Structural Exploration of Constraint-Based Scheduling Problems

A. Kovács[2] , A. Márkus[1], J. Váncza[1,2]
[1]Computer and Automation Research Institute, Hungarian Academy of Sciences
[2]Budapest University of Technology and Economics, Budapest, Hungary

Abstract
Constraint-based scheduling (CBS) is a highly efficient and widely applied method in solving resource-constrained project scheduling problems – including job-shop scheduling problems. However, many recent studies suggest that practical problems highly differ from benchmark instances used in theoretical scheduling research, generated intentionally to challenge solvers. Although industrial problems require a rich and large-size model, they are simple in the sense that they often have a loosely connected structure of easy and hard subproblems. In this paper, we present how we extended a general CBS framework by methods that exploit these structural properties. Our system was validated on industrial job-shop scheduling problem instances containing up to 2000 tasks, and succeeded in solving problem instances that previously remained unsolvable due to their sheer size.

Keywords:
Scheduling, Constraint programming

1 INTRODUCTION

In this paper, we address detailed **job-shop scheduling** problems of real-life complexity and size. In general, the problem of scheduling is to order competing tasks on finite resources. Beyond satisfying various temporal and resource capacity constraints, the solution should approach optimality with respect to some optimization criterion.

Close-to-optimal solution of scheduling problems requires expressive and flexible models and efficient, customized solution methods. Recently matured techniques for modeling and solving scheduling problems apply the method of **constraint programming** (CP). This new computing paradigm supports declarative representation on the one hand, and enables the efficient integration of reasoning, search and optimization methods on the other hand. Recent advances in **constraint-based scheduling** (CBS) resulted in a rich set of generic modeling tools as well as in a portfolio of robust inference and search methods for handling and solving scheduling problems efficiently [4,10].

However, even the most current systems often fail to solve industrial scheduling problems – which sometimes include an order of magnitude more tasks than typical benchmarks – to an acceptable range of the optimum. In this paper, we suggest that despite their large size, by exploiting their structural characteristics these problems become tractable. Opposed to the widely used benchmarks, in a real factory

- jobs visit resources in a sequence more or less determined by the manufacturing technology applied;
- there are several similar or even equivalent jobs;
- different product families often use basically different (though not disjoint) sets of resources;
- on the other way around, members of the same family are produced in a similar way, using common resources in the same order;
- the factory has many non-bottleneck resources, and
- typically, there are many non-critical jobs, too.

Consequently, real-life problems often have a specific structure of loosely connected, **easy and hard subproblems**. The solutions of some of these subproblems are interchangeable what we regard a sign of **symmetry**.

Our goal is to improve the efficiency of CSP methods. However, we are not interested simply in solving large problem instances. Instead, our focus is set on large scheduling problem instances that come from **real-life applications**. The aim is to detect and exploit the above-mentioned internal structural properties of scheduling problems during the solution process. Solving hard combinatorial optimization problems – like job-shop scheduling – is hopeless without making such structural exploration efforts. Broadly speaking, there is "no free lunch" in optimization: one can hope that the solution algorithm performs better than blind search only if the search operators are correlated to the structural features of the problem at hand [18].

In what follows, we give an overview of constraint-based scheduling in a nutshell. In the next section, our actual scheduling model is defined. In Section 4, we define any-case consistency, a specific structural property of subproblems that are relatively easy to solve. Then we give an algorithm for finding such components of scheduling problems. In Section 6, we define our notion of symmetry and present its role in making the solution process more efficient. Section 7 evaluates computational experiments and gives a comparative analysis of CBS algorithms running on industrial data without and with our suggested extensions. Finally, conclusions are drawn.

2 ISSUES IN CONSTRAINT-BASED SCHEDULING

2.1 Constraint programming

Constraint programming is a declarative programming paradigm for solving combinatorial satisfiability and optimization problems [1]. A constraint-based model consists of **variables** (e.g., start time of tasks), **domains** (possible values) of variables, a set of **constraints** over the variables and an optimization **objective**. The domains are typically finite sets of nonnegative integers. The solution process is aimed at finding those values of the variables (in their corresponding domains) that satisfy all the constraints and are the best according to the given optimization criterion.

The basic idea of solving a constraint program is to reduce it to an equivalent problem from which the solution can be extracted relatively easily. This reduction is made by **constraint propagation**. Propagation is a **destructive** technique: it removes inconsistent values from the domains of the variables, i.e., values that provably cannot constitute a part of a solution. Propagators vary with the types of constraints and the actual notion of consistency. Propagation is executed iteratively, every time the domain of some variable is changed. Since the propagation machinery is incomplete, the solution has to be found by a search process. During search, new, artificial constraints are introduced that divide the original problem into separate alternatives. Search decisions and propagation are interwoven so that propagation can reduce the search space as soon as possible. If the constraint model becomes inconsistent in one of the alternatives, then work continues with the other ones, and – in the last resort – the system backtracks. The search may run for a single solution, for all solutions, or for proving that there is no solution.

Note that in contrast to destructive techniques, one may apply during the solution process **constructive** techniques as well – algorithms that bind variables to provably consistent values.

2.2 Constraint-based scheduling

Constraint-based scheduling (CBS) is one of the most widespread practical applications of CP [4]. In a CBS model, variables are typically the start times of tasks and constraints link tasks, resources and time. Fundamental constraints are the temporal and resource constraints. Temporal constraints are the **precedences** between the tasks (as given e.g., in the routing, or implied by the Bill of Materials of complex products), the **durations** and the **time windows** (earliest start, latest finish times) of the tasks. Further on, machine and human **resource requirements** of the particular tasks should be satisfied by limited resource capacities. The solution is an assignment of starting times to tasks such that all temporal and resource capacity constraints are observed.

CBS applies both the generic propagation mechanisms of CP and domain specific propagators that fit the actual temporal and resource constraints of the scheduling problem.

Duration and precedence constraints of the tasks can be handled together by specific propagators that remove inconsistent values from the tasks' time windows. For instance, if task A is to be executed before task B, then the earliest start time of B should be at most the earliest start time plus the duration of A. Once the time window of a task is reduced, propagation tries to narrow the time windows of all the other tasks that are linked to this task by precedence constraints. The propagators that work on such temporal constraints are in fact versions of the standard, so-called **arc-B-consistency** algorithm of CP [2].

For propagating resource constraints, a widely used method is **edge finding** [2,4]. Given a particular resource and a set of tasks requiring this resource, edge finding tries to deduce which activities must be (or cannot be) scheduled first (or last) in this set. The algorithm investigates time windows where the total demand of tasks exceeds the capacity of the resource. The conclusions drawn are of two types: new precedence constraints are posted between some tasks, and the time windows of some tasks are tightened. Note that the application of edge finding may prompt the further call of temporal propagators and *vice versa*.

2.3 Constructive methods in CP

In the early years of CP, a wide variety of constructive approaches were developed to solve constraint satisfaction problems by themselves alone [16]. For example, [8] presents a synthesis algorithm that incrementally builds lattices representing partial solutions for 1, 2, etc. variables, until a complete solution is found. Later on, these early techniques were completely played down by systems relying on the destructive technique of constraint propagation.

Since that time, numerous application specific constructive algorithms were suggested as extensions to various constraint solver systems. All these methods were aimed at increasing the efficiency of the solvers. E.g., for resource constrained project scheduling, in [2] and [7], two **dominance rules** are suggested for the identification of tasks which can be scheduled before all other unscheduled tasks. A dominance rule to decompose the scheduling problem over time is also described, see [2]. In [5], 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.

In **symmetry breaking** two basic approaches compete. The first adds symmetry breaking constraints to the model before search, see e.g. [6]. For instance, row and column symmetries in matrix models can be eliminated by lexicographical ordering constraints. Other methods prune symmetric branches of the search tree during search [9]. However, algorithms belonging to the latter class often suffer from high memory complexity.

3 PROBLEM STATEMENT

3.1 The model

We define the job-shop scheduling problem as follows: There is a set of **tasks** T to be processed on a set of **resources** R. The resources are cumulative: i.e., there are several instances of them (e.g., homogeneous machine group) and thus they can process several tasks at the same time. Capacity of the resource $r \in R$ is denoted by $c(r) \in \mathbb{Z}^+$.

Each task $t \in T$ has a fixed duration $d(t) \in \mathbb{Z}^+$ and requires one unit of resource *r(t)* during the whole length of its execution. The tasks should be executed without preemption.

Tasks can be connected by end-to-start **precedence** constraints that determine together a directed acyclic graph of the tasks. The objective is to find start times for the tasks such that the **makespan**, i.e., the maximum of the end times is minimal.

3.2 The solution method

We solve this constrained optimization problem as a series of satisfiability problems in the course of a **dichotomic search**. In successive search runs it is checked whether solution exists for a given makespan. The trial value of the makespan is between two bounds. If UB is the smallest value of the makespan for which a solution is known and LB is the lowest value for which a solution can exist, then the trial value $(UB + LB)/2$ for the makespan 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 is found.

In the constraint-based representation of the problem one *start(t)* variable stands for the start time of each t task. The initial domain of these variables is the interval from time 0 to a large trial value of the makespan. These domains are later tightened by the propagators of the precedence and resource capacity constraints. When referring to these **time windows**, the notations *est(t)* and *lft(t)* will be used for the inferred earliest start and latest finish time of task t, respectively. For propagating precedence constraints, an arc-B-consistency algorithm, while for resource capacity constraints the edge-finding algorithm is applied.

During the search in the CP system, schedules are built chronologically, using the so-called **set times** strategy [10]. Accordingly, in each node of the search tree, an unscheduled task t is selected with minimal *est(t)*. Ties are broken by minimal *lft(t)*. Then, two sons of the search node are generated, according to the decisions whether *start(t)* is bound to *est(t)*, or t is postponed and can be scheduled only after the start time of another task has been bound.

We emphasize that scheduling cumulative resources is an especially hard problem, which can hardly be solved for optimality in real-life cases [2,3,7].

However, just in such problem instances the constraint graph consists of loosely connected components. We claim that current CBS systems can be made much more efficient if they are able to exploit this property. Below we give a formal definition of these relevant structural properties and present the algorithms that can detect and exploit them in a constructive way.

4 ANY-CASE CONSISTENCY

Any-case consistency is a notion to characterize a partial schedule that is consistent regardless of the remaining part of the scheduling problem.

A partial schedule PS is a decomposition of T into two complementary sets T_+^{PS} and T_-^{PS}, and a binding of the start times *start(t)* for the task $t \in T_+^{PS}$. For brevity, we will use *end(t)* to denote *start(t) + d(t)*. PS is called **dominant** iff it can be completed to a consistent schedule, i.e. there exists a schedule S, which satisfies all the above constraints and $\forall t \in T_+^{PS}$: $start^S(t) = start^{PS}(t)$.

Dominance can be exploited in constructive CP frameworks: if PS is dominant, then the start time of tasks $t \in T_+^{PS}$ can be bound to $start^{PS}(t)$, respectively, without the risk of loosing the solution. This means eliminating unnecessary branchings from the search tree, which can result in a significant speedup of the solution process.

In the construction of our dominance algorithm, we set out from the assumption that in loosely connected, large-size industrial problems, there are often subsets of T for which it is easy to find a consistent partial schedule PS by a fast heuristic. These subsets of T would constitute T_+^{PS}. In order to characterize that a PS is dominant, we define **any-case consistency** as follows.

We say that a precedence constraint $t1 \to t2$ of the model is any-case satisfied, iff

- $t1, t2 \in T_+^{PS}$ and $end(t1) \le start(t2)$,
- $t1 \in T_+^{PS}$, $t2 \in T_-^{PS}$ and $end(t1) \le est(t2)$,
- $t1 \in T_-^{PS}$, $t2 \in T_+^{PS}$ and $lft(t1) \le start(t2)$, or $t1, t2 \in T_-^{PS}$.

A resource capacity constraint is any-case satisfied in PS, iff for all time units τ, either

- the number of tasks t such that $t \in T_+^{PS}$ and $start(t) \le \tau \le end(t)$, or $t \in T_-^{PS}$ and $est(t) \le \tau \le lft(t)$ does not exceed *c(r)*, or
- no tasks $t \in T_+^{PS}$ use resource *r* at time τ.

If all the constraints in the model are satisfied in PS in the any-case sense, then PS is called any-case consistent.

Proposition 1: If PS is any-case consistent then it is also dominant.

Proof: The above definitions describe that all constraints within T_+^{PS} are satisfied in the traditional sense, and furthermore, the start times of tasks $t \in T_+^{PS}$ are consistent with all possible, yet unknown start times of tasks $t \in T_-^{PS}$. Hence, if there exists a solution S, then S' such that $\forall t \in T_+^{PS}: start^{S'}(t) = start^{PS}(t)$, $\forall t \in T_-^{PS}$ $start^{S'}(t) = start^S(t)$ is also a solution. Thus, PS is dominant. □

Note that whether a partial schedule is any-case consistent depends on the task time windows, and, consequently, also on the trial value of the makespan. Thus, this kind of dominance can not be applied e.g. in a branch and bound search, where constraints are posted on the makespan during search.

5 A DOMINANCE ALGORITHM

The **dominance algorithm** works on partial schedules and tries to separate those components of the problem that are any-case consistent. The algorithm looks for such a component of the scheduling problem

- that is relatively easy to solve, and
- whose solution has no influence on the other parts of the problem.

The proposed algorithm runs once in each search node within the standard CBS system, using the actual time windows *[est(t), lft(t)]* of the tasks.

If the dominance algorithm succeeds, then the *start(t)* variables of the tasks belonging to the any-case consistent component are bound immediately. This step reduces the size of the search tree since no more decisions should be made concerning these tasks. Consider Fig. 1: if an any-case consistent component is found in a node, then the search process does not need to explore the subtrees below this node and can directly jump to the next critical decision point.

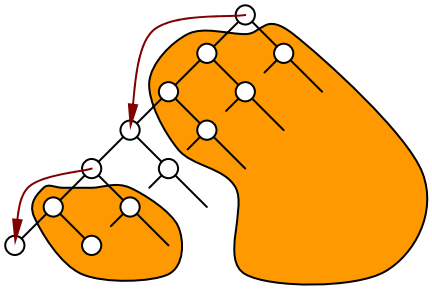

Figure 1. The effect of dominance on the search tree.

If some variables remain unbound, then the standard constraint solving procedure goes on.

5.1 Algorithmic details

The dominance algorithm works on the unscheduled tasks and builds up a schedule in a quick-and-dirty way, by the application of a fast heuristic. For this purpose we apply the rule-based **set times** heuristic (see Sect. 3). The algorithm handles tasks one by one, in the order suggested by the heuristic, and puts them either into the T_+^{PS} or into the T_-^{PS} sets.

Procedure MAIN
For each task t
  If $start(t)$ is bound, **then** add t to T_+^{PS}
  Else let $est_0(t) = est(t)$, $lft_0(t) = lft(t)$
While there are unscheduled tasks,
  Select an unscheduled t according to the heuristic
  Call INSERT(t)

If task t, selected by the heuristic, can be inserted into the partial schedule, then we assign it the earliest possible start time and add it to T_+^{PS}. Otherwise, it is added to T_-^{PS}.

Procedure INSERT (Task t)
Let τ be the first time unit such that $\tau \geq est(t)$ and there is free capacity on resource $r(t)$ in interval $[\tau, \tau+d(t)]$
If τ is found, **then call** INSERT_TPLUS(t, τ)
Else call INSERT_TMINUS(t)

Adding task to T_+^{PS} involves, at the same time, a scheduling decision. The effects of this decision are propagated by temporal propagation to the t' successors of task t. Hence, time windows of the successors are narrowed. Tasks t' for which there remains at most one valid start time are also added to T_+^{PS} and scheduled immediately.

Procedure INSERT_TPLUS(Task t, Time τ)
Add t to T_+^{PS}
Let $start(t) = \tau$, $end(t) = \tau + d(t)$
For each successor t' of t
  Let $est(t') \geq end(t)$
  If $est(t') + d(t') = lft(t')$, **then call** INSERT($t'$)

On the other hand, the addition of t to T_-^{PS} constrains other unscheduled tasks and members of T_+^{PS}. So as to make the T_+^{PS} and T_-^{PS} sets independent, in this case t should occupy its complete time window. This decision is followed by propagation that reduces the time windows of all the tasks that are related to t.

However, this propagation step may result in a too narrow time window for some tasks. Such tasks have to be put into the T_-^{PS} set. Note that this may require the removal of some task from T_+^{PS} into T_-^{PS}. Further on, moving a task from T_+^{PS} into T_-^{PS} may initiate a chain of other removal steps.

Procedure INSERT_TMINUS(Task t)
Add t to T_-^{PS}
For each successor t' of t
  Let $est(t') \geq lft_n(t)$
For each task t' which is a predecessor of t **or** is under processing on resource $r(t)$ in $[est_0(t), lft_0(t)]$
  If the current binding of $start(t')$ hurts any constraint,
    then
    Remove t' from T_+^{PS}
    Call INSERT_TMINUS(t')

The algorithm always finishes with an any-case consistent schedule. However, in the worst case, when working with heavily connected problems, T_+^{PS} may become empty.

5.2 An Illustrative Example

In the following, an example is presented to demonstrate the working of our algorithm described above. Suppose there are 3 jobs, each consisting of 2 or 3 tasks, to be scheduled on three unary resources. The task durations and time windows received from the constraint-based solver are indicated in Table 1 and Fig. 2. Tasks belonging to the same job are fully ordered by end-to-start precedence constraints.

Note that this example is moderately realistic: the edge-finding propagator of the constraint-based scheduler could further tighten the time windows, but, in order to keep the example compact, we dispense with this possibility.

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	$R2$

Table. 1. Parameters of the sample problem.

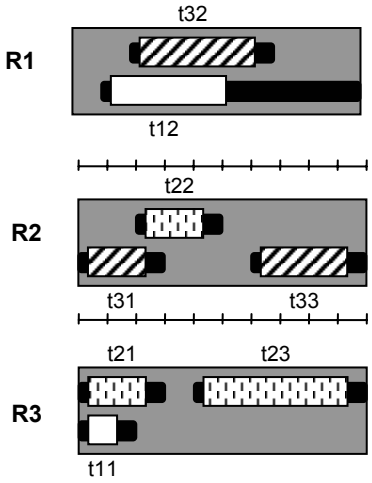

Figure 2. The sample problem.

The dominance algorithm begins by assigning start times to tasks in chronological order, according to the applied priority rule: *t11, t31* and *t21*. At this moment, the slack of tasks *t22* and *t23* vanish. Hence, they are fixed at the only valid interval left. Task *t12* is scheduled next, see Fig. 3. All these tasks are added to T_+.

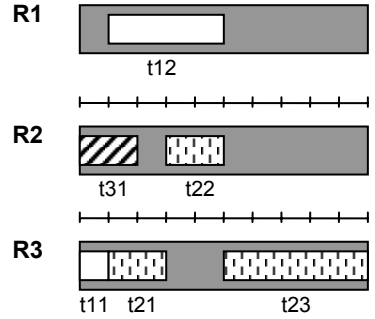

Figure 3. Building the partial schedule.

However, by fixing the start time of *t12* to *1*, the time window of *t32* should be tightened to a smaller length than its duration. Thus, it must be added to T_-. Since the time window of *t32* conflicts with the previously fixed start time of *t12*, the latter must also be transferred to T_-. Furthermore, there is only one valid start time left for *t33*, to which it has got bound now, see Fig. 4.

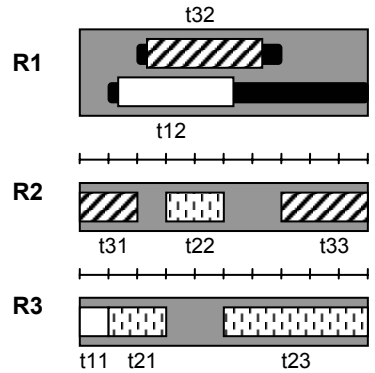

Figure 4. The any-case consistent partial schedule.

The previous investigations lead to an any-time consistent partial schedule, with all the tasks assigned to either T_+ or T_-. Since it is dominant, the start times of the 6 tasks belonging to T_+ can be bound in the CBS system. After a constraint propagation round, the solver infers valid start times for the two remaining tasks, and hence, an optimal solution with makespan 10 is found, see Fig. 5.

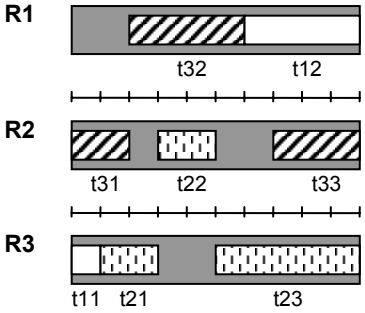

Figure 5. The final schedule.

6 SYMMETRY BREAKING

In general, **symmetry** is a mapping between two equivalent solutions of the same problem. Constraint-based scheduling problems often have symmetrical solutions. For instance, consider two independent jobs with the same routing: in different solutions, the start times are interchangeable between the respective tasks. **Breaking a symmetry** means eliminating all but one of the symmetric counterparts.

Symmetry breaking has twofold significance in solving large-scale scheduling problems. On the one hand, it prunes the search tree by reducing the number of symmetric branches. On the other hand, it strengthens propagation by converting resource capacity constraints to precedence constraints.

Let P and Q denote two isomorphic subsets of T, such that they are not connected by a directed precedence path. P and Q are considered isomorphic iff there exists a bijection $\beta : P \to Q$ such that

$$\forall p \in P : R(p) \equiv R(\beta(p)),$$
$$d(p) = d(\beta(p)), \text{ and}$$
$$\forall p_1, p_2 \in P : (p_1 \to p_2) \Leftrightarrow (\beta(p_1) \to \beta(p_2)).$$

Furthermore, suppose that there are only incoming precedence constraints to P and only outgoing precedence constraints from Q.

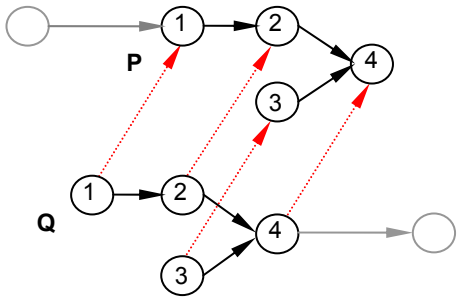

Figure 6. Symmetry breaking.

Proposition 2: If there exists a solution S of the scheduling problem, then there also exists a solution S' which satisfies the precedence constraints $\forall p \in P : \beta(p) \rightarrow p$.

If the resource *R(p)* is unary, then the corresponding $\beta(p) \rightarrow p$ constraint is an end-to-start precedence, otherwise, it is a start-to-start precedence. If neither P nor Q have predecessor or successors, then the direction of precedence constraints can be chosen arbitrarily. Note that both the standard CBS system and the dominance algorithm can be trivially extended to handle start-to-start precedence constraints.

Proof: Let us construct the desired solution S' departing from S by swapping each pair of tasks *p*, $\beta(p)$ where $\beta(p) \rightarrow p$ is not satisfied. Now all resource capacity constraints are satisfied, because the durations and resource requirements of the swapped pairs of tasks are the same. Precedence constraints $p1 \rightarrow p2$ cannot be hurt in S', either, since it was satisfied for the tasks at the place of *p1* and *p2* in *S*, either directly or through the chain $p1 \rightarrow \beta(p2) \rightarrow p2$. □

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. Hence, the symmetries are broken between jobs aimed at the fabrication of identical end products, or products that belong to the same family and have overlapping routings. Hence, P and Q stand for the sections of two jobs which fall into the scheduling horizon, and the job containing Q is in a slightly more advanced state. These symmetries can easily be found with the help of some appropriate task identifiers.

7 EXPERIMENTS

The above algorithms were prepared as a part of efforts to improve the efficiency of the detailed job-shop scheduler module of our integrated production planner and scheduler system [12, 13, 14, 17]. As in practical applications (see also [11]), the main goal of scheduling is to unfold medium-term production plans into executable schedules.

The starting point of the implementation was the constraint-based scheduler of Ilog. It was extended by the symmetry breaker as a pre-processor, and the dominance algorithm, run once in each search node. Both were encoded in C++.

Our test problem instances originate from an industrial partner that manufactures mechanical parts of high value. The products can be ordered into 4 larger product families, and their tree-structured routings contain 30 to 430 manufacturing tasks. They are executed on one of the cc. 100 different resources, out of which cc. 25 are homogeneous machine groups of 2 to 10 machines, while the rest constitutes of unary machines. The job-shop level scheduling problems were solved with a horizon of one week. The problem instances were subject to certain simplifications in order to match the problem definition above.

Four systems participated in the test: DS denotes a dichotomizing search using only tools of the commercial CP solver. First, it was extended by the symmetry breaker (DS+SB), then by the dominance algorithm (DS+Dom). In the last system, all components were switched on (DS+SB+Dom).

Test runs were performed on two sets of data. Benchmark set 1 consists of 30 instances received from the industrial partner, each containing from 150 up to 990 tasks. The time limit was set to 120 seconds. Even the simplest algorithm, DS could find optimal solutions for but one problem. The symmetry breaker further improved on its results, but the systems incorporating the dominance algorithm were the definite winners, thanks to an extremely low number of search nodes. In cases where the first solution proved optimal, these two systems could find it without any search. The results are presented in Table 2, with separate rows for instances which could be solved to optimality (+) and those which not (-).

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

Table 2: Test results on benchmark set 1.

A set of 10 larger problem instances – with 840 to 1930 tasks – was generated by merging several problems from benchmark set 1. Note that it turned the highly precedence-constrained instances into highly resource-constrained problems, which put the dominance algorithm in disadvantage due to wider time windows. Alike on benchmark set 1, the dominance algorithm significantly reduced the size of the search tree, although, this required a considerable investment of time, compared to the number of search nodes. Nevertheless, only the complete system could solve 9 problem instances out of the 10, see Table 3.

Method	Number of instances	Avg. search nodes	Avg. search time (sec)	Mean error (%)
DS (+)	8	2350	42.75	-
DS (-)	2	14190	120.0	8.8
DS + SB (+)	8	2015	45.25	-
DS + SB (-)	2	12405	120.0	5.8
DS + Dom (+)	7	8.6	32.0	-
DS + Dom (-)	3	642	120.0	5.0
DS + SB + Dom (+)	9	62	48.3	-
DS + SB + Dom (-)	1	672	120.0	8.3

Table 3: Test results on benchmark set 2.

8 CONCLUSIONS

Constraint-based scheduling enables an efficient combination of general purpose reasoning and search techniques as well as the use of special heuristics. This paper discussed the significance of constructive algorithms in the solution of large size industrial job scheduling problems. A system based on a commercial constraint-based scheduler, extended by a symmetry breaker and a dominance algorithm was proposed and

validated on industrial data. Our solution method can generate a series of improving solutions, with a more and more narrowing gap between the lower and upper bound of the schedules. The algorithm can be stopped at any time.

The first results are very promising: the constructive algorithms reduced the size of the search tree by at least an order of magnitude. The response time of the system is also reasonable, which is a more and more important pre-requisite of practical scheduling [15].

The weakness of the current implementation is the relatively high running time of the dominance algorithm. The adjustment of the algorithms to handle the richer models with human operators, setup and transportation times, etc., is also a precondition of the applicability of the approach on real-life scheduling problems.

All in all, we claim that a deeper understanding of the structural properties of practical scheduling problems could help much in improving the scalability of scheduling systems.

9 ACKNOWLEDGEMENTS

This work has been supported by the grants “Digital Factory” NRDP No. 2/040/2001 and IKTA 4-00181/2000. The authors wish to thank Tamás Kis for his valuable comments.

10 REFERENCES

[1] Apt, K.R., 2003, Principles of Constraint Programming, Cambridge University Press.

[2] Baptiste, P., Le Pape, C., 2000, Constraint Propagation and Decomposition Techniques for Highly Disjunctive and Highly Cumulative Project Scheduling Problems, Constraints 5(1/2): 119-139.

[3] Baptiste, P., Le Pape, C., Nuijten, W., 1999, Satisfiability Tests and Time-bound Adjustments for Cumulative Scheduling Problems, Annals of Operation Research 92:305–333.

[4] Baptiste, P., Le Pape, C., Nuijten, W., 2001, Constraint-based Scheduling, Kluwer Academic Publishers.

[5] Baptiste, P., Peridy, L., Pinson, E., 2003, 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: 1–11.

[6] Crawford, J., Luks, G., Ginsberg, M., Roy, A., 1996, Symmetry Breaking Predicates for Search Problems, In Proc. of the 5th Int. Conf. on Knowledge Representation and Reasoning, pp. 148-159.

[7] Demeulemeester, E.L., Herroelen, W.S., 2002, Project Scheduling: A Research Handbook, Kluwer Academic Publishers.

[8] Freuder, E.C., 1978, Synthesizing Constraint Expressions, Communications ACM 21(11): 958-966.

[9] Gent, I.P., Smith, B.M., 2000, Symmetry Breaking in Constraint Programming, In Proc. of the 14th European Conference on Artificial Intelligence, pp. 599-603.

[10] Ilog, 2001, Ilog Scheduler 5.1 User’s Manual.

[11] Kempf, K., Uzsoy, R., Smith, S., Gary, K., 2000, Evaluation and Comparison of Production Schedules, Computers in Industry, 42:203-220.

[12] Kovács, A., Váncza, J., Monostori, L., Kádár, B., Pfeiffer, A., 2003, Real-Life Scheduling Using Constraint Programming and Simulation, In Intelligent Manufacturing Systems, pp. 213-218, Elsevier.

[13] Kovács, A., 2003, A Novel Approach to Aggregate Scheduling in Project-Oriented Manufacturing. Int. Conference on Artificial Intelligence Planning and Scheduling, Doctoral Consortium, Trento, Italy, pp. 63-67.

[14] Márkus, A., Váncza, J., Kis, T., Kovács, A., 2003, Project Scheduling Approach to Production Planning, Annals of the CIRP, 52/1:359-362.

[15] McKay, K.N., Wiers, V.C.S., 1999, Unifying the Theory and Practice of Production Scheduling, Journal of Manufacturing Systems, 18(4):241-255.

[16] Tsang, E., 1993, Foundations of Constraint Satisfaction. Academic Press.

[17] Váncza, J., Kis, T., Kovács, A., 2003, Aggregation - The Key to Integrating Production Planning and Scheduling. Annals of the CIRP, 53/1, in print.

[18] Wolpert, D.H., Macready, W.G., 1997, No-Free Lunch Theorems for Optimization, IEEE Trans. on Evolutionary Computation, 1(1):67-82.