The Knowledge Engineering Review, Vol. 15:1, 2000, 85–100
Printed in the United Kingdom

Applying integer programming to AI planning*

THOMAS VOSSEN[1], MICHAEL BALL[1], AMNON LOTEM[2] and DANA NAU[2]

[1]*Robert H. Smith School of Business and Institute for Systems Research, Email: {tvossen, mball}@rhsmith.umd.edu;*
[2]*Department of Computer Science and Institute for Systems Research, University of Maryland, College Park, MD 20742, USA, Email: {lotem, nau}@cs.umd.edu*

Abstract

Despite the historical difference in focus between AI planning techniques and Integer Programming (IP) techniques, recent research has shown that IP techniques show significant promise in their ability to solve AI planning problems. This paper provides approaches to encode AI planning problems as IP problems, describes some of the more significant issues that arise in using IP for AI planning, and discusses promising directions for future research.

1 Introduction

AI planning is concerned with developing automated methods for generating and reasoning about sequences of actions to perform certain tasks or achieve certain goals. Planning problems are similar, but in some sense more general, than scheduling problems, which have received long-standing attention in the operations research community. What separates AI planning from scheduling is that solving a planning problem typically involves determining both *what* actions to do and *when* to do those actions. Scheduling problems, on other hand, typically involve "only" the order in which a prespecified set of actions should occur. This difference in scope, however, has also led to a different focus: solving planning problems in AI typically amounts to solving hard *feasibility* problems, whereas solving scheduling problems in OR usually amounts to solving hard *optimization* problems.

Integer Programming (IP) has been used to address a variety of hard combinatorial optimization problems, including certain classes of scheduling problems, such as crew and fleet scheduling. Despite the historical difference in focus between AI planning techniques and IP techniques, recent research has shown that IP techniques show significant promise in their ability to solve AI planning problems (e.g. Kautz and Walser (1999); Bockmayr and Dimopoulos (1998). Therefore, we feel that this is an important and fertile area for future research, on topics such as (1) how to choose and combine techniques from AI planning and IP optimization, in order to better solve AI planning problems, and (2) how to make use of IP's ability to solve optimization problems with numerical constraints, to extend the scope of AI planning to include planning problems that involve numeric computations.

While most real-world planners usually rely on domain-specific information and application-specific techniques that capture the structure of the problem at hand, the simplest version of the planning problem is completely domain-independent. It assumes that the environment is static and deterministic, and its formulation is given by three inputs: a description of the *initial state* of the

This work was supported in part, by the following grants and contracts: Army Research Laboratory DAAL01-97-K0135, Naval Research Laboratory N00173981G007, Air Force Research Laboratory F306029910013, and NSF DMI-9713718.

world, a description of the *goal* state, and a description of the possible actions that can be performed and their effects on the state of the world.

Most of the methods used to solve such problems are based on the so-called STRIPS representation, which is described in Section 2. STRIPS (Fikes and Nilsson, 1971) was an early AI planning system. Its plan representation technique provided a simple yet general format for specifying operators, and a clear semantics for change over time. Thus, although the STRIPS planning algorithm itself is no longer used, its way representing planning problems and their solutions is used in most subsequent AI planning algorithms (the primary exception being Hierarchical Task Network (HTN) planning (Russell and Norvig, 1995, pp. 371–380)).

Even though the development of planners that are based on the STRIPS representation has received considerable attention for nearly three decades now, recent developments have had a significant impact on the field. Two approaches in particular, the Graphplan system (Blum and Furst, 1995) and the SATplan system (Kautz and Selman, 1996) have been able to quickly solve planning problems that are orders of magnitude harder than what was considered the state of the art less than 10 years ago. The Graphplan system aims to exploit potential parallelism inherent in most plans. It is based on the notion of a ''planning graph'', which allows for a compact representation of the problem and for the efficient inference of many useful constraints inherent in the problem. In the SATplan approach, planning problems are converted into propositional formulae. The resulting satisfiability problem is then solved by a general systematic or stochastic SAT solver. It should be noted that both approaches are complementary, as the planning graph representation can readily be expressed in terms of propositional clauses (Kautz and Selman, 1996, 1999).

To a large extent, the success of the SATplan approach coincides with recent advances in the development of satisfiability solvers, which has resulted in powerful new general reasoning algorithms, that has been used not only for planning problems but also for many other difficult tasks (i.e. graph coloring, circuit verification). In spite of this general applicability, the propositional representations used in SAT solvers also have some inherent limitations. One problem is how to incorporate numeric constraints. For instance, converting a boolean linear inequality into a propositional representation may require an exponential number of clauses. An example of this is given by Hooker (1994) (taken from Barth (1993)), who reports that the boolean inequality

$$\begin{aligned} 300x_1 + 300x_2 + 285x_3 + 265x_4 + 265x_5 + 230x_6 \\ + 230x_7 + 190x_8 + 200x_9 + 400x_{10} + 200x_{11} + 400x_{12} \\ + 200x_{13} + 400x_{14} + 200x_{15} + 400x_{16} + 200x_{30} + 400x_{31} \leqslant 2700 \end{aligned}$$

expands to 117,520 non-redundant logical propositions. Numerical constraints (such as capacity and durational constraints), however, do arise in many practical, real-world domains, and the ability to incorporate these constraints would therefore significantly enhance the power of domain-independent planners.

To extend propositional representations of AI planning problems to incorporate numerical constraints, it is therefore necessary to find efficient ways to

1. Integrate numerical constraint representations with propositional representations for the AI planning problem.
2. Integrate numerical constraint solving *techniques* with propositional reasoning methods for the AI planning problem.

Numerical constraints have received ample attention in the field of Integer Programming. The field of Integer Programming has a long history in OR, and incorporates a declarative framework consisting of linear constraints and a linear objective function, together with powerful solution techniques based on the linear programming relaxation of the problem.

The general problem of integrating logic-based methodologies and mathematical programming techniques has received considerable attention recently. Two notable approaches include the ''Mixed Logical/Linear Programming'' framework developed by Hooker and Osorio (1997), in which the structure of the constraints indicates how linear programming and propositional

reasoning can interact to solve the problem, and the "Branch and Infer" framework by Bockmayr and Kasper (1998), which combines Integer and Constraint Logic Programming techniques.

Another approach, however, might be to incorporate both the numerical constraints and the propositional constraints into a single Integer Programming formulation. While this may appear outdated given the abovementioned attention to integrated frameworks, it should be noted that these frameworks are developed to enable the flexible and intuitive representations for large classes of problems that have significant numerical and logical components. For a specific problem class however, i.e., the AI planning problem, the formulation as an Integer Program may prove to be useful, in that it may lead to more efficient representations of the problem than would be obtained otherwise.

The formulation of propositional representations as Integer Programs itself is quite straightforward: it is well known that propositional clauses can be converted into 0–1 inequalities (see, for instance, Blair et al (1986) or Hooker (1988)). For example, the clause

$$x_1 \vee \neg x_2 \vee x_3$$

is equivalent to the 0–1 inequality

$$x_1 + (1 - x_2) + x_3 \geq 1; x_1, x_2, x_3 \in \{0, 1\}.$$

While this representation has been used to solve general satisfiability problems with Integer Programming methods, it should be noted that this has not been very successful in comparison with logic-based SAT solvers. For the specific problem class of AI planning problems, however, this conversion of the propositional representation may not necessarily be the best one.

Our research is primarily concerned with the development of "strong" Integer Programming formulations for the AI planning problem. Specifically, we use information about the problem structure that is specific to AI planning problems, so as to yield representations that are more efficient. It should be emphasized that the use of strong IP formulations is not limited to solving AI planning problems by Branch and Bound methods for Integer Programming alone. Strong IP formulations may also prove useful when using Integer Local Search methods (Walser, 1998) or CLP methods for pseudo-boolean constraints (Barth, 1995). In addition, strong formulations may lead to much better lower bounds when used as an admissible heuristic in partial-order planning (Bylander, 1997).

In the remainder of this paper, we discuss the use of Integer Programming formulations in AI planning in greater detail. In the next section, we review the STRIPS representation for AI planning, the SATplan approach, and Integer Programming formulations and techniques. Section 3 discusses the formulation of AI planning problems as Integer Programs, and section 4 provides a discussion of the issues that arise when solving the resulting Integer Programs. We conclude with an overview of future research directions.

2 Background

In this section, we define the STRIPS representation for AI planning, discuss the SATplan approach to solving STRIPS planning problems, and give a brief overview of Integer Programming formulations and techniques.

2.1 STRIPS representation

For our purposes, a STRIPS-style planning problem is a triple $R = (S, G, O)$ as defined below:

- S, the *initial state*, is a finite collections of ground atoms of some first-order language $\mathcal{L}$. Intuitively, S is a conjunct that tells us which ground atoms are true at the beginning of the planning problem: if a ground atom A is in the state S, then A is true in the state S, and if $B \notin S$, then B is false in the state S. Thus, S is simply an Herbrand interpretation (cf. Shoenfield, 1967)

for the language $\mathcal{L}$ and hence each formula of first-order logic is either satisfied or not satisfied in S according to the usual first-order logic definition of satisfactiion.

- G, the *goal condition*, is a conjunction of ground atoms in $\mathcal{L}$. G represens the conditions that we want to be true at the end of the planning problem.
- O is a set of *planning operators*. Each planning operator $o \in O$ is a 4-tuple (Name_o, Pre_o, Del_o, Add_o), where:
  - Name_o, the *name* of the operator, is a syntactic expression of the form $o(X_1, \ldots, X_n)$, where each X_i is a variable symbol of $\mathcal{L}$.
  - Pre_o, the *precondition list*, is a finite set of literals (i.e., atoms and negated atoms) such that each variable symbol is in the list $X_1, \ldots, X_n$ given above. Pre_o gives the conditions that a state must satisfy in order for the operator to be applicable to that state.
  - Del_o, the *delete list*, is a finite set of atoms such that each variable symbol is in the list $X_1, \ldots, X_n$ given above. Pre_o gives the conditions that the operator will delete when we apply it to a state.
  - Add_o, the *add list*, is a finite set of atoms such that each variable symbol is in the list $X_1, \ldots, X_n$ given above.[1] Pre_o gives the conditions that the operator will add when we apply it to a state.

  Just as with the initial state, each of the sets Pre_o, Del_o, Add_o is taken to be a conjunct.[2]

An *action* (also sometimes called a *step*) is a ground instance $\alpha = (\text{Name}_\alpha, \text{Pre}_\alpha, \text{Del}_\alpha, \text{Add}_\alpha)$ of any planning operator $o \in O$. If α is an action and S satisfies P_α, then α is *applicable* to a state S, and the *result* of applying α to s is defined to be the following state:

$$\text{result}(S, \alpha) = (S - D_\alpha) \cup A_\alpha.$$

Given a planning problem (S, G, O), a *plan* is a sequence $(\alpha_1, \ldots, \alpha_k)$ of actions. The plan is *applicable* to the state S if the following state is defined:

$$R = \text{result}(\text{result}(\ldots(\text{result}(\text{result}(S, \alpha_1), \alpha_2), \ldots), \alpha_{k-1}), \alpha_k).$$

If the plan is applicable to S, then it solves the planning problem (S, G, O) if the state R satisfies the goal condition G.

Note that in the STRIPS formulation, the notion of time is very simple: time consists of a finite sequence of time instants (0 to n), each action takes exactly one unit of time to perform, and only one action can be performed at each time instant.

A simple example of a planning problem using the STRIPS representation is the so-called rocket domain (see Blum and Furst (1997)). The rocket domain has three operators: Load, Unload and Move. A piece of cargo can be loaded into the rocket if the rocket and the cargo are in the same location. A rocket may move if it has fuel, but performing the move operation uses up the fuel. The operators may be defined as follows:

Name: $MOVE(R, X, Y)$
Pre: $AT(R, X)$, $HAS\text{-}FUEL(R)$
Del: $AT(R, X)$, $HAS\text{-}FUEL(R)$
Add: $AT(R, Y)$

Name: $LOAD(R, X, C)$
Pre: $AT(R, X)$, $AT(C, X)$
Del: $AT(C, X)$
Add: $IN(C, R)$

[1] In some recent versions of the STRIPS operator representation (e.g., Weld (1999)), the Add and Del lists are combined into a single list of literals called the "Effects" list, in which positive literals represent additions and negative literals represent deletions. However, that representation is basically equivalent to the one we are using here.

[2] The original STRIPS planning system (Fikes and Nilsson, 1971) allowed some of these sets to be more general logical expressions rather than just conjuncts; but this capability was disallowed in subsequent versions of STRIPS (Nilsson, 1980).

Name: $UNLOAD(R, X, C)$
Pre: $AT(R, X)$, $IN(C, R)$
Del: $IN(C, R)$
Add: $AT(C, R)$

A typical problem might have one or more rockets and some cargo in a start location with a goal of moving the cargo to some number of destinations.

2.2 SAT-based planning systems

In recent years, satisfiability testing has become one of the most successful methods for solving state-space planning problems (Weld, 1999). Kautz and Selman (1996) showed using their SATplan planner, that for a wide range of planning problems, compiling a problem into a propositional logic, and solving it using a general randomized SAT solver is competitive with, or superior to, solving the problem with the best specialized planning systems. Two main factors contributed to the success of SATplan:

- The particular conventions on how to represent a planning problem in propositional logic (the "*encoding*").
- The use of new general powerful SAT solvers, such as Walksat (Selman, Kautz and Cohen 1996), for solving the propositional logic representation of the planning problem.

Since 1996, SAT-based planners have continued to improve, benefiting from a continuous progress in the performance of general SAT engines, and from improved ways to encode the planning problem in propositional logic. The next subsections describes briefly how a planning problem can be encoded in propositional logic, and which kind of SAT solvers are being used nowadays in SAT-based planners. For a more detailed survey, see Weld (1999).

2.2.1 Encoding a planning problem in propositional logic

Most propositional-logic encodings of planning problems are based on the STRIPS-style planning problems described in section 2.1, with one important difference. In the STRIPS representation of a plan, one action can be performed at each time instant. However, in most propositional-logic representations, more than one action can be performed at the same time, as long as the actions do not conflict with each other (i.e., if they do not assert contradictory effects, and the effects of one do not negate the preconditions of the other). Thus, the length of a plan (the number of time units required to perform it) may differ from the number of actions in the plan.

To encode a planning problem in propositional logic, the basic idea is to start with a STRIPS-style problem description, and create a propositional formula that includes encodings of the initial state and the goal, axioms describing the preconditions of the actions that might occur at each time instance, axioms describing what will change and what will remain the same if those actions are performed, and a statement that a plan of length n exists for some n. Thus the formula will be satisfiable if a plan of length n exists; and in this case, the truth values of the propositions will tell what the plan is. The planning system starts with encoding the problem for an initial plan length n_0 and invokes a SAT engine to solve the propositional logic problem. If no solution is found for n_0 the planner increases incrementally the length of the plan that is being searched, until a plan is found.

In the basic encoding (the **regular** encoding), a logical variable represents either an action occurring at a specific time (e.g., fly(airplane1, BMI, JFK, 3)) or a time-varying condition (*fluent*) holding at a a particular time (e.g., at(airplane1, JFK, 4)).[3] An action starts at the specified time instance t and ends at the next time instance, where $t \in 0, \ldots, n-1$. A fluent corresponds to a time instance $t \in 0, \ldots, n$.

[3] AI researchers first used the term "fluent" to refer to an object or condition X whose value varies over time (cf. Russell and Norvig (1995)). However, in the literature on satisfiability methods for planning, it has become common to use the term "fluent" to refer to the instance X_t of X at the time t. Thus, that is what we do here.

The number of variables representing actions can be reduced by using an alternative action representation such as **simple action splitting** (Kautz and Selman, 1996), **overloaded splitting** or **bitwise** representation (Ernst, Millstein and Weld, 1997). However, these representations usually limit the ability to have parallel actions in the plan.

The encoding includes a set of axioms, defined over the action and fluent variables, to express the following features of a planning problem:

- *The initial state and goals*. These indicate which fluents are true and which are false at time 0. and which fluents should be true at time n.
- *Action schemata*. The execution of a ground action at tune t implies that its preconditions hold at time t, and its effects hold at time $t+1$. For example, the action of flying *airplane*1 from BWI to JFK at time t implies that at time t *airplane*1 is at BWI, and at time $t+1$ it is at JFK but not at BWI. Thus, we get the following axiom $\forall t \in 0, \ldots, n-1$:

$$fly(airplane1, BWI, JFK, t) \Rightarrow \\ at(airplane1, BWI, t) \wedge \neg at(airplane1, BWI, t+1) \wedge at(airplane1, JFK, t+1)$$

- *Frame axioms*. These describe what fluents will not change actions occur, as described below.

Frame axioms are expressed either by classical frame axioms or by explanatory frame axioms. **Classical frame axioms** (McCarthy and Hayes, 1969) declare which fluents are left unchanged by each action. **Explanatory frame axioms** (Haas, 1987) specify the the set of actions that could have occurred to achieve a state change. For example, having *airplane*l at time $t+1$ in JFK, but not having it in JFK at time t implies that *airplane*1 flew from some other airport to JFK at time t:

$$\neg at(airplane1, JFK, t) \wedge at(airplane1, JFK, t+1) \Rightarrow \\ fly(airplane1, BWI, JFK, t) \vee \cdots \vee fly(airplane1, Reagan, JFK, t)$$

Explanatory frame axioms have the advantage that they permit parallelism: several actions may occur simultaneously. **Conflict** exclusion axioms should be specified to prevent contradicting actions from occurring at the same time. For example, *airplane*1 cannot perform flights to JFK from Dulles and BWI at the same time, thus, either it does not perform the BWI–JFK flight or it does not perform the Dulles–JFK flight:

$$\neg fly(airplane1, BWI, JFK, t) \vee \neg fly(airplane1, Dulles, JFK, t)$$

Empirical results (Ernst, Millstein and Weld, 1997) showed that in most cases, explanatory frame axioms with conflict exclusions are superior to classical frame axioms.

Kautz and Selman suggested several additional encodings:

- The *Graphplan-based* encoding (Kautz, McAllester and Selman 1996), which is similar to the combination of the *regular* representation of actions and *explanatory* frame axioms. In this encoding, actions imply their preconditions, but there are no axioms in which actions imply their effects. The Graphplan-based encoding is used by the Blackbox planner (Kautz and Selman, 1998). Blackbox constructs first a planning graph (Blum and Furst, 1997) and then converts the graph into a CNF. In that way the planner benefits from the powerful simplification algorithm employed by Grapliplan to construct the graph, prior to invoking the SAT engine. As a result, Blackbox is currently one of the fastest planners.
- The *state-based encoding*, in which variables corresponding to actions are eliminated (Kautz and Selman 1996).
- The *causal-encoding*, which is based on a representation used by partial-order planners (Kautz, McAllester and Selman 1996).

As shown by Huang, Selman and Kautz (1999), declarative domain specific control knowledge can be added to the encoding of the planning problem to speed-up the planning process.

2.2.2 SAT solvers

General purpose SAT solvers, are used to solve the propositional-logic representation of the planning problem. A short survey of SAT solvers appears in Weld (1999). In brief, SAT solvers can be classified according to their search method: *systematic* or *stochastic*. Systematic solvers are complete while stochastic solvers are not. *Satz* (Li and Annbulagan, 1997) is one of the fastest systematic solvers available. It is based on the DPLL algorithm (Davis et. al., 1962), but uses more sophisticated heuristics for selecting the next variable to branch on. Blackbox also uses a randomized version of *Satz*, which in fact substantially improved its performance (Gomes et al., 1998).

Currently, *Walksat* (Selman, Katz and Cohen (1996)) is one of the fastest stochastic search algorithms for satisfiability testing. It searches locally, and uses random moves to escape from local minima.

2.3 Integer programming

A *mixed integer program* can be represented as

$$\begin{array}{lll} \text{Min} & cx & (0) \\ \text{subject to:} & & \\ & Ax \geq b & (1) \\ & x_1, \ldots, x_p \in Z^+ & (2) \\ & x_{p+1}, \ldots, x_n \in R^+ & (3) \end{array}$$

where A is a $(m \times n)$ matrix, b a m-dimensional column vector, c a n-dimensional row vector, and x a n-dimensional column vector. If $x_1, \ldots, x_p \in \{0, 1\}$, we get a mixed 0–1 program. A vector x^* which satisfies constraints (1)–(3) is called a "feasible solution". If x^*, furthermore, minimizes the objective function (0), it called an optimal solution and cx^* the optimal value.

(Mixed) Integer Programming has a long history in the field of operations research. It provides a very rich modeling capability, and a large number of problems in resource allocation, facility location, distribution, production, reliability and design have been represented by mixed integer programming models (see Nemhauser and Wolsey (1988)). A key notion in solving Integer Programs is the *linear relaxation* of the mixed integer program, in which the integrality constraints are. omitted, i.e.,

$$\begin{array}{ll} \text{Min} & cx \\ \text{subject to:} & \\ & Ax \geq b \\ & x_1, \ldots, x_n \in R^+ \end{array}$$

Solving the LP relaxation, which can be done efficiently in practice, may guide solving the Integer Program. In fact, one of the basic results in Integer Programming Theory states that every Integer Program can be represented by a set of constraints such that the LP relaxation will find the optimal (integer) solution. This "ideal" formulation of the Integer Program corresponds to the *convex hull* of the set of feasible (integer) solutions (see Wolsey (1998) for more details on this concept). In practice, however, using the convex hull formulation is hardly ever possible. Usually, it is very hard to characterize the convex hull and, moreover, an exponential number of inequalities might be required to characterize the convex hull.

While using the convex hull formulation is normally not possible, it is desirable to find "strong" formulations, in the sense that the LP relaxation closely approximates the convex hull of the Integer Program. In particular, given two formulations $\{\min cx \mid A^1 x \geq b, x \in Z^n\}$ and $\{\min cx \mid A^2 x \geq b, x \in Z^n\}$, formulation 1 is stronger than formulation 2 if $\{x \in R^n \mid A^1 x \geq b, x \geq 0\} \subseteq \{x \in R \mid A^{2\cdot} x \geq b, x \geq 0\}$. Stronger formulations are more likely to yield integer solutions and produce better lower bounds for the optimal value of the integer program.

The most effective current approach for solving general integer programs uses the LP relaxation together with branch and bound algorithms. Therefore, the key to the effectiveness of the LP relaxation lies in improving the underlying tree search. The LP relaxation is typically solved at every node in the search tree. Search can be terminated at a node (1) if LP relaxation value indicates that further search could only uncover solutions with objective function values inferior to the best known, (2) if the LP is infeasible, which in turn implies the integer program is infeasible, and (3) if the LP yields an integer solution. Another role the LP relaxation plays is that it provides information useful in deciding which variables to branch on. Since stronger formulations will give better lower bounds and are more likely to yield integer solutions, they have the potential of greatly reducing the number of nodes in the search tree.

3 Integer programming formulations for AI planning

In this section, we discuss various formulations for the AI planning problem using the STRIPS representation. In formulating the IP models, we shall use the following sets

- $\text{pre}_f \subseteq \mathcal{A}$ for all $f \in \mathcal{F}$
  pre_f represents the set of actions which have fluent f as a precondition;
- $\text{add}_f \subseteq \mathcal{A}$ for all $f \in \mathcal{F}$.
  add_f represents the set of actions which have fluent f as an add effect;
- $\text{del}_f \subseteq \mathcal{A}$ for all $f \in \mathcal{F}$.
  del_f represents the set of actions that delete fluent f.

3.1 SATplan-based IP formulations

Since propositional clauses can easily be expressed as linear inequalities, it is straightforward to express SATplan encodings as Integer Programming Models. So, as a first attempt we applied this conversion to the Graphplan-based propositional encoding. This is the regular encoding described in section 2.2.1, which includes axioms for the initial state and the goals, action schemata axioms which express that a ground action implies its preconditions, explanatory frame axioms, and conflict exclusion axioms. The resulting formulation is summarized as follows:

Variables For all $f \in \mathcal{F}$, $i \in 1, \ldots, t+1$ we have fluent variables, which are defined as

$$x_{f,i} = \begin{cases} 1 & \text{if fluent } f \text{ is true in period } i, \\ 0 & \text{otherwise.} \end{cases}$$

For all $a \in \mathcal{A}, i \in 1, \ldots, t$, we have action variables, which are defined as

$$y_{a,i} = \begin{cases} 1 & \text{if action } a \text{ is carried out in period } i \\ 0 & \text{otherwise.} \end{cases}$$

We note that the action variables include the "no-op" maintain operators from Graphplan for each time step and fact, which simply have that fact both as a precondition and as an add effect. "no-op" actions are necessary to propagate the fluent values.

Constraints The constraints are separated into different classes, which can be outlined as follows:

- **Initial/Goal State Constraints** These constraints set the requirements on the-initial and final period, i.e.,

$$x_{f,1} = \begin{cases} 1 & \text{if } f \in \mathcal{I}, \\ 0 & \text{if } f \notin \mathcal{I}. \end{cases}$$
$$x_{f,t+1} = 1 \quad \text{if } f \in \mathcal{G}.$$

- **Precondition Constraints** Actions should imply their preconditions, which is expressed as follows:

$$y_{a,i} \leq x_{f,i} \quad \forall a \in \text{pre}_f, i \in 1, \ldots, t.$$

- **Explanatory Frame Conditions** Backward chaining is expressed as

$$x_{f,i+1} \leq \sum_{a \in \text{add}_f} y_{a,i} \quad \forall i \in 1, \ldots, t, f \in \mathcal{F}.$$

- **Conflict Exclusion Constraints** Actions conflict if one deletes a precondition or add effect of the other. The exclusiveness of conflicting actions is expressed as

$$y_{a,i} + y_{a',i} \leq 1,$$

for all $i \in 1, \ldots, t$, and all a, a' for which there exist $f \in \mathcal{F}$ such that $a \in \text{del}_f$ and $a' \in \text{pre}_f \cup \text{add}_f$.

Objective Function The objective function was set to minimize the number of actions in the plan. It should be noted that, in theory, we could have chosen any objective function, since the constraints guarantee a feasible solution. In practice, however, the choice of an objective function can significantly impact performance.

In addition, we made the following two modifications in the formulation. First, we used the notion of clique inequalities to strengthen the formulation. The basic idea behind this is that the inequalities $x_1 + x_2 \leq 1$, $x_2 + x_3 \leq 1$ and $x_1 + x_3 \leq 1$ can be replaced by a single inequality $x_1 + x_2 + x_3 \leq 1$. This leads to a formulation which is not only more compact, but also stronger, in the sense that the fractional solution $x_1 = x_2 = x_3 = \frac{1}{2}$ is feasible in the first set of inequalities, but not in the second. It should be noted that the ability to detect clique inequalities is available in most of today's commercial solvers.

Secondly, we did not restrict all variables to be 0–1 integers. Specifically, the integrality of the fluent variables $x_{f,i}$ was relaxed, that is, the constraints $x_{f,i} \in \{0, 1\}$ were replaced by $0 \leq x_{f,i} \leq 1$. This is possible because the integrality of these variables is implied by the integrality of the action variables. We note that as a consequence, none of the fluent variables will be selected in the branch and bound tree.

3.2 An alternative formulation

We now describe an alternative formulation of the planning problem; which we shall refer to as the "state-change formulation". The differences with respect to the formulation described in the previous section are twofold. First, the original fluent variables are "compiled away" and suitably defined "state-change" variables are introduced instead. As we will see, this results in a stronger representation of the exclusion constraints. Secondly, we more explicitly restrict the possible propagation of fluents through "no-op"-actions, so as to reduce the number of equivalent feasible solutions. It should be noted that the resulting formulation is similar in spirit to the action-based propositional encodings discussed by Kautz and Selman (1996). However, the introduction of state-change variables prevents the polynomial increase in constraints (clauses) that results from compiling away the fluent variables.

Before giving this formulation, we again first define the variables. The action variables are the same as before, i.e.,

$$y_{a,i} = \begin{cases} 1 & \text{if action } a \text{ is executed in period } i, \\ 0 & \text{otherwise} \end{cases}$$

for all $a \in \mathcal{A}$, $i \in 1, \ldots, t$. Now, however, the "no-op" actions are *not* included, but represented separately by variables $x_{f,i}^{\text{maintain}}$, for all $f \in \mathcal{F}$, $i \in 1, \ldots, t$.

To express the possible state changes, we introduce auxiliary variables $x_{f,i}^{\text{pre-add}}$ $x_{f,i}^{\text{pre-del}}$ and $x_{f,i}^{\text{add}}$, which are defined logically as

$$x_{f,i}^{\text{pre-add}} \equiv \bigvee_{a \in \text{pre}_f/\text{del}_f} y_{a,i},$$
$$x_{f,i}^{\text{pre-del}} \equiv \bigvee_{a \in \text{pre}_f \cap \text{del}_f} y_{a,i},$$
$$x_{f,i}^{\text{add}} \equiv \bigvee_{a \in \text{add}_f/\text{pre}_f} y_{a,i}.$$

Informally, $x_{f,i}^{\text{pre-add}} = 1$ if and only if an action is executed in period i that has f as a precondition but does not delete it. We note that the execution of such an action at a given time step implicitly asserts that the value fluent f is propagated. Similarly, $x_{f,t}^{\text{pre-del}} = 1$ if and only if an action is executed in period i that has f both as a precondition and a delete effect. $x_{f,i}^{\text{add}} = 1$ if and only if an action is executed in period i that has f as an add effect but not as a precondition.

To give an example as to how these variables may be interpreted, consider the rocket domain given in section 2.1. Considering the fluent $AT(R, X)$, we may interpret the auxiliary variables as follows (time step indices are omitted):

$$\begin{aligned} X_{AT(R,X)}^{\text{pre-add}} &\equiv LOAD(R, X, C1) \vee LOAD(R, X, C2) \vee \cdots \\ &\quad \vee UNLOAD(R, X, C1) \vee UNLOAD(R, X, C2) \vee \cdots \\ x_{AT(R,X)}^{\text{pre-del}} &\equiv MOVE(R, X, Y) \vee MOVE(R, X, Z) \vee \cdots \\ x_{AT(R,X)}^{\text{add}} &\equiv MOVE(R, Y, X) \vee MOVE(R, Z, X) \vee \cdots. \end{aligned}$$

The logical interpretation of the auxiliary variables is represented in the IP formulation by the following constraints:

$$\sum_{a \in \text{pre}_f/\text{del}_f} y_{a,i} \geq x_{f,i}^{\text{pre-add}} \tag{1}$$
$$y_{a,i} \leq x_{f,i}^{\text{pre-add}} \quad \forall \quad a \in \text{pre}_f/\text{del}_f \tag{2}$$
$$\sum_{a \in \text{add}_f/\text{pre}_f} y_{a,i} \geq x_{f,i}^{\text{add}} \tag{3}$$
$$y_{a,i} \leq x_{f,i}^{\text{add}} \quad \forall \quad a \in \text{add}_f/\text{pre}_f \tag{4}$$
$$\sum_{a \in \text{pre}_f \cap \text{del}_f} = x_{f,i}^{\text{pre-del}} \tag{5}$$

for all $f \in \mathcal{F}$, $i \in 1, \ldots, t$. The equality in the definition of $x_{f,i}^{\text{pre-del}}$ follows from the fact that all actions that have f both as a precondition and as a del effect are mutually exclusive. As a consequence, these variables can in fact be substituted out, although for reasons of clarity we shall not do so here.

The remaining conflict exclusion constraints can easily be expressed in terms of the auxiliary variables, by stating that $x_{f,i}^{\text{pre-del}}$ is mutually exclusive with $x_{f,i}^{\text{add}}$, $x_{f,i}^{\text{pre-add}}$ and $x_{f,i}^{\text{maintain}}$. However, to strengthen the formulation we assert that $x_{f,i}^{\text{maintain}}$ is mutually, exclusive with $x_{f,i}^{\text{add}}$ and $x_{f,i}^{\text{pre-add}}$. Informally, this means that a fluent can only be propagated at a time step if no action that adds it is executed. The resulting constraints are as follows:

$$x_{f,i}^{\text{add}} + x_{f,i}^{\text{maintain}} + x_{f,i}^{\text{pre-del}} \leq 1 \tag{6}$$
$$x_{f,i}^{\text{pre-add}} + x_{f,i}^{\text{maintain}} + x_{p,i}^{\text{pre-del}} \leq 1 \tag{7}$$

for all $f \in \mathcal{F}$, $i \in 1, \ldots, t$.

The explanatory frame axioms can also be expressed in terms of the auxiliary variables. Since all auxiliary variables that assert the precondition of a fact f at a certain time step (i.e., $x_{f,i}^{\text{pre-add}}$, $x_{f,i}^{\text{maintain}}$ and $x_{f,i}^{\text{pre-del}}$) are mutually exclusive, we have the following constraint:

$$x_{f,i}^{\text{pre-add}} + x_{f,i}^{\text{maintain}} + x_{p,i}^{\text{pre-del}} \leq x_{f,i-1}^{\text{add}} + x_{f,i-1}^{\text{pre-add}} + x_{f,i-1}^{\text{maintain}} \tag{8}$$

for all $f \in \mathcal{F}$, $i \in 1, \ldots, t$.

Finally, we can express the initial/goal state constraints as

$$x_{f,t}^{\text{add}} + x_{f,t}^{\text{pre-add}} + x_{f,t}^{\text{maintain}} \geq 1 \tag{9}$$

for all $f \in \mathcal{G}$, and

$$x_{f,0}^{\text{add}} = \begin{cases} 1 & \text{if } f \in \mathcal{I}, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

The objective function is again set to minimize the number of actions. Also, the integrality requirement of the auxiliary variables variables was again relaxed, as it is implied by the integrality of the action variables.

3.3 Experimental results

We tested the IP formulations on a variety of planning problems from the Blackbox software distribution, and compared the results with those obtained by Blackbox using the systematic Satz solver. The integer programs were solved using Cplex 6.0, a widely used LP/IP solver. In solving the integer programs, we used all of Cplex's default settings, except the following: the initial LP optimum was obtained by solving the dual problem (this proved to be more efficient), and the variable selection strategy used was "pseudo-reduced cost". In addition, the solver was terminated as soon as a feasible integer solution was found. AIl problems were run on a Sun Ultra workstation, with 196 Mhz CPU and 512 MB RAM.

We compared the performance of the SATplan-based IP formulation and the state-change IP formulation with Blackbox. The results are shown in Table 1. "Nodes" represents the number of nodes visited in the branch and bound procedure, and "iterations" the number of simplex iterations performed. AIl times are in seconds. It should be noted that, both for the IP formulations and BlackBox, the results shown are for the problem of finding a feasible solution given the number of tune steps (i.e., t is known in advance and given).

As shown in Table 1, the state-change formulation led to a significant improvement in performance. Whereas the SATPLANbased IP formulation solved only the smallest problems, the state-change formulations solved all, and required both fewer nodes and less computation time. In comparison to Blackbox the state-change formulation usually required more time, though on the whole the numbers of nodes explored were similar. Still, in several cases the number of nodes explored by the statechange formulation was markedly higher than by Blackbox. We believe that this is largely due to the difference in *preprocessing* methods employed by both methods. In particular, Blackbox creates a planning graph prior to instantiating the CNF, which may be viewed as a powerful simplification technique.

To analyze the effect of this preprocessing step, we have also done some preliminary studies on running the planning graph expansion prior to generating the IP formulation. In the IP formulation, we eliminated all variables that corresponded to actions and fluents which did not occur in the planning graph. Our results (which will be described in detail in a forthcoming paper) show that the use of the planning graph indeed has a significant effect on the performance of the IP, both in terms of number of nodes explored and overall computation time, reducing the computation time to an amount that is competitive with Blackbox.

Table 1 Experimental results: IP formulations vs. systematic Blackbox solver

	SATPLAN IP			State-change IP			BlackBox/Satz	
Problem	nodes	its.	time	nodes	its.	time	nodes	time
anomaly	59	1471	3.1	3	161	0.1	3	0.55
bw-12step	*	*	*	4	2037	9.7	3	2.42
bw-large.a	**	**	**	4	4261	36	38	20.8
bw-large.b	**	**	**	28	89048	2500	–	–
att-log2	491	2748	4.9	24	177	0.57	7	0.56
att-log3	99	1296	12.1	33	406	4.2	16	0.58
att-log4	1179	20778	101.4	40	961	5.7	23	0.56
rocket.a	*	*	*	213	40877	140	234	3.37
rocket.b	**	**	**	73	18492	67	630	6.38
log-easy	*	*	*	102	2505	5.8	16	0.62
logistics.a	**	**	**	40	9305	80	41	1.66
logistics.b	**	**	**	30	9532	92	46	2.37
logistics.c	**	**	**	285	89760	1400	39297	79.3

– denotes that no plan was found after 10 hours of computation time.
* denotes that the node limit of 2500 was reached without finding a feasible integer Solution.
** denotes that the resulting formulation was too large to be solved.

4 Issues in using integer programming for AI planning

In this section, we discuss issues that arise in using Integer Programming Models for AI planning problems. In particular, we discuss the importance of "direct" IP formulations and the importance of preprocessing.

4.1 Importance of direct IP formulations

In the previous section, we presented two formulations for the AI planning problem: a conversion of the propositional SAT encoding into linear equalities, and a more "direct" formulation which considers the state changes in the AI planning problem.

While the SATplan-based encoding is straightforward, the importance of using a direct encoding becomes clear when we compare the strength of both formulations. An indication of the strength of the respective formulations can be found by examining the LP relaxations of the problems (see section 2.3). Since the objective function that is used is to minimize the number of actions in the plan, the value of the LP relaxation may be viewed as a lower bound on the number of actions required in the plan. The results for the SATplan-based and the State-change formulation are shown in Table 2. In almost all cases, the state-change formulation has a much higher lower bound. This indicates that its formulation is indeed much stronger, which is critical in solving the Integer Program.

4.2 Importance of preprocessing

While the state-change formulation significantly reduces the number of constraints, the resulting formulations can still be huge, due to the instantiation of all ground actions at each time step. In fact, the large size of the formulations might lead one to believe that IP approaches are not practical. However, we found that the size of the formulation was significantly reduced by standard IP preprocessing, as done by Cplex (version 6). Table 3 shows the effects of preprocessing, and gives the number of variables and clauses before and after standard IP preprocessing.

Table 2 LP relaxation values

Problem	SATplan-based	State-change
anomaly	2.62	5
bw-12step	2.33	5
bw-large.a	*	12
bw-large.b	*	16
att-log2	2.19	6.75
att-log3	1.57	6.75
att-log4	2.89	10.7
rocket.a	12.73	20.6
rocket.b	*	20.6
log-easy	5.28	19.25
logistics.a	*	42.8
logistics.b	*	30.9
logistics.c	*	38.9

*denotes that the resulting formulation was too large to be solved.

In addition, preliminary results clearly appear to indicate the importance of more advanced preprocessing techniques, i.e., the use of planning graph expansion as a simplification algorithm. The importance of preprocessing is in fact studied in a recent paper by Kautz and Selman (1999), who compare various aspects of clausal simplifying techniques in both Graphplan and BlackBox. Given the importance of preprocessing, an interesting topic for further research would would be to analyze the similarities and differences between propositional simplification techniques and planning graph expansion with techniques which are commonly used in IP preprocessing (e.g. Savelsbergh, 1994). For example, the use of mutex calculations in the expansion of the planning

Table 3 Effect of preprocessing

	Before preprocessing		After preprocessing	
Problem	Variables	Constraints	Variables	Constraints
anomaly	600	992	206	435
bw-12step	4752	8224	1682	3700
bw-large.a	7392	12936	2763	6162
bw-large.b	15912	28040	6473	14503
att-log2	8596	12323	256	434
att-log3	44494	66023	2164	4164
att-log4	48900	72822	2557	4918
rocket.a	23744	36018	1573	3007
rocket.b	64009	99074	3686	6994
log-easy	32121	48657	1361	2423
logistics.a	64009	99074	3686	6994
logistics.b	102856	158969	4273	7807
logistics.c	102856	158971	5485	10249

graph appears to be similar to the use of conflict graphs in IP preprocessing (see Atamturk et al., 1998).

4.3 IP solvers

To a large extent, the success of SAT-based planning systems coincides with recent advances in the development of satisfiability solvers, which has resulted in powerful new general reasoning algorithms. However, a similar development has occurred for general LP and IP solvers. There are currently several companies packaging commercial solvers, and the size of the LPs and IN that can be solved has increased dramatically over the last few years. In fact, the most recent versions of Cplex (perhaps the best known commercial solver) typically solve LPs and IN one to two orders of magnitude faster than just one and a half years ago. As the capabilities of these solvers continues to grow, the use of Integer Programming models for AI planning problems may become more and more attractive, in particular if strong formulations can be identified.

5 Research Directions

Although Selman et al. (1997) reported difficulty in making effective use of IP techniques for general propositional reasoning, our results indicate that for a specific problem, i.e., the AI planning problem domain, IP techniques may potentially work well. However, to fully exploit the possibilities of using IP methods for solving AI planning problems and extending the AI planning domain, several issues need further investigation. Research directions that appear to be particularly promising are the following:

- **Incorporation of numerical constraints**
  IP models may provide a natural means of incorporating numeric constraints and objectives into the planning formulation. This capability would be important in many application domains, but it is not available in most existing approaches to AI planning except for a few recent approaches. More specifically, the following questions need to be answered:
  1. *What kinds of numerical conditions are needed for AI planning, and what is the best way to represent them?* Some work is already being done in this area, for use in reasoning about resources (Koehler, 1998; Kautz and Walser, 1999; Wolfman and Weld, 1999). The basic idea in all these approaches is that actions can produce, consume, or provide resources, and that actions' preconditions are extended by resource requirements.[4] In view of this, one obvious research direction is to incorporate the general framework for reasoning about resources into our IP formulations.
  2. *What classes of Integer Programming problems will these numerical conditions produce, and how efficiently can they be solved?* The way in which numeric constraints for AI planning may have a significant influence on the performance, much in the same way as we saw with the various IP formulations. Therefore, the development of strong IP representations that capture common numeric constraints that arise in the planning domain is an issue for further research.

- **Hybrid Solution Approaches**
  Recent research has shown that Integer Programming provides a novel approach for solving AI planning problems. Probably the most natural and fundamental question to consider is how other AI planning techniques can be married with integer programming methods to generate hybrid approaches. This issue is really part of the broader question of how the problem solving methods developed within AI/computer science community can be combined with the techniques developed within the optimization/operations research community. As we mentioned earlier

[4] It should be noted that both the work by Kautz and Walser (1999) and the work by Wolfman and Weld (1999) is based on LP-based techniques.

there is already work progressing in this area, e.g., Hooker and Osorio (1997) and Bockmayr and Kasper (1998). The starting point for such work is to understand the fundamental differences (if any) in the techniques used. For example, it has already been observed that certain integer programming preprocessing techniques are similar to unit resolution in SAT approaches and the creation of a planning graph in the Graphplan system. Once fundamental differences and equivalence are understood then one can begin to construct combined approaches with superior performance.

- **Using Integer Programming to find feasible solutions**
  While the possibility of extending the AI planning framework to allow for complex objective functions is one of the main attractions of using Integer Programming, an interesting question in itself is the use of integer programming to simply find feasible plans. So far, our approach was to minimize the number of actions in the plan, but we terminated after the first feasible solution was found. However, it would certainly seem that there are interesting issues here worth investigating. One could start by asking the question of what is the value of an LP relaxation when the objective function is of minimal interest. Within the context of branch and bound, one can view the LP relaxation as providing four "services": (i) pruning a node based on objective function value; (ii) pruning a node when the LP (and consequently the IP) is infeasible; (iii) pruning a node with the LP generates an integer solution; (iv) guiding branch selection. It would seem that the LP relaxation's value relative to (i) is greatly diminished when the objective functions is of little interest. However, (ii) and (iii) are clearly still important. Certainly, one may wish to investigate alternate branch selection strategies. Another interesting question would be to consider alternate objective functions and even to dynamically change the objective function. The overall research problem would be to redesign branch and bound under the assumption that one is "only" interested in finding a feasible solution.

- **Derivation of cutting planes**
  A final topic for further research concerns the further strengthening of the IP formulation. While the state-change formulation is much stronger than the SATplan-based encoding, the gap between the LP relaxation value and the best feasible solution is sometimes still rather large. Therefore, methods that can automatically derive cutting planes, which cut off part of the linear relaxation without eliminating feasible solutions, at each node of the branch and bound tree may further increase the effectiveness of IP techniques. Of course, a key to the effectiveness of this approach would be the generation of cutting planes that represent strong constraints.

References

Atamturk, A, Nemhauser, GL and Savelsbergh, MWP, 1998. "Conflict Graphs in Integer Programming" *Report LEC-98-03*, Georgia Institute of Technology.

Barth, P, 1993. "Linear 0–1 inequalities and extended clauses" Manuscript, Max-Planck-Institut fur Informatik, W-6600 Saarbrucken, Germany.

Barth, P, 1995. *Logic-based 0–1 constraint programming*. Kluwer Academic.

Blair, CE, Jeroslow, RG and Lowe, JK, 1986. "Some results and experiments in programming techniques for propositional reasoning" *Computers and Operations Research* **13** 633–645.

Bockmayr, A and Kasper, T, 1998. "A unifying framework for integer and finite domain constraint programming" *INFORMS J on Computing* **10**(3) 287–300.

Bockmayr, A and Dimopoulos, Y, 1998. "Mixed Integer Programming Models for Planning Problems" *CP'98 Workshop on Constraint Problem Reformulation*.

Bylander, T, 1997. "A Linear Programming Heuristic for Optimal Planning" *Proceedings of the 14th National Conference of the American Association for Artificial Intelligence* 694–699.

Blum, AL and Furst, ML, 1997. Fast planning through planning graph analysis" *Artificial Intelligence* **90**(1–2) 281–300.

Davis, M, Logemann, G and Loveland, D, 1962. "A machine program for theorem proving" *Comm. ACM* **5** 394–397.

Ernst, M, Millstein, T and Weld, D, 1997. "Automatic satcompilation of planning problems" *Proceedings of the 15th International Joint Conference on Artificial Intelligence* 1169–1176.

Fikes, R and Nilsson, NJ, 1971. "STRIPS: A new approach to the application of theorem proving to problem solving" *Artificial Intelligence* **2** 189–208.

Gomes, CP, Selman, B and Kautz, H, 1999, "Boosting Combinatorial Search Through Randomization" *Proceedings of the 15th National Conference of the American Association for Artificial Intelligence* 431–437.

Haas, A, 1987. "The case for domain-specific frame axioms" *The Frame Problem in Artificial Intelligence, Proceedings of the 1987 Workshop* Morgan Kaufman.

Hooker, JN, 1988. "A quantitative approach to logical inference" *Decision Support Systems* **4** 45–69.

Hooker, JN, 1994. "Logic-based methods for optimization" in A Borning (ed), *Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* **874** 336–349.

Hooker, JN and Osorio, M, 1997. "Mixed logical/linear programming" *Discrete Applied Mathematics* to appear.

Huang, T-C, Selman, B and Kautz, H, 1999. *Proceedings of the 17th National Conference of the American Association for Artificial Intelligence* 511–517.

Kautz, H, McAllester, D and Selman, B, 1996. "Encoding plans in propositional logic" *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning* 374–384.

Kautz, H and Selman, B, 1996. "Pushing the envelope: Planning, propositional logic, and stochastic search" *Proceedings of the 13th National Conference of the American Association for Artificial Intelligence* 1194–1201.

Kautz, H and Selman, B, 1998. "BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving" *Working notes of the Workshop on Planning as Combinatorial Search*, held in conjunction with *AIPS-98*, Pittsburgh, PA.

Kautz, H and Selman, B, 1999. "Unifying SAT-based and graph-based planning" *Proceedings of the 17th National Conference of the American Association for Artificial Intelligence* 318–325.

Kautz, H and Walser, JP, 1999. "State-space Planning by Integer Optimization" *Proceedings of the 17th National Conference of the American Association for Artificial Intelligence* 526–533.

Koehler, J, 1998. "Planning under Resource Constraints" *Proceedings of the 13th European Conference on Artificial Intelligence* 489–493.

Li, C and Anbulagan, 1997. Heuristics based on unit propagation for satisfiability problems" *Proceedings of the 15th International Conference on Artificial Intelligence* 366–371.

McCarthy, J and Hayes, P, 1969. "Some philosophical problems from the standpoint of artificial intelligence" *Machine Intelligence 4* 463–502. Edinburgh University Press.

Nau, DS, Smith, SJ and Kutluhan Erol, ??, 1998. "Control strategies in HTN planning: theory versus practice" *Proceedings of the 16th National Conference of the American Association for Artificial Intelligence* 1127–1133.

Nemhauser, GL and Wolsey, LA, 1988. *Integer and Combinatorial Optimization*. John Wiley, New York.

Nilsson, N, 1980. *Principles of Artificial Intelligence*. Morgan Kaufmann.

Russel, S and Norvig, P, 1995. *Artificial Intelligence, A Modern Approach*. Prentice-Hall.

Savelsbergh, MWP, 1994. "Preprocessing and probing for mixed integer programming problems" *ORSA J Computing* **6** 445–454.

Selman, B, Kautz, H and Cohen, B. 1996. "Local search strategies for satisfiability testing" *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **26** 531–532.

Selman, B, Kautz, H and McAllester, D, 1997. "Ten challenges in propositional reasoning and search" *Proceedings of the 15th International Joint Conference on Artificial Intelligence* 50–54.

Shoenfield, J, 1967. *Mathematical Logic*. Academic Press.

Walser, J, 1998. "Domain-independent Local Search for Linear Integer Optimization" *PhD Dissertation*, Universitat des Saarlandes.

Weld, D, 1999. "Recent advances in AI planning" *AI Magazine* to appear.

Wolfman, SA and Weld, DS, 199. "The LPSAT Engine and its Application to Resource Planning" *Proceedings of the 15th International Joint Conference on Artificial Intelligence* 310–317.

Wolsey, L, 1998. *Integer Programming*. John Wiley, New York.