Planning as Branch and Bound: A Constraint Programming Implementation

Héctor Palacios[1] and Héctor Geffner[2]

[1] Universidad Simón Bolívar, Departamento de Computación
Caracas 1080-A, Venezuela
hlp@ldc.usb.ve

[2] Universitat Pompeu Fabra, Departamento de Tecnología – ICREA
08003 Barcelona, Spain
hector.geffner@tecn.upf.es

Abstract

Branching and lower bounds are two key notions in heuristic search and combinatorial optimization. Branching refers to the way the space of solutions is searched, while lower bounds refer to approximate cost measures used for focusing and pruning the search. In AI Planning, admissible heuristics or lower bounds have received considerable attention and most current optimal planners use them, either explicitly or implicitly (e.g., by relying on a plan graph). Branching, on the other hand, has received less attention and is seldom discussed explicitly. In this paper, we make the notion of branching in planning explicit and relate it to branching schemes used in combinatorial optimization. We analyze from this perspective the relationship between heuristic-search, constraint-based and partial-order planning, and between planning and scheduling. We also introduce a branch-and-bound formulation for planning that handles actions with durations and use unary resources, and consider a range of theories that stand halfway between planning and scheduling. The goals are twofold: to make sense of various techniques that have been found useful in planning and scheduling in a unified framework, and to lay the ground for systems that can effectively combine both capabilities. We have also implemented a planner based on this formulation on top of a constraint programming language and present some preliminary results.

Keywords: artificial intelligence, planning, branch and bound, constraint programming, scheduling, heuristics, branching

1. states σ are TSPs
2. initial state σ_0 is original TSP
3. lower bound $f(\sigma)$ computed by solving AP relaxation of σ; result yields one or more subtours
4. terminal goal states are states whose relaxation yields single tour
5. non-terminal states σ expanded by selecting an edge $i \rightarrow j$ from a subtour and generating the children σ_{ij}^{+} and σ_{ij}^{-}: the first, forcing the edge $i \rightarrow j$ in the solution, the second, excluding it.

Fig. 1. Basic Branch and Bound Scheme for Asymmetric TSPs

1 Branch and Bound in Combinatorial Optimization

The notion of *lower bounds* used in Combinatorial Optimization [1, 12] is familiar in AI where they are called *admissible heuristics* [33, 34]. Branching, on the other hand, is a less familiar notion, and the word does not even appear in the index of AI textbooks. This is probably due to the close correspondence between branching and *action application* in the search problems most often considered in AI such as the 15-puzzle or Rubik's Cube, where a state is *expanded* by applying all possible actions either forward or backward. We will refer to these forms of branching as *branching on actions* or *directional branching.* Directional branching is pervasive in AI yet, as known in combinatorial optimization and constraint programming, it's not always the best way for structuring the search.

1.1 Traveling Salesman Problem

The TSP is the problem of finding a tour with minimum cost through a given number of cities [1]. One of the most powerful approaches for solving it relies on the lower bound obtained from a relaxation of the TSP known as the *assignment problem* [1, 43].[3] The assignment problem or AP is the problem of assigning to each 'city' i a unique next 'city' $next(i) = j$ so that the sum of the distances c_{ij} from city i to city j is minimized. Such an assignment may result in a unique tour or multiple disjoint tours. In both cases, the cost of the assignment is a *lower bound* on the cost of the TSP. In the first case, in addition, the lower bound is *exact* and the optimal assignment represents an optimal tour. These properties are used in the standard branch-and-bound scheme for the TSP shown in Fig. 1 [1, 43]. Notice that branching is not done by applying the actions of going from one city to its neighbors but by making *commitments*; namely, forcing certain edges in or out of the solution. These commitments form a *partial tour* very much as the causal links and precedence constraints define *partial plans* in partial order planning [41].

This branch-and-bound *scheme* can be used in the context of a number of branch-and-bound *algorithms* such as IDA* (Iterative Deepening A*) or DFS B&B (Depth First Search Branch and Bound)[25]. In all branch-and-bound algorithms, the main operation is the *evaluation of the pruning condition* $f(\sigma) \leq B$ for a bound B and each node σ; if the condition is not satisfied, node σ is pruned.

For computing *optimal* solutions, a branch-and-bound scheme must be *sound* and *complete* in the following sense: goal states must represent solutions to the problem, and some goal states must represent *optimal* solutions. Provided these two conditions, any admissible branch-and-bound algorithm will find an optimal solution. Both properties are easy to verify for the scheme in Fig. 1.

1.2 Job Shop Scheduling

The job-shop problem (JSP) is defined by a set of jobs $j_1, \ldots, j_m$, each consisting of a chain of tasks $t_{i1}, \ldots, t_{in}$, $i = 1, \ldots, m$, with durations $D(t_{ij})$ that must be executed in order over a set of n unary resources $R(t_{ij})$ [35]. A *feasible schedule* is an assignment of times $T(t_{ij})$ to each task t_{ij} so that the *precedence constraints* in the jobs and the *resource constraints* are all satisfied. Such constraints can be written as

$$t_{ij} \prec t_{i,j+1} \quad \text{(precedence constraints)} \tag{1}$$

[3] Actually, this method is best for *asymmetric* TSPs when costs between pairs of cities are not symmetric.

1. states σ are simple temporal problems (STPs)
2. initial state σ_0 given by precedence constraints in jobs
3. lower bound $f(\sigma)$ and relaxed schedule h_σ obtained by solving STP σ; $f(\sigma) = \infty$ if STP inconsistent
4. terminal states of two types: σ is a *dead-end* if $f(\sigma) = \infty$ and a *goal state* if no pairs of tasks in conflict in h_σ
5. children generated from σ by selecting pair of tasks t and t' in conflict in σ, and generating two children $\sigma + \{t \prec t'\}$ and $\sigma + \{t' \prec t\}$

Fig. 2. Basic Branch and Bound Scheme for Job Shop

for $i \in [1 \ldots m]$, $j \in [1 \ldots n-1]$, and

$$R(t_{ij}) = R(t_{kl}) \;\Rightarrow\; t_{ij} \prec t_{kl} \;\vee\; t_{kl} \prec t_{ij} \quad \text{(resource constraints)} \tag{2}$$

for $i, k \in [1 \ldots m]$, $j, l \in [1 \ldots n]$, where $t \prec t'$ stands for $T(t) + D(t) \leq T(t')$.

An *optimal schedule* is a feasible schedule with *min makespan* where the makespan is the time at which all tasks have been completed.

Once again, it's possible to formulate a directional branching scheme in which actions (sets of parallel tasks) are applied either forward or backward, yet the resulting branching factor is too high and more effective schemes have been developed. Here we'll follow the approach in [10], that extends ideas from [7] and [39]. In this scheme, each state σ is a relaxed JSP that includes all precedence constraints in the original JSP but no resource constraints. The relaxed problem corresponds to a *Simple Temporal Problem* [14], which is tractable and can be solved by a number of shortest-path and constraint-propagation algorithms[9, 19]. The algorithms determine the consistency of the STP σ and if so return lower bounds $h_\sigma(t)$ on the times at which each task t may start, and hence, a lower bound $f(\sigma)$ on the makespan. We'll refer to the schedule in which each task t_{ij} is executed at its lower bound $h_\sigma(t_{ij})$ as the *relaxed schedule* and denote it by h_σ. It's known that the relaxed schedule satisfies the STP σ, and thus is a solution of the original JSP iff it satisfies the resource constraints (2). Otherwise, there must be a pair of tasks t and t' *in conflict* in σ; namely, tasks that use the same resource and whose execution overlaps in h_σ. These ideas underlie the basic branch-and-bound scheme for the Job Shop shown in Fig. 2. For more sophisticated techniques, see [2].

2 Branch-and-Bound Schemes in AI Planning

In traditional AI planning, planning tasks are expressed in Strips by means of a set A of atoms or boolean variables of interest, a set O of operators, an initial situation $I \subseteq A$, and a goal situation $G \subseteq A$ [17]. Operators a are defined by means of three sets of atoms, an add list $add(a)$ with the atoms that become true after doing action a, a delete list $del(a)$ with the atoms that become false after a, and the precondition list $pre(a)$ with the atoms that need to be true for the action a to be applicable. The basic planning task is to find a sequence of operators or plan that maps the initial situation I into the goal G.

Branching in AI planning is discussed implicitly in terms of the *space* in which the search for plans is done. State or directional planners are said to search in the space of states, while partial order planners are said to search in the space of plans. This is a useful distinction, yet it does not always reveal what these approaches have in common or what they have in common with other approaches such as those based on SAT (Propositional Satisfability) or CSP (Constraint Satisfaction Problems) formulations (yet see [37]). All planners, indeed, search in the space of plans; directional planners just exploit a decomposition property for which a partial plan tail or head σ can be summarized by the state s obtained by regressing the goal or progressing the initial state through the partial plan σ. Indeed, the completion of σ is independent of σ given the state s, and similarly, the estimated cost $f(\sigma)$ of the best completion of σ can be decomposed into the accumulated cost $g(\sigma)$, that depends only on σ, and the heuristic cost $h(s)$, that depends only on s. This decomposition does not hold in non-directional planners, yet the idea of using an estimated cost function $f(\sigma)$ or similar mechanism for guiding and pruning the search remains feasible and necessary for scaling up.

2.1 State Planning

Modern state planners for *sequential planning* branch on the set of applicable operators either forward from the initial state or backward from the goal, and use an heuristic function $h(s)$ for guiding or pruning the search. A useful heuristic can be obtained by ignoring deletes and assuming that *the cost to achieve a set of atoms is given by the sum of the costs of achieving each atom in the set.* This 'additive' heuristic is not admissible but is often quite effective [30, 6]. An admissible (lower bound) 'max' heuristic can be obtained by approximating *the cost to achieve a set of atoms by the cost of the most costly atom in the set.* A family of informative and admissible heuristics h^m, for $m = 1, 2, \ldots$ that generalizes the 'max' heuristic is formulated in [20]. The heuristic h^m approximates *the cost of a set of atoms C by the cost of the most costly subset of size m in C.* For $m = 1$, the heuristic h^m is equivalent to the max heuristic, while for $m > 1$, more informative but more expensive heuristics are obtained. A formulation of these heuristics for *parallel planning* is also developed in [20] where it's shown that the heuristic h^m, for $m = 2$, is equivalent to the heuristic underlying Graphplan [3]. More recently, [21] shows how the heuristics h^m can be extended to estimate *makespan* (completion time) in a temporal setting where actions can be executed concurrently and have different durations. The equation characterizing the temporal heuristic h^m_T for $m = 1$ is

$$h^1_T(C) = \begin{cases} 0 & \text{if } C \subseteq s_0 \text{, else} \\ \min_{a \in O(p)} [D(a) + h^1_T(prec(a))] & \text{if } C = \{p\} \text{, else} \\ \max_{p \in C} h^1_T(\{p\}) & \text{if } |C| > 1 \end{cases} \tag{3}$$

where $O(p)$ stands for the set of actions a that add p and $D(a)$ stands for the duration of a. The measures $h^m_T(C)$ are all lower bounds on the time needed to make the set of atoms C true. The regression-based temporal planner in [21] uses the heuristic h^m_T for $m = 2$. Below, for simplicity, we use the h^1_T estimator. Equations like (3) above are solved in polynomial time using variations of single source shortest path algorithms.

[20] reports results for an optimal planner based on regression search and the heuristic h^2. The planner is competitive and often superior to the best Graphplan and SAT/CSP planners in the *sequential setting,* but is not as good as the latter in the *parallel setting.* The problem is that the *branching factor* grows exponentially in either forward or backward *parallel* planning. Indeed, if there are n primitive operators applicable in a state s, there are up to 2^n possible *parallel actions.*

2.2 Partial Order Planning

Partial order Planning (POP) refers to a non-directional branching scheme used in AI Planning for many years [41]. POP planners are not competitive with modern planners because the latter rely on some form of heuristic estimation or pruning mechanisms, while POP is a pure branching scheme. The performance of POP planners can be enhanced through the addition of heuristic estimators (e.g., see [32]), although deriving *effective lower bounds* in the POP setting appears to be more difficult than deriving similar bounds in state planning. We consider partial-order planning here because it represents a branching scheme that is particularly suited for more expressive forms of planning such as *temporal planning with resources.* Indeed, two of the most expressive *temporal planners,* IxTeT [26] and RAX [22] are based on partial-order planning schemes. The potential advantages of POP in this setting have been discussed in [38]. The difficulty of deriving useful lower bounds will be addressed below where we show how the h^m_T heuristics can be modified to estimate the *completion time of partial plans.*

2.3 SAT and CSP Branch and Bound

Most SAT and CSP formulations of optimal planning (e.g., [23, 40, 16, 36, 28]) can be understood as branch-and-bound schemes in which the *states* σ are partial variable assignments and *branching* is performed by selecting a variable and extending the partial assignment with each of its possible values. However, rather than computing explicit *lower bounds* $f(\sigma)$ and evaluating the pruning condition $f(\sigma) \leq B$ for a bound B as in standard branch-and-bound algorithms, SAT and CSP formulations represent the condition $f(\sigma) \leq B$ as an explicit *constraint* which is checked for consistency using some form of constraint propagation. For example, in a SAT planner such as Blackbox [23], when the goal is $G = \{p, q\}$ and the bound is 10, this condition takes the form of two clauses p_{10} and q_{10} that are added to the theory. Explicit computation of lower bounds and consistency checking through constraint propagation are two alternative methods for pruning the search which as shown in [8, 18], can often be combined.

3 New Branch-and-Bound Formulations

The notions of branching and bounds make the relationships between the different approaches in planning more explicit and relate planning with other combinatorial optimization problems. Now, we'll take advantage of this view to introduce some novel branch-and-bound formulations that integrates a number of the ideas we have discussed: non-directional plans, lower bounds, and constraints.

3.1 Preliminary Definitions

We consider a simple extension of Strips where each action a has a duration $D(a)$ and uses a set of unary resources $R(a)$. We assume durations to be positive integers except for the dummy actions *Start* and *End*, as used in partial-order planning, that have zero durations.

Two actions a and a' are *mutex* when a) a and a' require a common resource, i.e., $R(a) \cap R(a') \neq \emptyset$, or b) a and a' interact destructively, i.e., a deletes a precondition or positive effect of a', or a' deletes a precondition or positive effect of a [3].

A *schedule* P is a finite set of time stamped actions $\langle a_i, t_i \rangle$, $i = 1, \ldots, n$, where a_i is an action and t_i is a non-negative integer. We say that a_i *precedes* a_j in P, and write $a_i \prec a_j$, if $t_i + D(a_i) \leq t_j$, and say that a_i and a_j *overlap* in P when $a_i \not\prec a_j$ and $a_j \not\prec a_i$. A schedule P is a *valid plan* iff mutex actions do not overlap in P and for every action a_i its preconditions $p \in prec(a)$ are true at time t_i. This condition is defined inductively from $t = 0$ in a direct way.

From now on, a plan will refer to a *valid* plan. The *makespan* of a plan P is the min time at which all goals are true. We are interested in computing a valid plan P with minimum makespan.

3.2 Disjunctive Branching for Positive Theories

We'll assume initially a class of domains where actions have durations and use unary resources but have *no deletes.* We call these theories *positive.* Positive domains are restricted from the point of view of planning, but are quite general from a scheduling point of view; indeed, they stand for a generalization of the job shop where there may be alternative tasks for achieving a job, alternative resources, arbitrary preconditions, etc. We'll exploit two properties of positive domains that make the formulation simpler, namely that

1. causal links are not needed for preserving the truth of atoms, and
2. no operator needs to be executed more than once.

The second property generalizes the condition found in most scheduling problems in the literature where *all* operators are executed *exactly* once. Thus positive theories, stand halfway between temporal planning and the job shop.

The branching scheme for the job shop (Fig. 2) can be easily generalized to positive theories. The main departure is the definition of the initial state σ_0. In the job shop, σ_0 is defined as the set of precedence constraints

$$T(t_{ij+1}) \geq D(t_{ij}) + T(t_{ij}) \tag{4}$$

for each pair of successive tasks t_{ij} and t_{ij+1} in the jobs. In planning, tasks (actions) are not ordered *explicitly* by precedence constraints but *implicitly* by their preconditions. This implicit ordering can be rendered explicit by means of equations similar to the ones characterizing the temporal estimators h^m_T. For example, Equation 3 for h^1_T can be rewritten as:

$$h^1_T(C) = \begin{cases} \min_{a \in O(p)} [D(a) + h^1_T(a)] & \text{if } C = \{p\}\text{, else} \\ \max_{p \in C} h^1_T(\{p\}) & \text{if } |C| > 1 \end{cases} \tag{5}$$

where we assume now the presence of the actions *Start* and *End*,[4] and $h^1_T(a)$ is a lower bound on the time needed to start the execution of action a

$$h^1_T(a) = h^1_T(prec(a)) \tag{6}$$

[4] $h^1_T(Start) = 0$, $D(Start) = 0$, and $Start \in O(p)$ if p is true in the initial situation.

1. states σ are Extended STPs
2. initial state σ_0 given by precondition constraints (8)
3. relaxed plan h_σ obtained by solving ESTP σ; $f(\sigma) = h_\sigma(End)$
4. terminal goal states σ if mutex constraints (9) not violated by any pair of actions a and a' in relaxed schedule h_σ
5. branching from non-terminal σ done by picking one such pair of actions a and a' and generating children $\sigma + \{a \prec a'\}$ and $\sigma + \{a' \prec a\}$

Fig. 3. Branch and Bound Scheme for Positive Planning Theories

These equations on lower bounds can be made to look similar to precedence constraints (4) by means of two transformations. First, we project the equations on actions by unfolding the left-hand-side of Equation 6 to get

$$h^1_T(a) = \max_{p \in prec(a)} \left\{ \min_{a' \in O(p)} [D(a') + h^1_T(a')] \right\} \tag{7}$$

Second, we express the resulting equation on lower bounds as an equation on *temporal variables* $T(a)$, where $T(a)$ is a variable that stands for the *time at which action a is executed* in the plan and whose domain is the set of non-negative integers extended with ∞. Intuitively, $T(a) = \infty$ means that action a is not executed in the plan and we assume that $a' \prec a$ for all a' when $T(a) = \infty$.

Recasting Equation 7 on the temporal variables $T(a)$, we obtain the set of *temporal constraints:*

$$T(a) \geq \min_{a' \in O(p)} [D(a') + T(a')] \quad \text{for each } p \in prec(a) \tag{8}$$

These constraints are similar to the precedence constraints (4) for the job shop except for the **min** operator, and we call them *precondition constraints.* It's not difficult to show that precedence and precondition constraints have similar computational properties: they are tractable, they can be solved by simple variations of shortest path algorithms, and the lower bounds $h(a)$ obtained for each temporal variable $T(a)$ define a consistent solution. We'll call the theories that combine precondition and precedence constraints, Extended STPs. Due to the inclusion of ∞ in the domain of the temporal variables, ESTPs are always consistent as the assignment $T(a) = \infty$ is always a solution. Of course, we are interested in the lowest consistent value of these variables, and in particular in the lower bound of the variable $T(End)$ which provides the lower bound on the state; i.e., $f(\sigma) = h_\sigma(End)$.

The branch-and-bound scheme for *positive planning theories* (Fig. 3) follows from the scheme for the job-shop (Fig. 2) due to the similarity between feasible schedules in the job-shop and valid plans in positive theories. To show this, let's *represent* an assignment over the variables $T(a)$ by a list of pairs $P = \langle a_i, t_i \rangle_i$ so that $T(a_k) = t_k \neq \infty$ if $a_i \in P$, and $T(a_k) = \infty$ if $a_k \notin P$. Then we have that:

Proposition 1. *$P = \langle a_i, t_i \rangle_i$ is a* valid plan *iff the assignment P satisfies the precondition constraints (8) and the mutex constraints*

$$mutex(a, a') \Longrightarrow a \prec a' \ \vee \ a' \prec a' \tag{9}$$

Valid plans in this setting are thus similar to feasible schedules in the job shop, from which the similarity between the corresponding branch-and-bound schemes follows.

3.3 Causal Link Branching in Presence of Deletes

Positive theories are more general than the job-shop as they involve arbitrary preconditions and positive postconditions, and not all actions need to be scheduled. Still the restriction of 'no deletes' is a strong one from the point of view of planning. This restriction guarantees that causal links are not needed and that no operator needs to be scheduled more than once (conditions 1 and 2 above). Now we will relax the first assumption. We'll refer to a plan in which no operator is scheduled more than once as a *canonical plan.* The scheme we develop below is suitable for solving planning theories in which *some optimal plans are canonical.* Theories such as the `blocks-world` are canonical in this sense, some instances of `logistics` are not. Similarly, the theories considered in scheduling (where all tasks are scheduled *exactly* once) and positive planning theories are canonical too.

1. states $\sigma = \langle Precs, Doms \rangle$, where $Precs$ are precedence and precondition constraints, and $Doms$ stand for the domains of the temporal and support variables $T(a)$ and $S(p, a)$
2. initial state $\sigma_0 = \langle Precs_0, Doms_0 \rangle$, where $Precs_0$ stands for the precondition constraints (10) and $Doms_0$ for initial domains of temporal and support variables
3. relaxed schedule h_σ computed by solving ESTP $Precs$; lower bound $f(\sigma) = h_\sigma(End)$
4. terminal goals states $\sigma = \langle Precs, Doms \rangle$ if mutex constraints (9) violated by no action pair (a, a'), and causal link constraints (11) violated by no action triplet (a, a', a'') in relaxed schedule h_σ
5. children generated from non-terminal state $\sigma = \langle Prec, Doms \rangle$ by selecting a mutex conflict (a, a') and branching $[a \prec a'; a' \prec a]$, or a causal link conflict (a, a', a'') and branching $[S(p, a) \neq a'; S(p, a) = a', a'' \prec a; S(p, a) = a', a' \prec a'']$ (updating $Precs$ and $Doms$ accordingly).

Fig. 4. Branch and Bound for Canonical Planning

In the presence of deletes, the scheme above is neither sound nor complete, as the preconditions of an action may be deleted before the action is executed. *Causal links,* as introduced in [29], allow us to detect such conditions and fix them. A causal link $a \rightarrow_p a'$ states that action a precedes a', makes its precondition p true, and no action a'' that deletes p is scheduled between a and a'.

Causal links can be used as in POP, yet we'll use them differently in order to get *better lower bounds.* We introduce finite domain variables $S(p, a)$ for each precondition p of action a which we call *support variables.* Initially, the domain $D(p, a)$ of the support variable $S(p, a)$ is $O(p)$, i.e., the set of operators that add p, yet this domain changes dynamically during the search. In particular, a causal link $a' \rightarrow_p a$, is asserted by setting $S(p, a) = a'$ and retracted by setting $S(p, a) \neq a'$. Provided this representation of causal links, the precondition constraints can be written as

$$T(a) \geq \min_{a' \in D(a,p)} [T(a') + D(a')] \ \text{ for each } p \in pre(a) \tag{10}$$

where the minimization is done over the *dynamic* domain $D(a, p)$ and not over the *static* domain $O(p)$. The planning task can then be expressed as a *constraint-satisfaction problem* as follows.

Proposition 2. *$P = \langle a_i, t_i \rangle_i$ is a valid canonical plan iff the assignment P over the temporal variables $T(a)$ and* some *assignment over the support variables $S(p, a)$ jointly satisfy 1) the precondition constraints (10), 2) the mutex constraints (9), and 3) the causal link constraints*

$$T(a) \neq \infty \ \&\ S(p, a) = a' \ \&\ p \in del(a'') \ \Rightarrow \ a'' \prec a' \ \vee \ a \prec a'' \tag{11}$$

Like mutex constraints, and unlike preconditions constraints, causal link constraints are *disjunctive* and thus *intractable.* As a result, the resulting branch-and-bound scheme computes relaxed schedules h_σ by considering the precondition and posted precedence constraints only, and branches over the disjunctions on either the mutex or causal link constraints that are violated in h_σ (we call them the mutex and causal link conflicts; causal link conflicts are similar to *threats* in partial-order planning). The resulting scheme is shown in Fig. 4.

3.4 Goal Oriented Branching

In partial-order-planning and planners such IxTeT and RAX, the set of operators *Steps* to be scheduled is built incrementally, starting with the actions *Start* and *End*, and checking for conflicts within *Steps* only. This makes the search more goal-oriented, something that pays off when the set of relevant actions is small in comparison with the set of all available actions. This modification can be easily included in the branch-and-bound scheme above resulting in scheme shown in Fig. 5.

1. states $\sigma = \langle Steps, Precs, Doms \rangle$, **where** $Steps$ **is a set of actions**, and $Precs$ and $Doms$ as before
2. initial state $\sigma_0 = \langle \{Start, End\}, Precs_0, Doms_0 \rangle$ with $Prec_0$ and $Doms_0$ as before
3. relaxed schedule h_σ computed by solving ESTP $Precs$; lower bound $f(\sigma) = h_\sigma(End)$
4. terminal goals states $\sigma = \langle Steps, Precs, Doms \rangle$, **if** $|D(p,a)| = 1$ **for all** $a \in Steps$, and in relaxed schedule h_σ, no mutex or causal link conflicts **for actions in** $Steps$**.**
5. children generated from non-terminal state $\sigma = \langle Prec, Doms \rangle$ by selecting a mutex conflict (a, a') and branching $[a \prec a'; a' \prec a]$, or by selecting a causal link conflict (a, a', a'') for which $S(p, a') = a$ and branching $[a'' \prec a; a' \prec a'']$, or **by selecting a domain** $|D(p,a)| > 1$ **for** $a \in Steps$ **and an action** $a' \in D(p,a)$ **and branching** $[S(p,a) = a'; S(p,a) \neq a']$.

Fig. 5. Goal-oriented branch-and-bound scheme for canonical plans

4 Implementation

In order to test the feasibility of these ideas above, we have implemented a planner based on the goal-oriented branching scheme shown in Fig. 5. The planner has the flavor of a partial-order planner yet it builds (optimal) parallel plans (only actions with unit durations are currently handled) and uses the h^1 (h^1_T) heuristic estimator in the form of the precondition constraints (10), among other differences. Also, as we will see, while its performance is not as good as the best optimal SAT and CSP parallel planners, it is as good as or better than Graphplan and HSP parallel planners, something that doesn't hold for (optimal) POP planners. This, and the fact that the implementation can be improved and can be extended to handle actions with non-unit durations, make the approach computationally appealing and worth of further exploration. A limitation of the current planner is that it computes (optimal) *canonical* plans only, hence plans where the same ground action is done multiple times are not considered. This limitation is discussed below.

The planner is implemented on top of the constraint logic programming language GNU Prolog [15] with some routines written in C++. The reasons for a constraint-based implementation are basically two. First, branching decisions are *constraints* and CP languages offer built-in, *incremental mechanisms* for propagating their effects over the temporal and support variables. Second, CP languages offer primitives that allow for *limited forms of reasoning over disjunctions* such as those appearing in *mutex and causal link constraints,* which as we will see, allow for further pruning.

In the planner, the computation of the lower bounds $f(\sigma)$ for partial plans σ and the evaluation of the pruning condition $f(\sigma) \leq B$ for a bound B, are replaced by constraint-propagation and consistency checking over the problem constraints and the goal constraint $T(End) \leq B$ that states that the goal must be achieved by time B.

The problem constraints include the precondition constraints, that currently capture the h^1 estimator, and the constraints that are posted dynamically during the search, that include precedence constraints over the temporal variables, and equality and inequality constraints over the support variables. The states σ in the search, which stand for partial plans, correspond in the implementation to the domains of the temporal and support variables and the constraints over them.

The planner performs a Depth First Search with bound B, which is increased by one when the search fails. The initial value of B is given by the lower bound of the variable $T(End)$ in the initial state σ_0. Branching in this search is done according to the scheme in Fig. 5: a 'flaw' in the partial plan σ is selected and each of its possible fixes is tried until a terminal goal state is obtained or an inconsistency is found. Flaws are of three types: mutex conflicts, causal link conflicts, and open preconditions, and each step is followed by constraint propagation.

This basic implementation does not compete with modern optimal parallel planners such as Graphplan or Blackbox. An important limitation is the use of the h^1 (h^1_T) estimator which is too weak, as it assumes the time needed to make a set of atoms true is given by the time needed to make each *individual atom* in the set true. The h^2 estimator takes into account interactions between *pairs of atoms* and results in a more informed lower bound. Planners such as Graphplan and Blackbox do indeed use the h^2 estimator (for actions with unit duration) which is implicitly encoded in the plan graph [20]. The same is true for the optimal parallel and temporal heuristic search planners HSPr* [20] and TP4 [21]. In our formulation, however, encoding the h^2 estimator results in several thousands of constraints and variables, something that seems to be beyond the capabilities of the CP tools we tried. We thus settled for a compromise and used the h^2 estimator computed in the initial state to prune the domain of the temporal variables $T(a)$ only. More precisely, we compute the $h^2(a)$ values of all actions a, and set the constraints $T(a) \geq h^2(a)$.

In the basic scheme, disjunctive mutex and causal link constraints play a passive role in the search: they are checked in every state, and among those violated by the relaxed plan (the lower bounds of the temporal variables $T(a)$), *one* disjunction is selected for creating a split. In the implementation, we take advantage of the ability to perform a limited form of inference over disjunctions, and post *all* such disjunctions. Then when a disjunct $A \prec B$ becomes false in the search, the other disjunct becomes immediately true, saving splits and backtracks. (see also [32]).

This form of disjunctive reasoning is actually extended in three ways. First, we compute the transitive closure of all posted precedence constraints so that checking the consistency of a precedence relation is done by a lookup operation. Second, we make use of the h^2 values computed over all actions a and all atom pairs p, q to identify *implicit* mutex and causal link conflicts, that generate additional valid disjunctions. [5] Third, and finally, for the purpose of detecting mutex and causal link conflicts and posting the corresponding disjunctions, we treat the support variables $S(p, a)$, that represent the (possibly undetermined) action that supports precondition p of a, as if they were normal actions. Thus for example, when a' is an action in *Steps* that deletes p such that $S(p, a) \prec a' \prec a$ is consistent, a disjunction $a' \prec T(p, a) \vee a \prec a'$ is posted, where $T(p, a)$ is a temporal variable that encodes the starting time of the (possibly undetermined) action that supports precondition p of a.[6] Some additional constraints are used to keep the support variables $S(p, a)$ and the temporal variables $T(p, a)$ in sync. In particular, $a' \prec T(p, a)$ implies $S(p, a) \neq a'$, and $a' \prec T(p, a)$ iff $a' \prec a''$ for all $a'' \in D(p, a)$, where $D(p, a)$ is the domain of the support variable $S(p, a)$. Similar constraints follow for '$\succ$' in place of '$\prec$'.

The planner, that we call BBP for branch-and-bound planner, accommodates the basic algorithm, the extensions above, and a simple 'flaw' selection heuristic based on the criticality of the flaw measured in terms of the slack and the number of conflicts of the actions involved. While further work is needed to tune up the code and the heuristics, the performance of the current planner is promising. Table 6 shows the performance of the planner over a set of widely used benchmarks, in relation to other modern, domain-independent optimal parallel and temporal planners such as Graphplan, Blackbox, IPP [24], HSPr*, and TP4 (we don't include STAN [27] in the comparison, as it uses mechanisms that are adapted to these types of domains). BBP is not as good as Blackbox, yet it appears to be as good or better than Graphplan and the heuristic search planners. At the same time, these results are preliminary and we believe that they can be improved substantially.

5 Discussion

There are three ways in which we believe the current implementation can be improved. First, we need to devote more time to tune up the variable and value selection heuristics (namely, the heuristics for selecting flaws and for ordering the repairs). These heuristics are known to have a big impact on performance. Second, we need to tune up the propagation rules. Right now, a few tens of nodes are generated by second, and thus, while the overall number of nodes is often much smaller than in other planners, this does not always translate in faster times. Third, the precondition constraints used for capturing the h^1_T lower bounds need to be extended to for capturing the more informed h^2_T bounds.

The extensions for accommodating actions with non-unit durations are not implemented yet but are minor. In any case, we believe that *the performance of a temporal planner should approach the performance of the best parallel planners when only actions of unit durations are considered.* No optimal temporal planner has this property yet, nonetheless, we believe that this approach offers a promising alternative.

[5] More precisely, we treat the atom pair p, q as mutually exclusive (mutex) when $h^2(p, q) > B$, where B is the current bound, and say that an action a *e-deletes* an atom p when a deletes p or when a has a precondition or positive effect q that is mutex with p. Then, it can be shown that mutex and causal link constraints remain true when the delete lists of actions a are replaced by extended delete lists, i.e., the lists of atoms p e-deleted by a.

[6] Actually, this is more subtle when a' is one of the possible values of $S(p, a)$. In such case, the disjunction becomes $S(p, a) = a' \vee (S(p, a) \neq a' \ \wedge \ (a' \prec T(p, a) \vee a \prec a'))$.

Problem	BBP	BBOX	GPLAN	IPP	TP4	HSPR*
bw-large.a	3.04	4.8	0.35	0.15	0.92	0.22
bw-large.b	25.8	-/-	26.53	6.06	-	32
bw-large.c	-	-/-	-/-	-/-	-	-
rocket.ext.a	40.63	2.67	-/-	16.36	184	85
rocket.ext.b	54.13	3.81	-/-	39.86	148	52
logistics.a	2.08	2.83	-/-	-/-	-	-
logistics.b	1094.97	10.65	-/-	-/-	-	-
logistics.c	NC	1743.7	-/-	-/-	-	-

Fig. 6. Results of some optimal parallel and temporal planners over set of benchmarks. Entries report times measured in seconds. All experiments ran on a PIII 500 MHZ machine with 250M RAM per run. Runs cut after 2000 secs. Symbols '-' and '-/-' mean that planner ran out of time or memory respectively. The Graphplan (GPlan) implementation used from the Blackbox (BBox) distribution; Blackbox ran with Satz Solver. NC above means that the optimal plans are not canonical.

The most serious limitation in the planner and in its underlying formulation is the consideration of canonical plans only where ground actions are executed at most once. One way to overcome this limitation is to replace the dynamic precondition constraints (10), that depend on the *current* domain $D(a,p)$ of the support variables $S(a,p)$, by the static precondition constraints (8) that depend on their *initial* domains only. In this way, arbitrary number of action 'copies' can be added to the plan, but the resulting lower bound propagation becomes weaker. On the other hand, if it is known that actions can be repeated, say, 2 times, then it is sufficient to add a copy a' of each action a in the domain, along with the constraint that a' cannot be in the plan when a is not (this is a form of symmetry pruning). Still, the best way to go from canonical to non-canonical plans in this setting remains open. Nonetheless, the focus on plans in which every action is done *at most once,* is a sensible way to approach the integration of planning and scheduling where actions are normally done *exactly* once.

6 Related Work

The view of *planning as branch-and-bound* is an extension of the view of planning as heuristic search [30, 5, 6] which is tied to directional branching schemes. The use of heuristics in non-linear planning has been considered recently in [32] where a high-performance non-optimal POP planner is introduced. While our BBP planner has been developed independently, there are a number of ideas that are common to both planners such as the ideas of exploiting mutex information and reasoning with disjunctions. Rintanen in [36] explains the limitations of Graphplan and the benefits of Blackbox in terms of a non-directional branching scheme. His approach is further elaborated in [28]. Systems that accommodate planning and scheduling capabilities include IxTeT [26] and RAX [22] (yet see also [31, 11, 13, 42]) although their domain-independent performance is weak due to the use of poor lower bounds. CPlan [40] is a high performance constraint-based planner that requires a highly tuned model of the domain to operate. Finally, [4] is a planner based on an integer programming formulation that does IP branch-and-bound using a standard IP solver.

7 Summary

Heuristic search planning appears currently as the best approach for sequential planning. On the other hand, for temporal or parallel planning, the directional branching schemes used in heuristic search planning lead to a large branching factor and weak performance. In this paper we have laid out a framework for integrating lower bounds, as used in heuristic search planners, with non-directional branching schemes, as found in in partial-order planning and SAT/CSP formulations. We have also tried to clarify the relationship between planning and scheduling techniques by considering a number of planning theories whose expressive power lies somewhere between the job-shop and temporal planning. We have also implemented a planner on top of a constraint programming system, and obtained some preliminary results that show the potential of this approach for developing high-performance domain-independent temporal planners that exploit recent advances in classical planning.

Acknowledgements. Part of this work was done while the second author paid visits to Nasa Ames and the Universita di Genova. He wants to thank Nicola Muscettola and Enrico Giunchiglia for their hospitality and a number of useful discussions about these topics. We've also benefited from discussions with P. Haslum, P. Laborie, C. Beck, S. Kambhampati, D. Smith, A. Jonsson, J. Frank, and P. Morris. We also thank Daniel Diaz for help with GNU Prolog.

References

1. E. Balas and P. Toth. Branch and bound methods. In E. L. Lawler *et al.*, editor, *The Traveling Salesman Problem*, pages 361–401. John Wiley and Sons, Essex, 1985.
2. J. Beck and M. Fox. A generic framework for constraint-directed scheduling. *AI Magazine*, 19(4), 1998.
3. A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, pages 1636–1642. Morgan Kaufmann, 1995.
4. A. Bockmayr and Y. Dimopoulos. Integer programs and valid inequalities for planning problems. In *Proceedings of ECP-99*. Springer, 1999.
5. B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proceedings of ECP-99*, pages 359–371. Springer, 1999.
6. B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.
7. J. Carlier and E. Pinson. An algorithm for solving the job shop scheduling problem. *Management Science*, 35(2), 1989.
8. Y. Caseau and F. Laburthe. Improved CLP scheduling with task intervals. In *Proc. ICLP-94*, pages 369–383. MIT Press, 1994.
9. R. Cervoni, A. Cesta, and A. Oddi. Managing dynamic temporal constraint networks. In *Proc. AIPS-94*, pages 13–20, 1999.
10. A. Cesta, A. Oddi, and S. Smith. Profile based algorithms to solve multicapacited metric scheduling problems. In *Proc. AIPS-98*, 1998.
11. S. Chien *et al.* Aspen – automating space mission operations using automated planning and scheduling. In *Proc. SpaceOps2000*, Toulouse, France, 2000.
12. William J. Cook and William H. Cunningham. *Combinatorial Optimization*. John Wiley and Sons, 1997.
13. K. Currie and A. Tate. O-plan: The open planning architecture. *Artificial Intelligence*, 52(1):49–86, 1991.
14. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
15. D. Diaz and P. Codognet. Design and implementation of the GNU-Prolog system. *Journal of Functional and Logic Programming*, 2001. System at `http://pauillac.inria.fr/ diaz/gnu-prolog`.
16. Minh Binh Do and Subbarao Kambhampati. Solving planning-graph by compiling it into CSP. In *Proc. AIPS-00*, pages 82–91, 2000.
17. R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971.
18. F. Focacci, A. Lodi, and M. Milano. Solving TSPs with time windows with constraints. In *Proc. ICLP-99*. MIT Press, 1999.
19. A. Gerevini, A. Perini, and F. Ricci. Incremental algorithms for managing temporal constraints. Technical report, IRST, Italy, 1996. Tec. Rep. IRST-9605-07.
20. P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proc. of the Fifth International Conference on AI Planning Systems (AIPS-2000)*, pages 70–82, 2000.
21. P. Haslum and H. Geffner. Heuristic planning with time and resources. In *Proc. ECP-01*, 2001.
22. A. Jonsson, P. Morris, N. Muscettla, and K. Rajan. Planning in interplanetary space: Theory and practice. In *Proc. AIPS-2000*, 2000.
23. H. Kautz and B. Selman. Unifying SAT-based and Graph-based planning. In T. Dean, editor, *Proceedings IJCAI-99*, pages 318–327. Morgan Kaufmann, 1999.
24. J. Koehler, B. Nebel, J. Hoffman, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In S. Steel and R. Alami, editors, *Recent Advances in AI Planning. Proc. 4th European Conf. on Planning (ECP-97). Lect. Notes in AI 1348*, pages 273–285. Springer, 1997.
25. R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.
26. P. Laborie and M. Ghallab. Planning with sharable resources constraints. In C. Mellish, editor, *Proc. IJCAI-95*, pages 1643–1649. Morgan Kaufmann, 1995.
27. D. Long and M. Fox. The efficient implementation of the plan-graph. *JAIR*, 10:85–115, 1999.
28. S. Marcugini M. Baioletti and A. Milani. DPPlan: An algorithm for fast solution extraction from a planning graph. In *Proc. AIPS-2000*, 2000.
29. D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of AAAI-91*, pages 634–639, Anaheim, CA, 1991. AAAI Press.

30. D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proc. Third Int. Conf. on AI Planning Systems (AIPS-96)*, 1996.
31. N. Muscettola. HSTS: Integrating planning and scheduling. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.
32. Xuan Long Nguyen and Subbarao Kambhampati. Reviving partial order planning. In *Proc. IJCAI-01*, 2001.
33. N. Nilsson. *Principles of Artificial Intelligence*. Tioga, 1980.
34. J. Pearl. *Heuristics*. Addison Wesley, 1983.
35. M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 1995.
36. J. Rintanen. A planning algorithm not based on directional search. In *Proceedings KR'98*, pages 617–624. Morgan Kaufmann, 1998.
37. B. Srivastava S. Kambhampati. Universal classical planner: An algorithm for unifying state-space and plan-space planning. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning*. IOS Press (Amsterdam), 1996.
38. D. Smith, J. Frank, and A. Jonsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1), 2000.
39. S. Smith and C. Cheng. Slack-based heuristics for the constraint satisfaction scheduling. In *Proc. AAAI-93*, pages 139–144, 1993.
40. P. Van Beek and X. Chen. CPlan: a constraint programming approach to planning. In *Proc. National Conference on Artificial Intelligence (AAAI-99)*, pages 585–590. AAAI Press/MIT Press, 1999.
41. Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.
42. D. Wilkins. *Practical Planning: Extending the classical AI paradigm*. M. Kaufmann, 1988.
43. W. Zhang and R. Korf. Performance of linear-space search algorithms. *Artificial Intelligence*, 79:241–292, 1995.