Preference-based Search for Scheduling

Ulrich Junker
ILOG
1681, route des Dolines
06560 Valbonne
France
junker@ilog.fr

Abstract

Preference-based search (PBS) is a new search procedure for solving combinatorial optimization problems. Given a set of preferences between search decisions, PBS searches through a space of preferred solutions, which is tighter than the space of all solutions. The definition of preferred solutions is based on work in non-monotonic reasoning (Brewka 1989; Geffner & Pearl 1992; Grosof 1991) on priorities between defaults. The basic idea of PBS is quite simple: Always pick a locally best decision α. Either make the decision α or make other locally best decisions that allow to deduce $\neg\alpha$ and thus represent a counterargument for α. If there is no possible counterargument then PBS does not explore the subtree of $\neg\alpha$. This pruning of the search space is obtained by non-monotonic inference rules that are inspired by Doyle's TMS and that detect decisions belonging to all or no preferred solution. We show that PBS can optimally solve various important scheduling problems.

Keywords: search, non-monotonic reasoning, scheduling, constraint satisfaction

Introduction

The standard approach for solving combinatorial optimization problems by constraint satisfaction is to apply a Branch-and-Bound method. Each time a solution is found during tree search, an upper bound on the objective is decreased. A lower bound is maintained in intermediate search states, e.g. by encapsulating an OR-algorithm inside a constraint. Furthermore, we can use value-ordering heuristics that select locally better choices first. Unfortunately, if the bounds are far from the optimum, the search space will not be pruned much. If the number of solutions is high, this process will take a long time to converge via the optimum. The problem is that the objective is not sufficiently exploited during search. Since we systematically search through the space of all solutions, locally best choices are more and more abandoned when search is progressing.

Preferred solutions as elaborated in non-monotonic reasoning (Brewka 1989) impose a tighter constraint on the set of all solutions even if the bounds on the objective are not yet tight. The definition of preferred solutions is based on preferences between the possible search decisions. In scheduling, for example, we prefer to assign smaller start times to an activity a. If $t_1 < t_2$ then we prefer the decision $s(a) = t_1$ to $s(a) = t_2$. These preferences define a strict partial order $\prec$ on the set of all possible decisions. A preferred solution is then defined as follows: Firstly, we choose a total order that is a superset of the given partial order. Secondly, we visit the search decisions in this order starting with the best decision. During this process, we will make some of the decisions and abandon the others. When we visit a decision then we check whether it is consistent w.r.t. the initial problem and the already made decisions. If yes then we make the decision. Otherwise, we abandon it. The set of made decisions that is obtained after visiting all possible decisions is a preferred solution. Different preferred solutions can be obtained by choosing different orders.

The same preferred solution can, however, be obtained by several orders. Enumerating all orders therefore is not a good method for computing all preferred solutions. We will elaborate a search procedure which determines each preferred solution exactly once, but which still profits from the preferences between search decisions in order to reduce the search effort. We call this procedure preference-based search (PBS). The basic idea of PBS is quite simple: In each search state, PBS always picks a locally best decision α. PBS branches as follows. It either makes the decision α or it makes other locally best decisions that allow to deduce $\neg\alpha$ and thus represent a counterargument for α. Hence, PBS abandons a locally best decision only if it finds a reason (i.e. a counterargument) for this. If PBS does not find a counterargument for α then it considers $\neg\alpha$ an unjustified choice and does not explore the subtree of $\neg\alpha$ in contrast to standard systematic tree search methods. PBS prunes the search space by applying non-monotonic inference rules that are inspired by Doyle's TMS and that detect decisions belonging to all or no preferred solution. Preference-based search thus does no longer search through the space of all solutions, but only through the space of preferred solutions. It is important to note that PBS is a systematic search procedure for exploring the preferred solutions. It explores all the preferred solutions and does not explore the same preferred solution twice.

PBS itself is a general search procedure that just needs a set of preferences between the possible search decisions.

It can be used as search algorithm for all AI problems that can be treated with Brewka's approach (Brewka 1989). Examples are inheritance of defaults, but also diagnosis and configuration.

In this paper, we show that PBS can also be used to solve optimization problems. Two different definitions for preferred solutions have been proposed in the literature. In (Junker & Brewka 1991), we show that Brewka's definition (Brewka 1989) is stricter than the definition used by Geffner and Grosof (Geffner & Pearl 1992; Grosof 1991). Interestingly, both are useful for solving optimization problems. Brewka's definition can be used for certain scheduling problems, which have a non-linear objective. Grosof/Geffner's definition can be used for integer programming where the objective is linear. The PBS procedure presented in this paper follows Brewka's approach and we consequently apply it to selected scheduling problems.

For each problem, we seek a set of preferences that preserves optimality in the following sense: At least one optimal solution has to be a preferred one w.r.t. the given preferences. If an optimization problem has this property we can use PBS to solve it. For project scheduling problems with precedence constraints and multiple-capacitated resources, we prefer to assign smaller start times to an activity and show that these preferences preserve optimality. The preferred solutions that are obtained by these preferences correspond to the left-shifted schedules.

If optimality is preserved then PBS is a systematic search procedure for the considered optimization problem. That means PBS is able to prove optimality. If PBS does not find a preferred solution that has an objective strictly better than v^* then the optimal value of the objective is not better than v^*. Thus, PBS also reduces the search effort for the optimality proof, which often is very time-consuming.

Compared to existing work in scheduling, we can state that PBS is a generalization and improvement of the schedule-or-postpone method (Le Pape *et al.* 1994; ILOG 1997). This method assigns start times in chronological order, which allows to do efficient constraint propagation in presence of resources with large or time-dependent capacities. PBS gives a clear semantics to the schedule-or-postpone method and shows how to do additional pruning of the search space by analyzing conflicts between start time assignments.

Another method that exploits conflicts between activities is the precedence-constraint-posting approach (PCP) of Cheng and Smith (Smith & Cheng 1993; Laborie & Ghallab 1995; Cesta, Oddi, & Smith 1999). PCP does not assign start times, but posts precedence constraints between pairs of activities that are potentially in conflict. Thus, it also provides an elegant way to search through the space of all left-shifted schedules.

The purpose of this paper is to elaborate the PBS algorithm in a general way and to show that it can be used for scheduling. An empirical comparison between the PBS-scheduler, i.e. the improved schedule-or-postpone method, and the precedence-constraint-posting approach, requires further work and is beyond the scope of this paper.

The paper has five parts: First, we introduce the definition of preferred solutions. Second, we develop the PBS-algorithm. Third, we give preferences for selected scheduling problems. Fourth, we briefly discuss an implementation of PBS for job-shop problems. Finally, we discuss related and future work.

Preferred Solutions

In this section, we give a precise definition of preferred solutions for combinatorial problems that are formulated as constraint satisfaction problems. A constraint satisfaction problem (CSP) consists of a set of variables on which a set of constraints is formulated. Each constraint consists of a relation and a tuple of variables. In this paper, we simply suppose that the set Γ of all constraints of a CSP is given. This set includes domain membership constraints of the form $x \in D$, which are usually introduced separately. We also suppose that a consistency checker is given that checks whether a set of constraints is consistent or not.

We furthermore suppose that a set $\mathcal{A}$ of all possible decisions is given that can be made for solving the problem. A decision can be an arbitrary constraint. For example, we consider constraints of the form $x = v$ which represent an assignment of value v to variable x. We now consider a set of preferences between the decisions in $\mathcal{A}$. In scheduling, for example, we prefer to assign smaller start times to an activity a. If $t_1 < t_2$ then we prefer the decision $s(a) = t_1$ to $s(a) = t_2$. These preferences define a strict partial order $\prec$ on the set $\mathcal{A}$ of all possible decisions.

Definition 1 *Let $<$ be a total order on $\mathcal{A}$ that is a superset of $\prec$. Let $\alpha_1, \ldots, \alpha_n$ be an enumeration of the elements of $\mathcal{A}$ in increasing $<$-order. We then define $A_0 := \emptyset$ and*

$$A_i := \begin{cases} A_{i-1} \cup \{\alpha_i\} & \text{if } \Gamma \cup A_{i-1} \cup \{\alpha_i\} \text{ is consistent} \\ A_{i-1} & \text{otherwise} \end{cases}$$

A_n is a preferred solution.

Thus, we choose a total order $<$ that completes the given partial order. We then visit the decisions of $\mathcal{A}$ in increasing $<$-order and make a decision if it is consistent w.r.t. the decisions made earlier: Thus, a decision can only be retracted if it is inconsistent w.r.t. better decisions. Each suitable total order $<$ defines a preferred solution. The same preferred solution can, however, be obtained by several orders.

Preference-based Search (PBS)

In this section, we introduce two versions of preference-based search. The basic version just exploits preferences, whereas the more sophisticated version also exploits counterarguments and prunes the search space even more.

PBS1: Decide-or-Refute

Since the same preferred solution can be obtained by several orders, enumerating all orders is not a good method for computing all preferred solutions. In order to compute each preferred solution exactly once, we proceed as follows. In each step, we select a best decision α among a set U of unexplored decisions. We also say that α is a $\prec$-best element of U since U does not contain an element β such that $\beta \prec \alpha$.

Algorithm *PBS1*(Γ, $\mathcal{A}$, $\prec$)

1. $A := \emptyset$; $U := \mathcal{A}$; $Q := \emptyset$;
2. **while** $Q \cup U \neq \emptyset$ **do**
3. $B := \{\alpha \in Q \cup U \mid \nexists \beta \in Q \cup U : \beta \prec \alpha\}$;
4. **if** there is an $\alpha \in B \cap Q$ s.t.
5. $\Gamma \cup A \cup \{\alpha\}$ is inconsistent
6. **then** $Q := Q - \{\alpha\}$ and continue;
7. **if** $U = \emptyset$ and $Q \neq \emptyset$ **then** fail;
8. **if** $B \cap U = \emptyset$ **then** fail;
9. **if** there is an $\alpha \in B \cap U$ s.t.
10. $\Gamma \cup A \cup \{\alpha\}$ is inconsistent
11. **then** $U := U - \{\alpha\}$ and continue;
12. select an $\alpha \in B \cap U$ and set $U := U - \{\alpha\}$;
13. **choose** $A := A \cup \{\alpha\}$ **or** $Q := Q \cup \{\alpha\}$;
14. return A;

Figure 1: Algorithm PBS1

If α is inconsistent w.r.t. a set A of already made decisions then we just drop it. Otherwise, we branch. We either make the decision α by adding it to A or we add it to a set Q of decisions that have to be refuted by subsequent decisions. The set Q thus represents a set of *'refutation queries'*, which impose an additional constraint on the preferred solutions we are computing. We are only interested in preferred solutions that do not contain elements of Q. If, in a subsequent state, an element β of Q is inconsistent w.r.t. an extended set A then we can remove β from Q. The set Q also imposes a constraint on the elements that can be selected in a search state. As long as an element β is in Q, we can't select the decisions that are $\prec$-worse than β. If we can't select any element any more then a dead-end is reached. The nondeterministic algorithm in figure 1 describes this behaviour.

This algorithm can be implemented by a tree search procedure. The following theorem states that PBS1 systematically searches through the space of all preferred solutions:

Theorem 1 *Algorithm PBS1 always terminates. Each successful run returns a preferred solution and each preferred solution is returned by exactly one successful run.*

The algorithm terminates since $2 \cdot \mid U \mid + \mid Q \mid$ is decreased in each iteration. In each step, the algorithm maintains a state $S := (A, Q, U)$. We say that X is a *preferred solution of the state* S iff X is a superset of A and disjoint to Q and $X - A$ is a preferred solution of the problem $\Gamma \cup A$ with decisions $Q \cup U$ and preferences $\prec \cap (Q \cup U)^2$. Following properties allow to show that PBS1 computes exactly the preferred solutions of state S if PBS1 is in the state S:

P1 if $U = \emptyset$ and $Q = \emptyset$ then A is a preferred solution of (A, Q, U).

P2 if $\alpha \in Q$ and α is a $\prec$-best element of $Q \cup U$ and $\Gamma \cup A \cup \{\alpha\}$ is inconsistent then X is a preferred solution of (A, Q, U) iff X is a preferred solution of $(A, Q - \{\alpha\}, U)$.

P3 if $\alpha \in U$ and α is a $\prec$-best element of $Q \cup U$ and $\Gamma \cup A \cup \{\alpha\}$ is inconsistent then X is a preferred solution of (A, Q, U) iff X is a preferred solution of $(A, Q, U - \{\alpha\})$.

P4 if all $\prec$-best elements α of $Q \cup U$ are in Q and are consistent w.r.t. $\Gamma \cup A$ then (A, Q, U) has no preferred solution.

P5 if $\alpha \in U$ and α is a $\prec$-best element of $Q \cup U$ then X is a preferred solution of (A, Q, U) iff X is a preferred solution of $(A, Q \cup \{\alpha\}, U - \{\alpha\})$ or of $(A \cup \{\alpha\}, Q, U - \{\alpha\})$.

PBS2: Reasoning with Conflicts

As shown in the last section, preference-based search maintains a set Q of refutation queries. For each refutation query α, it has to make a set of decisions that are inconsistent w.r.t. α and that represent a counterargument to α. For this purpose, PBS1 can use any unexplored decision in U except those that are $\prec$-worse than α. Now suppose that there is no such counterargument. In this case, we can make as many additional decisions as possible without being able to refute α. Hence, PBS1 will not find any preferred solution that does not contain α. The problem is that PBS1 detects this only after having explored all elements of U (except those that are worse than α) and this can require a longer search.

In this section, we elaborate a set of non-monotonic inference rules that detect elements that belong to all preferred solutions or to none of them. They help to detect deadends earlier and avoid useless branching. In order to do this, the rules exploit the possible conflicts between decisions. A subset C of $\mathcal{A}$ is a *conflict* iff $\Gamma \cup C$ is inconsistent. A conflict C is *minimal* if no proper subset of C is a conflict. If C is a minimal conflict containing α then $C - \{\alpha\}$ is a *counterargument* for α. In order to refute α, we can, for example, make the decisions in $C - \{\alpha\}$. If the set of all possible decisions $\mathcal{A}$ does not contain any conflict then we are not able to refute any element. As a consequence, there is only a single preferred solution, namely $\mathcal{A}$.

Hence, multiple preferred solutions are only obtained if there are conflicts between decisions. Conflicts can therefore help to search for different preferred solution. Different techniques for exploiting conflicts have been proposed.

1. Hitting trees (Reiter 1987) pick a minimal conflict $\{\gamma_1, \ldots, \gamma_k\}$ in each search node and consider all possible ways to resolve it. For each element γ_i, a son node is introduced and γ_i is removed from the set of possible decisions in the son node. The set $\{\gamma_1, \ldots, \gamma_k\} - \{\gamma_i\}$ is a counterargument for γ_i. Elements of this counterargument can, however, be retracted in subsequent steps. If this happens then the retracted element γ_i may be consistent w.r.t. a solution. With other words, we can obtain unjustified retractions in the end.
2. TMS-based provers for default logic (Junker & Konolige 1990) require that all counterarguments for a default are pre-computed. If no counterargument for a default can ever be applied then a TMS-labelling algorithm labels the default with IN, which means that the default belongs to all TMS-labellings (i.e. all solutions).

Whereas the hitting tree approach picks a fixed counterargument in order to refute a decision α, the TMS-based approach just checks whether there exists a potential counterargument. We show how to integrate the TMS-based labelling rules into PBS. The TMS-based approach requires the precomputation of all possible counterarguments for a decision α. Since there is an exponential number of conflicts in the

Algorithm *PBS2*(Γ, $\mathcal{A}$, $\prec$)

1. $A := \emptyset$; $U := \mathcal{A}$; $Q := \emptyset$;
2. **while** $Q \cup U \neq \emptyset$ **do**
3. $B := \{\alpha \in Q \cup U \mid \nexists \beta \in Q \cup U : \beta \prec \alpha\}$;
4. **if** there is an $\alpha \in B \cap Q$ s.t.
5. $\Gamma \cup A \cup \{\alpha\}$ is inconsistent
6. **then** $Q := Q - \{\alpha\}$ and continue;
7. **if** $U = \emptyset$ and $Q \neq \emptyset$ **then** fail;
8. **if** $B \cap U = \emptyset$ **then** fail;
9. **if** there is an $\alpha \in B \cap U$ s.t.
10. $\Gamma \cup A \cup \{\alpha\}$ is inconsistent
11. **then** $U := U - \{\alpha\}$ and continue;
12. **if** there is an $\alpha \in Q$ s.t. $A \cup U - \mathit{succ}(\{\alpha\})$
13. does not contain a counterargument to α
14. **then** fail;
15. **if** there is an $\alpha \in B \cap U$ s.t. $A \cup U - \mathit{succ}(\{\alpha\})$
16. does not contain a counterargument to α
17. **then** $A := A \cup \{\alpha\}$; $U := U - \{\alpha\}$; continue;
18. select an $\alpha \in B \cap U$ and set $U := U - \{\alpha\}$;
19. **choose** $A := A \cup \{\alpha\}$ **or** $Q := Q \cup \{\alpha\}$;
20. return A;

Figure 2: Algorithm PBS2

worst-case, this approach is not feasible. Instead of precomputing all counterarguments, we check for the absence of counterarguments during search. Even the check for the absence of a counterargument can computationally be very hard. We do not need to detect the absence of counterarguments in all cases. Instead, we can check sufficient conditions for the absence of counterarguments if this is computationally cheaper. We thus approximate the subproblem of counterargument checking as follows:

Definition 2 *Let X be a set of possible decisions and α be a decision to be refuted. A* counterargument checker *is a procedure that satisfies following condition: If applying the checker to X and α returns false then X does not contain a counterargument for α.*

Depending on the problem, we can implement very specific counterargument checkers and choose the sufficient conditions they are checking. We now extend algorithm PBS1 as follows: If an element α of Q has no counterargument in the set $A \cup U - \{\beta \in U \mid \alpha \prec \beta\}$ then we can never refute α and we have reached a dead-end. If we show the same property for an element α of U then we know in advance that we can never refute α. Hence, each preferred solution of the given state contains α and it is not necessary to branch. The resulting algorithm is given in figure 2.

In some cases, the set $A \cup U - \{\beta \in U \mid \alpha \prec \beta\}$ may contain a counterargument for α, but it involves decisions in U that can never be made. Below, we list some more complex properties that allow to limit the set of possible counterarguments even further. In the following, let B be the set of $\prec$-best elements of $Q \cup U$ and $\mathit{succ}(X) := \{\alpha \in U \mid \exists \beta \in X : \beta \prec \alpha\}$ the set of elements of U that are worse than the elements in X:

P6 if $\alpha \in Q$ and $A \cup U - \mathit{succ}(\{\alpha\})$ does not contain a counterargument for α then (A, Q, U) has no preferred solution.

P7 if $\alpha \in U$ and α is a $\prec$-best element of $Q \cup U$ and $A \cup U - \mathit{succ}(\{\alpha\})$ does not contain a counterargument for α then X is a preferred solution of (A, Q, U) iff X is a preferred solution of $(A \cup \{\alpha\}, Q, U - \{\alpha\})$.

P8 if $\alpha \in Q$ and $\beta \in U$ and each counterargument C for α with $C \subseteq A \cup U - \mathit{succ}(\{\alpha\})$ is also a counterargument for β then X is a preferred solution of (A, Q, U) iff X is a preferred solution of $(A, Q \cup \{\beta\}, U - \{\beta\})$.

P9 if there is a $Y \subseteq Q$ s.t. no $\alpha \in Y$ has a counterargument in $A \cup U - \mathit{succ}(Y)$ then (A, Q, U) has no preferred solution.

P10 if there is a $Y \subseteq B \cap U$ s.t. no $\alpha \in Y$ has a counterargument in $A \cup U - \mathit{succ}(Y)$ then X is a preferred solution of (A, Q, U) iff there is an $\alpha \in Y$ s.t. X is a preferred solution of $(A \cup \{\alpha\}, Q, U - \{\alpha\})$.

P11 if there is a $Y \subseteq U$ s.t. for all $\beta \in Y$ there exists an $\alpha \in Q$ s.t. each counterargument C for α with $C \subseteq A \cup U - \mathit{succ}(Y)$ is also a counterargument for β then X is a preferred solution of (A, Q, U) iff X is a preferred solution of $(A, Q \cup Y, U - Y)$.

Properties P6 and P7 are those used in PBS2. P6 allows to detect failures and P7 allows to detect elements that belong to all preferred solutions. P8 allows to detect elements that belong to no preferred solution. P9, P10, and P11 are generalizations of these rules that consider a set Y of decisions instead of a single one. Finding suitable sets Y that allow the application of P9, P10, and P11 is a non-trivial task. Again we can check sufficient conditions for applying P9, P10, and P11. For specific problems such as scheduling there are efficient techniques to identify some sets Y for P9, P10, and P11.

Preferences in Scheduling

We now apply PBS to selected scheduling problems, which are optimization problems. In addition to a set Γ of constraints, these problems have an objective z, which is an integer variable. v^* is the *optimal value* of the optimizations problem iff 1. the objective z is greater than or equal to v^* in all solutions (i.e. Γ implies $z \geq v^*$) and 2. the objective is equal to v^* in some solution (i.e. $\Gamma \cup \{z = v^*\}$ is consistent). A solution with an objective of v^* is called *optimal solution*.

Our purpose is to find a set of preferences that *preserves optimality* in the following sense: At least one optimal solution has to be a preferred one w.r.t. the given preferences. If an optimization problem has this property we can use PBS to solve it. We show that important scheduling problems have this property.

Resource-Constrained Project Scheduling

First we consider project scheduling problems that consist of a set of activities with fixed durations, a set of precedence constraints between the activities, and a set of resources with discrete capacities. For each activity, we introduce a start variable $s(a)$. If $d(a)$ is the fixed duration of the activity then $s(a) + d(a)$ denotes its end time. Activities may have release dates and due dates which leads to constraints of the

form $s(a) \geq rd(a)$ and $s(a) + d(a) \leq dd(a)$. If an activity a_1 precedes an activity a_2 then the end time of a_1 is smaller than or equal to the start time of a_2, i.e. $s(a_1) + d(a_1) \leq s(a_2)$. Furthermore, an activity can require a fixed capacity of a given resource. For each time t, the sum of the required capacities of all activities that require a resource r and that are executed during t is smaller than or equal to the capacity of the resource. The objective of the problem is to minimize the makespan, i.e. the latest end time of the activities.

We now introduce preferences between start time assignments of the form $s(a) = t$. Let $\prec_1$ be the smallest strict partial order between those assignments that satisfies:

$$\begin{array}{lll} (s(a) = t_1) \prec_1 (s(a) = t_2) & \text{if} & t_1 < t_2 \\ (s(a_1) = t_1) \prec_1 (s(a_2) = t_2) & \text{if} & t_1 + d(a_1) \leq t_2 \end{array}$$

Let $\prec_1$ be the transitive closure of the relation $\leftarrow_1$. The preferred solutions obtained by $\prec_1$ correspond to the left-shifted schedules.

Since PBS constantly does consistency checks it is necessary that those consistency checks are cheap and can be implemented by constraint propagation. Constraints that make the given problem difficult (e.g. due date constraints) are better treated separately.

Theorem 2 *Let Γ be a set of precedence, release date, and resource constraints and Δ be a set of due date constraints for a project scheduling problem. If $\Gamma \cup \Delta$ has a solution then there exists a preferred solution X of Γ and $\prec_1$ that is an optimal solution of $\Gamma \cup \Delta$.*

As usual, we can use an upper bound on the objective to prune the search for an optimal solution. For preferred solutions, we proceed as follows: Suppose we found a preferred solution that satisfies Δ and that has an objective value of v. If PBS explores a state $S := (A, Q, U)$ such that $\Gamma \cup A \cup \Delta \cup \{z < v\}$ is inconsistent then no preferred solution of this state will satisfy Δ and have a better objective than v. In this case, we can prune the subtree of S.

Alternatives Resources

PBS can also be applied to alternative resources. In this case, the resource required by an activity is not given, but belongs to a set of alternative resources. These resources can differ in speed and the duration $d(a, r)$ of an activity a is dependent on the resource r. The chosen resource of a is represented by a constraint variable $r(a)$. The end time of an activity now is $s(a) + d(a, r)$ if $r(a) = r$. A search decision now will assign and schedule an activity and has the form $s(a) = t \wedge r(a) = r$. Of course, we prefer faster resources for an activity a. If the alternative resources have task-independent preferences (i.e. if $d(a_1, r_1) < d(a_1, r_2)$ implies $d(a_2, r_1) < d(a_2, r_2)$ for all a_1, a_2, r_1, r_2) then we can prove that following preferences preserve optimality: Let $\prec_2$ be the smallest strict partial order between those assignments such that:

$$(s(a) = t_1 \wedge r(a) = r_1) \prec_2 (s(a) = t_2 \wedge r(a) = r_2)$$
$$\text{if } t_1 + d(a, r_1) < t_2 + d(a, r_2)$$

$$(s(a_1) = t_1 \wedge r(a_1) = r) \prec_2 (s(a_2) = t_2 \wedge r(a_2) = r)$$
$$\text{if } t_1 + d(a_1, r) \leq t_2$$

Implementation for Job-shop Problems

For first experiments, we implemented a special version of PBS2 for the special case of job-shop scheduling with ILOG SCHEDULER (ILOG 1997). The operations of PBS2 have been expressed in terms of activities and their earliest start times. We also obtain intuitive specializations of properties P6,P7, and P11.

First experimental results are very encouraging. The difficult job-shop problem MT20 has been solved in 56 sec. and with 69344 backtracks (on a Pentium II/300MHz using Linux). This includes the proof of optimality. If additionally the edge-finder of ILOG SCHEDULER is used, MT20 is solved in 6 sec. and with 3542 backtracks. Further improvements might be obtained by variable ordering heuristics and search methods such as LDS.

Related Work

The schedule-or-postpone method (Le Pape *et al.* 1994; ILOG 1997), which is used in numerous industrial project scheduling applications, can directly be seen as an implementation of the algorithm PBS1 for project scheduling problems and demonstrates its effectiveness. On the other hand, PBS provides a theoretical foundation for this method and PBS2 improves the schedule-or-postpone method significantly.

PBS has some concepts in common with Squeaky-wheel optimization (SWO) (Joslin & Clements 1998). SWO performs a local search through a space of possible prioritizations and uses them to produce greedy solutions. Depending on critical elements (trouble-makers) in the greedy solutions priorities are changed. A greedy solution can be seen as a preferred solution and its prioritization corresponds to the total order that produced the preferred solution. In contrast to SWO, PBS is a systematic search procedure which visits each preferred solution only once, whereas different prioritizations can lead to the same preferred solution.

Similar to PBS, heuristic repair (Minton *et al.* 1992) only abandons current (best) choices if it finds a reason for this in form of a violated constraint. For example, consider three activities a, b, c of duration 10 that all require the same resource of capacity 1. If the current start times are $s(a) = 0$, $s(b) = 5$, $s(c) = 10$ then a and b are in conflict. We might decide to push a after the end of b by changing $s(a)$ to 15. Next we treat the conflict between b and c and push b after the end of c by setting $s(b)$ to 20. Unfortunately, the reason for delaying a is no longer valid. Since we reproduce the same pattern after each step, we will even run into an infinite repair chain. PBS avoids this problem.

A standard approach for solving project scheduling problems consists in solving conflicts between activities by adding additional precedence constraints (cf. e.g. (Cesta, Oddi, & Smith 1999)). In contrast to this, PBS allows to do start time assignments which results into tighter constraints and more efficient propagation.

Conclusion

We have developed a systematic search procedure for combinatorial optimization problems. PBS uses a set of pref-

erences in order to search through a space of preferred solutions, which is tighter than the space of all solutions. PBS uses non-monotonic inference rules that are inspired by Doyle's TMS and that exploit properties of preferred solution for doing the additional pruning of the search space.

We applied PBS to scheduling. We determined a set of preferences for project scheduling problems which preserve optimality (i.e. guarantee that at least one optimal solution is a preferred one). First experimental results for the job-shop problem MT20 show that PBS is capable to optimally solve difficult optimization problems. Future work is needed to compare PBS with other techniques for solving project scheduling problems (e.g. dominance rules) and to see for which advanced scheduling problems (e.g. with alternative resources) it can provide interesting results.

Future work will be spent to develop a variant of the PBS algorithm that is able to determine the 'G-preferred solutions' as defined by (Geffner & Pearl 1992; Grosof 1991). This will enable us to apply the PBS-idea to other optimization problems such as integer programming, assignment problems, and set covering.

Acknowledgements

I would like to thank the anonymous reviewers for very helpful comments. Without the moral support from my family, this paper would not have been written.

References

Brewka, G. 1989. Preferred subtheories: An extended logical framework for default reasoning. In *IJCAI-89*, 1043–1048. Detroit, MI: Morgan Kaufmann.

Cesta, A.; Oddi, A.; and Smith, S. F. 1999. An iterative sampling procedure for resource constrained project scheduling with time windows. In *IJCAI-99*, 1022–1029.

Geffner, H., and Pearl, J. 1992. Conditional entailment: Bridging two approaches to default reasoning. *Artificial Intelligence* 53:209–244.

Grosof, B. 1991. Generalizing prioritization. In *KR'91*, 289–300. Cambridge, MA: Morgan Kaufmann.

ILOG. 1997. Ilog Scheduler. Reference manual and User manual. V4.0, ILOG.

Joslin, D. E., and Clements, D. P. 1998. "Squeaky Wheel" Optimization. In *AAAI-98*, 340–346.

Junker, U., and Brewka, G. 1991. Handling partially ordered defaults in TMS. In Kruse, R., and Siegel, P., eds., *Symbolic and Quantitative Aspects for Uncertainty. Proceedings of the European Conference ECSQAU*. Berlin: Springer, LNCS 548. 211–218.

Junker, U., and Konolige, K. 1990. Computing the extensions of autoepistemic and default logics with a truth maintenance system. In *AAAI-90*, 278–283. Boston, MA: MIT press.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *IJCAI-95*.

Le Pape, C.; Couronné, P.; Vergamini, D.; and Gosselin, V. 1994. Time-versus-capacity compromises in project scheduling. In *Proc. of Thirteenth Workshop of the UK Planning Special Interest Group*.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling. *Artificial Intelligence* 58:161–205.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32:57–952.

Smith, S., and Cheng, C. 1993. Slack-based heuristics for constraint satisfaction scheduling. In *AAAI-93*.