A Constraint-Based Method for Project Scheduling with Time Windows*

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

[1] IP-CNR, National Research Council of Italy
Viale Marx 15, I-00137 Rome, Italy, {*cesta, oddi*}*@ip.rm.cnr.it*
Phone: +39-06-86090-209

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

Abstract

This paper presents a heuristic algorithm for solving RCPSP/max, the resource constrained project scheduling problem with generalized precedence relations. The algorithm relies, at its core, on a constraint satisfaction problem solving (CSP) search procedure, which generates a consistent set of activity start times by incrementally removing resource conflicts from an otherwise temporally feasible solution. Key to the effectiveness of the CSP search procedure is its heuristic strategy for conflict selection. A conflict sampling method biased toward selection of minimal conflict sets that involve activities with higher-capacity requests is introduced, and coupled with a non-deterministic choice heuristic to guide the base conflict resolution process. This CSP search is then embedded within a larger iterative-sampling search framework to broaden search space coverage and promote solution optimization. The efficacy of the overall heuristic algorithm is demonstrated empirically on a large set of previously studied RCPSP/max benchmark problems.

* Corresponding author: Amedeo Cesta, IP-CNR, Viale Marx 15, I-00137 Rome, Italy, phone: +39-06-86090-209, fax: +39-06-824737, e-mail: cesta@ip.rm.cnr.it.

Contents

1 Introduction

In this paper we present a constraint-based procedure for solving the scheduling problem known in the Operations Research (OR) literature as the Resource Constrained Project Scheduling Problem with Generalized Precedence Relations (or RCPSP/max). This scheduling problem derives from a project management environment in which activities represent steps that must be performed to achieve completion of the project and are subject to partial order constraints that reflect dependencies on project progression. Drawing from previous work in resource-constrained scheduling, we develop a basic constraint satisfaction problem solving (CSP) search procedure for scheduling with cumulative resources. Our procedure proceeds by iteratively detecting and leveling "resource contention peaks", i.e., sets of activities that temporally overlap and whose total resource requirement exceed resource capacity. Extending previous "profile-based" scheduling approaches [7, 9], we define new search control heuristics, based on the intuition that the most critical conflicts to resolve first are those involving activities with large resource capacity requirements (since these conflicts generally have the few resolution alternatives). A given conflict is resolved, as in previous work, by posting a precedence constraint between two of the competing activities. To solve the RCPSP/max problem, we repeatedly apply this core CSP procedure within a larger iterative-sampling search process.

1.1 The RCPSP/max Problem

The Resource Constrained Project Scheduling Problem (RCPSP) has been widely studied in Operations Research (OR) literature (see [4] for a recent survey). RCPSP/max is a specific formulation of the basic problem underlying a number of scheduling applications [23] and is considered particularly difficult, due to the presence of temporal separation constraints (in particular maximum time lags) between project activities. In fact, finding a feasible schedule alone is NP-hard [3].

The RCPSP/max can be formalized as follows:

- a set V of n activities must be executed, where each activity j has a fixed duration d_j. Each activity has a start-time S_j and a completion-time C_j that satisfies the constraint $S_j + d_j = C_j$.
- a set E of temporal constraints exists between various activity pairs $\langle i, j \rangle$ of the form $S_j - S_i \in [T_{ij}^{min}, T_{ij}^{max}]$, called start-to-start constraints (time lags or *generalized precedence relations* between activities). [1]
- a set R of renewable resources are available, where each resource r_k has a integer capacity $c_k \geq 1$.
- execution of an activity j requires capacity from one or more resources. For each resource r_k the integer $rc_{j,k}$ represents the required capacity (or *size*) of activity j.

[1] Note that since activity durations are constant values, end-to-end, end-to-start, and start-to-end constraints between activities can all be represented in start-to-start form.

A schedule S is an assignment of values to the start-times of all activities in V ($S = (S_1, \ldots, S_n)$). A schedule is *time-feasible* if all temporal constraints are satisfied (all constraints $S_j - S_i \in [T_{ij}^{min}, T_{ij}^{max}]$ and $S_j + d_j = C_j$ hold). A schedule is *resource-feasible* if all resource constraints are satisfied (let $A(S,t) = \{i \in V | S_i \leq t < S_i + d_i\}$ be the set of activities which are in progress at time t and $r_k(S,t) = \sum_{j \in A(S,t)} rc_{j,k}$ the usage of resource r_k at that same time; for each t the constraint $r_k(S,t) \leq c_k$ must hold). A schedule is *feasible* if both sets of constraints are satisfied. The RCPSP/max optimization problem, then, is to find a feasible schedule with *minimum makespan* MK, where $MK(S) = max_{i \in V}\{C_i\}$.

1.2 A CSP Approach to RCPSP/max

Scheduling problems such as RCPSP/max can be seen as a special type of Constraint Satisfaction Problem (CSP) [22]. An instance of a CSP involves a set of variables $X = \{X_1, X_2, \ldots, X_n\}$, a domain D_i for each variable and a set of constraints $C = \{C_1, C_2, \ldots, C_q\}$ s.t. $C_i \subseteq D_1 \times D_2 \times \cdots \times D_n$, which define feasible combinations of domain values. A solution is an assignment of domain values to all variables which is consistent with all imposed constraints. A general algorithmic template for solving a CSP is shown in Figure 1. It can be seen as an iterative search procedure where the current (partial) solution is extended on each cycle by assigning a value to a new variable.

CSPsolver(Problem)
1. CreateCSPForProblem
2. **while not**(solved **or** infeasible) **do**
3. RemoveInconsistentValues
4. SelectDecisionVariable
5. SelectValueForVariable
6. **end**

Figure 1: Basic CSP Search Procedure

As each new decision is made during this search, a set of "propagation rules" removes elements from domains D_i which cannot be contained in any feasible extension of the current partial solution (Step 3 of the algorithm). In general though, it is not possible to remove all inconsistent values through propagation alone. Choices must be made between possible values for some variables, giving rise to the need for variable and value ordering (or selection) heuristics (Steps 4 and 5 in the figure). Such "search control" heuristics generally depend on characteristics of the specific problem at hand, but there are "domain independent" criteria which are commonly used to define heuristics. A standard variable ordering criterion prefers the *most constrained variable* —the variable most difficult to instantiate, or equivalently, the variable that is most likely to lead to an infeasible state. For value selection, it is quite common to choose the *least constraining value* —the value that leaves open as many values as possible for any remaining unassigned variables. The algorithm in Figure 1 describes a greedy (and partial) CSP

procedure. It is frequently embedded within a backtracking framework to define a complete solution procedure [28, 10]. But it is also possible to exploit its use within other search frameworks (e.g., limited discrepancy search [17], random restart [25, 26], etc.).

Formulating RCPSP/max as a CSP. Within the CSP scheduling literature, there are two main approaches to formulating a scheduling problem as a constraint satisfaction problem. The first (and perhaps most direct) formulation is a *start time assignment* model [28, 26]. Under this model, decision variables are time points that designate the start times of various activities and CSP search focuses on determination of a consistent assignment of start time values. The RCPSP/max problem can be straightforwardly formulated in this way. Starting from the definition in Section 1.1, a variable is introduced for each activity; hence V is the set of CSP variables. For each variable i a domain $D_i = [0, H - d_i]$ is assigned (where H is an upper bound on the scheduling horizon), specifying its possible start times. Two types of constraint combine to further restrict the values that may be assigned to the set V of variables: (1) binary constraints (involving pairs of variables) for representing the start-to-start temporal relations between activities; and (2) n-ary constraints to describe the capacity constraints that each resource imposes on all feasible schedules.

The second CSP formulation of a scheduling problem is a *precedence constraint posting model* [31, 10]. In this case, decision variables correspond alternatively to the various ordering decisions that need to be made between sets of activities that are competing for the same resources, and CSP search focuses on specifying (or posting) a consistent set of precedence constraints that eliminates any possibility of resource contention. One principal advantage of this sequencing approach is that it avoids over-commitment, as activities need not be anchored to specific start times during the search or in the final solution.

To formulate RCPSP/max under this model, a variable is defined for each set of activities that simultaneously require resource r_k with a total capacity requirement $> c_k$ (i.e., each potential resource conflict). For each such variable i, the domain $D_i = \{pc_{i,1}, \ldots\}$ consists of the set of precedence constraints that can be feasibly posted to eliminate the conflict. Operationally, the ongoing determination of domain D_i for each ordering decision variable i requires propagation of activity start times in an underlying time point network. Thus, a precedence constraint posting model can be seen as a *meta-CSP* formulation, which utilizes a start-time assignment model as a *ground-CSP* representation. We take this approach to developing our procedure for solving RCPSP/max.

Contribution of the paper. In this paper, we develop a constraint posting scheduling procedure for RCPSP/max. The main contribution of this work lies in the development of effective search heuristics for this class of problem, which use analysis of interactions between time and resource capacity constraints to productively bias variable and value selection decisions. Central to this analysis and to the heuristics that are defined is the notion of Minimal Critical Sets. A Minimal Critical Set (MCS) is defined as a *a*

set of activities that simultaneously require a resource r_k with a combined capacity requirement greater than the resource's total capacity, such that the combined requirement of any subset is less or equal to the resource capacity. From the definition of an MCS, it follows that the posting of a single precedence between some pair of activities in the MCS is sufficient to eliminate the conflict. By isolating MCSs as the decision variables of the meta-CSP, and by defining a solution procedure that iteratively selects and resolves an MCS in the current solution until no such conflicts remain, it is possible to consistently focus the search on the most critical conflicts and minimize the number of constraints that must be posted to obtain a feasible solution.

As suggested above, the basic search procedure is best summarized as a two layered algorithm: (a) the ground layer, or *ground-CSP*, represents the temporal aspects of the problem, in the form of a quantitative temporal constraint network [14]; (b) the higher layer is a *meta-CSP*, where resource conflicts are represented and reasoned about. The search proceeds by iterating performing the following sequence of steps: propagation of temporal consequences through the ground-CSP; computation of the meta-CSP; selection of a conflict in the meta-CSP; resolution of the conflict by imposing a new precedence constraint in the ground-CSP.

A similar search schema was first used in [10] for scheduling problems with binary (disjunctive) resources. Under these resource assumptions, the meta-CSP of conflicts is computable in a fairly efficient manner (polynomial cost), since resource conflicts are represented by pairs of temporally overlapping activities that require the same resource. However, adaptation of the approach to RCPSP/max, and more generally to scheduling problems with non-binary (cumulative) resources, requires that more attention be given to the meta-CSP computation. Under these extended assumptions, complete computation of the meta-CSP (i.e., collection of all MCSs) is exponential in the size of total resource capacity and some form of approximation becomes essential for practical solution.

To address this computational problem, we integrate and extend two key ideas from previous research. (1) after propagation in the ground-CSP we extract the earliest start times of all temporal variables, and this so-called ESS (Earliest Start Schedule) is used as a basis for computing MCSs (i.e., the meta-CSP). (2) a polynomial MCS sampling method is introduced, which extracts a subset of MCSs and overcomes the exponential worst case cost of complete enumeration. This sampling method capitalizes on a heuristic criterion that acts to extract the most critical MCS first. The result is an efficient greedy procedure for generating feasible solutions to RCPSP/max problems.

To solve RCPSP/max problems to optimality, we encapsulate this function within an optimization schema. One possible way in which the effectiveness of a greedy (partial) solution procedure can be enhanced is through the addition of backtracking or restarting mechanisms, which serve to broaden the search in the event of failure. Exploiting the approach taken in [27], we introduce a randomized variant of the conflict selection heuristic and embed the core resolution procedure within a larger, iterative sampling

search. Iterative sampling provides a framework for finding "lower makespan" solutions.

The overall procedure is called ISES (for Iterative Sampling Earliest Solutions). In this paper ISES is compared with the best existing approaches to RCPSP/max. It outperforms several systematic and heuristic approaches on different reference problem sets. In particular, ISES is shown to perform quite well in situations where, due to problem characteristics such as heavy resource contention, the search space is quite large and becomes a serious obstacle for systematic approaches. In this case, our non-systematic random approach demonstrates an advantageous trend in comparison to the best known systematic algorithm (in terms of both the number and the quality of the solutions provided) as problem size is increased.

Plan of the paper. The remainder of the paper is organized as follows. In Section 2, we first review related research in both Operations Research and Constraint Programming. We next describe, in turn, the two principal components of our approach: the greedy resolution procedure and the encompassing optimization cycle. Section 3 introduces the constraint-based conflict resolution algorithm, justifying its design, providing basic definitions and summarizing the technical details of both ground and meta-CSP models. Section 4 describes how randomization is introduced and how iterative sampling is used to search for better solutions. In Section 5, we report comparative performance results on a set of published benchmark problems. A general discussion of the various results closes the paper.

2 Related Research

In this section we briefly summarize state-of-the-art OR approaches to RCPSP/max which we will later use as a basis for comparative evaluation of our approach. We also summarize related CSP scheduling research, which provides the context and underpinnings of our approach.

2.1 Overview of OR Approaches

The RCPSP/max problem has been the subject of considerable recent investigation within the OR community. An early analysis appeared in [3] aimed principally at investigating the mathematical properties of the problem, proposed a solution approach quite similar in many respects to the CSP based procedure we introduce below. It similarly conceptualized the solution space as a network of temporal constraints, and introduced notions of *forbidden sets* and *reduced forbidden sets* to designate resource conflicts and resource conflicts with a minimal number of activities respectively. A systematic Branch and Bound (B&B) procedure was sketched that proceeds by extending the set of precedence relations in the problem to eliminate all reduced forbidden sets in an initial, time-feasible solution. Unfortunately the computational analysis is quite limited and few results on small instances are only mentioned.

More recent B&B approaches to RCPSP/max have retained the idea of extending a time-feasible solution by adding precedence relations, but analyze the evolving solution differently. In [13] and [30], conflicts are considered in increasing chronological time order and, for each of them, a set of "minimal delaying activities" are detected for resolving conflicts.

Other B&B algorithms work differently. For example [21] solves resource conflicts by increasing release dates (minimum time lags relative to beginning of the project) for certain activities instead of introducing precedence relations. A very recent proposal [15] obtained strong results using a combination of techniques: it couples a set of temporal and resource constraint propagation rules with a binary branching schema that exploits particular properties of the current partial solution to successively fix and/or delay the start-times of various activities.

Though most work has focused on exact solution procedures, approximate RCPSP/max procedures have also been developed. The heuristic presented in [16] has recently obtained very good results on RCPSP/max benchmark problems (see below). Despite being classified as a "priority rules" approach, this work uses a two step method: (a) a sophisticated decomposition analysis is performed to identify "critical sub-components" which can be scheduled independently (a technique introduced in [24]), and (b) the scheduled sub-components (partial schedules) are integrated into one using a set of priority rules. Note that the first step uses properties due to the presence of maximum time lags.

2.2 Overview of CSP Approaches

A considerable amount of work has focused on the development of CSP models and solution procedures for scheduling problems, although the specific RCPSP/max problem has not been addressed.

Early work in this area focused on techniques for solving the classical job-shop scheduling problem [28, 31, 5, 1], yielding a number of different constraint propagation rules and search control heuristics, and several high performance solution procedures. More recent research has demonstrated the flexibility of CSP models to accommodate extended classes of scheduling problems. Some work has focused on solution procedures for scheduling problems with generalized precedence relations (i.e., maximum and minimum metric separation constraints between activities) [10, 27]. A number of other efforts have focused on CSP scheduling algorithms for problems with cumulative (or multi-capacity) resources [2, 25, 6, 12]. Much of this work has focused on constraint propagation techniques that exploit the structure of cumulative resource constraints, and are hence capable of stronger inference. Less attention has been paid to development of effective heuristics for managing the scheduling search process.

Our previous research with cumulative scheduling problems, alternatively, has focused principally on the development of heuristics for managing the search process, and several specific ideas integrated in the ISES algorithm have evolved from techniques first proposed in different contexts. In [7], we addressed a machine scheduling problem called the *Multi-Capacitated Metric Scheduling Problem (MCM-SP)*, which

simultaneously involves both cumulative resources and generalized temporal relations. MCM-SP exhibits a problem structure wherein each individual job consists of a linear sequence of activities. Each activity of a given job requires one unit of capacity on a single resource to be executed. Formally, MCM-SP is an extension of classical Job-Shop Scheduling Problem (JSSP) and of the Multi-Capacitated JSSP introduced in [25]. In this context, the notion of computing conflicts through observation of an earliest start schedule (ESS) was first explored, and shown to outperform basic strategies that operate with respect to a more general, upper-bound computation of resource consumption.

At the same time, the conflict identification technique used in [7] was simpler than the MCS-based analysis used in this paper, being based directly on the pairwise analysis previously used to solve disjunctive scheduling problems [10]. The current MCS-based approach integrates ideas from a previously reported clique-based approach to resource reasoning [20], which, due to its more global perspective, provides a basis for more effective search control. While introducing the idea of MCSs as resource conflicts, the work of [20] proposes a combinatorial (and effectively intractable) systematic approach to MCS computation (clique detection on a "activity intersection graph"). This computational difficulty has motivated our development of effective, approximate, "conflict sampling" methods.

A preliminary version of this work appeared in [9]. With respect to that work, the current paper introduces and emphasizes a different strategy for conflict sampling, which tends to isolate pairs of activities that must be sequenced in any feasible solution. We also include a more comprehensive experimental analysis, using a wider set of parameters and demonstrating the improved performance of the new sampling procedure.

3 Constructing a Feasible Solution

As sketched in the introduction the ISES algorithm consists of two parts, a basic greedy algorithm described in this section and a optimization cycle based on random restart described in the next section. The resolution algorithm presented below follows the precedence constraint posting schema previously mentioned. The approach is further distinguished by problem solving schema based on a ground-CSP that considers the temporal aspects of the problem and a meta-CSP that reasons about the resource conflicts. The general procedure first computes an initial, time-feasible solution (ignoring all resource constraints). Resource constraints are then super-imposed, giving rise to a set of resource *conflicts* (i.e., sets of activities competing for the same resource capacity over some time interval) and hence a set of sequencing (or conflict resolution) decisions.

Figure 2 presents the basic greedy procedure used. It accepts a problem instance (`Problem`) and an upper bound on the overall makespan (`Horizon`). In describing the algorithm we first clarify its two layered structure by introducing the ground- and meta-CSP and a number of connected definitions.

```
ESA(Problem, Horizon)
1.   ground-CSP ← CreateCSP(Problem)
2.   ground-CSP ← PostContraint(ground-CSP, Horizon)
3.   loop
4.        Propagate(ground-CSP)
5.        meta-CSP ← ComputeResourceConflict(ground-CSP)
6.        if Empty(meta-CSP)
7.        then return(ExtractSolution(ground-CSP))
8.        else
9.             if Unsolvable(meta-CSP)
10.            then return(EmptySolution)
11.            else
12.                 Conflict ← SelectConflict(meta-CSP)
13.                 PrecedenceConstraint ← SelectPrecedence(Conflict)
14.                 ground-CSP ← PostCostraint(ground-CSP, PrecedenceConstraint)
15.  end-loop
```

Figure 2: Basic CSP Search Procedure

Ground-CSP: Temporal Analysis. The first step of ESA produces a Temporal Constraint Network [14] to represent the temporal constraints of the problem (Line 1). A temporal constraint network is a specialized CSP formulation where variables represent time points (or temporal events), their values are time instants, and constraints are binary inequalities that bound the distance between two time points (given two time points tp_k, tp_l a distance constraint is $tp_k - tp_l \leq d_{kl}$ with d_{kl} an integer value). Time-points tp_j are used to represent the start-time variables S_j of all activities in the RCPSP/max instance, as well as the begin and end of the temporal horizon. Temporal constraints in the problem are mapped immediately to distance constraints between appropriate time-points, and an additional constraint is posted to bound the maximum temporal horizon (Line 2 of the ESA procedure). The temporal constraint network described here corresponds to the so-called Simple Temporal Problem (STP) [14]. A STP has two useful properties: (1) the domains of variables are convex intervals of values (i.e., the set of possible values of each time point tp_j that is consistent with all distance constraints corresponds to an interval $[lb_j, ub_j]$); and (2) the algorithm for computing such intervals (i.e., propagating the effects of the various constraints) is polynomial in the number of temporal variables.

A time feasible solution for the RCPSP/max is extracted from a temporal network by choosing a consistent value for each time point. One simple approach [14] is to choose for each activity $j \in V$ the earliest possible start times ($S_j = lb_j$).

Definition: Earliest Start Schedule (ESS). An ESS is a consistent temporal assignment to the set of start time variables S_j such that for each activity $j \in V$ the value $S_j = lb_j$ is chosen. For a given activity $j \in V$, we will indicate with $est(j)$ the earliest start time of the activity j ($est(j) = lb_j$) and with

$eft(j) = est(j) + d_j$ its earliest finishing time.

If an ESS, which is temporally feasible by definition, is also resource feasible, then we have found a solution for the RCPSP/max. The theoretical results in [3] show that an optimal makespan solution is an ESS. This result gives a theoretical grounding to our general heuristic. In fact, one distinguishing aspect of our approach is the fact that we compute resource conflicts relative to the ESS instead of, for example, reasoning from the complete temporal constraint network in the ground-CSP. The greedy algorithm is referred to as ESA (Earliest Start Algorithm) because it iteratively extends the current ESS until one without resource conflicts is found.

Meta-CSP: Resource Analysis. Over the basic temporal network formulation, we super-impose resource capacity constraints to identify the set of outstanding conflicts. This conflict set is resolved incrementally by repeated posting of precedence constraints between pairs of competing activities.

The basic cycle for transforming an initial ESS into a conflict-free solution proceeds as follows:

1. propagate the current ground-CSP (temporal network) to compute current bounds for all temporal variables (line 4 of Figure 2);

2. consider the resource constraints in the RCPSP/max and identify the set of resource capacity violations implied by the current ground-CSP (i.e., construct the current meta-CSP of resource conflicts —line 5).

3. select a resource conflict in the meta-CSP by applying a *variable ordering heuristic* (line 12).

4. solve the conflict by applying a *value ordering heuristic* and choosing best ranked value (lines 13-14). As indicated above, this step has the correspondent action in the ground-CSP of imposing a new precedence constraint between some pair of competing activities.

If the meta-CSP has no unassigned variables, there are no remaining resource conflicts and a feasible solution has been found. If, alternatively, the set of feasible orderings of a conflict becomes empty (i.e., some variable in the meta-CSP has no possible values), then there is no feasible completion to the current partial solution. To understand how the meta-CSP is computed we need two further definitions:

Definition: Contention Peak. Given an ESS, a *contention peak* (or simply a *peak*) on resource r_k is a set of activities $P = \{j_1, j_2 \ldots j_p\}$ that simultaneously require r_k (i.e., they temporally overlap: $\min_{j \in P}\{eft(j)\} > \max_{j \in P}\{est(j)\}$ holds) and whose combined capacity requirement is $> c_k$.

Definition: Minimal Critical Set (MCS). A MCS is a peak on a resource r_k such that *any* subset of its activities has a combined requirement $\leq c_k$.

The important advantage of isolating MCSs, as mentioned earlier, is that a single precedence relation

between any pair of activities in the MCS eliminates the resource conflict. (Such constraints are posted between the end-time of one activity and the start-time of the other to avoid overlap.) The meta-CSP is defined by taking the current set of MCSs as variables, and the set of precedence constraints that can be feasibly posted between some pair of activities in a given MCS as the domain of the corresponding variable. To be feasible, a precedence constraint must necessarily satisfy all temporal and resource constraints of the original problem.

The meta-CSP is computed using the procedure `ComputeResourceConflict` (line 5 of Figure 2), which expands as shown in Figure 3: The ESS is computed from the ground-CSP, the peaks in the ESS are collected; the MCSs are sampled from the current list of peaks.

ComputeResourceConflict(ground-CSP)
1. ESS $\leftarrow$ ComputeESS(ground-CSP)
2. Peak-List $\leftarrow$ ComputePeaks(ESS)
3. MCS-List $\leftarrow$ SampleMCSs(Peak-List)
4. **end**

Figure 3: Three steps for computing the meta-CSP of resource conflicts

A further distinction between the approach taken in this paper and that of the work previously reported in [7] is worth noting at this point. Both approaches use what we called in [7] profile-based scheduling procedures. In [7] we pursued the broad idea that a solution is obtained when all the contention peaks have been eliminated by the current solution. In general a peak is eliminated by "leveling" it, that is by posting one or more precedence constraints between pairs of activities contributing to the peak. In the present paper we similarly attempt to create solutions without peaks, but the crucial choice of which pair of activities to order is made after a further MCS analysis. This additional analysis is performed to avoid the problem of adding precedence relations that do not have a strong peak leveling effect. The difference can be illustrated with a simple example. Suppose we have a simple problem consisting of a single resource with capacity $c = 8$, and a set of four activities $V = \{j_1[5], j_2[5], j_3[1], j_4[1]\}$ (capacity requirements shown in brackets). All activities have the same duration equal to 10, start time 0 and have to be scheduled in the time window $[0, 100]$. If any pair of activities can be chosen, then the following pairs are all possible choices: $\{\langle j_1, j_2\rangle, \langle j_1, j_3\rangle, \langle j_1, j_4\rangle, \langle j_2, j_3\rangle, \langle j_2, j_4\rangle, \langle j_3, j_4\rangle\}$. However, it can be immediately seen that the only pair that must be ordered *in any solution* is the pair $\langle j_1, j_2\rangle$ because by itself it exceeds the capacity c. One observed shortcoming of previous profile-based approaches to cumulative scheduling has been their tendency to post unnecessary ordering constraints, following from the use of conflict selection and resolution heuristics that rely strictly on pairwise analysis of competing activities [7]. Alternatively, the use of MCS-based analysis in the previous short examples leads directly to detection of the single set $\langle j_1, j_2\rangle$ which, once ordered, resolves the resource conflict. The importance of MCS analysis has been pointed out in [20]; a specific contribution of this paper is the use of such analysis

to characterize detected peaks in the ESS. In this way we obtain an effective trade-off between the quality of the conflict analysis and the time needed to perform it.

Implementation of ESA. The rest of this section gives a more detailed description of three crucial aspects of the ESA procedure. Subsection 3.1 deals with constraint propagation; Subsection 3.2 illustrates the computation of the meta-CSP; and Subsection 3.3 discusses variable and value ordering heuristics that direct the search for a conflict free solution.

3.1 Constraint Propagation

ESA, like any CSP search procedure, interleaves processes of solution refinement, which acts to assign values to variables, and constraint propagation, which, after each new assignment, computes implications with respect to other CSP variables and eliminates inconsistent values. In a scheduling problem like RCPSP/max both temporal and resource constraints can be sources of propagation and performed in the propagation step performed at Line 4 of Figure 2.

Time constraints. A basic aspect of ESA consists of maintaining a temporally consistent network of time points. The network is constrained initially by the set of temporal constraints in the RCPSP/max definition and further constrained as additional precedence constraints are posted during the search. Path consistency in this network is dynamically maintained *via* all pairs shortest path computation, making distance information between any pair of time variables available for use in focusing the search. Path consistency enforced at ground-CSP level allows us to detect (at meta-CSP level) in constant time the set of feasible ordering associate to a decision variable.

Resource constraints. We do not make use of resource constraint propagation rules as is done in some other recent approaches to disjunctive and cumulative scheduling problems (e.g., [2, 25, 26]). Such resource propagation rules are aimed at synthesizing further domain reductions by a specialized reasoning on the temporal information and the resource constraints. We have chosen not to incorporate these more sophisticated forms of propagation, since they play a role complementary to that of the search control heuristics and procedures of principal interest in this paper. It is worth remarking that such propagation could be added in a transparent way: at formal level redefining the ground-CSP to be not a purely temporal problem but a mixed time and resource problem; at algorithmic level by modifying the `Propagate` function at Line 4 adding further deduction rules corresponding to resource constraint propagation. The effect is to further prune the search space (and potentially improve reported results).

3.2 Computing Resource Conflicts

The approach taken to identification of resource conflicts attempts to reconcile two, typically conflicting desiderata: (1) on one hand to always take the decision centers on the most critical precedence constraint to post and (2) on the other to minimize the amount of time spent in the analysis that leads to this decision.

The need for this tradeoff arises due to the computational complexity inherent in complete computation of the meta-CSP (specifically the set of all outstanding MCSs). MCS analysis has been used in [20], where MCSs are seen as particular cliques that are collected via systematic search of an activity "intersection graph" (which is constructed starting from the temporal information in the equivalent of our ground-CSP). The unfortunate drawback of this approach is the exponential nature of the intersection graph search, which prohibits use of this basic approach on scheduling problems of any interesting size. In [8], it is shown that much of the advantage of this type of global conflict analysis can be retained by using an approximate procedure for computing MCSs. But the pragmatic cost of recomputing MCSs across all resources at each iteration of the CSP resolution procedure nonetheless remains high and significantly limits scalability.

In ESA, we improve on this approach to exploiting MCS analysis in two ways. First, we achieve an even better computational tradeoff by instead integrating the use of MCS analysis into a profile-based scheduling framework on the current ESS. As sketched in Figure 3, on each iteration of the search, we first compute contention peaks (which is quadratic in the number of activities) to isolate those areas of the solution where conflicts (i.e., MCSs) should be computed. Next we generate a set of MCSs for each peak collecting them in the meta-CSP that is ready for being solved.

The number of MCSs contained in a given peak can still be quite large (in the worst case $\binom{|V|}{c_k+1}$), and, accordingly, ESA also utilizes an approximate (heuristic) scheme for computing MCSs. In fact, the second way in which ESA improves on previous use of MCS analysis is in its use of a more rational "MCS sampling strategy". In [8], as well as in an earlier version of ESA reported in [9], MCSs were generated in a fairly ad hoc manner, motivated primarily by the desire to limit the overall number of generated MCS. In this paper we introduce a different approximate procedure for generating MCSs, which is designed to bias generation toward those MCSs closest to an unresolvable state, and hence those which are most "critical" to resolve next. In general, an MCS's distance from an unresolvable state depends on two (not necessarily independent) factors: (1) the number of different precedence constraints that could be posted to resolve it, and (2) the temporal flexibility that is retained in the solution after it is resolved. The MCS generation process employed in ESA considers the first factor. It favors those MCSs that contain activities with the largest resource capacity requirements, and these MCSs tend to be resolvable in the fewest number of ways.

3.2.1 Collecting Peaks

To collect MCSs, ISES uses a two step algorithm which first detects *peaks* in resource usage and then samples MCSs within these detected peaks.

Given a partial solution S and a resource r let $\Pi_r = \{P_1, P_2, \ldots, P_p\}$ be the set of all elements which satisfy the definition of peak. Note that this set is generally quite large. For example, given a peak $P_i \in \Pi_r$, all subsets of P_i whose total capacity requirement exceeds c_k are also contained in Π_r. To promote diversity in the set of MCSs that are ultimately generated and raise the level of heuristic information, we define a peak detection approach that avoids sampling peaks which are either subsets of other peaks or have a large percentage of activities in common. Specifically, we follow a strategy of collecting sets of *maximal peaks* that is, sets of activities such that none of the sets is a subset of the others.

The algorithm takes as input set A_{r_k} for each resource r_k ($k = 1 \ldots n_r$), representing the set of activities requiring r_k, and produces the set of elements PS_k ($k = 1 \ldots n_r$), where each $PS_k = \{P_1, P_2, \ldots, P_{p_k}\}$ represents a set of peaks on r_k. The algorithm `CollectPeaks` proceeds in two main steps:

1. the set of activities $j \in A_{r_k}$ are sorted according to $est(j)$.

2. Activities are sequentially removed from A_{r_k} and collected in the current peak $P = \{j_1, j_2, \ldots, j_p\}$ until the activities in P overlap each other —that is, until the condition $\min_{i \in P}\{eft(i)\} > \max_{i \in P}\{est(i)\}$ holds. When the previous test fails, that is, when an activity i is removed from A_{r_k} and it does not overlaps with all the activities in P, then three further steps are executed: (a) if the total capacity requirement of all activities in P exceeds the resource capacity ($\sum_{j \in P} rc_{j,k} > c_k$), then P is collected in PS_k; (b) P is updated to remove all activities which do not overlap with the last selected i; (c) i is inserted in P. The collecting process continues until A_{r_k} becomes empty.

3.2.2 Sampling MCSs

The conflict sampling procedure accepts as input a set of peaks and two parameters, δ and s_f, and returns a set of minimal critical sets. δ controls the cardinality (number of activities) in any sampled MCS and s_f bounds the overall number of MCSs that are sampled and returned.

Given a peak P on a resource r_k, we are interested in sampling MCSs of the smallest possible cardinality. In fact this is one of the heuristic criteria that is used to bias the sampling process toward critical MCSs. Let M_P be the set of all MCSs in P and m_P the minimal cardinality of any MCS in M_P. M_P can be at most partitioned in $(c_k + 1) - m_P + 1$ subsets of MCSs with equivalent cardinality. $M_P = M_0 \cup M_1 \cup \cdots \cup M_{(c_k+1)-m_P}$, where M_0 is the set of MCSs with the minimum cardinality, and in general M_δ, with $(0 \leq \delta \leq (c_k + 1) - m_P)$, is the set of MCSs with cardinality minimal plus δ. Some of the previous subsets may be empty, but in general their cardinality has a lower bound of 2 and an upper

bound of $c_k + 1$ (in the case all activities of the MCS have a unit capacity requirement). The parameter δ is used to tune the search of critical MCSs; in fact, we can choose to detect only those MCSs in the set M_0 (MCSs of minimal cardinality) or to augment this set with the elements in the sets $M_1, M_2 \ldots M_\delta$.

To sample elements in $M_0 \cup M_1 \cup \cdots \cup M_\delta$, we use the following algorithm. For each peak P we impose a total order on its activities such that for each pair of activities $j_i, j_l \in P$ the relation $j_i \prec j_l$ holds if $rc_{j_i,k} \geq rc_{j_l,k}$ (i.e., the activities contributing to the peak are sorted in decreasing order of their resource capacity requirements $rc_{j,k}$). For example, let r_k be a resource with capacity $c_k = 7$, and P= {j_1[5], j_2[3], j_3[3], j_4[2], j_5[1], j_6[1], j_7[1]} be a sorted peak on resource r_k (values in square brackets represent resource requirements). The total order imposed on P by $\prec$ also induces a lexicographical order on the set M_P (the set of all the MCSs in P). This order is generated in a manner equivalent to sorting a set of English words into alphabetical order by the alphabetical order of their constituent letters. Specifically, assume that two generic MCSs $mcs_i = \{j_{i1}, j_{i2}, \ldots, j_{in}\}$ and $mcs_l = \{j_{l1}, j_{l2}, \ldots, j_{lm}\}$ are each represented as a string of elements j_i which are totally sorted according to $\prec$. Then the relation $mcs_i \prec mcs_l$ holds if an index k exists such that $j_{ip} = j_{lp}$ for $p = 1..(k-1)$ and $j_{ik} \prec j_{lk}$. Considering the previous example, if we choose a value $\delta = 1$ (implying that we want to sample MCSs with maximal cardinality $m_P + 1$; in this case since $m_P = 2$, maximal cardinality $2 + 1 = 3$), the order imposed on the activities in P induces the following lexicographical order within the set of MCSs contained in $M_0 \cup M_1$: $\{j_1[5], j_2[3]\} \prec \{j_1[5], j_3[3]\} \prec \{j_1[5], j_4[2], j_5[1]\} \prec \{j_1[5], j_4[2], j_6[1]\} \prec \{j_1[5], j_4[2], j_7[1]\} \prec \{j_2[3], j_3[3], j_4[2]\}$. Observe that the first elements in the sorted order are those MCSs which contain the fewest activities, and hence are likely to be good candidates as critical conflicts.

To lexicographically sample MCSs within a given peak P, a Depth-First Search procedure is iteratively applied until either all elements in the set $M_0 \cup M_1 \cup \cdots \cup M_\delta$ have been sampled or a maximum number of $s_f\,|P|$ elements have been collected. The linear (with respect to the peak's cardinality) upper bound $s_f\,|P|$ on the number of elements sampled is enforced as a further hedge against combinatorial explosion of the search.

3.3 Conflict Selection and Removal

Having generated a MCS choice set (or a meta-CSP in our abstract schema) the next step of ESA is to identify one particular MCS for resolution. If at least one resolvable MCS remains outstanding, then the selected MCS is removed by selecting and posting an additional precedence constraint between two of the competing activities in the MCS (Lines 12-14 in Figure 2).

The heuristics that govern both conflict selection and conflict resolution in ESA implement the widely-used principle of selecting first the variable which is most constrained, and setting it to the value that is least constraining. The constrainedness is measured by observing the so-called *temporal flexibility* in the current solution.

Temporal Flexibility. The intuition behind the concept of temporal flexibility is that the greater the number of "temporal positions" that the time variables in a solution may assume with respect to each other (i.e., how many consistent assignments remain in the ground-CSP), the greater the ability to accommodate new ordering constraints. Given two different configurations A and B of the ground-CSP, we assume that the probability of obtaining a complete solution from A is greater than it is from B if A contains a greater amount of "free temporal space". This relatively simple heuristic strategy has proved to be quite effective in solving several planning and scheduling problems (e.g., [10, 20, 27, 7]). Alternative strategies might be considered for prioritizing the resolution of outstanding MCSs (e.g., solving MCSs chronologically). We have chosen to use a criticality estimator based on temporal flexibility because it is naturally suited with the constraint-satisfaction problem solving style.

Variable and value ordering. As indicated above, candidate MCSs are ordered according to the temporal flexibility they contain (a function of the degree to which constituent activities can be reciprocally shifted in time). The less flexibility a MCS has, the more critical it is to resolve first. The greater the flexibility that is retained after posting a precedence constraint that resolves a MCS, the more desirable it is to post that constraint. Note that the use of temporal flexibility as a measure of criticality and choice criteria complements the heuristic bias used to drive the MCS sampling process (i.e., toward generating MCSs with few resolution alternatives).

To quantify the notion of temporal flexibility, the heuristic estimator K suggested in [20] is used. Given a candidate MCS and a set $\{pc_1 \ldots pc_k\}$ of precedence constraints that could be posted between pairs of activities in the MCS, $K(\text{MCS})$ is defined as $K(\text{MCS})^{-1} = \sum_{i=1}^{k}(1 + commit(pc_i) - commit(pc_{min}))^{-1}$ where $commit(pc_i)$ ranges from 0 to 1 and estimates the loss in temporal flexibility as a result of posting constraint pc_i, and pc_{min} is the precedence constraint with the minimum value of $commit(pc)$. Note that $K(\text{MCS})$ takes on its highest value of 1 in those cases where only one specific precedence constraint can be feasibly posted to resolve the conflict. In general, the closer an MCS is to being unresolvable, the higher the value of $K(\text{MCS})$. It is worth noting that an MCS that is "close to a unresolvable state" is one for which very few consistent activity start-time assignments remain (relative to other MCSs), so it represents a critical case to solve first. When choosing which MCS to solve next, we focus on the area closest to failure (i.e., the conflict selection heuristic `SelectConflict` chooses the MCS with the highest K value). The rationale here is that if the solution becomes further constrained by resolving a more temporally flexible MCS, the probability increases that less temporally flexible MCSs will eventually reach an unresolvable state. Opposite reasoning applies in the case of value selection (which pair of activities to order and how within selected MCS). In this case we attempt to retain as much temporal flexibility as possible. The conflict resolution heuristic (`SelectPrecedence`) simply chooses the pair with pc_{min} and orders it in a way similar to min-slack heuristics proposed in [31].

4 Iterative Sampling for Better Makespan

The ESA resolution procedure, as defined above, is a deterministic (partial) solution procedure with no recourse in the event that an unresolved conflict is encountered. To provide a capability for expanding the search in such cases without incurring the combinatorial overhead of a conventional backtracking search, we define a random counterpart of our conflict selection heuristic (in the style of [27]) and embed the resulting "`restartESA`" procedure within an iterative sampling search framework. This choice is motivated by the observation that in many cases systematic backtracking search can explore large subtrees without finding any solution. On the other hand, if we compare the whole search tree created by a systematic search algorithm with the non systematic tree explored by repeatedly restarting a randomized search algorithm, we see that the randomized procedure is able to reach "different and distant" leaves in the search tree. This latter property could be an advantage when problem solutions are uniformly distributed within the set of search tree leaves interleaved with large subtrees which do not contain any problem solution.

To make ESA suitable to random restart its function `SelectConflict` is modified according to the following rationale. Recall that an MCS is selected after two steps: (1) a subset of MCSs is sampled from the resource peaks in the current ESS; (2) the MCSs are ranked according to the K estimator and the one with the highest value of K, called K_{max}, is chosen. The estimator K gives a measure of the criticality of a MCS according to the principle of temporal flexibility. However, considering the overall set of MCSs sampled at each solution step, it could well be the case that several MCSs have heuristic evaluations "quite close" to each other and in particular to K_{max}. In such cases, the heuristic information used to compute K is not very selective, and we may consider this subset of elements to be *heuristically equivalent*. This last observation is the basis of our approach to randomization.

The core ESA procedure is transformed into a random procedure by redefining `SelectConflict` to proceed in two-steps: (1) at each solution step, a set of "equivalent" MCSs are first identified, and then (2) one of these MCSs is randomly selected. As in the deterministic variant, the selected MCS is then resolved by posting a precedence constraint. The set of equivalent Ks is created by introducing the parameter $\beta \in [0, 1]$, called *acceptance factor*, and considering as equivalent to K_{max}, all MCSs whose K is within the interval $K_{max}(1 - \beta) \leq K(\text{MCS}) \leq K_{max}$. The conflict to be resolved is randomly selected from this set, resulting in a non-deterministic yet heuristically-biased choice. Successive calls to ESA are intended to explore heuristically equivalent paths through the search space.

Figure 4 depicts the complete iterative sampling algorithm for generating a feasible solution. It is designed simply to invoke the `restartESA` resolution procedure a fixed number (`MaxRestart`) of times, rather than predicating any restarts on a failure to produce a feasible solution. Given that the broader objective in this paper is makespan minimization, each restart provides a new opportunity to produce a different feasible solution with lower makespan.

```
restartESA(Problem, MaxRestart, Horizon)
1.   S_best ← EmptySolution
2.   Sol ← EmptySolution
     /* MK(EmptySolution)=+∞ */
3.   Counter ← 1
4.   While Counter ≤ MaxRestart and MK(Sol) > mk_0 do
5.        Sol ← ESA(Problem, Horizon)
6.        if MK(Sol) < MK(S_best)
7.            then S_best ← Sol
8.        Counter ← Counter + 1
9.   end-while
10.  return(S_best)
11.  end
```

Figure 4: The Restarting ESA Procedure

4.1 The ISES Algorithm

Though the restarting procedure just described does in fact retain the smallest makespan solution generated across calls to `restart`ESA, its principal role is to produce a feasible solution relative to a given upper-bound horizon. In this section, we define an RCPSP/max optimization procedure based on use of this feasible solution generator.

Similar to other CSP procedures for makespan minimization (e.g., [11]), we adopt a multi-pass approach; the makespan of the feasible solution generator is repeatedly applied to solve problems with increasingly smaller temporal horizons, until it is no longer possible to find a feasible solution or until the solution makespan is equal to a known lower-bound of the optimal value mk_{opt}. For example, in the case the algorithm finds a solution with a makespan equal to $\mathtt{mk}_0$ (the makespan of the "infinite capacity" solution to `Problem` which ignores all resource constraints) the search process can immediately stop and return the optimal solution found.

Figure 5 shows the specific multi-pass version of the base ESA procedure we have defined, called ISES (Iterative Sampling Earliest Solutions). ISES is composed of two basic steps. First, a feasible solution is found by invoking `restart`ESA with a horizon value (`MaxH`) much greater than the lower bound $\mathtt{mk}_0$. In other words we use the restart algorithm (Step 2 of Figure 5) to find a quite tight upper-bound of the optimal makespan. We preferred this choice instead of a procedure to estimate an upper bound of the optimal makespan mainly for two motivations: first we have lower upper bound values and second the computational time to find this first horizon is generally negligible with respect to the overall computation time.

Successive calls are then made to `restart`ESA, each time substituting the new best makespan found on the previous call as the new problem horizon. The iteration stops when either (1) a call to `restart`ESA

ISES(Problem, MaxRestart, MaxH)

1. $S_{best} \leftarrow$ EmptySolution
   / find a first feasible solution */*
2. Sol $\leftarrow$ **restartESA**(Problem, MaxRestart, MaxH)
3. **if** Sol $\neq$ EmptySolution **then**
   / improve the current makespan */*
4. **repeat**
5. $S_{best} \leftarrow$ Sol
6. Sol $\leftarrow$ **restartESA**(Problem, MaxRestart, MK(S_{best}))
7. **until** $\mathtt{mk}_0 <$ MK(Sol) $<$ MK(S_{best})
8. **return**(S_{best})
9. **end**

Figure 5: The ISES Optimization Algorithm

returns an empty solution, (2) a lower-bound solution makespan is obtained, or (3) a solution is returned which does not improve the previous best. During an initial tuning phase of the ISES algorithm this "dynamic backward", multi-pass approach was found to outperform alternative schemes where the horizon parameter for successive calls was uniformly varied between established lower and upper bound values. These latter approaches were found to be more expensive computationally without significant improvement in makespan minimization performance.

5 Experimental Evaluation

To evaluate the effectiveness of the ISES algorithm, we consider its performance on two sets of reference problems taken from the RCPSP/max problem repository [2]:

Problem set A. This is the benchmark problem set described in [19]. It consists of three sets of 270 problems each, named $J10$, $J20$ and $J30$, with problems of 10, 20 and 30 activities respectively and 5 resources. The numbers of solvable instance are 187, 184, and 185 respectively. The rest of the problems are provably infeasible.

Problem set B. This is the benchmark problem set introduced in [30] and used by the recent B&B approaches. It consists of 1080 problems with 100 activities and 5 resources, of which 1059 are feasible and the rest provably infeasible.

For both of these problem sets, lower bounds on makespan are known for each problem [18], providing a common reference point for measuring deviation from optimal solutions. In the repository, the current best makespan result for each problem is also available. These results are also used for comparison

[2] Available at `ftp://ftp.wior.uni-karlsruhe.de/pub/ProGen-max/pspmaxlib/`

below, together with performance results obtained with a number of different algorithms reported in the literature.

Both problem sets were generated by PROGEN/max, a flexible random networks generator [29] capable of creating project scheduling problems of varying structure, constrainedness and difficulty. Due to differences in the generation parameters used in each case, there are important differences in the characteristics of the problems in each set. The parameter settings used to generate Problem Set A are closer than Problem Set B's to the settings which produce the "hardest possible" problems [16]. In particular, the problems in Set A exhibit higher levels of resource contention (i.e., higher contention peaks) than those in Set B, along with increased parallelism in project activities (i.e., increased sequencing flexibility). Given these properties, we can expect deeper search trees (i.e., a larger search space) in solving problems from Set A than for problems in Set B, since a greater number of ordering decisions are needed to build up a feasible solution. As we will see, application of the B&B algorithm of [30] provides indirect confirmation of this fact; this algorithm performs comparably on both Problem Set B and the J30 problems of Problem Set A, even though Set B contains larger problem instances. Hence, the problems in Problem Set A are actually very challenging.

5.1 Experimental Design

The algorithms compared to ISES below have been implemented in C/C++ and run on a Pentium 200 with an imposed time limit of 100 seconds per problem. Our current implementation of ISES is in Allegro Common Lisp and the reported results are obtained on a SUN UltraSparc 30 (266MHz). The C++ and Lisp implementations are not directly comparable, but we have nonetheless imposed the same 100 seconds time limit originally used to evaluate the other algorithms in all the experiments.

Results are presented separately for problem Sets A and B. For each set, a two-step experimental analysis has been performed, consisting of a *preliminary phase* to first examine the performance effects of different ISES parameter settings, and an *intensive phase* where the performance of ISES is evaluated more comprehensively using the best ISES parameter settings found. Four parameters were varied during the runs conducted in the preliminary phase: the δ and s_f parameters of the MCS sampling strategy, the β parameter used to randomize the conflict selection heuristic, and N_{rst}, the maximum number of restarts attempted to produce a feasible solution. The maximum horizon `MaxH` was not varied; in all experiments reported in the paper this parameter is set to $5\times$ $\mathtt{mk}_0$. In this way we obtain a sufficiently large horizon to quickly find a first solution. During the subsequent intensive phase, each problem is solved multiple times using different random seeds, to minimize stochastic effects and better calibrate ISES performance. (This is similar to the approach taken in [26] which also evaluates a random restart algorithm.) Two types of results are reported in these experiments: (1) the average, obtained using the average result on each problem over different n runs; (2) the best, obtained using the best result on any single problem

over the n runs.

5.2 Evaluation Criteria

Using the data available from the repository, we compute the following performance measures for purposes of comparative analysis:

- $\Delta_{LB}\%$ – the average relative deviation from the known lower bound. This is the standard baseline performance metric used in the OR literature.
- N_{opt} – the number of optimal solutions found (i.e., solutions that either equal the lower bound or are proved optimal by a B&B);
- N_{feas} – the number of problems solved to feasibility.

We also include two additional metrics that are specific of ISES:

- N_{impr} – the number of solution makespan that improves the current best. This is a value that underscores problems where it performs as the best algorithm.
- CPU – it gives a measure of average computational requirements in seconds on classes of problems.

In the case of the intensive phase experiments, we report average performance as the main result and indicate the best performance between round brackets.

5.3 Problem Set A

For purposes of conducting our preliminary parameter study, we restrict attention to the most difficult subset of problems in Problem Set A, the J30 problems. For these problems, the current best reported results fall on average (8.9%) above lower bound values. Following some initial tuning, we performed a single run of ISES on each of the 270 instances of J30 for each combination of the following parameter values: $N_{rst} \in \{10, 30\}$; $\beta \in \{0.05, 0.1, 0.2, 0.5, 1.0\}$; $\delta \in \{0, 1, 2\}$; $s_f \in \{1, 10\}$. Results are given in Table 1.

Examining these results, we can make several preliminary observations:

- the heuristic ESA strategy coupled with iterative sampling strategy ISES feasibly solves between 184 and 185 problems in all runs, independently of the different settings. This is interesting because 185 coincides with the feasible problems and the B&B approach of [30] solved only 183 problems within 100 seconds.
- Although 10 restarts of ESA obtains good results, it is constantly outperformed by the 30 restart configuration at extra computational expense. We also conducted selected experiments with $N_{rst} =$

Table 1: Set A - preliminary experiments (J30 problem set)

β	δ	s_f	$N_{rst} = 10$				$N_{rst} = 30$			
			$\Delta_{LB}\%$	N_{opt}	N_{feas}	Cpu_{sec}	$\Delta_{LB}\%$	N_{opt}	N_{feas}	Cpu_{sec}
0.05	0	1	12.46	83	184	10.21	12.10	84	184	23.88
	0	10	12.46	83	184	11.57	12.05	85	184	24.68
	1	1	12.41	84	184	12.88	12.15	84	184	26.74
	1	10	12.37	83	184	15.30	12.67	80	184	28.54
	2	1	12.46	84	184	12.69	12.25	82	184	26.85
	2	10	12.33	83	184	15.86	12.41	79	184	28.46
0.1	0	1	11.95	81	185	10.47	11.45	82	185	23.69
	0	10	11.89	82	185	11.51	11.61	85	185	23.51
	1	1	11.90	82	185	12.30	11.64	83	185	26.25
	1	10	12.10	83	185	14.10	11.81	82	185	27.50
	2	1	11.95	82	185	12.15	11.57	83	185	26.78
	2	10	11.92	83	185	15.37	11.96	81	185	28.12
0.2	0	1	11.69	86	185	9.53	11.14	86	184	22.96
	0	10	11.56	81	184	10.52	11.16	90	185	23.87
	1	1	11.70	84	184	11.60	11.21	87	185	25.50
	1	10	11.71	81	184	13.34	11.49	87	185	26.28
	2	1	11.69	82	184	11.61	11.31	87	184	25.14
	2	10	11.67	84	184	13.97	11.44	86	185	27.26
0.5	0	1	11.60	89	185	9.83	11.03	91	185	22.11
	0	10	11.63	88	185	9.34	10.78	90	185	22.73
	1	1	11.61	87	185	10.87	11.01	91	185	23.95
	1	10	11.76	85	185	11.99	11.21	89	185	25.40
	2	1	11.55	86	185	11.14	11.01	89	185	24.22
	2	10	11.60	86	185	12.92	11.26	89	185	25.58
1.0	0	1	11.84	87	185	9.12	11.06	96	185	22.01
	0	10	11.85	83	185	9.27	11.32	93	185	21.83
	1	1	11.90	85	185	11.11	11.26	90	185	24.01
	1	10	12.02	84	185	12.06	11.36	90	185	26.03
	2	1	12.06	86	185	10.99	11.21	94	185	25.14
	2	10	12.03	90	185	12.16	11.50	92	185	26.10

50. However, in these cases, significant further improvement was not obtained, suggesting the existence of a search "saturation point".

- As expected, $\delta = 0$ is the best choice for solution of the RCPSP/max. This confirms the intuition that smaller MCSs containing the activities with highest resource demand are most critical to resolve first, and the utility of focusing variable ordering (i.e., MCS selection) heuristics as narrowly as possible on this set of MCSs.

- The role of the temporal flexibility does not play as big a role in defining MCS criticality as has been observed and described in other domains (e.g., [10, 7]). In fact, the value of β appears to only slightly influence performance. Nevertheless, we observe that too small a value of β significantly degrades performance and the best performance is obtained for β values of 0.5 and 1. Looking for confirmation of this trend suggests the first extensive experiment (below).

- The influence of the sampling factor is indefinite. It seems generally that $s_f = 1$ is sufficient but the absolute best result is obtained for $\beta = 0.5$ and $s_f = 10$. This is another aspect that suggests further analysis.

Given these results and observations, we settle on values of $\delta = 0$ and $N_{rst} = 30$, and undertake a more intensive evaluation of combinations of parameter values for β and s_f across multiple runs of the non-deterministic ISES procedure. Specifically, we apply ISES to each problem in J30 five times (starting from five different random seeds) for each combination of the following parameter values: $s_f \in \{1, 10\}$, and $\beta \in \{0.1, 0.5, 1.0\}$. As indicated above, our goals are twofold:

1. to examine more closely the impact of varying the sampling factor (which in this case will varying the total number of minimum cardinality MCSs that are collected to form the conflict choice set), and

2. to more extensively evaluate the performance impact (or lack thereof) of the "K" estimator, which measures the temporal flexibility associated with an MCS and is used to define the heuristic for choosing which conflict to resolve next. Recall that when $\beta = 0$, strong bias is given to the ordering of candidate MCSs provided by the K estimator, whereas with $\beta = 1$ this heuristic bias is completely disregarded and the choice is random. With $\beta = 0.5$, the heuristic bias is modulated by a fair amount of non-determinism.

The results of these experiments are given in Table 2. Both average and best values obtained are reported (the latter appear in brackets). We also include the number of new "best solutions" obtained (N_{impr}).

We can make the following observations from these extended tests:

- Under all parameter configurations some number of new best solutions are obtained with ISES (N_{impr}), indicating its effectiveness as a RCPSP/max solution procedure.

Table 2: Set A - Intensive Experiments ($\delta = 0$, $N_{rst} = 30$)

β	s_f	$\Delta_{LB}\%$	N_{opt}	N_{feas}	N_{impr}	Cpu_{sec}
0.1	1	11.58 (11.06)	83.60 (85)	185.0 (185)	1.40 (3)	23.74
	10	11.52 (11.10)	84.4 (88)	185.0 (185)	1.80 (2)	23.64
0.5	1	10.99 (10.37)	91.4 (96)	185.0 (185)	2.4 (6)	22.68
	10	11.00 (10.24)	91.0 (97)	185.0 (185)	2.20 (5)	25.57
1	1	11.12 (10.27)	93.0 (101)	185.0 (185)	2.60 (5)	21.24
	10	11.15 (10.34)	92.60 (101)	185.0 (185)	2.20 (3)	21.06

- The role of β and in turn the utility of temporal flexibility as a basis for estimating MCS criticality is confirmed. Strong reliance on K ($\beta = 0.1$) results in too narrow of a search and restricts the algorithms ability to find good (i.e., lower makespan) solutions.

- Nevertheless, the K ordering does appear to provide some positive effect when coupled with a sufficient amount of non-determinism. For $\beta = 0.5$, the lowest average results and the largest number of new best solutions are obtained.

- The lowest best results are obtained alternatively with $\beta = 1$. Thus it seems that more broad-based exploration of the MCS conflict set produced by the heuristic sampling strategy ultimately provides better opportunities for lower makespan solutions but with higher variance in expected results. Some reliance on the K estimator increases the consistency of search results while potentially decreasing the reachability of some lower makespan solutions.

- There is an interesting observation (even if not so strong effect): the value β influences in different ways the average and best values. With higher value of β we observe a larger "scattering" of the makespan values around the average value, this fact can be deduced from the small values of the best $\Delta_{LB}\%$ in the case of $\beta \in \{0.5, 1\}$.

- The sampling factor s_f appears in fact to have no strong effect. The strong, single-run results obtained earlier for $\beta = 0.5$ and $s_f = 10$ do not persist when several runs are considered and appear to be simply a stochastic effect.

On the basis of these results, we determine the best configuration of parameter values for ISES to be: $\delta = 0$, $N_{rst} = 30$, $\beta = 0.5$, $s_f = 1$, and we use this configuration to evaluate ISES performance on the entire Problem Set A. Table 3 gives the results obtained by ISES with this configuration, along with (1) results previously obtained using the B&B approach of [30] (labeled $B\&B_{S98}$)[3], and (2) the current

[3] It is worth reminding that the B&B approaches here and in all the results in the paper are truncated at 100 seconds and as a consequence are not working as complete methods. This is common practice in the reported OR literature.

best solutions (labeled *C-BEST*), which, besides results of [30], also includes some results obtained with other, extended-computation, meta-heuristic solution approaches. It can be noted that:

- ISES shows a regular ability to solve the problems in the whole set (with the single exception of a single instance of J10). This again confirms the power of the heuristic conflict analysis performed by the basic strategy.
- As expected, the exact $B\&B_{S98}$ procedure finds better results than ISES on the smaller J10 and J20 instances. However on the larger J30 set, ISES begins to produce competitive results within specified time bounds, and finds some new best known solutions. On average, ISES solutions to the J30 problems are seen to be within 1.5% of those generated by $B\&B_{S98}$ from the standpoint of deviation from the lower bound.

Table 3: Set A (5 runs, 100 secs, $\delta = 0$, $\beta = 0.5$, $s_f = 1$)

Set	Algorithm	$\Delta_{LB}\%$	N_{opt}	N_{feas}	N_{impr}	Cpu_{sec}
J10	*ISES*	1.26 (0.99)	159.0 (164)	186.0 (186)	0	0.71
	$B\&B_{S98}$	0.0	187	187	–	–
	C-BEST	0.0	187	187	–	–
J20	*ISES*	5.37 (4.99)	117.8 (122)	184.0 (184)	0	4.48
	$B\&B_{S98}$	4.29	157	184	–	–
	C-BEST	3.97	158	184	–	–
J30	*ISES*	10.99 (10.37)	91.4 (96)	185.0 (185)	2.40 (6)	22.68
	$B\&B_{S98}$	9.56	115	183	–	–
	C-BEST	8.91	120	185	–	–

On balance, these results suggest that for RCPSP/max problems characterized by high contention (and hence exhibiting a large search space), ISES offers a scalable alternative to exact B&B approaches. As additional confirmation of this fact, it was found that the number of problems for which ISES generates lower makespan solutions than $B\&B_{S98}$ to steadily increase with the dimension of the problem (from 0 on J10 problems, to 7 on J20 problems, to 28 on J30 problems).

5.4 Problem Set B

The test set B consists of 1080 problems with 100 activities and 5 resources, of which 1059 are feasible and the rest are provably infeasible. We draw on the experience gained on set A and fix $\delta = 0$ for all experiments. As before, we first conduct a preliminary analysis of parameter settings, where all results are obtained with a single run of ISES. The results of this preliminary analysis are given in Table 4.
One clear difference in the results obtained for Problem Set B is the effect of the number of restarts on solution quality. In fact, results for $N_{rst} = 10$ are consistently better than $N_{rst} = 30$. This behavior

Table 4: Set B - Preliminary experiments ($\delta = 0$)

β	s_f	$N_{rst} = 10$				$N_{rst} = 30$			
		$\Delta_{LB}\%$	N_{opt}	N_{feas}	Cpu_{sec}	$\Delta_{LB}\%$	N_{opt}	N_{feas}	Cpu_{sec}
0.1	1	7.99	667	1056	40.31	8.35	659	1058	49.52
	10	8.03	662	1057	41.21	8.34	657	1057	49.40
0.5	1	7.98	671	1057	40.89	8.34	661	1059	49.04
	10	8.00	670	1057	40.25	8.31	662	1058	48.88
1.0	1	8.16	668	1057	41.48	8.53	655	1058	49.18
	10	8.31	661	1058	41.56	8.47	660	1057	49.30

is a direct consequence of the 100 second bound on solution time, which is quite severe for the current ISES implementation (due primarily to the cost of temporal propagation in the 100 activities problem). We notice again a limited influence of the sampling factor, and decide to perform intensive, multi-run experimental analysis with the following settings: $N_{rst} = 10$, $\beta = 0.5$, $\delta = 0$ and $s_f = 1$.

Table 5 reports the intensive experimental analysis of performance results for ISES on Problem set B using the same metrics originally used in the B&B study of [30]. We compare the performance of ISES with all recently reported branch and bound approaches for RCPSP/max, including those of [13] (labeled $B\&B_{dRH}$), [21] (labeled $B\&B_{M98}$), [30] (labeled $B\&B_{S98}$), and [15] (labeled $B\&B_{D98}$). We also include the previously summarized heuristic procedure of [16] (labeled PR^{best}_{FN98}), until now the best non systematic solution to RCPSP/max. In the case of PR^{best}_{FN98}, no decomposition strategy/priority rule pair was found to outperform all others on all problems, so we include in Table 5 the results obtained by using the best performing decomposition scheme and, for each problem, taking the best solution found by 10 priority rules.

Table 5: Set B - Comparison with five independent runs

1080 Problems	$\Delta_{LB}\%$	N_{opt}	N_{feas}
ISES	7.95 (7.34)	669.8 (683)	1057.64 (1059)
$B\&B_{dRH}$	–	606	1009
$B\&B_{M98}$	10.30	701	1023
PR^{best}_{FN98}	8.00	613	1053
$B\&B_{S98}$	7.04	675	1059
$B\&B_{D98}$	4.40	769	1059
C-Best	4.09	781	1059

Same observations follow from Table 5:

- ISES is one of three approaches that is able to find all feasible solutions (98.06%) within the 100 sec-

ond time bound. In this regard, ISES outperforms PR_{FN98}, previously the best heuristic procedure known for RCPSP/max. In fact, ISES also appears to be more robust than PR_{FN98}; considering the 5 runs performed with ISES individually, no more than 3 feasible solutions were ever missed on a single run.

- all B&B approaches except $B\&B_{dRH}$ find higher percentages of optimal solutions than ISES. However, with regard to deviation from lower bound solutions Δ_{LB}% ISES ranks third behind $B\&B_{D98}$ and $B\&B_{S98}$, and is in fact fairly comparable to $B\&B_{S98}$ with a 100 second time limit. (Note that Δ_{LB}% is influenced by the number of feasible solutions found; solving the more difficult problems typically increases the deviation [15]).

The clearly dominating procedure on Problem Set B is $B\&B_{D98}$ of [15]. It is interesting to note that this approach exploits resource constraint propagation rules that could be straightforwardly added to ISES. We are currently investigating this possibility. In fact we do not necessarily consider our complete procedure as the best total solution; but instead view our work as contributing a "heuristic search module" which might be used in conjunction with a B&B framework such as [15] or [30] or even within an alternative search framework such as Limited Discrepancy Search (LDS) [17].

Considering again the differences between Problem Sets A and B, we observe the different average deviation results obtained by the B&B approach of [30]: 9.56% on Set A, 7.04% on Set B. These results reinforce the suggestion that Set A is of higher difficulty than Set B, and hence that, despite the larger problem size in terms of number of activities, the search spaces of Set B instances may not be of sufficient size to stress the B&B approaches. In any event, ISES can be seen to provide a strong heuristic solution.

6 Conclusions

In this paper, we have investigated the use of an iterative sampling procedure to solve RCPSP/max, a complex optimization problem. The ISES procedure uses a combination of constraint-guided and randomized search. At its core, a greedy feasible solution generator is formulated according to a constraint-posting CSP model, which acts to iteratively transform an initial time-feasible solution into one that is also resource-feasible. Within this procedure, analysis of "minimal critical sets" (MCSs) is used to identify where additional ordering constraints are required to avoid resource contention, and MCS analysis is integrated with principles related to preservation of temporal flexibility to provide heuristics for focusing the search. One key to the effectiveness of this resolution procedure is its use of an approximate procedure for computing outstanding MCSs, which avoids the exponential computation of systematic enumeration and identifies "critical" MCSs at a low polynomial cost.

Randomization is introduced into the core procedure in a way that incorporates the bias of these greedy search heuristics. This provides a basis for smoothing the decisions of the deterministic algorithm and

for broadening the search to include heuristically equivalent search paths in the space. An encompassing, iterative restarting search framework is defined to capitalize on these opportunities and enhance the probability of finding better quality solutions.

ISES was compared with the best existing approaches to RCPSP/max. In this comparison, ISES was found to outperform several systematic and heuristic approaches on different reference problem sets. ISES was shown to exhibit the best comparative performance in problems exhibiting heavy resource contention, where the search space is quite large and becomes a serious obstacle for even the best systematic approaches. As problem size was increased in these types of problems, a clearly favorable performance trend was observed in terms of both the number and the quality of solutions produced by ISES, indicating its potential as a scalable, heuristic solution procedure. It is interesting to note that the effectiveness of ISES derives principally from its composite search strategy and heuristics, rather than on a set of propagation rules for early pruning of the search space. The addition of the latter could be an interesting direction for future research.

Acknowledgments

Authors would like to thank Dr.Christoph Schwindt for creating the repository that made possible the experimental comparison described in this work and for giving prompt clarifications on its use. We also thank the various authors who have made available their updated unpublished works on this topic. A special thank to the JoH special issue reviewers for their constructive comments. 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. Stephen F. Smith's work has been sponsored in part by the US Department of Defense Advanced Research Projects Agency under contract F30602-97-20227, and by the CMU Robotics Institute.

References

[1] P. Baptiste and C. Le Pape. A Theoretical and Experimental Comparison of Constraint Propagation Trchniques for Disjunctive Scheduling. In *Proceedings of the* 14^{th} *International Joint Conference on Artificial Intelligence*, 1995.

[2] P. Baptiste, C. Le Pape, and W. Nuijten. Satifiability Tests and Time-Bound Adjustments for Cumulative Scheduling Problems. Technical report, Univerity of Compiégnie, 1997. to appear in Annals of Operations Research.

[3] M. Bartusch, R. H. Mohring, and F. J. Radermacher. Scheduling Project Networks with Resource Constraints and Time Windows. *Annals of Operations Research*, 16:201–240, 1988.

[4] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operations Research*, 1998. to appear.

[5] Y. Caseau and F. Laburthe. Improved CLP Scheduling with Task Intervals. In P. V. Hentenryck, editor, *Logic Programming, Proceedings of the Eleventh International Conference on Logic Programming*, pages 369–383, Santa Margherita Ligure, Italy, 1994. MIT Press.

[6] 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.

[7] A. Cesta, A. Oddi, and S. 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.

[8] A. Cesta, A. Oddi, and S. Smith. Scheduling Multi-Capacitated Resources under Complex Temporal Constraints. Technical Report CMU-RI-TR-98-17, Robotics Institute, Carnegie Mellon University, 1998.

[9] A. Cesta, A. Oddi, and S. Smith. An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows. In *Proceedings of the* 16^{th} *Int. Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.

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

[11] C. Cheng and S. Smith. Applying Constraint Satisfaction Techniques to Job Shop Scheduling. *Annals of Operations Research*, 70:327–357, 1997.

[12] J. Crawford. An Approach to Resource Constrained Project Scheduling. In *Proceedings of the 1996 Artificial Intelligence and Manufacturing Research Planning Workshop*, 1996.

[13] B. De Reyck and W. Herroelen. A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations. *European Journal of Operations Research*, 111(1):152–174, 1998.

[14] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[15] U. Dorndorf, E. Pesch, and T. Phan Huy. A Time-Oriented Branch-and-Bound Algorithm for Resource-Constrained Project Scheduling with Generalized Precedence Relations. Technical report, University of Bonn, Faculty of Economics, 1998.

[16] B. Franck and K. Neumann. Resource Constrained Project Scheduling Problems with Time Windows – Structural Questions and Priority-Rule Methods. Technical Report WIOR-492, Universität Karlsruhe, 1998. (Revised November 1998).

[17] W. Harvey and M. Ginsberg. Limited Discrepancy Search. In *Proceedings of the* 14^{th} *Int. Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

[18] R. Heilmann and C. Schwindt. Lower Bounds for RCPSP/max. Technical Report WIOR-511, Universität Karlsruhe, 1997.

[19] R. Kolisch, C. Schwindt, and A. Sprecher. Benchmark Instances for Project Scheduling Problems. In J. Weglarz, editor, *Handbook on Recent Advances in Project Scheduling*. Kluwer, 1998.

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

[21] R. Möhring, F. Stork, and M. Uetz. Resource Constrained Project Scheduling with Time Windows: A Branching Scheme Based on Dynamic Release Dates. Technical Report 596/1998, Fachbereich Mathematick, Technische Universität Berlin, 1998.

[22] U. Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Sciences*, 7:95–132, 1974.

[23] K. Neumann and C. Schwindt. Activity-on-Node Networks with Minimal and Maximal Time Lags and Their Application to Make-to-Order Production. *Operation Research Spektrum*, 19:205–217, 1997.

[24] K. Neumann and J. Zhan. Heuristics for the minimum project-duration problem with minimal and maximal time lags under fixed resource constraints. *Journal of Intelligent Manufacturing*, 6:145–154, 1995.

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

[26] W. Nuijten and C. Le Pape. Constraint-Based Job Shop Scheduling with ILOG-SCHEDULER. *Journal of Heuristics*, 3:271–286, 1998.

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

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

[29] C. Schwindt. Generation of Resource Constrained Project Scheduling Problems with Minimal and Maximal Time Lags. Technical Report WIOR-489, Universität Karlsruhe, 1996.

[30] C. Schwindt. A Branch and Bound Algorithm for the Resource-Constrained Project Duration Problem Subject to Temporal Constraints. Technical Report WIOR-544, Universität Karlsruhe, 1998.

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