used to reduce the horizon and increase the problem constrainedness. We use either $m_v = 1$ or $m_v = 0.7$.

Experimental Results. In this section we consider the performance characteristics of ESTA and CCSA on sets of MCM-SP problem instances of increasing scale, varying both the temporal horizon (affecting tightness of temporal constraints) and resource capacity levels. Each of the 4 experiments summarized below consist of 50 random problem instances of each of 5 problem sizes.

Experiment 1. $m_v = 1$ and $c_j \in [2, 3]$.

$m_v = 1$, $c_j \in [2,3]$						
Method	Problem	N_{ps}	RB	M_{ks}	N_{LC}	CPU
ESTA	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
CCSA	5×5	50	9.5	203.2	18.3	2.5
	10×5	50	7.8	299.1	118.6	40.9
	15×5	50	6.6	448.2	264.8	162.4
	20×5	50	5.6	597.4	447.3	430.3
	25×5	50	4.7	754.3	677.3	925.6

Experiment 1 provides the basic scenario also used in [2] for comparison of alternatives. ESTA and CCSA perform almost equivalently on this problem set with respect to solution quality (ESTA is actually slightly better). This is significant, since CCSA CPU times are an order of magnitude higher than ESTA's. CCSA also posts considerably more leveling constraints than ESTA; interestingly, however, these leveling constraints are much more "well chosen" (as indicated by the relatively high values obtained for RB).

Experiment 2. To investigate the impact of increased resource capacity levels on algorithm performance, a 2nd problem set was generated with settings $m_v = 1$ and $c_j \in [2, 5]$. Results are given below.

$m_v = 1$, $c_j \in [2,5]$						
Method	Problem	N_{ps}	RB	M_{ks}	N_{LC}	CPU
ESTA	5×5	49	10.1	205.7	0.8	0.9
	10×5	50	9.5	259.3	10.6	4.7
	15×5	50	6.6	357.7	41.0	14.7
	20×5	50	5.5	457.2	94.3	33.8
	25×5	50	4.9	563.8	160.3	65.3
CCSA	5×5	50	10.0	201.7	10.4	1.7
	10×5	50	8.9	261.6	85.4	44.9
	15×5	50	6.9	405.2	204.9	200.7
	20×5	50	6.3	536.5	355.9	544.5
	25×5	50	5.8	667.6	533.8	1152.1

Two points are noteworthy here. First, though ESTA does miss one solution out of the total set of 250, it consistently produces solutions with better makespan that does CCSA. Second, the ratio between CCSA's CPU times and ESTA's CPU time increases on this problem set, with CCSA requiring greater computation and ESTA requiring slightly less. This fact is explained by the observation that the complexity of the `Clique-Tree` algorithm increases as the size of MCSs increases; in contrast, ESTA needs to post fewer leveling constraints to solve these problem instances (compare N_{LC} values in the case of Experiment 1 and 2).

Experiment 3. In this experiment, resource capacity levels are kept as in experiment 2, but the overall scheduling horizon is tightened, i.e., $m_v = 0.7$ and $c_j \in [2,5]$, producing problems that are more temporally constrained.

$m_v = 0.7$, $c_j \in [2,5]$						
Method	Problem	N_{ps}	RB	M_{ks}	N_{LC}	CPU
ESTA	5×5	49	4.4	202.0	0.9	1.0
	10×5	50	8.8	252.1	11.2	5.0
	15×5	50	7.0	338.2	43.4	15.1
	20×5	50	5.6	437.8	94.8	34.6
	25×5	49	4.9	527.4	159.3	66.2
CCSA	5×5	49	4.5	200.4	3.9	1.1
	10×5	50	10.3	220.3	90.1	42.6
	15×5	50	8.0	295.6	231.4	216.7
	20×5	50	6.5	408.4	406.1	594.6
	25×5	50	5.8	507.7	599.4	1241.0

In this case, the computation times of both algorithms remain about the same. However, CCSA is seen to find slightly better quality solutions.

Experiment 4. Finally, we consider the case of tighter temporal horizon with lower resource capacity levels — $m_v = 0.7$ and $c_j \in [2,3]$ — yielding the most constrained problem set of all.

$m_v = 0.7$, $c_j \in [2,3]$						
Method	Problem	N_{ps}	RB	M_{ks}	N_{LC}	CPU
ESTA	5×5	48	4.2	203.6	1.8	1.0
	10×5	50	6.4	271.8	21.3	5.5
	15×5	47	4.6	384.7	73.1	18.4
	20×5	44	4.0	483.9	143.8	41.9
	25×5	38	3.3	598.0	247.4	84.0
CCSA	5×5	48	4.3	201.7	7.6	1.3
	10×5	50	8.7	232.3	121.5	38.4
	15×5	50	6.2	333.5	295.5	177.7
	20×5	50	5.1	448.3	503.1	469.9
	25×5	50	4.4	555.2	736.0	972.6

In this last case, performance trends are clearly reversed; CCSA consistently finds better quality solutions and ESTA solves less problems. Thus CCSA appears well suited for solution of more highly constrained problems.

One final observation relating to the leverage provided by the sampling strategy employed in CCSA is appropriate. If we examine the ratio between CPU times in moving from one problem size to the next, we see that it decreases as function of N_{jobs}. This fact can be directly attributed to the sampling strategy, since cutoffs are more limiting at higher problem sizes. Thus, while drastically reducing the number of MCSs considered, the sampling strategy nonetheless enables CCSA to consistently find good quality solutions.

5 Conclusions

In this paper, we developed and analyzed least-commitment CSP-based procedures for solving scheduling problems with metric temporal constraints and multiple capacitated resources, referred to formally as the Multiple Capacitated Metric Scheduling Problem (MCM-SP). We considered both profile-based and clique-based solution approaches to this class of scheduling problem, setting both within a common CSP search framework to facilitate comparison. In both cases, we started from previously developed techniques for reasoning with multiple capacitated resources. With respect to profile-based approaches, we adopted the highly efficient, two-stage ESTA procedure, which was previously shown to significantly outperform basic profile-based methods. In the case of clique-based methods, the procedure originally outlined in [7] was taken as a starting point and a heuristic variant was developed. This development was necessitated by early recognition of the impracticality of applying exhaustive search procedures to this class of scheduling problem, and led to specification of the CCSA procedure, which incorporates a highly effective

heuristic sampling strategy to bound the conflict detection search.

Experimental results obtained on a range of problems of increasing scale, varying both the tightness of temporal constraints and size of resource capacity levels, have indicated the relative strengths of ESTA and CCSA. From the standpoint of computational cost, CCSA (even with use of the sampling procedure) is considerably more expensive than ESTA, and the differential increases as resource capacity levels are increased. In this circumstance, both algorithms perform comparably with respect to solution quality, suggesting the advantage of ESTA in this region of the problem space. This is quite impressive, given the relatively simple idea that motivates ESTA's design. In highly constrained regions of the problem space, on the other hand, CCSA consistently produces better quality solutions and the performance of ESTA falters.

Given the performance potential of CCSA as a solution approach to the MCM-SP, one obvious area for further investigation would concern the development of more efficient clique-detection algorithms. For example, computational leverage might be gained by relying on particular properties of the PIG_j and DIG_j graphs such as the fact that they are weakly chordal graphs.

Two other approaches in the literature address multi-capacitated scheduling problems which differ from MCM-SP. As previously mentioned [9] has developed procedures to solve the MCJSSP problem, a class of scheduling problem which assumes only qualitative separation constraints (i.e., precedence) between pairs of activities and does not allow metric quantification. In [1] the *cumulative scheduling problem* is defined, which differs from MCM-SP in that for each activity a constraint is imposed only on the product *duration* $\times$ *resource requirement* and because again the separation constraints between pair of subsequent activities are not quantitative. However, despite these differences in the types of problems considered, it would be interesting to examine if the techniques developed in this work can be extended and applied to the MCM-SP.

One extension that we are currently working on is a relaxation of the current assumption regarding $rc_{a_k,j}$ to allow values greater than one. This restriction has been introduced to simplify the algorithms and to focus the attention on the pure comparison between profile-based and clique-based approaches. But we also recognize the necessity of this more flexible modeling assumption in many practical domains.

Acknowledgments

Amedeo Cesta and Angelo Oddi's work is supported by Italian Space Agency, by CNR Committee 12 on Information Technology (Project SCI*SIA), and CNR Committee 4 on Biology and Medicine. 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

[1] Y. Caseau and F. Laburthe. Cumulative scheduling with task intervals. In M. Maher, editor, *Logic Programming, Proceedings of the Joint International Conference and Symposium on Logic Programming*, Bonn, Germany, 1996. MIT Press.

[2] A. Cesta, A. Oddi, and S.F. Smith. 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. `http://www.cs.cmu.edu/afs/cs/user/sfs/www/AIPS98/aips-98.html`.

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

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

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

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

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

[8] N. Muscettola. Scheduling by Iterative Partition of Bottleneck Conflicts. In *Proc. 9th IEEE Conference on AI Applications*, 1993.

[9] W.P.M. Nuijten and E.H.L. Aarts. A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling. *European Journal of Operational Research*, 90(2):269–284, 1996.

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

[11] N. Sadeh. *Look-ahead Techniques for Micro-opportunistic Job-shop Scheduling.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.

[12] S.F. Smith and C. Cheng. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Proceedings 11th National Conference on AI (AAAI-93)*, 1993.

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