# Planning as Branch and Bound and its Relation to Constraint-based Approaches

Héctor Geffner
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
hector@usb.ve

***** DRAFT *****

**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 recently 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 the literature.

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. The goals are twofold: to make sense of the various techniques that have been found useful in planning and scheduling, and to lay the ground for systems that can effectively combine both.

## 1 Introduction

In this paper we focus on the problem of planning with Strips actions that have durations and require unary resources. This problem is a generalization of problems found in planning and scheduling. On the one hand, it generalizes the job-shop [35], on the other, it generalizes sequential and parallel planning [5]. The resulting planning theories are challenging from a computational point of view and capture more closely the type of planning tasks that are relevant in practice. Currently, there are few planners with these capabilities (e.g., IxTeT [27] and RAX [23]) and the performance of such planners appears rather weak. Indeed, such planners are based on the ideas of partial-order planning, an approach that prevailed in planning research for a number of years [40], but which recently has been dominated by approaches based on Graphplan, SAT, and heuristic-search formulations [5, 24, 6].

The goal of this paper is to identify the ideas that have been found useful in planning, in particular, heuristics, constraint propagation, and non-linear plans, and to analyze how these ideas can be fit together in order to have an *effective temporal planner.* For this, we make use of two key notions in heuristic search and combinatorial optimization: *branching* and *lower bounds* [34, 1, 12]. 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 recently, and most optimal planners currently use them, either explicitly or implicitly (e.g., by relying on a plan graph) [6, 19]. 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 argue that the directional branching schemes used in heuristic search planning are not adequate for *parallel* and *temporal planning,* and introduce an alternative branch-and-bound formulation that is more convenient. The formulation integrates ideas from heuristic search, constraint-based, and partial-order planning in a coherent way, and provides a suitable platform for integrating planning and scheduling [37].

This work is related to a number of recent proposals, in particular, those dealing with the use of constraint-propagation methods for extracting plans from the plan graph [36], and the use of heuristic estimators to guide partial-order planners [31]. We aim, however, for a general framework based on the notions of branching and bounds, where these and other proposals can be understood and integrated.

The paper is organized as follows. First, we illustrate the notions branching and bounds by considering two problems from combinatorial optimization: the traveling salesman problem and the job shop (Sect. 2). In both cases, it's possible to do directional branching yet that's not as effective as constructing 'partial' tours and schedules – very much as it's done in partial-order planning – and using lower bounds on such partial solutions to prune the search. Then we review some of the lower bounds that have been proposed recently in sequential and temporal planning (Sect. 3), and analyze heuristic-search, partial-order, and SAT/CSP planners as branch-and-bound schemes (Sect. 4). We then show how *lower-bounds* as used in heuristic-search approaches, *branching* as done in partial-order planning, and *constraints* as used in SAT and CSP approaches can all be integrated in a general branch-and-bound scheme for temporal planning (Sect. 5). We finally discuss implementation-related issues and refinements, and related work (Sect. 6 and 7).

## 2 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* [32, 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. Indeed in problems like the 15-puzzle or Rubik's Cube, a state is *expanded* by applying all possible actions, in planning, a state is expanded by applying all possible actions either forward or backward, etc. We will refer to these forms of branching as *branching on actions* or *directional branching.* Directional branching is pervasive in AI yet it's not always the best way for structuring the search. To emphasize this point, we consider two problems from combinatorial optimization where non-directional branching schemes yield superior performance. In both cases, the branching scheme is tightly integrated with lower bound computations.

### 2.1 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, 42].[1] 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 branch-and-bound scheme for the TSP shown in Fig. 1. [2]

This branch-and-bound scheme can be used in the context of a number of branch-and-bound *algorithms* such as IDA* or Depth First Search Branch and Bound [26]. For example, a complete IDA* algorithm for the TSP can be obtained by defining the selection of the edge $i \rightarrow j$ on which to branch and the order in which the two branches are considered (Step 5). In all branch-and-bound algorithms, the main operation is the *evaluation of the pruning condition* $f(\sigma) \leq C$ for a threshold cost $C$, and they differ from one another in the state $\sigma$ selected for expansion (e.g., depth-first, best-first, etc.) and the way the threshold is adjusted (decreased until no solutions found, increased until solution found, etc.).

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

[2] This scheme can be improved by noticing that the AP relaxation generates the same assignment, and thus the same subtours, for $\sigma$ and $\sigma^+$ (see [1, 42]).

1. states $\sigma$ are TSPs
2. initial state $\sigma_0$ is original TSP
3. lower bound $f(\sigma)$ computed by solving AP relaxation of $\sigma$; result yields one or more subtours
4. terminal goal states are states whose relaxation yields single tour
5. non-terminal states $\sigma$ expanded by selecting an edge $i \to j$ from a subtour and generating the children $\sigma_{ij}^{+}$ and $\sigma_{ij}^{-}$: the first, forcing the edge $i \to j$ in the solution, the second, excluding the edge $i \to j$ from the solution (namely, $\sigma_{ij}^{-}$ obtained by excluding the edge $i \to j$ from $\sigma$, while $\sigma_{ij}^{+}$ obtained by excluding all links $i \to j'$ and $i' \to j$ for all $j' \neq j$ and $i' \neq i$)

Figure 1: Branch and Bound Scheme for Asymmetric TSPs

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 be guaranteed to find an optimal solution. The first property is simple to verify for the scheme above: all the terminal states represent single tours and thus solutions to the problem. At the same time, splits cannot exclude optimal solutions (namely, an optimal solution for $\sigma$ will be clearly an optimal solution of one of its two children), and if $\sigma$ is a terminal state, then the AP relaxation yields one optimal solution (even if there might be many). So clearly, the branch-and-bound scheme is sound and complete in the sense above.

Notice that branching is not done by applying the actions of going from one city to its neighbors but by making *commitments*; namely, that certain edges are to be forced 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. These are forms of non-directional branching or *branching on commitments* as opposed to *branching on actions*. Of course, both are compatible with the use of lower bounds for pruning the search.

## 2.2 Job Shop Scheduling

The job shop is relevant to our discussion for two reasons. First, as the TSP, it provides a clear, well understood domain where it's feasible to branch on actions but where that's not the most effective thing to do. Second, the job shop is a special case of the general planning task considered in this paper, and the ideas we find useful here will be usable in the more general setting.

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.

1. states $\sigma$ are simple temporal problems (STPs)
2. initial state $\sigma_0$ given by precedence constraints in jobs
3. lower bound $f(\sigma)$ and relaxed schedule $h_\sigma$ obtained by solving STP $\sigma$; $f(\sigma) = \infty$ if STP inconsistent
4. terminal states of two types: $\sigma$ is a *dead-end* if $f(\sigma) = \infty$ and a *goal state* if no pairs of tasks in conflict in $h_\sigma$
5. children generated from $\sigma$ by selecting pair of tasks $t$ and $t'$ in conflict in $\sigma$, and generating two children $\sigma + \{t \prec t'\}$ and $\sigma + \{t' \prec t\}$

Figure 2: Branch and Bound Scheme for Job Shop

Such constraints can be written as

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

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

$$t \prec t' \text{ stands for } T(t) + D(t) \leq T(t')$$

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 would be too high and more effective schemes have been developed. Here we'll follow the approach in [11], that extends ideas from [8] and [38]. In this scheme, each state $\sigma$ 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 [14, 10, 18]. The algorithms determine the consistency of the STP $\sigma$ and if so return lower bounds $h_\sigma(t)$ on the times in 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 tasked $t_{ij}$ is executed at its lower bound $h_\sigma(t_{ij})$ as the *relaxed schedule* and denote it by $h_\sigma$. It's known that the relaxed schedule satisfies the STP $\sigma$, 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 $\sigma$; namely, tasks that use the same resource and whose execution overlaps in $h_\sigma$. This suggests the branch-and-bound scheme shown in Fig. 2.

It's simple to verify that this branch-and-bound scheme is also sound and complete in the sense defined above: terminal goal states are solutions to the problem, and some of them are optimal solutions. Like the scheme for the TSP, this scheme shows many of the features that make a branch-and-bound scheme powerful: a tight integration between branching and lower bound computations, a small number of children, fast and potentially incremental computation of lower bounds, etc.

# 3 Bounds in AI Planning

A recent development in AI planning is the notion of *heuristic estimators* and their automatic extraction from Strips and ADL problem encodings [30, 7]. Heuristic estimators provide approximate measures of the 'cost to go' and can be used to guide the search for plans in the context of algorithms such as A*, IDA*, and Hill-Climbing. Two examples of planners based on these ideas are HSP and FF [6, 22]. In both, the heuristic function is derived from a planning problem represented in Strips or ADL by solving a *relaxed* planning problem in which delete lists are ignored. Since this relaxed problem is still intractable, HSP and FF make further approximations. For example, HSP, makes the assumption that subgoals are *independent.* As a result, *the cost to achieve a set of atoms is approximated by the sum of the costs to achieve each of the atoms in the set.* The resulting heuristic function, called the *additive* heuristic, is informative and plugged into standard heuristic search algorithms yields planners that are very competitive [6]. The additive heuristic, however, is not a *lower bound,* and hence it's not useful to find *optimal* plans.

## 3.1 Heuristics $h^m$

A family of *admissible heuristics* (lower bounds) $h^m$, for $m = 1, 2, \ldots$ for sequential and parallel Strips planning is formulated in [19]. 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 $C$*. Thus, for $m = 1$, $h^m$ approximates the cost of a set of atoms by the cost of the most costly *atom* in the set, for $m = 2$, $h^m$ approximates the cost of the set by the cost of the most costly *atom pair* in the set, and so on. Formally, the function $h^m(C)$ for a set of atoms $C$ is defined by the functional equation

$$h^m(C) = \begin{cases} 0 & \text{if } C \subseteq s_0, \text{ else} \\ \min_{B \in R(C)} \; [1 + h^m(B)] & \text{if } |C| \leq m, \text{ else} \\ \max_{B \subset C, |B| = m} \; h^m(B) & \text{if } |C| > m \end{cases} \tag{3}$$

where $R(C)$, called the regression set of $C$, stands for the states $B$ that can be obtained by regressing $C$ through one of the actions.[3] For *sequential planning*, the actions stand for the primitive operators, while for *parallel planning*, the actions stand for sets of compatible primitive operators (see [19] for details).

[3] More precisely, $B \in R(C)$ iff for some action $a$ s.t. $add(a) \cap C \neq \emptyset$ and $del(a) \cap C = \emptyset$, $B = C - add(a) + pre(a)$. The notion of regression is familiar in AI and corresponds to the 'inverse' application of actions; namely, $B \in R(C)$ iff $B$ is a minimal set of atoms that would lead to $C$ by applying one of the actions [32].

The higher the value of $m$, the more accurate the heuristic but the more expensive its computation. The computation of $h^m$ is similar to the computation of shortest paths in a graph whose nodes are the different atom sets of size equal to or smaller than $m$. Thus its complexity is a low polynomial in $|A|^m$ where $|A|$ is the number of atoms. As an illustration, for $m = 1$, Equation 3 above simplifies to

$$h^1(C) = \begin{cases} 0 & \text{if } C \subseteq s_0\text{, else} \\ \min_{o \in O(p)} [1 + h^1(prec(o))] & \text{if } C = \{p\}\text{, else} \\ \max_{p \in C} h^1(\{p\}) & \text{if } |C| > 1 \end{cases} \tag{4}$$

where $p$ is an atom and $O(p)$ stands for the operators $o$ that add $p$. Similar expressions can be obtained for $m = 2$, $m = 3$, etc, yet computing $h^m$ for $m > 2$ is expensive and does not appear to pay off in practice.

## 3.2 Graphplan Heuristic

Graphplan is perhaps the first modern planner and had a significant in planning research [5]. Graphplan builds first a plan graph made of a number of propositional and action layers, and then starting from the last layer attempts to extract a plan from the plan graph. If successful, the plan is returned, if not, a new attempt is made by extending the graph by one layer and so successively. As argued in [6] and [19], Graphplan can be understood accurately as an heuristic search planner that combines an IDA* regression search for parallel plans with an admissible heuristic function $h_G$ represented in the plan graph. For a set of atoms $C$, $h_G(C)$ is the index of the first layer in the plan graph in which all atoms in $C$ appear without a mutex. This heuristic is equivalent to the $h^2$ heuristic for parallel planning; namely $h_G = h^2$ [19].

## 3.3 Heuristics for Temporal Planning

[20] 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 for $m = 1$ becomes

$$h^1_T(C) = \begin{cases} 0 & \text{if } C \subseteq s_0\text{, else} \\ \min_{o \in O(p)} [D(o) + h^1_T(prec(o))] & \text{if } C = \{p\}\text{, else} \\ \max_{p \in C} h^1_T(\{p\}) & \text{if } |C| > 1 \end{cases} \tag{5}$$

where the only change from the *parallel* estimator $h^1$ to the *temporal* estimator $h^1_T$ is the substitution of the fixed cost 1 by $D(o)$, the temporal duration of the operator $o$. For $m = 2$, the temporal estimator $h^2_T$ departs from parallel $h^2$ in other ways; see [20] for details. While the $h^1$ estimators are often too weak and the $h^2$ estimators are normally preferred, in this paper, for simplicity, we discuss branching schemes in the context of the former only. The generalization to higher-order estimators is direct but the details are involved. The measures $h^m_T(C)$ are all lower bounds on the time needed to make $C$ true.

# 4 Branch-and-Bound Schemes in AI Planning

While few approaches in AI Planning have been described as branch-and-bound schemes, most can be understood in those terms and there are interesting lessons that can be learned from that perspective. Clearly optimal *state-based planners* such as HSPr$^*$ and Graphplan are branch-and-bound planners combining $h^2$ lower bounds with regression branching. *Partial order planning*, on the other hand, is a smart branching scheme for planning but which does not rely on any lower bounds. Finally, *SAT* and *CSP* planners, are characterized by branching schemes based on variable-splitting, and lower bound computation based on constraint propagation. In some cases, constraint propagation is just the *mechanism for incrementally computing the lower bounds,* in others, constraint propagation provides *an incomplete evaluation of the pruning condition* $f^*(\sigma) \leq C$ used in branch-and-bound algorithms, where $f^*$ is the exact bound and $C$ is a threshold cost. In this section we elaborate this view, and in the next, we introduce a new branch-and-bound formulation that combines several of these ideas.

## 4.1 State Planning

Total and partial order planners both search in the *space of plans.* However, while total order planners build *linear plans* from the 'beginning' (forward planning) or 'end' (regression planning), partial order planners build plans non-linearly. In both cases, the state $\sigma$ in the search is a *partial plan,* yet since *partial linear plans* are either plan prefixes or suffixes, and standard markovian assumptions hold, partial linear plans can be *summarized* by suitably defined *states* represented by sets of atoms. This is why linear planners are often called *state planners.*

State planners for *sequential planning* branch on the set of applicable operators either forward or backward. The heuristics $h^m$ provide useful lower bounds in this setting. In state-based planning, the evaluation function $f(s)$ is obtained by combining two terms, the accumulated cost $g(s)$ and a lower bound $h(s)$ on the 'cost to go'. This is typical in state-based search but different from what we have seen in the TSP and the JSP (and what'll see in other planning approaches) where there is no 'accumulated cost' and $f = h$.

[19] 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.* SAT and CSP formulations, are not affected by this explosion. In SAT, all variables are binary, and thus branching on variables always makes for a branching factor of 2. If this branching scheme is supplemented by good and efficient lower bound computations, then it may represent a more powerful approach (yet see below).

### 4.2 Partial Order Planning

Partial order Planning (POP) refers to a non-directional branching scheme used in Planning [40]. POP planners are currently not competitive with modern planners because the latter all rely on some form of heuristic estimation or lower bound computation. POP on the other hand, is a pure and blind branching scheme. The performance of POP planners can be enhanced through the addition of heuristic estimators (e.g., see [31]), 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 suitable for more expressive forms of planning such as *temporal planning with resources.* Indeed, two of the most expressive *temporal planners,* IxTeT [27] and RAX [23] are based on partial-order planning schemes. The potential advantages of POP in this setting have been discussed in [37]. 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 *completion time of partial plans.*

The state in POP is a partial plan $\sigma = \langle Steps, Prec, CL, Open \rangle$, where *Steps* is a set of actions, *Prec* is a set of precedence constraints on *Steps*, $CL$ is a set of causal links, and *Open* is a set of open preconditions.[4] A *causal link* $a \rightarrow_p a'$ states that action $a$ supports the precondition $p$ of action $a$ in $\sigma$, while an *open precondition* is a pair $(p, a)$ such that $p$ is a precondition of action $a$ currently unsupported. With these notions, POP branching can be described as a process of finding and repairing 'flaws'. An *open precondition* $(p, a)$ is repaired by selecting an action $a'$ that adds $p$, and adding the precedence constraint $a' \prec a$ to *Prec* and the causal link $a' \rightarrow_p a$ to $CL$ ($a'$ should also be added to *Steps*). Similarly, a *threat* occurs when there is a causal link $a_1 \rightarrow_p a_2$ in $CL$ and an action $a \in Steps$ that deletes $p$ such that the ordering $a_1 \prec a' \prec a_2$ is consistent. This flaw is repaired my making this ordering inconsistent; i.e., by placing either the precedence constraint $a' \prec a_1$ or $a_2 \prec a'$ in *Prec*. The resulting branching scheme is shown in Fig. 3. The actions *Start* and *End* are dummy actions that summarize the information in the initial situation $I$ and the goals $G$; namely, *Start* is defined as an operator with empty preconditions and add list equal to $I$, and *End* is defined as an operator with preconditions equal to $G$ and empty postconditions.

It's interesting to compare POP branching (Fig. 3) with the branch-and-bound scheme for the job shop (Fig. 2). In the job-shop, the state $\sigma$ corresponds to a set of precedence constraints *Prec*, while in POP it also includes a set of actions *Steps*, a set of causal links $CL$, and a set of open preconditions *Open*. Consider, however, planning theories that satisfy one of these conditions:

1. for each atom $p$ there is single action that adds $p$
2. deletes lists are all empty

If condition 1 holds, we can remove all 'open preconditions' flaws in one shot without backtracking, and thus can remove the fields *Steps* and *Open* from

[4] We assume that all actions are grounded.

1. states $\sigma$ are partial plans $\langle Steps, Prec, CL, Open\rangle$
2. $\sigma_0$ is $\langle\{Start, End\}, \{Start \prec End\}, \emptyset, \{(G_i, End)_i\}\rangle$, where the $G_i$'s are the goals
3. terminal states of two types: $\sigma$ is a *dead-end* if $Prec$ inconsistent, and a *goal state* if $\sigma$ has no flaws
4. children $\sigma_i$ generated from $\sigma$ by selecting a flaw in $\sigma$ and extending $\sigma$ with each possible repair (see text)

Figure 3: Branching scheme in partial-order planning

$\sigma$. Similarly, causal links $CL$ in $\sigma$ are not required if condition 2 is satisfied. Of course, there is no much planning left to be done in the classical setting when conditions 1 and 2 are satisfied. Yet that situation is meaningful when actions have durations and use unary resources. Indeed, those theories correspond closely to the job shop and will be considered in Section 5.

### 4.3 SAT and CSP Branch and Bound

Most SAT and CSP formulations of optimal planning can be understood as branch-and-bound schemes in which the *states* $\sigma$ 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 then plugging these bounds in the pruning condition $f(\sigma) \leq C$ in branch-and-bound algorithms, SAT and CSP formulations often bypass the computation of explicit lower bounds and provide instead a *sound but incomplete evaluation of the pruning condition* $f^*(\sigma) \leq C$ *for the optimal bound* $f^*(\sigma)$. These pruning condition is represented by one or more formulas or constraints and its violation is detected by inferring an *inconsistency.* For example, in a SAT planner such as Blackbox, the pruning condition becomes the pair of clauses $p_{15}$ and $q_{15}$ when the goal is $G = \{p, q\}$ and Blackbox is trying to find a plan in 14 time steps. Inference is done by some form of *constraint propagation* in a theory that includes all the constraints in the problem, and in certain cases, some derived constraints. For example, Blackbox does inference by unit resolution over a theory that is obtained from the plan graph, which incorporates derived constraints such as propositional and action mutexes.

We say that a constraint-based implementation of a branch-and-bound algorithms performs *explicit* lower bound computations, when constraint propagation is used to compute the lower bounds $f(\sigma)$. For example, the RAX planner is a temporal planner that performs explicit lower bound computation by solving STPs in each state $\sigma$ by means constraint-propagation. In this case, constraint-propagation is the mechanism for computing the lower bounds incrementally. On the other hand, we say that a constraint-based branch-and-bound algorithm performs *implicit* lower bound computations, when constraint-propagation is used to evaluate the pruning condition $f^*(\sigma) \leq C$ directly. This is the case in most SAT and CSP planners. Notice that in the first case, constraint propagation is done over a *tractable theory that is completely solved;* in the second case, constraint propagation is done as a *limited form of inference over an intractable theory.* Once again, approaches to the job-shop that ignore resource constraints belong to the first class, while approaches that take resource constraints into account belong to the second class (e.g., [33, 2, 9]). To a certain extent, the novelty and success of recent constraint-based formulations of branch-and-bound algorithms for combinatorial optimization problems has to do with the additional pruning that can be obtained when explicit computation of lower bounds, as in traditional branch-and-bound algorithms, is replaced by implicit computations based on suitable propagation rules [21, 17].

# 5 New Branch-and-Bound Formulations

We have seen that the notions of branching and bounds allow us to clarify the relationships between the different approaches in planning and to relate planning with other combinatorial optimization problems. Now, we want to take advantage of this view to introduce a novel branch-and-bound formulation that integrates a number of the ideas we have discussed: partial order planning, lower bounds, and constraints. The formulation is quite general from a planning point of view, and it accommodates *actions with durations* and *unary resources.* While it's not as expressive as IxTeT [27] and RAX [23], it may scale up better due to the use of lower bounds. We introduce the formulation in several steps, but first define the notions that we'll use.

## 5.1 Preliminary Definitions

We consider a simple extension of the Strips language that accommodates concurrent actions with durations and unary resources. Each action or operation $a$ is characterized by a precondition, add, and delete lists $prec(a)$, $add(a)$, and $del(a)$. As usual, $prec(a)$ stands for the set of atoms that must be true for $a$ to be executed, while $add(a)$ (resp. $del(a)$) stands for the set of atoms that become true (resp. false) as a result of $a$. In addition, there is a *time duration* $D(a)$ and a set of *unary resources* required $R(a)$. We assume durations to be positive integers except for the dummy actions *Start* and *End* that have zero durations. Unary resources refer to resources that cannot be divided or shared (e.g., a machine that can perform one task at a time until completion) and the set of resources $R(a)$ may or may not be empty

Two actions $a$ and $a'$ are *mutex* when their executions cannot overlap. More precisely, $a$ and $a'$ are *mutex* if 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 $a$. Two actions are *compatible* if they are not mutex.

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. The same action can be executed more than once in $P$ if $a_i = a_j$ for $i \neq j$. In such a case, $a_i$ and $a_j$ refer to two *occurrences* of the same action. We say that $a_i$ *precedes* $a_j$ in $P$, and write $a_i \prec a_j$, when $t_i + D(a_i) \leq t_j$, 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 no two occurrences of mutex actions overlap in $P$ and for every action $a_i$ its preconditions $p \in prec(a)$ are true at time $t_i$. This condition is inductively defined as follows:

- $p$ is true at time $t = 0$ if $p$ is true in the initial situation
- $p$ is true at time $t > 0$ if $p$ is true at time $t - 1$ and no action $a$ in $P$ ending at $t$ deletes $p$, or some action $a'$ in $P$ ending at $t$ adds $p$

From now on, a plan will refer to a *valid* plan. The *makespan* of a plan $P$, $t_m(P)$, is the min time at which all goals are true, and the *end time* of $P$, $t_e(P)$, is the min time at which all actions in $P$ have terminated. Clearly, $t_m(P) \leq t_e(P)$ and normally $t_m(P) = t_e(P)$, yet for convenience we will allow plans in which this last condition does not hold. We'll call such plans non-normal and they can easily be normalized by excluding actions that terminate after the makespan.

We are interested in the task of computing a valid plan $P$ with minimum makespan that achieves a goal $G$ from a given initial situation $I$ and a given set of operators $O$. When there are no required resources and all action durations are equal, the task reduces to optimal *parallel planning* as supported in Graphplan [5] and Blackbox [25]. *Sequential planning* is obtained when all action durations are equal and there is a single unary resource common to all actions. Similarly, the *job shop* is obtained by having an action $a_{ij}$ for each task $t_{ij}$ with precondition $done(t_{ij-1})$ and postconditions $done(t_{ij})$, the exception being the first task in each job which has no preconditions.

## 5.2 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 will 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 where *all* operators are executed *exactly* once. Thus positive theories, stand halfway between 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 $\sigma_0$. In the job shop, $\sigma_0$ is defined as the set of precedence constraints

$$T(t_{ij}) + D(t_{ij}) \leq T(t_{ij+1}) \tag{6}$$

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 $h_T^m$. For example, Equation 4 for $h_T^1$ can be rewritten as:

$$h_T^1(C) = \begin{cases} \min_{a \in O(p)}[D(a) + h_T^1(a)] & \text{if } C = \{p\}\text{, else} \\ \max_{p \in C} h_T^1(\{p\}) & \text{if } |C| > 1 \end{cases} \tag{7}$$

where we assume now the presence of the actions *Start* and *End*,[5] and $h_T^1(a)$ is a lower bound on the time needed to execute action $a$

$$h_T^1(a) = h_T^1(prec(a)) \tag{8}$$

These equations on lower bounds can be made to look similar to precedence constraints (6) by means of two transformations. First, we project the equations on actions only by unfolding the left-hand-side of Equation 8 to get

$$h_T^1(a) = \max_{p \in prec(a)} \left\{ \min_{a' \in O(p)} [D(a') + h_T^1(a')] \right\} \tag{9}$$

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 $\infty$. Intuitively, $T(a) = \infty$ means that action $a$ is not executed in the plan and we assume $T(a) + D(a) = \infty$ if $T(a) = \infty$.[6]

Recasting Equation 9 on the temporal variables $T(a)$, we obtain the set of *temporal constraints:*

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

These constraints are similar to the precedence constraints (6) for the job shop except for the **min** operator, and we will call them *precondition constraints.* These precondition constraints are derived from the heuristic $h^1$ so we say that they are precondition constraints of degree 1. In principle, precondition constraints of degree $m > 1$ could be defined, but we won't use them in this paper.

It's not difficult to show that precedence and precondition constraints are similar from a computationally point of view: they are tractable, they can be solved by simple variations of shortest path algorithms, and the lower bounds

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

[6] This is so that $a' \prec a$ always holds when $T(a) = \infty$, even if $T(a') = \infty$.

1. states $\sigma$ are Extended STPs
2. initial state $\sigma_0$ given by precondition constraints (10)
3. relaxed plan $h_\sigma$ obtained by solving ESTP $\sigma$; $f(\sigma) = h_\sigma(End)$
4. terminal goal states $\sigma$ if mutex constraints (11) not violated by any pair of actions $a$ and $a'$ in relaxed schedule $h_\sigma$
5. branching from non-terminal $\sigma$ done by picking one such pair of actions $a$ and $a'$ and generating children $\sigma + \{a \prec a'\}$ and $\sigma + \{a' \prec a\}$

Figure 4: Branch and Bound Scheme for Positive Planning Theories

$h(a)$ obtained for each temporal variable $T(a)$ define a consistent solution. We'll call the theories that combine precondition and precedence constraints, Extended STPs. Note, however, that due to the inclusion of $\infty$ 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* is similar to 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 (10) and the mutex constraints*

$$mutex(a, a') \Longrightarrow a \prec a' \ \vee \ a' \prec a' \tag{11}$$

Due to the similarity between precondition and precedence constraints on the one hand, and resource and mutex constraints on the other, it's simple to recast the branch-and-bound scheme for the job-shop (Fig. 2) into a *sound* and *complete* scheme for positive theories.[7] This is shown in Fig. 4.

## 5.3 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 sched-

[7] In order to prove soundness, we need to show that in any relaxed schedule $h_\sigma$, all actions $a$ are scheduled after each precondition $p$ in $prec(a)$ has been established by some action $a_p$. This easily follows from (10). Completeness is more subtle; we need to show that *some* goal states represent optimal (valid) plans. This can be done by showing that the search tree contains paths that lead to the left-shifted optimal plans, i.e., the optimal plans in which the actions occur as early as possible (any plan can be 'left-shifted' by moving actions to the left in the time axis, from beginning to end). If $P$ is one such plan, such paths are the ones in which for each conflict between two actions $a$ and $a'$, the ordering $a \prec a'$ is selected when $a \prec a'$ in $P$ or $a$ is in $P$ but $a'$ is not.

uled. 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 (assumptions 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.

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 $a'$ and makes its precondition $p$ true, and that 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.* In POP, the state (partial plan) keeps track of the current causal links but not of the causal links that have been tried and failed. Yet, both sets of causal links are informative for computing lower bounds. For example, consider a proposition $p$ that is only 'added' by the *Start* action (i.e., $p$ holds in the initial situation) and by an action $a_p$ with duration 5. Then, if at some point, an open precondition $\langle p, a\rangle$ is selected for support and the causal link $Start \rightarrow_p a$ fails, it can be inferred immediately that $a$ must be scheduled after action $a_p$ and thus $T(a) \geq 5$.

For this reason, we introduce finite domain variables $S(p, a)$ for each precondition $p$ of action $a$ which we call them *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 will change dynamically during the search. In particular, a causal link $a' \rightarrow_p a$, is asserted by setting $S(p, a) = a'$, and hence $D(p, a) = \{a'\}$, and is retracted by setting $S(p, a) \neq a'$, hence deleting $a'$ from $D(p, a)$.

Provided this representation of causal links, the precondition constraints can be written as

$$T(a) \geq \min_{a' \in D(a,p)} [T(a') + D(a')] \;\; \text{for each } p \in pre(a) \tag{12}$$

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 dynamic precondition constraints (12), 2) the mutex constraints (11), and 3) the causal link constraints*

$$T(a) \neq \infty \;\&\; S(p, a) = a' \;\&\; p \in del(a'') \;\Rightarrow\; a'' \prec a' \;\vee\; a \prec a'' \tag{13}$$

1. states $\sigma = \langle Precs, Doms \rangle$, where $Precs$ are precedence and precondition constraints, and $Doms$ stand for the dynamic domains $D(p, a)$ for all $p \in prec(a)$ and all $a$
2. initial state $\sigma_0 = \langle Precs_0, Doms_0 \rangle$, where $Precs_0$ stands for the precondition constraints (12) and $Doms_0$ for the domains $D(p, a) = O(p)$
3. relaxed schedule $h_\sigma$ computed by solving ESTP $Precs$, over domains $Doms$; lower bound $f(\sigma) = h_\sigma(End)$
4. terminal goals states $\sigma = \langle Precs, Doms \rangle$ if mutex constraints (11) violated by no action pair $(a, a')$, and causal link constraints (13) violated by no action triplet $(a, a', a'')$ in relaxed schedule $h_\sigma$
5. children generated from non-terminal state $\sigma = \langle Prec, Doms \rangle$ by selecting a culprit $(a, a')$ and branching $[a \prec a'; a' \prec a]$, or by selecting a culprit $(a, a', a'')$ 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 (see text).

Figure 5: Branch and Bound for Canonical Planning

Like the disjunctive constraints for mutexes, and unlike the preconditions constraints, the causal link constraints are *intractable.* As a result, the branch-and-bound scheme computes relaxed schedules $h_\sigma$ by considering the precondition and posted precedence constraints only, and branches over the disjunctions in either mutex or causal link constraints that are violated in $h_\sigma$. The resulting scheme is shown in Fig. 5. The expression 'branching on $[Updates1; Updates2]$ in $\sigma = \langle Precs, Doms \rangle$' means that two children are generated, one in which $\sigma$ is updated with $Updates1$, the other in which is updated with $Update2$. If an update contains the expression $a \prec a'$, this constraint is added to $Precs$; if it contains the expression $S(p, a) = a'$ ($S(p, a) \neq a'$), then the domain $D(p, a)$ is updated to $D(p, a) := \{a'\}$ ($D(p, a) := D(p, a) - \{a'\}$).

Notice that unlike, partial-order planning branching schemes, we don't require a unique causal link support for all actions. Such unique support, may result however from the constraints $S(p, a) = a'$ and $S(p, a) \neq a'$ posted during the search. At the same time, we don't resolve all *potential conflicts* (threats), but the *actual* conflicts that result in the relaxed plan (a similar idea is used by [11] in scheduling). A final difference is that the search is not goal oriented. This may be a problem when the set of relevant actions is small in comparison with the set of all actions. Actually, the same occurs in SAT formulations [25] and CSP formulations over the plan graph [15, 36, 28].

## 5.4 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

1. states $\sigma = \langle Steps, Precs, Doms \rangle$, **where** $Steps$ **is a set of actions**, and $Precs$ and Doms are 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 $h_\sigma$ computed by solving ESTP $Precs$ over domains $Doms$; 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 $h_\sigma$, mutex constraint (11) is violated by no action pair $(a, a')$, and causal link constraint (13) is violated by no action triplet $(a, a', a'')$ **for** $a$,$a$,$a''$ **in** $Steps$
5. children generated from non-terminal state $\sigma = \langle Prec, Doms \rangle$ by selecting a culprit $(a, a')$ and branching $[a \prec a'; a' \prec a]$, or by selecting a culprit $(a, a', a'')$ and branching $[a'' \prec a; a' \prec a'']$, **or by selecting a domain** $|D(p, a)| > 1$ **for** $a \in Steps$, **and branching by setting** $S(p, a) = a_i$ **and adding** $a_i$ **to** $Steps$, **for each** $a_i \in D(p, a)$

Figure 6: Goal-oriented branch-and-bound for Canonical Plans

small in comparison with the set of all actions. These modifications can be easily included in the branch-and-bound scheme above to yield the scheme shown in Fig. 6. Which scheme is better remains an open empirical question, yet it's useful to make these distinctions explicit. For example, RAX is a constraint-based planner that does goal-oriented branching, while Blackbox and CSP-Graphplan are constraint-planners that do not.

### 5.5 Scheduling an Action Multiple Times

The schemes above are all restricted to compute canonical plans where each action is scheduled at most once. Planning theories where this restriction does not hold, however, are common. Nonetheless, extending the schemes for dealing with such theories is not difficult. In principle, if an action $a$ can appear twice in a plan, we can create two copies of $a$, $a_1$ and $a_2$, and all the schemes above will handle such copies correctly as if they were different actions. Actually, the same trick can be done as long as we have an upper bound on the number of occurrences of each action. Alternatively, when considering plans with makespan less than or equal to a threshold $C$, we can create copies $a_0$, $a_1$, $\ldots$, $a_m$ of an action $a$, where $m = C - D(a)$ and the domain of $T(a_i)$ is restricted to the single time point $i$. Something like this is done in Blackbox and Graphplan-CSP approaches where the actions $a_i$ are treated as boolean variables.

Creating copies of an action $a$ dynamically, as done normally in partial-order planning, is not sound in this framework. This is because the makespan and lower bound of a plan can decrease when copies are introduced. A way around this difficulty is to introduce a distinction between action templates and action copies. Action copies are created dynamically as in POP, and once a copy $a$ is created from a template $\mathbf{a}$, the precedence constraint $T(a) \geq T(\mathbf{a})$ is posted. Action copies can be included in plans but action templates can't: they can only appear in precondition constraints. The result is that the constraints that are posted during the search do not affect the lower bound of action templates and thus of future copies. This device, however, while general, may not be effective as the resulting bounds may be too weak.

# 6 From Formulation to Implementation

We have shown that the notions of branching and lower bounds provide a useful perspective for understanding and relating a number of ideas in planning. We have also presented a formulation of temporal planning with unary resources that integrates non-linear branching and lower bound computation, which may provide a suitable platform for building systems that combine planning and scheduling. Implementing such systems, however, whether for positive, canonical, or general theories, is not direct, and there are additional issues that need to be addressed. We hope that others may want to undertake the challenge. Here we discuss some of them.

- Variable (and value) selection heuristics are needed and they are often crucial for performance; namely, criteria for selecting the 'flaw' to fix and the order in which to try the repairs. Slack and contention based heuristics are common in scheduling research [4]. Such selection heuristics require not only lower bounds but *time windows;* i.e., lower and upper bounds.
- Lower and upper bounds can be obtained from the same computation with little overhead [14]. In addition, such computations have to be *incremental,* so that the time windows for the child are obtained incrementally from the time windows for the parent. Constraint-based implementation are well suited for this, in particular, if they match the performance of shortest-path algorithms such as Bellman-Ford [3]
- Mutex and causal link constraints, are intractable, and play a *passive role* in the schemes we have considered. Limited forms of inference over such constraints, however, as captured in recent CP implementation of branch-and-bound algorithms [33, 2, 9, 21, 17], may speed up the search considerably by pruning additional parts of the state space.
- The precondition constraints we have used have been defined from the $h^1_T$ heuristic estimator. Precondition constraints based on high-order estimators, in particular $h^2_T$ [20], will introduce more variables but may yield better bounds and better performance

It may also be useful to develop methods for analyzing planning domains, determining the number of times in which an action may have to be scheduled. This may be intractable in general, but some easy cases could be identified (e.g., if there are no deletes, no action need to be scheduled more than once, etc.) In particular, it seems that better lower bounds can be derived if the theory is known to be canonical.

# 7 Related Work

The view of *planning as branch-and-bound* is an extension of the view of planning as heuristic search [30, 6]. The latter is tied to directional branching schemes while the former is more general and includes partial-order and constraint-based planning too. In this framework, POP [29, 40] is a smart but blind branching scheme, while constraint-based planners [25, 15, 36, 28] are branch-and-bound planners in which the computation of explicit lower bounds $f(\sigma)$ is replaced by an incomplete evaluation of the pruning condition $f^*(\sigma) \leq C$ for the exact value function $f^*$. This implicit lower-bound schemes, have considerable promise as they can prune additional parts of the search space through the use of suitable propagation rules [33, 2, 9, 17] and are incremental (inference is deductive and branching just adds formulas). Constraint-based approaches, however, take many different forms and often differ along several dimensions; e.g.

- **branching:** some planners try to schedule all actions (e.g., Blackbox, CSP-Graphplan); others build the set of actions incrementally, as in POP (e.g., RAX)
- **surrogate constraints:** some planners do constraint-propagation over a theory extended with implicit constraints obtained from the plan graph (Blackbox, CSP-Graphplan) or by hand (CPlan [39]); others do it directly over the given theory (SATPlan)
- **propagation:** some planners do complete constraint-propagation over tractable theories (RAX), most others do incomplete propagation over intractable ones

The scheme we have presented that combines non-linear branching with lower bounds is related to [31], where an heuristic estimator is introduced to guide a partial-order planner. Similarly, [36], explains the limitations of Graphplan and the benefits of Blackbox in terms of a non-directional branching scheme.

Systems that accommodate planning and scheduling capabilities include IxTeT [27], RAX [23], Aspen [16], O-Plan [13] and SIPE [41]. The performance of these systems, however, does not appear to match the performance of modern planners and often rely on explicitly supplied control-knowledge. This is likely due to the weak lower bounds that are used when they are used at all. In planners such as IxTeT and RAX, such lower bounds could be boosted by including the *precondition constraints* (10) among the constraints that are checked for consistency in every state. Indeed, the precondition constraints capture the heuristics $h^m$ that have been found so useful in state planners such as Graphplan and HSPr* [19].

## 8 Conclusions

In this paper, we have analyzed the notion of branching in AI planning and have related it to branching schemes in combinatorial optimization. We have also analyzed heuristic-search, partial-order, and constraint-based planning from this perspective, trying to sort commonalities and differences. We have also argued that directional branching schemes as used in heuristic search planning are not suitable for parallel and temporal planning, and introduced a new branch-and-bound formulation that appears more convenient. 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 expect that some of the branch-and-bound schemes that we have presented can contribute to the development of systems that effectively integrate planning and scheduling capabilities.

**Acknowledgements.** Part of this work was done during visits to Nasa Ames and the Universita di Genova. I want to thank Nicola Muscettola and Enrico Giunchiglia for their hospitality and for a number of useful discussions about these topics. Other people I've discussed these issues with are Patrik Haslum, Rao Kambhampati, David Smith, Ari Jonsson, Jeremy Frank, Paul Morris, ...

## 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] P. Baptiste and C. Le Pape. A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In *Proc. IJCAI-95*, 1995.

[3] P. Baptiste, C. Le Pape, and W. Nuijten. Incorporating efficient operations research algorithms in constraint-based scheduling. In *Proc. ??*, 1995.

[4] J. Beck and M. Fox. A generic framework for constraint-directed scheduling. *AI Magazine*, 19(4), 1998.

[5] A. Blum and M. Furst. Fast planning through planning graph analysis. In C. Mellish, editor, *Proceedings of IJCAI-95*, pages 1636–1642. Morgan Kaufmann, 1995.

[6] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proceedings of ECP-99*, pages 359–371. Springer, 1999.

[7] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*, pages 714–719. MIT Press, 1997.

[8] J. Carlier and E. Pinson. An algorithm for solving the job shop scheduling problem. *Management Science*, 35(2), 1989.

[9] Y. Caseau and F. Laburthe. Improved CLP scheduling with task intervals. In *Proc. ICLP-94*, pages 369–383. MIT Press, 1994.

[10] R. Cervoni, A. Cesta, and A. Oddi. Managing dynamic temporal constraint networks. In *Proc. AIPS-94*, pages 13–20, 1999.

[11] A. Cesta, A. Oddi, and S. Smith. Profile based algorithms to solve multi-capacited metric scheduling problems. In *Proc. AIPS-98*, 1998.

[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] Minh Binh Do and Subbarao Kambhampati. Solving planning-graph by compiling it into CSP. In *Proc. AIPS-00*, pages 82–91, 2000.

[16] S. Chien *et al.* Aspen – automating space mission operations using automated planning and scheduling. In *Proc. SpaceOps2000*, Toulouse, France, 2000.

[17] F. Focacci, A. Lodi, and M. Milano. Solving TSPs with time windows with constraints. In *Proc. ICLP-99*. MIT Press, 1999.

[18] A. Gerevini, A. Perini, and F. Ricci. Incremental algorithms for managing temporal constraints. Technical report, IRST, Italy, 1996. Tec. Rep. IRST-9605-07.

[19] 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.

[20] P. Haslum and H. Geffner. Heuristic planning with time and resources. In *Proc. IJCAI-01 Workshop on Planning with Resources*, 2001. To appear.

[21] P. Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, 1999.

[22] Jörg Hoffmann. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proceedings of the 12th International Symposium on Methodologies for Intelligent Systems (ISMIS-00)*, pages 216–227. Springer, October 2000.

[23] A. Jonsson, P. Morris, N. Muscettla, and K. Rajan. Planning in interplanetary space: Theory and practice. In *Proc. AIPS-2000*, 2000.

[24] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, pages 1194–1201. AAAI Press / MIT Press, 1996.

[25] 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.

[26] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.

[27] P. Laborie and M. Ghallab. Planning with sharable resources constraints. In C. Mellish, editor, *Proc. IJCAI-95*, pages 1643–1649. Morgan Kaufmann, 1995.

[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] Xuan Long Nguyen and Subbarao Kambhampati. Reviving partial order planning. In *Proc. IJCAI-01*, 2001.

[32] N. Nilsson. *Principles of Artificial Intelligence*. Tioga, 1980.

[33] W. Nuijten. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*. PhD thesis, Eindhoven University of Technology, The Netherlands, 1994.

[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] D. Smith, J. Frank, and A. Jonsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1), 2000.

[38] S. Smith and C. Cheng. Slack-based heuristics for the constraint satisfaction scheduling. In *Proc. AAAI-93*, pages 139–144, 1993.

[39] 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.

[40] D. Weld. An introduction to least commitment planning. *AI Magazine*, 1994.

[41] D. Wilkins. *Practical Planning: Extending the classical AI paradigm*. M. Kaufmann, 1988.

[42] W. Zhang and R. Korf. Performance of linear-space search algorithms. *Artificial Intelligence*, 79:241–292, 1995.