Reviving Integer Programming Approaches for AI Planning: A Branch-and-Cut Framework

Menkes van den Briel
Department of Industrial Engineering
Arizona State University
Tempe AZ, 85287-8809
menkes@asu.edu

Thomas Vossen
Leeds School of Business
University of Colorado at Boulder
Boulder CO, 80309-0419
vossen@colorado.edu

Subbarao Kambhampati*
Department of Computer Science
Arizona State University
Tempe AZ, 85287-8809
rao@asu.edu

Abstract

The conventional wisdom in the planning community is that planners based on integer programming (IP) techniques cannot compete with satisfiability and constraint satisfaction based planners. In this paper we challenge this perception of IP techniques by presenting novel formulations that outperform the most efficient SAT-based planner that currently exists. We will present a series of IP formulations that (1) use multi-valued state variables that are represented by networks, and that (2) control the encoding length by progressively generalizing the notion of parallelism. The resulting IP encodings are solved within a branch-and-cut framework and yield impressive results.

Introduction

The use of integer programming (IP) to solve AI planning problems has an intuitive appeal, given its remarkable successes in similar problem domains such as scheduling, production planning and routing (Johnson, Nemhauser, & Savelsbergh 2000). In addition, one potential advantage is that IP techniques can provide a natural way to incorporate several important aspects of real-world planning problems, including numeric constraints and objective functions.

Nevertheless, the application of IP techniques to AI planning has only received limited attention. The first appears to have been Bylander (1997), who proposed a linear programming (LP) formulation that could be used as a heuristic in partial order planning. Vossen *et al.* (1999) discuss the importance of developing "strong" IP formulations, by comparing two formulations for classical planning. While a straightforward translation of sat-based encodings yields mediocre results, a less intuitive formulation based on the representation of state transitions results in considerable performance improvements. Dimopoulos (2001) discusses a number of ideas that further improve this IP formulation. A somewhat different approach that relies on domain-specific knowledge is proposed by Bockmayr and Dimopoulos (1998; 1999). The use of LP and IP has also been explored for non-classical planning. Dimopoulos and Gerevini (2002) describe an IP formulation for temporal planning and Wolfman and Weld (1999) use LP formulations in combination with a satisfiability-based planner to solve resource planning problems. Kautz and Walser (1999) use IP formulations for resource planning problems that incorporate action costs and complex objectives.

So far, none of these IP approaches have been able to produce a planner whose performance compares with today's most advanced satisfiability and constraint satisfaction-based planners. Indeed, the current conventional wisdom in the planning community is that IP techniques are not competitive with these approaches. In this paper we challenge this current perception by presenting novel IP formulations that outperform the best SAT-based planners. The formulations we propose rely on two key innovations:

1. We model changes in individual state variables during planning as flows in an appropriately defined network. As a consequence, the resulting IP formulations can be interpreted as a network flow problem with additional side constraints. While this idea can be used with any state variable representation, it is particularly synergistic with multi-valued state variables. We thus adapt existing methods to automatically convert PDDL domain encodings into multi-valued domain encodings.
2. One difficulty in scaling IP encodings has been the dependency between the size of the encoding and the length of the solution plan. This dependency often leads to encodings that are very large. To alleviate this dependency, we separate causal considerations from the action sequencing considerations, as in RealPlan (Srivastava, Kambhampati, & Do 2001), by generalizing the common notion of parallelism based on planning graphs. Planning graphs suggest that it should be possible to arrange parallel actions in any order with exactly the same outcome (Blum & Furst 1995). By relaxing this condition, we develop new concepts of parallelism that are similar yet strictly more general and powerful than the relaxation proposed by Cayrol *et al.* (2001) for the LCGP planner.

*We thank Malte Helmert for making available the translator of the Fast Downward planning system. This research is supported in part by the NSF grant IIS-0308139.

A naive encoding of this decoupling will not be effective as the sequencing phase will add exponentially many ordering constraints. Instead, we propose and implement a so-called *branch-and-cut* framework, in which certain constraints are dynamically generated and added to the formulation only when needed. This approach has been extremely successful for a number of large-scale optimization problems (Caprara & Fischetti 1997). We show that the performance of the resulting planning system is superior to Satplan04(Siege) (Kautz 2004), which is currently the most efficient SAT-based approach to planning. This is a significant result in that it forms the basis for other more advanced IP-based planning systems capable of handling numeric constraints and non-uniform action costs.

The remainder of this paper is organized as follows. In the next section, we introduce three progressively more general IP formulations. Subsequently, we discuss the branch-and-cut framework that is used to solve these formulations. After that, we provide experimental results for this approach, as well as a comparison with Satplan04(Siege). Finally, we conclude with a summary and a discussion of avenues for future research.

Integer Programming Models

This section describes a series of IP formulations for classical planning that progressively generalize the conditions on parallelism. As stated in the introduction, we use (multi-valued) state variables instead of the (binary-valued) propositional variables that were used in the formulations by Vossen *et al.* (1999). The use of multi-valued state variables is based on the SAS+ planning formalism (Bäckström & Nebel 1995). SAS+ is a planning formalism that uses multi-valued state variables instead of propositional atoms, and it uses a *prevail condition* on top of the regular pre- and post-conditions (pre-, add-, and delete-lists). A prevail is a condition imposed by an action that specifies for one or more state variables a specific value that must hold before and during the execution of that action. Another way to look at a prevail is that it implies the persistence of a specific value. To obtain a state variable description from a PDDL description of a planning problem we use the translator that is implemented in the planner Fast (Diagonally) Downward (Helmert 2005). This translator is a general purpose algorithm that transforms a classical planning problem into a multi-valued state variable description. It provides an efficient grounding that minimizes the state description length and is based on the ideas presented by Edelkamp and Helmert (1999).

We formalize the use of multi-valued state variables using the following notation:

- $C = \{c_1, ..., c_n\}$ is a set of *state variables*, where each state variable c has an associated (finite) domain D_c of possible values. A *state* s is a variable assignment over C given by the function s such that $s(c) \in D_c$ for all $c \in D_c$. A *partial variable assignment* over C is a function $s(c)$ on some subset of C such that $s(c) \in D_c$ where $s(c)$ is defined. The state I is called the *initial state*, and the partial variable assignment G is called the *goal*.
- $A = \{a_1, ..., a_m\}$ is a set of *actions* (operators), where an action is a partial variable assignment pair $\langle pre, eff \rangle$ of preconditions and effects respectively. For an action $a \in A$, the set of *state changes* SC_a is a set of partial variable assignment pairs (f, g) such that there exists a $c \in C$ with $s(c) = f \in pre$ and $s(c) = g \in eff$. Similarly, the set of *prevails* PR_a is a set of partial variable assignment pairs (f, f) such that there exists a $c \in C$ with $s(c) = f \in pre$ and for all $g \neq f$ we have $s(c) = g \notin eff$.

Also, we will assume that actions have at most one state change effect or prevail condition on each state variable. In other words, for each $a \in A$ and $c \in C$, we have $|SC_a(c)| + |PR_a(c)| \leq 1$, where $SC_a(c)$ and $PR_a(c)$, respectively, represent the set of state changes and prevails that an action a imposes on a specific state variable c. Finally, we assume a given maximum plan length T for each formulation.

To illustrate the use of multi-valued state variables, consider the Blocksworld domain with actions of the form $MOVE(x, y, z)$. In a propositional representation of this domain, the atoms are given by $OT(x)$, $CL(x)$, and $ON(x, y)$, which represent respectively that block x is on the table, clear, or on block y. The preconditions and effects of the $MOVE$ action for moving a block to another block in a propositional-based representation are given by $pre = \{ON(x, y), CL(x), CL(z)\}$, $add = \{ON(x, z), CL(y)\}$, and $del = \{ON(x, y)\}$. Multi-valued state variables naturally capture some of the mutual exclusion relations between actions, for example, those that correspond to different configurations of the same fluent. Therefore, one way to translate this domain into a state description is by creating the following multi-valued state variables

$$BELOW(x) := \{OT(x)\} \quad \cup \quad \bigcup_y \{ON(x, y)\} \quad \text{for all } x,$$

$$ONTOP(x) := \{CL(x)\} \quad \cup \quad \bigcup_y \{ON(y, x)\} \quad \text{for all } x.$$

The effects of the $MOVE(x, y, z)$ action for moving a block to another block are now given by

$BELOW(x)$**:** $ON(x, y) \rightarrow ON(x, z)$

$ONTOP(y)$**:** $ON(x, y) \rightarrow CL(y)$

$ONTOP(z)$**:** $CL(z) \rightarrow ON(x, z)$

$ONTOP(x)$**:** $CL(x) \rightarrow CL(x)$

where the first three are state change effects and the fourth a prevail condition. A possible initial state and goal are given in Figure 1.

It is interesting to note that there may be several ways to translate a propositional representation into a multi-valued state variable representation. For example, we could have obtained a different translation if

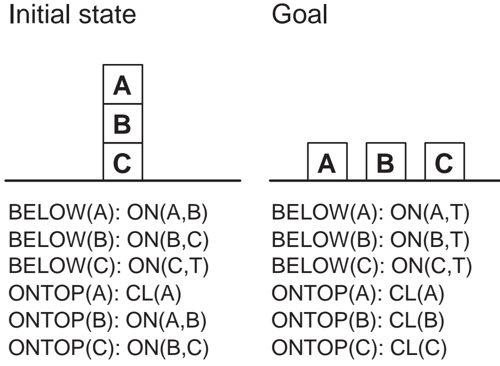


Figure 1: A Blocksworld instance

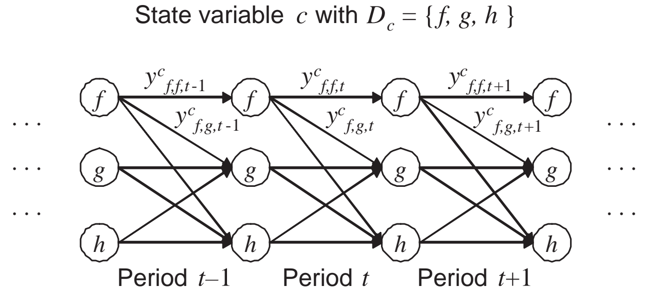


Figure 2: State change flow network

we had used a single dummy atom $ON(x, *)$ to replace all the $ON(x, y)$ atoms in each $BELOW(x)$ state variable. In that case, the dummy atom would simply represent a lifted version of the $ON(x, y)$ atoms, that is, it would represent that a block x is on some block y without specifying y. While we currently use the translator based on the ideas by Edelkamp and Helmert 1999, preliminary experiments indicate that the domain representation choice may impact the overall performance.

In the remainder of this section we first propose an IP formulation that uses multi-valued state variables together with the concept of parallelism as defined in Graphplan. Subsequently, we discuss two IP formulations that also use the multi-valued state variables, but in addition generalize Graphplan's concept of parallelism.

Single State Change (1SC) Formulation

This formulation uses Graphplan's parallelism and allows at most one state change for each state variable per plan step.

We use action and flow variables, which we define as:

- $x_{a,t} \in \{0, 1\}$, for $a \in A, 1 \le t \le T$; $x_{a,t}$ is equal to 1 if action a is executed at plan step t, and 0 otherwise.
- $y^c_{f,g,t} \in \{0, 1\}$, for $c \in C, f, g \in D_c, 1 \le t \le T$; $y^c_{f,g,t}$ is equal to 1 if the state of state variable c transitions from f to g at step t, and 0 otherwise.

Objective function

In classical planning it is sufficient to find a feasible plan. Since the constraints guarantee feasibility we could use a null objective, but when solving an IP encoding the choice of an objective function could significantly influence the search performance as it determines a search direction. Even though our objective function can be any linear expression, we choose to minimize the number of actions to guide the search.

$$MIN \sum_{a \in A} x_{a,t}$$

The constraints are separated into different constraint sets, which are given as follows.

State change flow constraints

In our formulation the state transitions of each multi-valued state variable are represented by paths in an appropriately defined network. In this network, nodes appear in stages (levels) and correspond to the possible values of the state variable at different plan steps. If we set up an IP encoding with a maximum plan length T, there will be $T + 1$ stages in the network (one stage for the initial state and T stages for the number of succeeding plan steps). Arcs link nodes between stages and correspond to state changes or persistence of values. Over the course of a plan, state variables transition from one value to the next. This corresponds to a path in the network, where the source node is a node in stage 0 (zero) that corresponds to the state variable's initial state. The sink node is a node in stage T that corresponds to the goal, and in case the goal is not defined for the state variable then the sink node could be any of the nodes in stage T.

Figure 2 displays a network corresponding to the state variable c with domain $D_c = \{f, g, h\}$. For each allowable value transition there is an arc in the state change network. The horizontal arcs, labeled by $y^c_{f,f,t}$, correspond to the persistence of value f within the state variable c at plan step t. The diagonal arcs, labeled by $y^c_{f,g,t}$, correspond to the state change effects of an action. In the network of Figure 2 there is no arc that connects the value g to f between two consecutive stages, so no action supports the state change (g, f) in c.

The resulting network flow constraints for each multi-valued state variable $c \in C$ are given as follows:

$$\sum_{g \in D_c} y^c_{f,g,1} = \mathbf{1}\{f \in I\} \quad \text{for } f \in D_c, \tag{1}$$

$$\sum_{h \in D_c} y^c_{g,h,t+1} = \sum_{f \in D_c} y^c_{f,g,t} \quad \text{for } g \in D_c, \; 1 \le t \le T-1 \tag{2}$$

$$\sum_{f \in D_c} y^c_{f,g,T} = 1 \quad \text{for } g \in D_c \cap G \tag{3}$$

The state change flow constraints define the underlying graph of the network. They ensure that (1) there is a supply of one unit of flow at the source nodes, (2)

there is a balance of flow at all intermediate nodes, and (3) there is a demand of one unit of flow at the sink nodes if a goal is defined

State change implication constraints
Actions may introduce interactions between the state variables. For instance, the effects of the $MOVE$ action in our Blocksworld example affect four different state variables. Action interactions link state variables to each other and must be represented by constraints. For each $c \in C, f, g \in D_c, f \neq g, 1 \leq t \leq T$ we have constraints linking all the actions state change effects over the corresponding network arcs.

$$\sum_{a \in A:(f,g) \in SC_a(c)} x_{a,t} \quad = \quad y^c_{f,g,t} \tag{4}$$

The state change effects of an action are tied to the flow variables. So, if an action $x_{a,t}$ with $(f, g) \in SC_a(c)$ is executed then the source-sink path in the state variable c must use the arc $y^c_{f,g,t}$. Likewise, if we choose to follow the arc $y^c_{f,g,t}$ in the source-sink path in c then there must be an action $x_{a,t}$ that has the state change (f, g) as one of its effects. The summation on the left hand side prevents two or more actions from interfering with each other, hence only one action may cause the state change (f, g) in a state variable c at step t.

Prevail implication constraints
Prevail conditions of an action link state variables in a similar way as the state change effects of action link them. For each $c \in C, f \in D_c, a \in A, (f, f) \in PR_a(c), 1 \leq t \leq T$ we have constraints linking the all the action prevail conditions to the corresponding network arcs.

$$x_{a,t} \quad \leq \quad y^c_{f,f,t} \tag{5}$$

In words, this constraint states that *if* action a is executed at step t ($x_{a,t} = 1$), *then* value f prevails within the state variable c during step t ($y^c_{f,f,t} = 1$).

Generalized Single State Change (g1SC) Formulation

The formulation described above uses the concept of parallelism as used in Graphplan (Blum & Furst 1995). As stated in this paper, the basic idea in this approach is that:

> "several actions may be specified to occur at the same time step so long as they do not interfere with each other. Specifically, we say that two actions interfere if one deletes a precondition or an addeffect of the other. Thus, in an actual plan these independent parallel actions could be arranged in any order with exactly the same outcome."

Here we will propose a *set of alternative conditions* for parallel actions that will ultimately lead to smaller encodings in terms of number of plan steps. We relax the condition that parallel actions can be arranged in any order by requiring a much weaker condition. In an actual plan parallel actions could be arranged as long as *there exists* a valid ordering. More specifically, within a plan step a set of actions is feasible if (1) there exists an ordering of the actions such that all preconditions are satisfied, and (2) there is at most one state change in each of the state variables.

To illustrate the basic concept, let us again examine the Blocksworld instance given in Figure 1. The obvious solution is to first execute action $MOVE(A, B, T)$ and then $MOVE(B, C, T)$. Clearly, this is not a solution that would be allowed *within a single step* under Graphplan's parallelism, since we cannot execute the actions in an *arbitrary* order (that is, $MOVE(B, C, T)$ cannot be executed unless $MOVE(A, B, T)$ is executed first). Yet, the number of state changes within any state variable is at most one, and while the two actions cannot be arranged in any order with exactly the same outcome, there does exists *some* ordering that is feasible. The key idea behind this example should be clear: while it may not be possible to find a set of actions that can be linearized in any order, there may nevertheless be an ordering of the actions that is viable. The question is, of course, how to incorporate this idea into an IP formulation.

This example illustrates that we are looking for a set of conditions that allow, within each plan step, those sets of actions for which:

- All the actions' preconditions are met,
- There exists an ordering of actions at each plan step that is feasible, and
- Within each state variable, the value is changed at most once.

The incorporation of these ideas only requires minor modifications to the single state change formulation. Specifically, we need to change the prevail implication constraints and add a new set of constraints which we call the cycle elimination constraints.

Prevail implication constraints
To incorporate the new set of alternative conditions on parallelism, we relax the prevail implication constraints. In particular, we need to ensure that for each state variable c, the value $f \in D_c$ holds if it is required by the prevail condition of action a at plan step t. There are three possibilities: (1) The value f holds for c throughout the period. (2) The value f holds initially for c, but the value is changed to a value other than f by another action. (3) The value f does not hold initially for c, but the value is changed to f by another action. In either of the three cases the value f holds at some point in step t so that the prevail condition for action a can be satisfied. In words, we can prevail a value f implicitly as long as there is a state change that includes f. As before, the prevail implication constraints link the action prevail

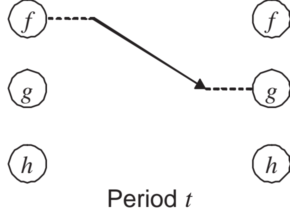


Figure 3: Generalized state change

conditions to the corresponding network arcs. For each $c \in C, f \in D_c, a \in A, (f, f) \in PR_a(c), 1 \leq t \leq T$ we have:

$$x_{a,t} \leq y^c_{f,f,t} + \sum_{g \in D_c, g \neq f} y^c_{f,g,t} + \sum_{g \in D_c, g \neq f} y^c_{g,f,t} \quad (6)$$

The interpretation of these constraints is that in each plan step we can prevail a value, change the value, and prevail the new value as shown in Figure 3, where the implicit prevails are indicated by the dashed lines.

Cycle elimination constraints

Figure 3 also indicates that there are implied orderings between actions. Actions that prevail the value f must be executed before the action that changes f into g. Likewise, the action that changes f into g must be executed before actions that prevail g. Hence, these prevail implication constraints ensure that actions can be linearized into some ordering. We just need to make sure that the actions can be linearized into a feasible ordering. The constraints outlined above indicate that there is an ordering between the actions, but this ordering could be cyclic and therefore infeasible. To make sure that an ordering is acyclic we start by creating a directed *implied precedence graph* $G = (V, E)$. In this graph the nodes $a \in V$ correspond to the actions, that is, $V = A$. We create an arc, an ordering, between two actions $(a, b) \in E$ if action a has to be executed before action b in time step t, or if b has to be executed after a. In particular, we have

$$\begin{aligned} E &= \bigcup_{\substack{a,b \in A, c \in C, f,g \in D_c, g \neq f: \\ (f,f) \in PR_a(c) \wedge (f,g) \in SC_b(c)}} (a, b) \\ &\cup \bigcup_{\substack{a,b \in A, c \in C, f,g \in D_c, g \neq f: \\ (f,g) \in SC_a(c) \wedge (g,g) \in PR_b(c)}} (a, b) \end{aligned}$$

The cycle elimination constraints ensure that the actions in the final solution can be linearized. They basically involve putting an n-ary mutex relation between the actions that are involved in each cycle. They are stated as follows. For each $1 \leq t \leq T$ we have:

$$\sum_{a \in V(\Delta)} x_{a,t} \leq |V(\Delta)| - 1 \text{ for all cycles } \Delta \in G \quad (7)$$

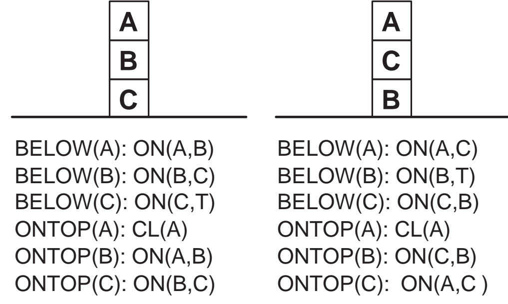


Figure 4: Another Blocksworld instance

Observe that the number of cycle elimination constraints grows exponentially in the number of actions. As a result, it will be impossible to solve the resulting formulation using standard approaches. We address this complication by implementing a branch-and-cut framework in which the cycle elimination constraints are added dynamically; this is discussed in the next section.

State Change Path (kSC) Formulation

Now, we further generalize the g1SC formulation by allowing more than one state change in each state variable. Our aim is to potentially further reduce the number of plan steps needed to execute the plan. To illustrate this idea we consider another Blocksworld example with the $MOVE(x, y, z)$ actions depicted in Figure 4.

In this case, the obvious solution is to execute the following sequence of actions, $MOVE(A, B, T)$, then $MOVE(B, C, T)$, then $MOVE(C, T, B)$, and lastly $MOVE(A, T, C)$. This solution would not be allowed *within a single step* under Graphplan's parallelism. Moreover, it would also not be allowed within a single step in the g1SC formulation. The reason for this is that the number of state changes within the $ONTOP(A)$ state variable is two. First, it changes from $ON(A, B)$ to $OT(A)$, and then it changes from $OT(A)$ to $ON(A, C)$.

As before, however, there does exists an ordering of the four actions that is feasible. The key idea behind this example is to show that we can allow multiple state changes in a single step. If we limit the state changes in a state variable to paths, that is, in a single step each value is visited at most once, then we can still use implied precedences to determine the ordering restrictions.

This formulation uses both the action and flow variables that have been described earlier, respectively $x_{a,t}$ and $y^c_{f,g,t}$. In addition, it uses linking variables, which are defined as:

- $z^c_{f,t} \in \{0, 1\}$, for $c \in C, f \in D_c, 1 \leq t \leq T - 1$; $z^c_{f,t}$ is equal to 1 if the value f links two consecutive plan steps, and 0 otherwise.

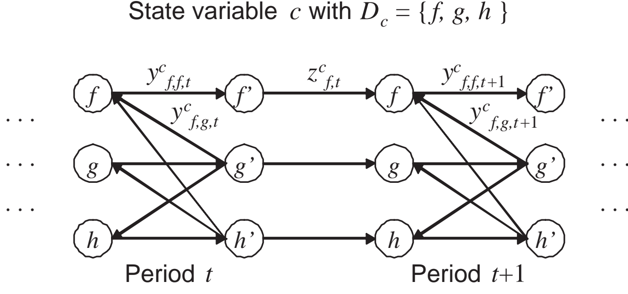


Figure 5: State change flow network that allows multiple state changes per period

The IP formulation of the kSC model is set up in a similar way as the previous models. The objective is still the minimization of the number of actions in the plan, but the constraints are somewhat different and stated below.

State change flow constraints
In this formulation we limit the number of state transitions for each plan step to k_c where $k_c = |D_c| - 1$ for each $c \in C$. Again the state transitions of each multi-valued state variable are represented by paths in an appropriately defined network. In this network, nodes appear in stages (levels) and correspond to the values of the state variable. However, each stage consists of two layers. If we set up an IP encoding with a maximum plan length T then there will be T stages. Arcs within a stage correspond to state changes or to the persistence or no-op of values. Arcs between stages just make sure that all stages are connected to each other.

Figure 5 displays a network corresponding to the state variable c with domain $D_c = \{f, g, h\}$ that allows multiple state changes per stage. The arcs going rightwards and labeled by $y^c_{f,f,t}$ correspond to the persistence or no-op of value f. The arcs going leftwards and labeled by $y^c_{f,g,t}$ correspond to the state changes. The arcs going rightwards and labeled $z^c_{f,t}$ connect two consecutive stages. Note that with unit capacity on the arcs, any path in the network can visit each node at most once. The resulting network flow constraints for each multi-valued state variable $c \in C$ are given as follows:

$$\sum_{g \in D_c : f \neq g} y^c_{g,f,1} + \mathbf{1}\{f \in I\} = y^c_{f,f,1} \quad (8)$$

$$y^c_{f,f,1} = \sum_{g \in D_c : f \neq g} y^c_{f,g,1} + z^c_{f,1} \quad (9)$$

$$\sum_{g \in D_c : f \neq g} y^c_{g,f,t} + z^c_{f,t-1} = y^c_{f,f,t} \quad (10)$$

$$y^c_{f,f,t} = \sum_{g \in D_c : f \neq g} y^c_{f,g,t} + z^c_{f,t} \quad \text{for } 1 \leq t \leq T-1 \quad (11)$$

$$\sum_{g \in D_c : f \neq g} y^c_{g,f,T} + z^c_{f,T-1} = y^c_{f,f,T} \quad (12)$$

$$y^c_{f,f,T} = \sum_{g \in D_c : f \neq g} y^c_{f,g,T} + \mathbf{1}\{f \in G\} \quad (13)$$

The state change flow constraints determine the network structure. There is a supply of one unit of flow at the source nodes (8), a demand of one unit of flow at the sink nodes (13), and there is a balance of flow at all the intermediate nodes (9) through (12).

State change and prevail Implications
The interactions that actions impose upon different state variables are again represented by the state change and prevail implication constraints that we defined for the generalized single state change formulation (g1SC). That is, for all $c \in C, f, g \in D_c, f \neq g, 1 \leq t \leq T$, we have:

$$\sum_{a \in A : (f,g) \in SC_a(C)} x_{a,t} = y^c_{f,g,t} \quad (14)$$

$$x_{a,t} \leq y^c_{f,f,t} \quad \text{for } c \in C, (f,f) \in PR_a(C). \quad (15)$$

Cycle elimination constraints
The implied precedence graph for this formulation is given by $G' = (V, E')$. It has an extra set of arcs to incorporate the implied precedences that are introduced when two actions imply a state change in the same class $c \in C$. The nodes $a \in V$ again correspond to actions, and there is an arc $(a, b) \in E'$ if action a has to be executed before action b in the same time step, or if b has to be executed after a. More specifically, we have

$$E' = E \cup \bigcup_{\substack{a,b \in A, c \in C, f,g,h \in D_c : \\ (f,g) \in SC_a(c) \wedge (g,h) \in SC_b(c)}} (a, b)$$

As before, we need to ensure that the actions in the solution plan can be linearized into a valid ordering, and for each $1 \leq t \leq T$ we have:

$$\sum_{a \in V(\Delta)} x_{a,t} \leq |V(\Delta)| - 1 \text{ for all cycles } \Delta \in G' \quad (16)$$

Branch-and-Cut Framework

IP formulations are usually solved with an LP-based branch-and-bound algorithm. The basic structure of LP-based branch-and-bound involves an enumeration tree, in which branches are pruned according to bounds provided by the LP relaxation. A relatively large number of solvers is available (the best-known is probably CPLEX (ILO 2002)), and in the last few years there have been significant improvements in their performance (Bixby 2002).

For our formulations this standard approach is largely ineffective due to the large number (exponential many) of cycle elimination constraints in (7) and

(16). While it is possible to reduce the number of constraints by introducing additional variables, the resulting formulations would still be intractable for all but the smallest problem instances. Therefore, we solve the IP formulations within a so-called *branch-and-cut* framework which considers the cycle elimination constraints implicitly. A branch-and-cut algorithm is a branch-and-bound algorithm in which certain constraints are generated dynamically throughout the branch-and-bound tree. If, after solving the LP relaxation, we are unable to prune the node on the basis of the LP solution, the branch-and-cut algorithm tries to find a violated cut, that is, a constraint that is valid but not satisfied by the current solution (this is also known as the *separation* problem). If one or more violated cuts are found, the constraints are added to the formulation and the LP is solved again. If none are found, the algorithm creates a branch in the enumeration tree (if the current LP solution is fractional) or generates a feasible solution (if the current LP solution is integral). Branch-and-cut algorithms have successfully been applied in solving hard large-scale IP problems in a wide variety of applications including scheduling, routing, graph partitioning, network design, and facility location problems (Caprara & Fischetti 1997).

In our implementation of the branch-and-cut algorithm, we start with an LP relaxation in which the cycle elimination constraints are omitted. Given a solution to the current LP relaxation (which may be fractional), the separation problem is then to determine whether the current solution violates one of the omitted cycle elimination constraints. If so, we identify one or more violated constraints and add them to the formulation. A separation problem involving cycle constraints occurs in numerous applications, for example, the traveling salesman problem (Padberg & Rinaldi 1991), and has received considerable attention. Algorithms for separating cycle constraints are well-known, and the general idea behind our approach is as follows:

1. Given a solution to the LP relaxation, determine the subgraph G_t for plan step t consisting of all the nodes a for which $x_{a,t} > 0$.
2. For all the arcs $(a, b) \in G_t$, define the weights $w_{a,b} := x_{a,t} + x_{b,t} - 1$.
3. Determine the shortest path distance $d_{a,b}$ for all pairs $((a, b) \in G_t)$ based on arc weights $\bar{w}_{a,b} := 1 - w_{a,b}$ (e.g. using the Floyd-Warshall all-pairs shortest path algorithm).
4. If $d_{a,b} - w_{b,a} < 0$ for some arc $(a, b) \in G_t$, there exists a violated cycle constraint.

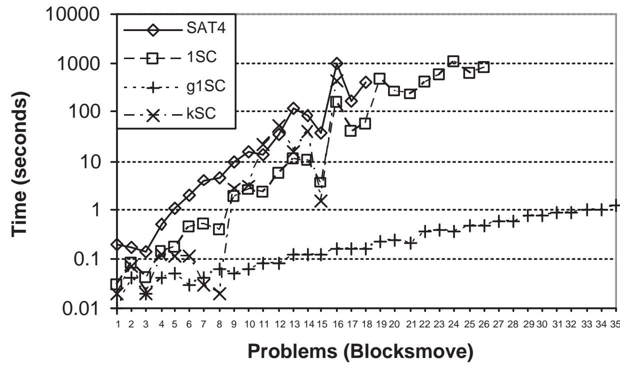
While the general principles behind branch-and-cut algorithms are rather straightforward, there are a number of algorithmic and implementation issues that may have a significant impact on overall performance. At the heart of these issues is the trade-off between computation time spent at each node in the enumeration tree and the number of nodes that are explored. One issue, for example, is to decide when to generate violated cuts. Another issue is which of the generated cuts (if any) should be added to the LP relaxation, and whether and when to delete constraints that were added to the LP before. In our implementation, we have only addressed these issues in a straightforward manner: cuts are generated at every node in the enumeration tree, the first cut found by the algorithm is added, and constraints are never deleted from the LP relaxation. Given the potential impact of more advanced strategies that has been observed in other application, however, we believe there still may be considerable room for improvement.

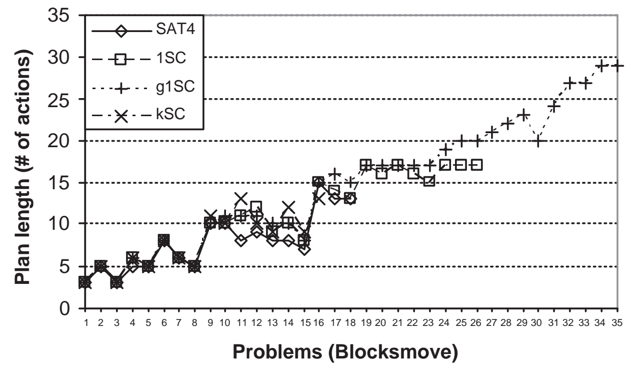
Empirical Results

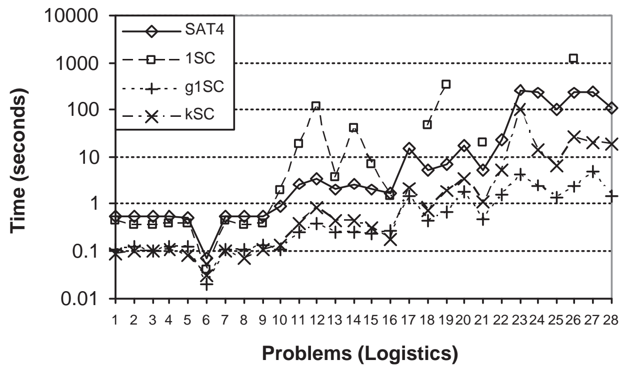
The various IP formulations resulted in three planning systems: the single state change formulation (1SC), the generalized single state change formulation (g1SC), and the state change path formulation (kSC). To evaluate the performance, we compared their results to Satplan04(Siege) (SAT4), and Optiplan(Van den Briel & Kambhampati 2004), an IP-based planner which uses a direct Graphplan encoding. Since our results consistently show that all our new IP encodings dominate Optiplan, we restrict our attention to the comparisons with SAT4. We use four domains: the Blocks and Logistics domain from the international planning competition 2000, and the Driverlog and the Zenotravel domain from international planning competition 2002.

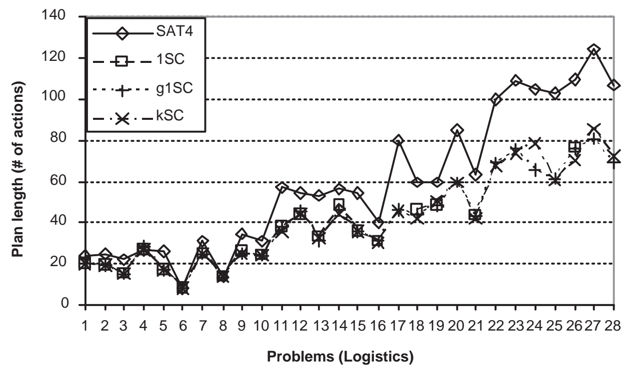
As a first step, we translated these domain instances into a multi-valued state variable description using the translator in the Fast (Diagonally) Downward planner (Helmert 2005). The resulting state description provided the input for our IP formulations. Each IP formulation started with a maximum plan length T equal to 1. If no plan was found, T was automatically increased by one. The IP formulation is thus repeatedly solved until the first feasible plan is found. The IP encodings were solved using ILOG CPLEX 8.1 (ILO 2002), a commercial LP/IP solver, and all tests were performed on a 2.67GHz Linux machine with 1Gb of memory. Each planning system was aborted if no solution had been found after 30 minutes.

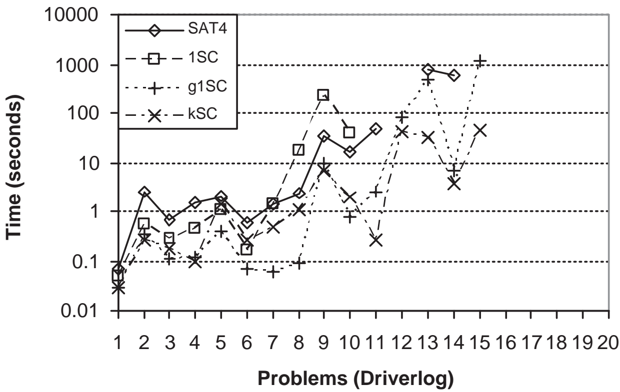
The results are summarized in Figure 6. The leftmost graphs in Figure 6 show the solution times for the different domains, using a logarithmic scale. The rightmost graphs show plan quality, that is, the number of actions in the plan that was found. Overall, the g1SC formulation exhibits the best performance; it consistently outperforms SAT4 by at least one order of magnitude (and oftentimes substantially more). The kSC formulation also performs substantially better than SAT4, except in the Zenotravel domain. In this domain, there are a number of cases in which it fails to converge to a feasible plan within the allotted time. The performance of the 1SC formulation which is using the same notion of parallelism as Graphplan is competitive with SAT4. In addition, both the g1SC and kSC formulations generate plans with fewer actions than SAT4, even though the IP formulations terminate as soon as the first feasible

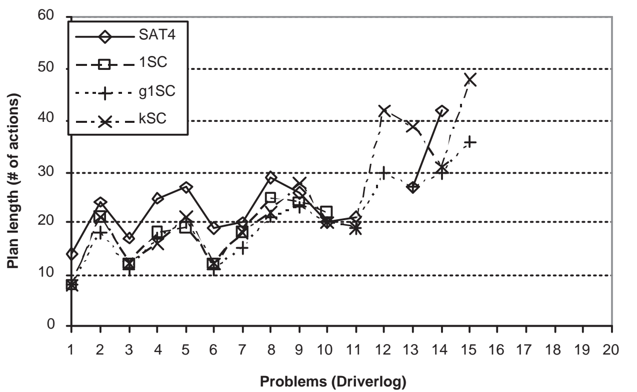


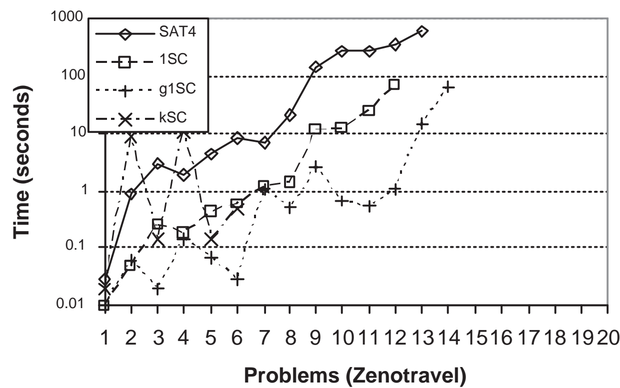












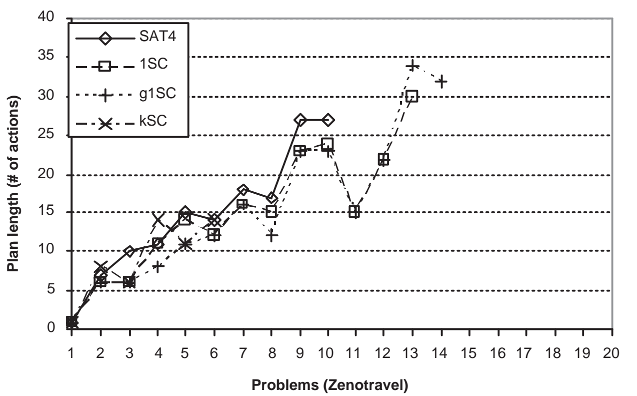


Figure 6: Problem results

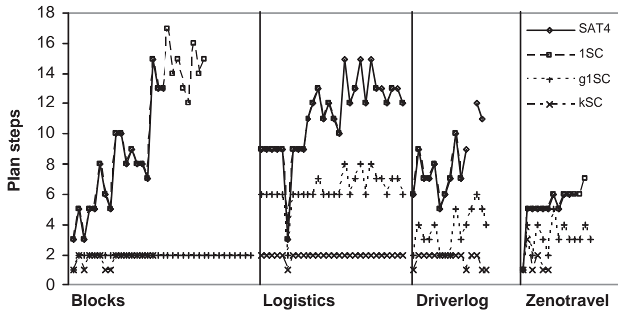


Figure 7: Number of plan steps needed to find a plan

solution is found.

Figure 7 depicts the number of plan steps that are needed to solve each problem. SAT4 and 1SC both use the Graphplan concept of parallelism, and therefore need the same number of plan steps to find a plan. The g1SC and kSC formulations, on the other hand, require significantly fewer plan steps, due to the relaxation of the ordering conditions. In fact, the kSC formulation usually needs only one or two steps (in only instance it requires three steps). While the cycle constraints may theoretically lead to huge formulations, an important advantage of the increased parallelism is therefore an actual *reduction* in the effective size of the resulting formulations. This is illustrated in Table 1, which shows the size of the IP formulations on selected problems after IP preprocessing (as performed by CPLEX). Table 1 also depicts the number of cuts needed by the kSC formulation to find a feasible plan. It shows that only a small fraction of the exponential many cycle constraints are generated by the branch-and-cut procedure. Surprisingly enough, however, no cuts were needed in the g1SC formulation for any of these problems. In fact, the g1SC formulation only required the generation of violated cycle constraints in a few of the Driverlog instances.

Overall, these initial results appear to indicate a tradeoff between the benefits that relaxing parallelism has on size, and the burden that generating additional cycle elimination constraints will have. The 1SC formulation presents one extreme, since Graphplan's concept of parallelism will never result in cycles. The kSC formulation, on the other hand, significantly relaxes the ordering conditions imposed under Graphplan. This, however, may require the generation of large numbers of violated cycle elimination constraints, which can be time-intensive. As such, the g1SC formulation may be viewed as a compromise between these extremes.

Related Work

For a planning formalism that supports multi-valued state variables we refer to the work by Bäckström and Nebel (1995). Planners that have successfully incorporated the use of multi-valued state variables include Fast (Diagonally) Downward (Helmert 2005) and MIPS

	1SC		g1SC		kSC		
Problem	rows	cols	rows	cols	rows	cols	cuts
blo05	403	667	109	127	137	192	19
blo10	1321	2511	25	40	311	610	372
blo15	1687	3546	29	47	426	738	85
blo20	5380	12891	124	189			
blo25	6878	18200	346	366			
blo30			299	240			
blo35			158	152			
log05	690	740	261	314	220	225	2
log10	929	990	193	254	217	215	14
log15	1897	2010	361	459	486	489	14
log20	4102	4322	1818	1954	861	880	89
log25	6974	7438	1789	2057	1675	1761	31
dri05	1324	1298	605	635	341	432	123
dri10	2901	2705	1165	1115	944	1243	43
dri15					1584	2326	448
zen05	540	1160	229	441	165	342	32
zen10	2253	5036	864	1726			

Table 1: Encoding size in number of rows (constraints) and number of columns (variables)

(Edelkamp & Helmert 2000).

The generalization of planning graph-based parallelism and subsequent constraint generation was first introduced by Dimopoulos *et al.* (1997) and used for experimentation by Rintanen (1998). Cayrol *et al.* (2001) also allow more general notions of parallelism in their planning system, called LCGP. Their approach, however, ensures that constraint generation is not needed after a solution is found. Other approaches that use dynamic constraint generation during search, but do not allow more parallelism, include RealPlan (Srivastava, Kambhampati, & Do 2001) and LPSAT (Wolfman & Weld 1999).

Both the use of multi-valued state variables and the introduction of more general parallelism concepts have received considerable attention. To the best of our knowledge, however, our work is the first that shows the potential benefits of combining the two.

Conclusions

Despite the potential flexibility offered by IP encodings for planning, in practice planners based on IP encodings have not been competitive with those based on CSP and SAT encodings. We believe that this state of affairs is more a reflection on the type of encodings that have been tried until now, rather than any inherent shortcomings of IP as a combinatorial substrate for planning. In this paper we introduced a sequence of novel IP formulations whose performance scale up and surpass that of state-of-the art SAT-based approaches to planning. The success of our encodings is based on three interleaved ideas: (1) modeling state changes in individual (multi-valued) state variables as flows in an appropriately defined network, (2) generalizing the notion of action parallelism to loosen the dependency between encoding length and solution length, and (3) us-

ing a branch and cut framework (rather than a branch and bound one), to allow for incremental addition of constraints during the solving phase.

We believe that our results are significant in that they could revive interest in IP encodings for planning. In the future, we intend to exploit the competitive foundation provided by our framework to explore more complex classes of planning problems that have natural affinity to IP encodings, including handling of numeric resource constraints and generation of cost sensitive plans (in the context of non-uniform action costs). We would also like to explore the interface between planning and scheduling by coupling IP-based schedulers to our planner (using the same general branch-and-cut framework). In the near term, we also plan to (1) improve the engineering of our branch-and-cut framework, (2) strengthen the IP formulations by taking into account mutexes (these will be different from Graphplan mutexes due to the different notion of parallelism), (3) further analyze the impact of more general notions of parallelism, and (4) increase the scale of problems that can be solved using column generation techniques.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Computational Intelligence* 11(4):625–655.

Bixby, R. 2002. Solving real-world linear programs: A decade and more of progress. *Operations Research* 50(1):3–15.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1636–1642.

Bockmayr, A., and Dimopoulos, Y. 1998. Mixed integer programming models for planning problems. In *Working notes of the CP-98 Constraint Problem Reformulation Workshop*.

Bockmayr, A., and Dimopoulos, Y. 1999. Integer programs and valid inequalities for planning problems. In *Proceedings of the European Conference on Planning (ECP-99)*, 239–251. Springer-Verlag.

Bylander, T. 1997. A linear programming heuristic for optimal planning. In *AAAI-97/IAAI-97 Proceedings*, 694–699.

Caprara, A., and Fischetti, M. 1997. *Annotated Bibliographies in Combinatorial Optimization*. John Wiley and Sons. chapter Branch and Cut Algorithms, 45–63.

Cayrol, M.; Régnier, P.; and Vidal, V. 2001. Least commitment in graphplan. *Artificial Intelligence* 130(1):85–118.

Dimopoulos, Y., and Gerevini, A. 2002. Temporal planning through mixed integer programming. In *Proceeding of the AIPS Workshop on Planning for Temporal Domains*, 2–8.

Dimopoulos, Y.; Nebel, B.; and Koehler, J. 1997. Encoding planning problems in nonmonotic logic programs. In *Proceedings of the 4th European Conference on Planning (ECP-97)*, 167–181.

Dimopoulos, Y. 2001. Improved integer programming models and heuristic search for ai planning. In *Proceedings of the European Conference on Planning (ECP-01)*, 301–313. Springer-Verlag.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *Proceedings of the European Conference on Planning (ECP-99)*, 135–147. Springer-Verlag.

Edelkamp, S., and Helmert, M. 2000. On the implementation of MIPS. In *Proceedings of the ICAPS Workshop on Model-Theoretic Approaches to Planning*.

Helmert, M. 2005. The Fast Downward planning system. Technical Report 217, Albert-Ludwigs-Universität Freiburg, Institut für Informatik. (Submitted to JAIR).

ILOG Inc., Mountain View, CA. 2002. *ILOG CPLEX 8.0 user's manual*.

Johnson, E.; Nemhauser, G.; and Savelsbergh, M. 2000. Progress in linear programming based branch-and-bound algorithms: An exposition. *INFORMS Journal on Computing* 12:2–23.

Kautz, H., and Walser, J. 1999. State-space planning by integer optimization. In *AAAI-99/IAAI-99 Proceedings*, 526–533.

Kautz, H. 2004. SATPLAN04: Planning as satisfiability. In *Working Notes on the International Planning Competition (IPC-2004)*, 44–45.

Padberg, M., and Rinaldi, G. 1991. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review* 33:60–100.

Rintanen, J. 1998. A planning algorithm not based on directional search. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, 617–624.

Srivastava, B.; Kambhampati, S.; and Do, M. 2001. Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in realplan. *Artificial Intelligence* 131(1-2):73–134.

Van den Briel, M., and Kambhampati, S. 2004. Optiplan: Unifying IP-based and graph-based planning. In *Working Notes on the International Planning Competition (IPC-2004)*, 18–20.

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the use of integer programming models in ai planning. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 304–309.

Wolfman, S., and Weld, D. 1999. The lpsat engine and its applicationto resource planning. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 310–317.