Completable Partial Solutions in Constraint Programming and Constraint-Based Scheduling

András Kovács[1] and József Váncza[2]

[1] Budapest University of Technology and Economics
Magyar tudósok körútja 2/d, 1117 Budapest, Hungary
akovacs@mit.bme.hu

[2] Computer and Automation Research Institute
Kende utca 13-17, 1111 Budapest, Hungary
vancza@sztaki.hu

Abstract. The paper introduces the notion of freely completable partial solutions to characterize constraint satisfaction problems that have components which are relatively easy to solve and are only loosely connected to the remaining parts of the problem. Discovering such partial solutions during the solution process can result in strongly pruned search trees. We give a general definition of freely completable partial solutions, and then apply it to resource-constrained project scheduling. In this domain, we suggest a heuristic algorithm that is able to construct freely completable partial schedules. The method – together with symmetry breaking applied before search – has been successfully tested on real-life resource-constrained project scheduling problems containing up to 2000 tasks.

1 Introduction

In this paper we address the problem of exploiting certain structural properties of constraint satisfaction problems in the course of the solution process. We suggest a method that looks for such a binding of a subset of the variables, which does not constrain the domain of the remaining variables in any way. This kind of bindings is called a *freely completable partial solution.*

Broadly speaking, freely completable partial solutions are traits of such constraint satisfaction problems (CSPs) that have some components which are relatively easy to solve and are only loosely connected to the remaining parts of the problem. Once detected, these partial solutions can be exploited well during the search for solutions: decisions in the easy-to-solve component of the problem can be eliminated, and search can be focused to making the relevant decisions only.

With pruning the search tree, the method may exclude even all but one solutions. In this way, it is closely related to *symmetry breaking*, except that our approach treats *all* solutions equivalent and does not necessitate the explicit declaration of symmetry functions. It can be applied in satisfiability problems and optimization problems which are solved as a series of satisfiability problems.

Our particular motivation was to improve the efficiency of *constraint-based scheduling* methods. By now, constraint programming provides attractive representation and solution methods for solving complex, real-life scheduling problems. However, even the most advanced systems are often unable to solve large problems – which may include an order of magnitude more tasks than typical benchmarks – to an acceptable range of the optimum. Industrial scheduling problems require rich and large-size models, but, at the same time, they can be simple in the sense that they have a loosely connected structure of easy and hard sub-problems. In a real factory, projects visit resources in sequences more or less determined by the manufacturing technology applied. There are product families, members of which are produced in a similar way, using common resources in the same order, while, on the other way around, different product families often use basically different (though not disjoint) sets of resources. Typically, there are many non-bottleneck resources and non-critical projects as well. Some of these properties (e.g., symmetries) can be detected even at the time of model building, but the problem structure remains hidden and can be discovered only at solution time.

In what follows we first discuss equivalence and consistency preserving transformations of CSPs. After summing up related works, a general definition of freely completable partial solutions is given in Sect. 3. Then we shortly present our approach to solving resource-constrained scheduling problems, give a problem-specific definition of freely completable partial schedules and propose a heuristic algorithm to construct partial solutions with such a property. Next we describe how we break symmetries in scheduling problems. Sect. 6. evaluates computational experiments and gives a comparative analysis of constraint-based scheduling algorithms that run on industrial, large-size problem instances without and with the suggested extensions. Finally, conclusions are drawn.

2 Transformations of Constraint Problems

Let there be given a constraint satisfaction problem Π as follows. $X = \{x_i\}$ denotes a finite set of *variables*. Each variable x_i can take a value from its *domain* D_i. There is a set of *constraints* C defined on the variables. The set of variables present in the N-ary constraint $c(x_{i_1}, \ldots, x_{i_N}) \in C$, or briefly c, is denoted by $X_c = \{x_{i_1}, \ldots, x_{i_N}\}$. The solution of a constraint program is a binding S of the variables, i.e., $\forall x_i \in X : x_i = v_i^S \in D_i$ such that all the constraints are satisfied, $\forall c \in C : c(v_{i_1}^S, \ldots, v_{i_N}^S) = \mathit{true}$.

The solution process of a constraint satisfaction problem generally consists of a tree search. Constraint programming earns its efficiency from the *transformations* of the constraint problem, such as domain reductions and addition of inferred constraints, performed within the search nodes.

2.1 Preserving Equivalence vs. Consistency

According to the definitions in [1], a transformation $\Pi \Rightarrow \Pi'$ is called *equivalence preserving* if for every binding S of the variables, S is a solution of Π iff it is also a solution of Π'. For example, constraint propagation and shaving preserve equivalence.

However, a wider set of transformations, i.e., the so-called *consistency preserving* transformations are eligible to solve problems when one has to decide only whether Π has a solution or not. A transformation $\Pi \Rightarrow \Pi'$ is defined to be consistency preserving, if it holds that Π' has a solution iff Π has a solution.

Current general purpose constraint solvers perform equivalence preserving transformations. The reason for that is rooted in their modular structure. *Local propagation algorithms* are attached to individual constraints, hence do not have a view of the entire model. They remove only such values from the variables' domains that cannot be part of any solution because violate the given constraint. In contrast, transformations which do not preserve equivalence, remove also values which *can* participate in some of the solutions. Without loosing the chance of finding a solution (or proving infeasibility), this is possible only with an overall, global view of the model.

2.2 Related Work

Recently, several efforts have been made to explore consistency preserving techniques. Typical transformations which preserve consistency, but do not retain equivalence, are the applications of *symmetry breaking techniques* and *dominance rules*.

In symmetry breaking two basic approaches compete. The first adds symmetry breaking constraints to the model before search, see e.g., [9]. For instance, row and column symmetries in matrix models can be eliminated by lexicographical ordering constraints. Other methods, such as the Symmetry Breaking During Search [15], Symmetry Breaking via Dominance Detection [13], or the Symmetry Excluding Search [2] algorithms, prune symmetric branches of the search tree during search. All of these general frameworks require an explicit declaration of the symmetries in the form of symmetry functions or a dominance checker.

In constraint-based scheduling, it is a common technique to apply dominance rules to prune the search tree. A dominance rule defines a property that must be satisfied at least by one of the optimal solutions. Hence, also the application of a dominance rule can be regarded as a transformation that preserves the consistency of the original problem. E.g., in the field of resource constrained project scheduling, two similar dominance rules are suggested in [3, 11] that bind the start time of a task to the earliest possible value if its predecessors are already processed and the given resource is not required by any other task at that time. Note that this assignment can also be seen as a freely completable solution. A dominance rule to decompose the scheduling problem over time is described in [3]. More complex – and more expensive – dominance rules are discussed by [12]. Several dominance rules as well as rules for the insertion of redundant precedence constraints are proposed for the problem of minimizing the number of late jobs on a single machine, see [5].

Early *solution synthesis* techniques of constraint solving can be regarded as precursors of our proposed method [20]. For example, [14] presents a synthesis algorithm that incrementally builds lattices representing partial solutions for one, two, etc. variables, until a complete solution is found. However, synthesis methods were aimed at finding complete solutions of the problem by themselves whereas we are content with constructing partial solutions that are freely completable.

A basically different approach to exploiting problem structure is by using branching strategies which identify and resolve the most critical subproblems in a CSP. In [7], various search heuristics are presented that extract information from the constraint-based model of job-shop scheduling problems (hence, they are referred to as *textures*) to drive the search heuristic. Similar, so-called *profile-based* analysis methods are suggested in [8] that are tailored to cumulative resource models. Alternatively, a clique-based approach is proposed by [18] to find those subsets of connected activities whose resource requirements can produce a conflict. The above approaches are in common that they point out – resembling the decision method of human schedulers – the most critical resources and/or activities time and again during the solution process.

3 Freely Completable Partial Solutions

In what follows we suggest a framework which performs consistency preserving transformations on structured constraint satisfaction problems by binding a subset of the variables. This binding is selected so that it does not constrain in any way the domains of the remaining variables. We call this kind of partial solutions freely completable, and characterize them formally as follows.

A partial solution PS is a binding of a subset $X^{PS} \subseteq X$ of the variables, $\forall x_i \in X^{PS} : x_i = v_i^{PS}$. We define PS freely completable, iff for each constraint $c \in C$:

- If $X_c \subseteq X^{PS}$, then $c(v_{i_1}^{PS}, \ldots, v_{i_N}^{PS}) = true$, i.e., c is satisfied.
- If $X_c \nsubseteq X^{PS} \wedge X_c \cap X^{PS} \neq \emptyset$, then let $D'_{i_k} = \{v_{i_k}^{PS}\}$ for $x_{i_k} \in X^{PS}$, and $D'_{i_k} = D_{i_k}$ for $x_{i_k} \notin X^{PS}$. Then, $\forall (u_{i_1}, \ldots, u_{i_N}) \in D'_{i_1} \times \ldots \times D'_{i_N}$: $c(u_{i_1}, \ldots, u_{i_N}) = true$. Note that this means that *all the possible* bindings of the variables not included in PS lead to the satisfaction of c.
- If $X_c \cap X^{PS} = \emptyset$, then we make no restrictions.

Proposition 1: If PS is a freely completable partial solution, then binding the variables $x_i \in X^{PS}$ to the values v_i^{PS}, respectively, is a consistency preserving transformation.

Proof: Suppose that there exists a solution S of the constraint program. Then, the preconditions in the above definition prescribe that the binding $x_i \in X^{PS}$: $x_i = v_i^{PS}, x_i \notin X^{PS} : x_i = v_i^S$ is also solution, because every constraint is satisfied in it. On the other hand, it is trivial that any solution of the transformed problem is a solution of the original problem, too. □

Note that whether a partial solution is freely completable or not, depends on *all* the constraints present in the model. In case of an optimization problem, this includes the constraints posted on the objective value as well. Thus, this transformation can not be applied e.g., within a branch and bound search, where such constraints are added during the search process.

A freely completable partial solution PS, apart from the trivial $X^{PS} = \emptyset$ case, does not necessary exist for constraint satisfaction problems, or it can be difficult to find. Notwithstanding, we claim that in structured, practical problems, fast and simple heuristics are often capable to generate such a PS. In what follows, this will be demonstrated for the case of constraint-based scheduling.

4 An Application in Constraint-Based Scheduling

We applied the above framework to solve *resource constrained project scheduling* problems. For that purpose, a commercial constraint-based scheduler [16] was extended by a heuristic algorithm for finding freely completable partial solutions during the search process. In addition, potential symmetries of similar projects were excluded by adding symmetry breaking constraints *before* search.

4.1 Problem Statement and Solution Approach

The scheduling problems are defined as follows. There is a set of tasks T to be processed on a set of cumulative resources R. Capacity of the resource $r \in R$ is denoted by $q(r) \in \mathbb{Z}^+$. Each task $t \in T$ has a fixed duration $d(t)$ and requires one unit of resource $r(t)$ during the whole length of its execution, without preemption. Tasks can be arbitrarily connected by end-to-start and start-to-start precedence constraints. These will be denoted by $(t_1 \rightarrow t_2)$ and $(t_1 \dashrightarrow t_2)$, respectively, and determine a directed acyclic graph of the tasks together. The objective is to find start times $start(t)$ for the tasks such that all the precedence and resource capacity constraints are observed and the makespan, i.e., the maximum of the tasks' end times, $end(t) = start(t) + d(t)$ is minimal.

We solve this constrained optimization problem as a series of satisfiability problems in the course of a *dichotomic search*. In successive search runs, the feasibility of the problem is checked for different trial values of the makespan. If UB is the smallest value of the makespan for which a solution is known and LB is the lowest value for which infeasibility has not been proven, then the trial value $\lfloor (UB + LB)/2 \rfloor$ is probed next. Then, depending on the outcome of the trial, either the value of UB or LB is updated. This step is iterated until the time limit is hit or $UB = LB$ is reached, which means that an optimal solution has been found.

Within each search run, the initial time window of each task $t \in T$, limited by its earliest start time $est(t)$ and latest finish time $lft(t)$, equals the interval from time 0 to the trial value of the makespan. In the constraint-based representation of the problem, one variable $start(t)$ stands for the start time of each task $t \in T$. The initial domain of $start(t)$ is the interval $[est(t), lft(t) - d(t)]$. These domains are later tightened by the propagators of the precedence and resource capacity constraints. For propagating precedence constraints, an *arc-B-consistency* algorithm, while for resource capacity constraints the *edge-finding* algorithm is applied [4].

During the search, we build schedules chronologically using the so-called *settimes* strategy [16]. This relies on the LFT priority rule [10], which works as follows. It selects the earliest time instant τ for which there exists a non-empty set $T_\tau \subseteq T$ of unscheduled tasks that can be started at time τ. A task $t \in T$ belongs to T_τ iff all its end-to-start predecessors have ended and all its start-to-start predecessors have started by τ, and there is at least one unit of resource $r(t)$ free in the interval $[\tau, \tau + d(t)]$. From T_τ, the task t^* with the smallest latest finish time $lft(t^*)$ is selected. The settimes branching algorithm then generates two sons of the current search node, according to the decisions whether $start(t^*)$ is bound to $est(t^*)$, or t^* is postponed.

4.2 Freely Completable Partial Schedules

A partial solution PS of a scheduling problem, i.e., a *partial schedule*, is a binding of the start time variables $start(t)$ of a subset of the tasks, which will be denoted by $T^{PS} \subseteq T$. According to the previous definitions, PS is called freely completable, if the following conditions hold for each constraint of the model.

For end-to-start precedence constraints $c : (t_1 \rightarrow t_2)$,

- $t_1, t_2 \in T^{PS}$ and $end(t_1) \leq start(t_2)$, i.e., c is satisfied, or
- $t_1 \in T^{PS}, t_2 \notin T^{PS}$ and $end(t_1) \leq est(t_2)$, i.e., c is satisfied irrespective of the value of $start(t_2)$, or
- $t_1 \notin T^{PS}, t_2 \in T^{PS}$ and $lft(t_1) \leq start(t_2)$, i.e., c is satisfied irrespective of the value of $start(t_1)$, or
- $t_1, t_2 \notin T^{PS}$, i.e., PS does not make any commitments on the start times of t_1 and t_2.

This definition can be extended to start-to-start precedence constraints $c : (t_1 \dashrightarrow t_2)$ likewise:

- $t_1, t_2 \in T^{PS}$ and $start(t_1) \leq start(t_2)$, or
- $t_1 \in T^{PS}, t_2 \notin T^{PS}$ and $start(t_1) \leq est(t_2)$, or
- $t_1 \notin T^{PS}, t_2 \in T^{PS}$ and $lft(t_1) - d(t_1) \leq start(t_2)$, or
- $t_1, t_2 \notin T^{PS}$.

To check resource capacity constraints, we define $M^+_{r,\tau}$ as the set of tasks $t \in T^{PS}$ which are under execution at time τ on resource r, while $M^-_{r,\tau}$ as the set of tasks $t \notin T^{PS}$ which *might be* under execution at the same time:

$$M^+_{r,\tau} = \{t | t \in T^{PS} \wedge r(t) = r \wedge (start(t) \leq \tau \leq end(t))\}$$
$$M^-_{r,\tau} = \{t | t \notin T^{PS} \wedge r(t) = r \wedge (est(t) \leq \tau \leq lft(t))\}$$

Now, one of the followings must hold for every resource $r \in R$ and for every time unit τ:

- $|M^+_{r,\tau}| + |M^-_{r,\tau}| \leq q(r)$, i.e., the constraint is satisfied at time τ irrespective of how PS will be complemented to a complete schedule, or
- $M^+_{r,\tau} = \emptyset$, i.e., PS does not make any commitment on r at time τ.

4.3 A Heuristic Algorithm

We applied the following heuristic algorithm to construct freely completable partial schedules. The algorithm is run once in each search node, with actual task time windows drawn from the constraint solver.

The method is based on the LFT priority rule-based scheduling algorithm, which also serves as the origin of the branching strategy. It was modified so that it generates freely completable partial schedules when it is unable to find a consistent complete schedule. The algorithm assigns start times to tasks in a chronological order, according to the priority rule, and adds the processed tasks to T^{PS}.

```
PROCEDURE FindAnyCaseConsistentPS()
    % Let $U$ be the set of tasks not yet scheduled.
    $U := \{t | t \in T : start(t)$ is not bound$\}$
    WHILE ($U \neq \emptyset$)
      Choose a task $t \in U$ and a start time $\tau$ using the LFT rule;
      Remove $t$ from $U$;
      IF $\tau + d(t) \leq lft(t)$ THEN
        $start(t) := \tau$;
        Add $t$ to $T^{PS}$
      ELSE
        FailOnTask($t$);

PROCEDURE FailOnTask($t$)
    IF $t \in T^{PS}$ THEN
      Remove $t$ from $T^{PS}$;
    FORALL task $t' \in T^{PS} : (t' \rightarrow t) \in C$
      IF $end(t') > est(t)$ THEN
        FailOnTask($t'$);
    FORALL task $t' \in T^{PS} : (t' \dashrightarrow t) \in C$
      IF $start(t') > est(t)$ THEN
        FailOnTask($t'$);
    FORALL task $t' \in T^{PS} : r(t') = r(t)$
      % Let $I$ be the time interval in which $t$ and $t'$ can be
      % processed concurrently.
      $I := [start(t'), end(t')] \cap [est(t), lft(t)]$;
      IF $\exists \tau \in I : |M^{+}_{r(t),\tau}| + |M^{-}_{r(t),\tau}| > q(r(t))$ THEN
        FailOnTask($t'$);
```

Fig. 1. The heuristic algorithm for constructing freely completable partial schedules.

Whenever the heuristic happens to assign an obviously infeasible start time to a task t, i.e., $start(t) > lft(t) - d(t)$, t is removed from T^{PS}. The removal is recursively continued on all tasks t' which are linked to t by a precedence or a resource capacity constraint, and the previously determined start time $start(t')$ of which can be incompatible with any value in the domain of $start(t)$. After having processed all the tasks, the algorithm returns with a freely completable partial schedule PS. In the best case, it produces a complete schedule, $T^{PS} = T$, while in the worst case, PS is an empty schedule, $T^{PS} = \emptyset$. The pseudo-code of the algorithm is presented in Fig. 1.

Certainly, this simple heuristic can be improved in many ways. First of all, we applied a small random perturbation on the LFT priority rule. This leads to slightly different runs in successive search nodes, which allows finding freely completable partial solutions which were missed in the ancestor nodes nodes. In experiments (see Sect. 6.), the modified rule, named LFT^{rand}, resulted in roughly 20 % smaller search trees than LFT.

The time spent for building potentially empty partial schedules can be further decreased by restricting the focus of the heuristic to partial schedules PS which obviate the actual branching in the given search node. Task t^*, whose immediate scheduling or postponement is the next search decision in the constraint-based solver, is already known before running the heuristic. This next branching would be eliminated by PS only if $t^* \in T^{PS}$. Otherwise, finding PS does not immediately contribute to decreasing the size of the search tree, and it is likely that PS will only be easier to find later, deeper in the search tree. Accordingly, when `FailOnTask` is called on t^*, the heuristic algorithm can be aborted and an empty schedule returned. These improvements can be realized by replacing one line and adding three lines to the pseudo-code of the basic algorithm, as shown in Fig. 2.

```
1 PROCEDURE FindAnyCaseConsistentPS()
...
5     Choose a task $t \in U$ and a start time $\tau$ using the LFT$^{rand}$ rule;
...

12 PROCEDURE FailOnTask($t$)
12A   IF $t$ is the task on which the branching is anticipated THEN
12B     $T^{PS} := \emptyset$;
12C     EXIT; % The next branching cannot be avoided.
...
```

Fig. 2. Improvements of the heuristic algorithm.

4.4 An Illustrative Example

In the following, an example is presented to demonstrate the working of the heuristic algorithm that constructs freely completable partial schedules. Suppose there are 3 projects, consisting of 8 tasks altogether, to be scheduled on three unary resources. Tasks belonging to the same project are fully ordered by end-to-start precedence constraints. The durations and resource requirements of the tasks are indicated in Fig. 3, together with the time windows received by the heuristic algorithm from the constraint-based solver in the root node of the search tree. The trial value of the makespan is 10.

t	d(t)	est(t)	lft(t)	r(t)
t11	1	0	2	R3
t12	4	1	10	R1
t21	2	0	3	R3
t22	2	2	5	R2
t23	5	4	10	R3
t31	2	0	3	R2
t32	4	2	7	R1
t33	3	6	10	R2

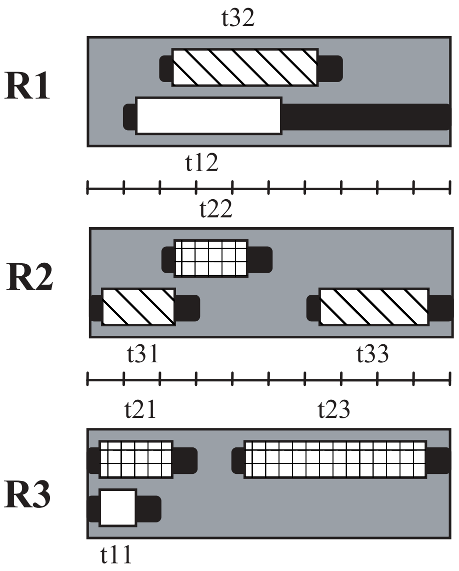

Fig. 3. Parameters of the sample problem.

Note that in order to be able to present a compact but non-trivial example, we switched off the edge-finding resource constraint propagator in the constraint solver engine, and used time-table propagation only.

The algorithm begins by assigning start times to tasks in chronological order, according to the LFT priority rule: $start(t11) = 0$, $start(t31) = 0$, $start(t21) = 1$, $start(t12) = 1$ and $start(t22) = 3$, see Fig. 4.a. All these tasks are added to T^{PS}.

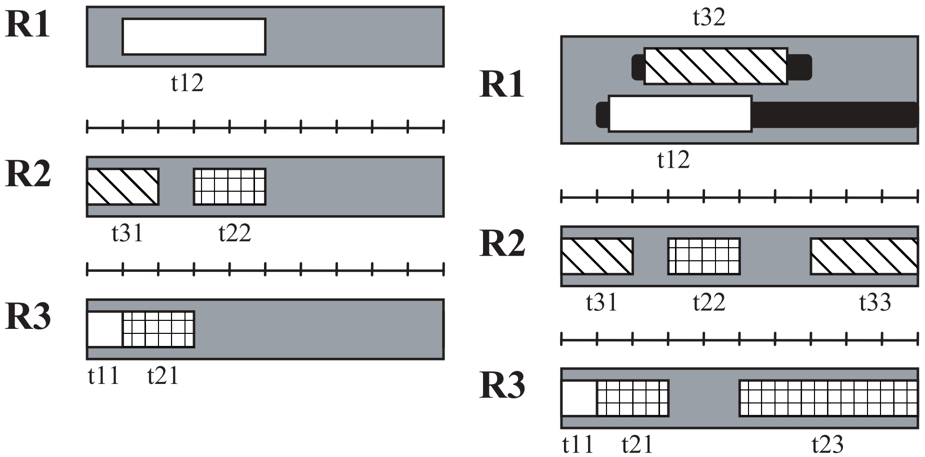

Fig. 4. a.) Building the partial schedule. b.) The freely completable partial schedule.

Now, it is the turn of $t32$. Unfortunately, its execution can start the soonest at time 5, and consequently, it cannot be completed within its time window. Hence, the function `FailOnTask` is called on $t32$, and recursively on all the tasks which could cause this failure. At this example, it only concerns $t12$ which is removed from T^{PS}. Then, further tasks are scheduled according to the LFT priority rule: start times are assigned to the two remaining tasks, $start(t23) = 5$ and $start(t33) = 7$. The heuristic algorithm stops at this point, and it returns the freely completable partial schedule PS with $T^{PS} = \{t11, t21, t22, t23, t31, t33\}$, see Fig. 4.b.

After having bound the start times of these tasks in the constraint-based solver, the solver continues the search process for the remaining two tasks. In the next search node, it infers the only remaining valid start times for $t12$ and $t32$ by propagation. This leads to an optimal solution for this problem, as shown at Fig. 5.

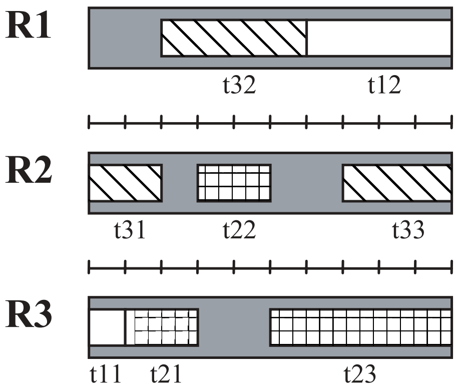

Fig. 5. The final schedule.

5 Breaking the Symmetries Between Similar Projects

In real industrial plants, products can often be ordered into a few number of product families. Members of the same family generally share parts of their routings, which introduces a huge number of symmetries in the scheduling problem. We exclude symmetries of similar projects by adding symmetry breaking constraints to the model *before* search, by using the following method.

Let P and Q denote two *isomorphic* subsets of T. P and Q are considered isomorphic iff their cardinality is the same and their tasks can be indexed such that

$$\begin{aligned} &\forall i \in [1, ..., n] : && d(p_i) = d(q_i) \wedge r(p_i) = r(q_i), \text{ and} \\ &\forall i, j \in [1, ..., n] : && (p_i \rightarrow p_j) \Leftrightarrow (q_i \rightarrow q_j) \wedge (p_i \dashrightarrow p_j) \Leftrightarrow (q_i \dashrightarrow q_j). \end{aligned}$$

Furthermore, suppose that there are no outgoing precedence constraints from P and no incoming precedence constraints to Q.

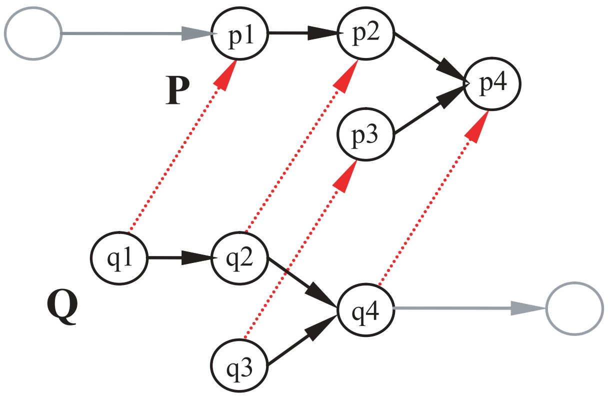

Fig. 6. Symmetry breaking.

Proposition 2: If there exists a solution S of the scheduling problem, then it also has a solution S' which satisfies all the precedence constraints $(q_i \rightarrow p_i)$ if resource $r(p_i)$ is unary, and $(q_i \dashrightarrow p_i)$ if resource $r(p_i)$ has a higher capacity.

Proof: Let us construct the desired solution S' departing from S by swapping each pair of tasks p_i, q_i for which the added precedence constraint is not satisfied:

$$\forall i \in [1, ..., n]: \qquad \begin{aligned} start(p_i)^{S'} &= max(start(p_i)^S, start(q_i)^S), \\ start(q_i)^{S'} &= min(start(p_i)^S, start(q_i)^S). \end{aligned}$$

Now, all resource capacity constraints are satisfied in S', because the durations and resource requirements of p_i and q_i are the same. End-to-start precedence constraints $(p_i \rightarrow p_j)$ cannot be violated in S', either, because

- If neither of the ith or jth pairs of tasks were swapped, then the start times of p_i and p_j are unchanged in S' w.r.t S;
- If only the ith pair of tasks was swapped, then $end(p_i)^{S'} = end(q_i)^S \leq start(q_j)^S \leq start(p_j)^S = start(p_j)^{S'}$;
- If only the jth pair of tasks was swapped, then $end(p_i)^{S'} = end(p_i)^S \leq start(p_j)^S \leq start(q_j)^S = start(p_j)^{S'}$;
- If both of the ith and jth pairs of tasks were swapped, then $end(p_i)^{S'} = end(q_i)^S \leq start(q_j)^S = start(p_j)^{S'}$.

For start-to-start precedence constraints, the proof is analogous. □

Note that by the iterative application of this proposition, an arbitrary number of symmetrical subsets can be fully ordered. In our system, we add precedence constraints to the model according to proposition 2. Thus, P and Q stand for the sections of two projects, which fall into the scheduling horizon, and where the project containing Q is in a slightly more advanced state. These symmetries can easily be found with the help of some appropriate task identifiers.

6 Experiments

The above algorithms were developed and implemented as part of the efforts to improve the efficiency of the job-shop level scheduler module of our integrated production planner and scheduler system [17, 19, 21]. This scheduler unfolds medium-term production plans into detailed schedules on a horizon of one week.

The starting point of the implementation was the constraint-based scheduler of Ilog [16]. It was extended by the symmetry breaker as a pre-processor, and the heuristic algorithm for constructing freely completable partial schedules, run once in each search node. Both extensions were encoded in C++. The experiments were run on a 1.6 GHz Pentium IV computer.

The test problem instances originate from an industrial partner that manufactures mechanical parts of high complexity. The products can be ordered into several product families. A project, aimed at the fabrication of an end product, usually contains 50 to 500 machining, assembly and inspection operations. The precedence relations between the tasks of a project form an in-tree. There are cc. 100 different unary and cumulative resources in the plant.

Four systems participated in the test: DS denotes a dichotomic search using only propagation algorithms of the commercial CP solver. First, it was extended by the symmetry breaker (DS+SB), then by the algorithm for building freely completable partial solutions (DS+FC). In the last system, all components were switched on (DS+SB+FC).

Test runs were performed on two sets of data. Problem set 1 consists of 30 instances received from the industrial partner, each containing from 150 up to 990 tasks. The solution time limit was set to 120 seconds. Even the simplest algorithm, DS could find optimal solutions for all but one problem. The symmetry breaker further improved on its results, but the systems exploiting freely completable partial solutions were the definite winners, thanks to an extremely low number of search nodes. In many cases, including those where the first solution proved to be optimal, these two systems could solve the problems without any search. The results are presented in Table 1, with separate rows for instances which could be solved to optimality (+) and those which could not (–). *Search time* and *search nodes* both include finding the solutions and proving optimality. *Error* is measured by the difference of the best known upper and lower bounds, in the percentage of the lower bound.

A set of 18 larger problem instances – with up to 2021 tasks – was generated by merging several problems from problem set 1. 14 of them were solvable with standard methods of DS. Just like on problem set 1, identifying the freely completable partial solutions of the problems significantly reduced the size of the search tree. The complete system could solve all the problem instances within the time limit. The detailed results are presented in Table 2 for each problem instance[1].

[1] An extended set of problem instances is available online at http://www.mit.bme.hu/~akovacs/projects/fcps/instances.html

Table 1. Results on problem set 1.

Method	Number of instances	Avg. search nodes	Avg. search time (sec)	Avg. Error (%)
DS (+)	29	282.5	2.00	-
DS (–)	1	59073.0	120.00	12.0
DS+SB (+)	30	272.1	1.67	-
DS+FC (+)	30	8.0	0.83	-
DS+SB+FC (+)	30	6.6	0.73	-

Table 2. Results on problem set 2.

Instance	Tasks	DS			DS+SB			DS+FC			DS+SB+FC		
		Nodes	Time (sec)	Error (%)	Nodes	Time (sec)	Error (%)	Nodes	Time (sec)	Error (%)	Nodes	Time (sec)	Error (%)
#1	836	836	14	-	836	11	-	0	8	-	0	0	-
#2	1027	1027	21	-	2054	22	-	0	11	-	0	1	-
#3	1138	2280	27	-	2276	27	-	13	11	-	0	10	-
#4	944	18650	120	7.1	12547	120	4.2	30	14	-	9	8	-
#5	1328	10294	120	11.0	9779	120	4.2	382	120	2.0	9	13	-
#6	639	24991	120	12.2	13785	65	-	2083	120	0.8	8	5	-
#7	1141	12334	120	8.9	2283	32	-	730	120	4.2	137	30	-
#8	994	1988	21	-	1988	22	-	0	7	-	0	8	-
#9	1932	3864	101	-	3857	110	-	0	22	-	0	24	-
#10	1876	3745	99	-	3745	106	-	18	28	-	81	55	-
#11	2021	2021	76	-	2021	82	-	0	30	-	0	33	-
#12	1637	1637	46	-	1637	50	-	0	20	-	0	23	-
#13	1771	1771	53	-	1771	59	-	0	24	-	0	24	-
#14	1337	4004	45	-	1337	27	-	794	112	-	212	32	-
#15	1592	3175	52	-	3184	55	-	525	106	-	0	16	-
#16	1098	1098	32	-	1098	40	-	0	18	-	73	29	-
#17	953	953	22	-	953	28	-	0	14	-	6	13	-
#18	819	819	17	-	811	22	-	0	11	-	0	13	-

The systems were also tested on Lawrence's job-shop benchmark problems la01-la20 [6]. Since these benchmarks basically lack the structural properties of industrial problems that our algorithms exploit, we did not expect the complete system to significantly improve on the performance of the commercial constraint-based scheduler. In fact, it turned out that freely completable partial solutions also exist in these benchmark instances, and our algorithms managed to decrease the size of the search tree by a factor of 7.3 on average, but this reduction did not always return the time invested in the construction of freely completable partial schedules.

7 Conclusions

In this paper we suggested general notions and specialized algorithms to treat constraint satisfaction problems that have relatively easy-to-solve and loosely connected sub-problems in their internal structure. We argued that solutions of such components should be discovered and separated as freely completable partial solutions by consistency preserving transformations.

We made this concept operational in the field of resource-constrained project scheduling. The method was validated on large-size practical scheduling problems, where only a few search decisions really matter. Such problems are hard to solve for pure propagation-based solvers because many search decisions produce equivalent choices. However, by constructing freely completable partial solutions we were able to avoid growing the search tree by branchings on irrelevant search decisions, and thus scheduling problems of large size became tractable.

We are currently extending the approach for other application areas of constraint programming, such as graph coloring. This requires the creation of heuristic algorithms that build freely completable partial solutions for the given problem class.

Acknowledgements

The authors wish to thank András Márkus and Tamás Kis for their valuable comments. This research has been supported by the grant OTKA T046509.

References

1. Apt, K.R.: Principles of Constraint Programming. Cambridge Univ. Press. (2003)
2. Backofen, R., Will, S.: Excluding Symmetries in Constraint-based Search, Constraints 7(3), pp. 333-349 (2002)
3. Baptiste, P., Le Pape, C.: Constraint Propagation and Decomposition Techniques for Highly Disjunctive and Highly Cumulative Project Scheduling Problems. Constraints 5(1/2), pp. 119-139. (2000)
4. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-based Scheduling. Kluwer Academic Publishers. (2001)
5. Baptiste, P., Peridy, L., Pinson, E.: A Branch and Bound to Minimize the Number of Late Jobs on a Single Machine with Release Time Constraints. European Journal of Operational Research 144(1), pp. 1-11. (2003)
6. Beasley, J.E: The OR-Library. http://www.ms.ic.ac.uk/info.html
7. Beck, J.Ch., Fox, M.S.: Dynamic Problem Structure Analysis as a Basis for Constraint-Directed Scheduling Heuristics. Artificial Intelligence 117, pp. 31-81. (2000)
8. Cesta, A., Oddi, A., Smith, S.F.: Profile-Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In Proc. of the 4th International Conference on Artificial Intelligence Planning Systems, pp. 214-223. (1998).
9. Crawford, J., Luks, G., Ginsberg, M., Roy, A.: Symmetry Breaking Predicates for Search Problems. In Proc. of the 5th Int. Conf. on Knowledge Representation and Reasoning, pp. 148-159. (1996)

10. Davis, E., Patterson, J.: A Comparision of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling. Management Science 21, pp. 944-955. (1975)
11. Demeulemeester, E.L., Herroelen, W.S.: A Branch-and-bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem. Management Science 38(12), pp. 1803-1818. (1992)
12. Demeulemeester, E.L., Herroelen, W.S.: Project Scheduling: A Research Handbook. Kluwer Academic Publishers. (2002)
13. Fahle, T., Schamberger, S., Sellmann, M.: Symmetry Breaking. In Principles and Practice of Constraint Programming – CP2001, pp. 93-107. (2001)
14. Freuder, E.C.: Synthesizing Constraint Expressions. Communications ACM 21(11), pp. 958-966. (1978)
15. Gent, I.P., Smith, B.M.: Symmetry Breaking in Constraint Programming. In Proc. of the 14th European Conference on Artificial Intelligence, pp. 599-603. (2000)
16. Ilog Scheduler 5.1 User's Manual. (2001)
17. Kovács, A.: A Novel Approach to Aggregate Scheduling in Project-Oriented Manufacturing. In Proc. of the 13th Int. Conference on Automated Planning and Scheduling, Doctoral Consortium, pp. 63-67. (2003)
18. Laborie, Ph., Ghallab, M.: Planning with Shareable Resource Constraints. In Proc. of the 14th Int. Joint Conference on Artificial Intelligence, pp. 1643-1649. (1995)
19. Márkus, A., Váncza, J., Kis, T., Kovács, A., Project Scheduling Approach to Production Planning, CIRP Annals - Manuf. Techn. 52(1), pp. 359-362. (2003)
20. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press. (1993)
21. Váncza, J., Kis, T., Kovács, A.: Aggregation – The Key to Integrating Production Planning and Scheduling. CIRP Annals - Manuf. Techn. 53(1), pp. 377-380. (2004)