Integer Optimization Models of AI Planning Problems

Henry Kautz
AT&T Shannon Labs
180 Park Avenue
Florham Park, NJ 07932, USA
kautz@research.att.com

Joachim P. Walser
i2 Technologies
11701 Luna Road
Dallas, TX 75234, USA
walser@i2.com

December 7, 1999

Abstract

This paper describes ILP-PLAN, a framework for solving AI planning problems represented as integer linear programs. ILP-PLAN extends the planning as satisfiability framework to handle plans with resources, action costs, and complex objective functions. We show that challenging planning problems can be effectively solved using both traditional branch-and-bound integer programming solvers and efficient new integer local search algorithms. ILP-PLAN can find better quality solutions for a set of hard benchmark logistics planning problems than had been found by any earlier system.

1 Introduction

In recent years there has been a growing awareness in the artificial intelligence (AI) and operations research research (OR) communities that the fields deal with similar kinds of combinatorial problems. The similarities are often hidden by the differences in problem representations that each field has traditionally used.

Work in planning in AI has its roots in symbolic logic, starting with the work by Green (Green 1969) on planning as theorem-proving in first-order logic. The basic statement of an AI planning problem is to find a sequence of actions that transform a given initial state into a desired goal state. States are described by lists of facts that hold true in the state. Each action is described a list of preconditions that must hold when it is applied, and a list of its effects in terms of making some facts true and others false. STRIPS notation (Fikes & Nilsson 1971) was introduced as a way to improve the efficiency of theorem-proving, and is still the most popular notation for representing planning problems in AI. For example, one of the best current planning systems, Graphplan (Blum & Furst 1995), takes STRIPS-style input, and several authors (Kautz & Selman 1996; Kambhampati, Lambrecht, & Parker 1997) have shown how the Graphplan algorithm can be viewed as a form of specialized theorem-proving.

The SATPLAN system (Kautz & Selman 1992; 1996; Weld 1999) showed that an effective strategy for solving AI planning problems is to directly represent problems in propositional logic and use satisfiability checking algorithms. By contrast, the roots of OR are in mathematical programming. The notion of planning in OR is more diverse and includes a variety of problems from production planning, facility location, or inventory management. Planning problems in OR are usually represented as either linear programs or integer linear programs.

At a high level the distinction between logical and mathematical representations corresponds to a distinction in the kinds of planning problems and solution techniques that each field studied. AI has concentrated on discrete feasibility problems, and developed algorithms based on backtracking and local search. OR has concentrated on optimization problems and has developed sophisticated mathematical techniques, many of which build on linear programming. From a more detailed view, however, there is a great deal of overlap in the kinds of problems and solution approaches that the two fields study: For once, there is no clear line between feasibility and optimization, and there is a growing interest in AI in real-world optimization problems. Historically, there has been a significant overlap in the area of local search such as simulated annealing and genetic algorithms. Branch-and-bound and backtracking algorithms are a central topic of integer programming in OR, and appear in the guise of constraint programming, a discipline that draws practitioners in equal number from both fields.

One of the main reasons for the increased interest in optimization problems in AI planning is that researchers are tackling more realistic problem domains. In almost any real domain different actions incur different costs, and low-cost solutions are most desirable. Furthermore, many domains involve actions that consume or produce *resources* that are most naturally modeled by numeric variables. Because both STRIPS and SATPLAN only include Boolean variables, modeling resources that can take on a wider range of values is problematic as it requires to encode each possible value using a distinct variable, or to make use of encoding schemes (*e.g.*, binary or Grey codes) that are computationally less favorable. These facts have led a number of researchers to extend the STRIPS representation with resources and action costs, and develop various specialized planning algorithms that take these factors into account, usually in a purely heuristic manner (see Section 4).

OR, of course, has long studied *general* optimization algorithms, particularly for linear and integer linear programs. However, it has not been obvious how to represent the kind of planning problems studied in AI in these formalisms. One goal of this paper, therefore, is to show how AI planning problems extended with costs and optimization criteria can be encoded as integer optimization problems. We will discuss extensions of the SATPLAN framework, where problems are specified using axioms in propositional logic, as well as compilation issues for the translation from an extended STRIPS representation, where problems are specified using operator schemas. We use the name ILP-PLAN to refer to this general approach to representing and solving AI planning problems as integer linear programs.

In summary, the ILP-PLAN framework provides a foundation for representing AI planning problems with resources, action costs, and complex objective functions as integer linear programming (ILP) models. ILP-PLAN does not commit to a specific approach for solving the planning models, and we will present empirical results on solving ILP problem encodings using both traditional branch-and-bound solvers and new efficient integer local search algorithms.

The first part of this paper will demonstrate how STRIPS-style planning problems extended with costs, resources, and optimality conditions can be represented as an ILP and solved using a LP-based branch-and-bound. For the experiments, use a commercial modeling language (AMPL) (Fourer, Gay, & Kernighan 1993) and a mixed integer programming package (ILOG CPLEX). We will argue that specifying a planning problem by a *combination* of STRIPS operators (to represent logical constraints between actions and their preconditions and effects) and linear inequalities (to represent resource usage and objective functions) is more elegant and natural than using *only* extend STRIPS operators or *only* linear inequalities.

One reason that the ILP approach has been rarely investigated in AI is that branch-and-bound is a relatively inefficient algorithm for solving problems that are mainly *logical* in nature. However, recently a new approach to integer programming has been developed by Walser (1997, 1998) based on randomized local search. This algorithm, WSAT(OIP), generalizes the Walksat algorithm for satisfiability (Selman, Kautz, & Cohen 1994) to a form of ILP's called "over-constrained integer programs (OIP)". In an OIP the objective function is represented by a set of "soft constraints" – that is, linear inequalities that may or may not hold for a feasible solution. Walser demonstrated that WSAT(OIP) is can outperform ILP branch-and-bound in a number of challenging domains, including sports scheduling and capacitated production planning (CLSP).[1]

The second part of this paper presents a case study of using WSAT(OIP) to find *better quality solutions* to a set of difficult logistics planning benchmark problems that have been frequently cited in recent literature on planning systems, including SATPLAN, Graphplan, and algorithms such as ASP and LRTA* (Bonet, Loerincs, & Geffner 1997). We will demonstrate that the known computational advantages of using local search on SATPLAN's best propositional encoding (Kautz & Selman 1996) of this domain can be retained, while the integer local search framework allows us to find solutions of lower cost. (The encoding used is called "state-based", and we will briefly describe how it differs from the more straightforward encoding used in the first part of the paper.) The ILP-PLAN approach improves on the best published solutions for this domain.

ILP-PLAN thus brings together work on generalizing the *expressive* power of SATPLAN with work on generalizing the Walksat *inference engine* originally employed by that system. ILP-PLAN also builds a bridge between AI or OR technology. It shows how powerful and sophisticated OR solvers can be applied to AI planning. As a contribution to OR, ILP-PLAN shows how STRIPS operators can be used to make the specification of complex optimization problems that involve action selection concise and easy to express.

[1]CLSP is unlike the kind of AI planning discussed in this paper in that it does not involve action selection, but rather optimizing the quantities of materials produced at different points in a production process.

2 ILP Translations of Extended STRIPS Problems

All of the “classical” AI planning systems mentioned above (the original STRIPS, Graphplan, and SATPLAN) solve what are called deterministic state-space planning problems. In this model *states* assign truth values to time-varying propositions called *fluents*. A state can thus be identified with the set of fluent that it assigns true. An *action* is a partial function over states. Actions are specified by a precondition, add list, and delete list of fluents. An action is only defined on states in which the precondition holds, and yields the state that is identical except that the fluents in the add list are assigned true and those in the delete list are assigned false. A planning problem consists of an initial state a set of actions, and a set of goals (also represented by a set of fluents). A solution to a planning problem is a sequence of actions whose application to the initial state yields a state that assigns all the goals to true. A *bounded* planning problem is a planning problem which also specifies a maximum number of actions that may appear in a solution.

These semantics can be refined to allow parallel actions as is done in Graphplan and SATPLAN. The set of states is assumed to form a linear sequence (corresponding to the natural numbers). The parallel composition of a set of actions is defined for a state if all possible sequential compositions of the actions are well-defined and equivalent. A necessary and sufficient condition for parallel compositionality is that no action deletes a precondition or effect of another. The *parallel length* of a solution is then the number of actions where such composite actions are counted as a single step, and the *sequential length* is the number of atomic actions. Koehler *et al.* (1997) and Anderson, Smith, and Weld (1998) have further extended STRIPS and the state-space model to handle actions with quantified conditional effects; in Section 2.4 below, we show that such actions can also be represented in ILP-PLAN in a straightforward manner.

The first major extension we make to classical AI planning is the introduction of quantitative *resources*. *Resource variables* are assigned a numeric value (integer or real-valued) by each state. Every resource variable has a global minimum and maximum value. Actions are extended with *resource preconditions* and *resource effects*. A resource precondition is a simple linear inequality that must hold in any states in which the action is applicable. We distinguish actions whose effects are to *consume* (decrease), *produce* (increase), or *provide* (set to an absolute value, regardless of it’s previous value) a resource.[2] As before, the parallel composition of a set of actions is defined for a state only if all possible sequential executions of the actions are well-defined and equivalent. In Section 2.2 below we will describe a set of necessary and sufficient conditions for a set of actions with resources to be parallel composable. One contribution of this work is demonstrating that these constraints can be captured using only linear inequalities, even though the most straightforward way of representing them uses *quadratic* equations.

The second extension to STRIPS is the addition of *optimization criteria* to the planning problems. We will want to find solutions that minimize one or more linear functions of resources, actions, and facts. Although resource consumption is a most

[2]In this paper we only consider resource effects involving a single variable, *e.g.*, $r = r + 3$, not general linear equations.

common objective function, note that we also allow such functions as the number of actions that occur, or the number of objects for which a predicate holds.

The ILP-PLAN encoding consists of three parts: constraints for the logical properties of actions; for resource usage; and for optimization objectives. After discussing each component and an implemented example, we will consider in more detail the expressivity of the framework in relation to extensions to STRIPS to handle conditional effects.

2.1 The Logical Component

Previous papers (Kautz, McAllester, & Selman 1996; Ernst, Millstein, & Weld 1997) have described how the logical properties of bounded-time STRIPS planning problems can be represented in clausal form. The following kinds of axioms are used:

1. Actions imply their preconditions and effects (meaning that for an action occurrence at time t, its preconditions must hold at t and its effects hold at $t+1$);

2. Non-parallel-composable actions may not occur at the same time step;

3. Explanatory frame axioms (Haas 1987; Schubert 1989) which state that if a fluent changes its truth value between states, then one of the actions that changes it must have occurred;

4. The initial state holds at time 0 and the goals hold at a given final time point n.

Following are schemas for axioms of types (1)–(3), where the Boolean variable a_i means ground action a occurs at time i; p_i means fluent p holds at time i; and pre(a), add(a), and del(a) are the sets of fluents in the precondition, add, and delete lists of ground action a respectively.

$$
\begin{aligned}
a_i &\rightarrow p_i \quad \forall p \in \text{pre}(a) && (1)\\
a_i &\rightarrow p_{i+1} \quad \forall p \in \text{add}(a) && (2)\\
a_i &\rightarrow \neg p_{i+1} \quad \forall p \in \text{del}(a) && (3)\\
\neg a_i &\vee \neg b_i \quad \forall a, b \ . \ \text{del}(a) \cap (\text{pre}(b) \cup \text{add}(b)) \neq \emptyset && (4)\\
(\neg p_i \wedge p_{i+1}) &\rightarrow \bigvee_{p \in \text{add(a)}} a_i &&\\
(p_i \wedge \neg p_{i+1}) &\rightarrow \bigvee_{p \in \text{del(a)}} a_i && (5)
\end{aligned}
$$

The most straightforward way to capture these constraints in ILP is to simply individually convert each clause into an inequality over 0/1 variables (Hooker 1988). For example, $(p \vee \neg q)$ becomes $p + (1 - q) \geq 1$. We used this translation in the experiments reported in this paper. Recent work by Vossen *et al.* (1999) describes a more complex ILP translation of these logical constraints that usually results in sets of equations with stronger linear relaxations. The technique improves the performance of branch-and-bound solvers, and could be combined with the conventions for encodings resources and optimality conditions described below.

2.2 The Resource Usage and Optimization Components

The second set of constraints maintains the value of each resource at each point in time, and makes sure that parallel actions are free of resource conflicts. The following are a set of necessary and sufficient conditions for a set $\mathcal{A}$ of actions to be parallel composable (Koehler 1998).[3]

- The set $\mathcal{A}$ is logically conflict-free, as defined above.
- If $\mathcal{A}$ contains a provider for r, it contains no other effect for r.
- The value of r at i minus the sum of the consumers of r in $\mathcal{A}$ satisfies the global lower bound for r.
- The value of r at i plus the sum of the producers of r in $\mathcal{A}$ satisfies the global upper bound for r.
- For any action a in $\mathcal{A}$ with a resource precondition that puts a lower bound on r, the value of r at i minus the sum of the consumers of r in $\mathcal{A}$ *other than* a satisfies that precondition. Likewise for an a in $\mathcal{A}$ with a resource precondition that puts an upper bound on r, the value of r at i plus the sum of the producers of r *other than* a in $\mathcal{A}$ satisfies the precondition.

These conditions ensure that every way of sequencing a set of parallel actions is well-defined and equivalent. Note too that explicit resource preconditions on actions are redundant if they are the same as the global bounds on the resource.

Now we consider the ILP encoding of these constraints. For each resource r we introduce a set of numeric variables r_i that stand for the quantity of r at the start of step i. Let M and N be minimum and maximum bounds on r. For each ground operator a we use the variable a_i to represent the *action* of that ground operator occurring at time i. For each consumer, producer, or provider ground operator a let c_a be the amount by which the operator decreases, increases, or sets r. Let *Prod* and *Con* be the sets of producing and consuming ground operators for r respectively. For simplicity we will say there is exactly one providing ground operator, s, which resets r to its maximum N. Finally, let *Prec* be the set of ground operators b that have a resource precondition, *e.g.* a lower bound p_b.

The main technical challenge is to find a *linear* encoding of domains with providing actions. The most straightforward equation for updating the value of a resource is *quadratic*:

$$r_{i+1} = s_i N + (1 - s_i)\left(r_i - \sum_{a \in Con} c_a a_i + \sum_{a \in Prod} c_a a_i\right) \tag{6}$$

That is, if the provider occurs, the resource is set to N, otherwise it is linearly updated by the consumers and producers. To overcome this problem we introduce a new set of

[3]Note that composability is now relative to a particular state; *e.g.*, two actions may be parallel composable from states where the value of a resource is sufficiently large.

```
**resource** $0 \leq \textit{fuel} \leq C$
**action** fly($a, b$: airport):
    **precondition**: $\textit{at-plane}_a$
    **effects**: $\textit{at-plane}_b, \neg\textit{at-plane}_a$
        $\textit{fuel} \mathrel{-}= D_{ab} \cdot F$
        $\forall$passenger $p$: $\textit{boarded}_p$
                    **effects**: $\textit{at}_{pb}, \neg\textit{at}_{pa}$
**action** refuel:
    **effects**: $\textit{fuel} = C$
```

Figure 1: Airplane example with conditional effects. Notice that refueling fills the tank to capacity.

variables k_i that stand for amount of resource created by provider s_i if it occurs. We call the k_i *provider reset* variables. Then the resource conflict constraints ($\forall i$) are:

$$\begin{aligned}
r_{i+1} &= r_i + k_i - \sum_{a \in \textit{Con}} c_a a_i + \sum_{a \in \textit{Prod}} c_a a_i && (7)\\
s_i &\rightarrow \neg a_i \quad \forall a \in \textit{Prod} \cup \textit{Con} && (8)\\
s_i &\rightarrow (r_{i+1} \geq N) && (9)\\
\neg s_i &\rightarrow (k_i = 0) && (10)\\
M &\leq r_i \leq N && (11)\\
b_i p_b &\leq r_i - \sum_{a \in \textit{Con} \setminus \{b\}} c_a a_i \quad \forall b \in \textit{Prec} && (12)
\end{aligned}$$

(7) propagates the value of the resource from one time step to the next. (8) makes providers exclusive of other actions that affect the resource, while (9) and (10) establish the amount of resource created by a providing action (if any). (11) enforce the global bounds on the resource. (12) enforces resource preconditions for the (most common) case where the precondition places a lower bound on the resource. k_i is not further constrained and gets set by propagating the resource balance (7).

Equations (8)–(10) mix logical and mathematical connectives. Again some care must be taken to expand these constraints in linear form. They can be rewritten as follows:

$$\begin{aligned}
a_i + s_i &\leq 1 && (13)\\
0 &\leq r_{i+1} - N s_i && (14)\\
0 &\leq N s_i - k_i && (15)
\end{aligned}$$

Finally, resource optimization constraints are simply arbitrary linear inequalities over all the variables described above (not being limited to resource variables).

Indices	Definition
i	action step (L is the last step)
a, b	airports
p	passenger
Constants	**Definition**
C, F	tank capacity, fuel use per dist. unit
D_{ab}	Distance from a to b
Variables	**Definition**
fly_{abi}	flight from a to b occurs in step i
refuel_i	refuel occurs at step i
$\mathit{refuel\_amount}_i$	provider reset variable $\in [0, C]$
fuel_i	plane's fuel level $\in [0, C]$
board_{ai}	boarding of all checked passengers at airport a occurs at step i
$\mathit{deplane}_{pi}$	deplaning of p in step i
at_{pai}	person p is at airport a at step i
$\mathit{at\text{-}plane}_{ai}$	plane is at airport a at step i
$\mathit{checked\text{-}in}_{pi}$	person p has checked-in status at i
$\mathit{boarded}_{pi}$	person p has boarded status at i

Table 1: Parameters in the ILP translation of the airplane example. All variables binary unless declared otherwise.

2.3 Example: Resource Optimization Planning.

We illustrate this translation with a a modified version of the airplane example from (Penberthy & Weld 1992) and (Koehler 1998). It simplifies the original example by ignoring timing aspects but extends it for optimization of passenger routings. The scenario is a plane that can fly between a number of different airports and consumes fuel. Passengers with checked-in status at the location of the plane can be boarded. Boarded passengers move with the plane until they are deplaned, which can occur individually in our variation. The ILP-PLAN version of the example extends the task from a decision problem (with resources) to resource optimization: An explicit optimization objective is included to minimize resource usage, in this case "fuel". Figure 1 and Table 1 describe some of the operators and variables used.

The following inequalities state the resource aspect of the model ($1 \leq i < L$).

$$\mathit{refuel}_i \rightarrow (\mathit{fuel}_{i+1} \geq C) \tag{16}$$

$$\neg\mathit{refuel}_i \rightarrow (\mathit{refuel\_amount}_i = 0) \tag{17}$$

$$\mathit{fuel}_{i+1} = \mathit{fuel}_i + \mathit{refuel\_amount}_i - \sum_{a,b:a\neq b} \mathit{fly}_{abi} D_{ab} F \tag{18}$$

$$\mathit{refuel}_i + \sum_{a,b:a\neq b} \mathit{fly}_{abi} \leq 1 \tag{19}$$

(16) and (17) link the decision variables for refueling with the fuel and refuel amounts

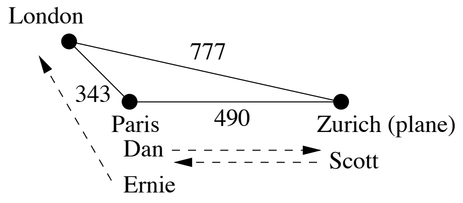

Figure 2: Scenario of initial state and travel destinations (arrows) of airplane-a. The resource-optimal plan found by CPLEX given the state-based ILP encoding has 8 steps: board Scott, fly to Paris, deplane Scott, refuel || board Dan and Ernie, fly to London, deplane Ernie, fly to Zürich, deplane Dan.

and (18) states the fuel balance. As described above, (16) and (17) can be directly translated to linear inequalities. (19) is a compact way of making refuel (a provider) and flying (a consumer) mutually exclusive. (Using a single equation rather than a set of pairwise mutual exclusion inequalities leads to a stronger linear relaxation.)

The optimization objective is stated as

$$\textbf{minimize}\ \ \mathit{fuel}_1 - \mathit{fuel}_L + \sum_i \mathit{refuel\_amount}_i,$$

where $\mathit{refuel\_amount}_i$ is the provider reset variable corresponding to the *refuel* action at time i.

Figure 2 shows an example problem and Tables 2 and 3 report experimental results on several instances using CPLEX 6.5. CPLEX standard automatic parameter settings were used except for experiments with CPLEX's "probing" heuristic. (Probing is a proprietary variable selection heuristic.) Additionally, we attempted different types of automatic cutting plane generation implemented in Cplex, *i.e.*, cover cuts, flow cover cuts, clique cuts, GUB cover cuts, and implied bound cuts, but none of these improved performance.

Finding optimal solutions to the Airplane domain turns out to be computationally challenging. Therefore, to reduce the computational effort, we first reduced the number of necessary time steps by relaxing the conditions for parallel actions given in section 2: In our ILP model, instead of requiring that all possible sequentializations be equivalent, we allow for parallel actions whenever there exists *one* valid sequentialization that achieves the correct effects. In the following experiments, we thus allow for parallelizing *e.g.* a 'fly' and a 'deplane' action because we can always construct a feasible plan from the parallel solution.

Each instance in the test set includes one more city than the previous, and we see from Table 2 that the solution time increases in each case by about an order of magnitude. It has been suggested in the literature (Giunchiglia, Massarotto, & Sebastiani 1998) that performance of backtracking solvers on AI planning problems can be improved by restricting branching to variables representing actions. We tried this in the column labeled "fv-relax" by relaxing the integrality of the fluent variables. (A branch and bound solver like CPLEX only branches on integer variables.) Surprisingly, this degraded performance, both in terms of the number of nodes expanded and the over-

	[std]	[fv-relax]	[av-relax]	[av-rlx+prob]
airplane-a	1s: 8	2s: 38	2s: 15	2s: 4
airplane-b	11s: 364	19s: 726	10s: 336	12s: 238
airplane-c	206s: 1331	296s: 1730	181s: 1205	130s: 667

Table 2: Experimental results for the airplane example, basic encoding and two encoding variants. Results in seconds (:) number of explored MIP nodes. All experiments were run on an HP-8000 using CPLEX 6.5. The columns report results for all fluent and action variables integer (except resource variables [std]), relaxed integrality of fluent variables [fv-relax], relaxed integrality of action variables [av-relax], and the use of a probing heuristic [av-rlx+prob].

all running time. However, as shown in the "av-relax" column, relaxing integrality of the action variables, and thus branching only on *fluent* variables, *does* lead to a significant decrease in the number of nodes expanded. Finally, combining relaxing action variables with use of CPLEX's probing heuristic (column "av-rlx+probe") decreases the number of nodes expanded by about 50%, and also reduces the total solution time by about 30% on the largest instance. (Improvement using probing without relaxing action variables was marginal.) Although a full examination of this phenomena awaits future research, it is interesting to note that SATPLAN encodings considered in Section 3 below also boost performance by emphasizing the role of the fluent variables.[4]

problem / steps	acts	p/a	vars	cnstrs	min.fuel	LP-lb
airplane-a / 5	10	3	134	551	805	387.91
airplane-b / 7	15	4	304	1774	897	109.21
airplane-c / 9	21	5	576	4405	2096.5	33.24

Table 3: Problem specification of the airplane example. Columns are problem name and minimum number of times steps in a parallel plan, the number of actions in such a minimal solution, the number of passengers and airports (p/a), problem size in number of variables and constraints, the minimal fuel consumption in an optimal IP solution, and the LP lower bound.

Table 3 provides an explanation for the rapid increase in the CPLEX solution time as the problem size grows. The last two columns compare the fuel usage in an optimal solution and the minimum fuel usage in the linear relaxation of the problem instance. We see that the linear relaxation provides a very weak lower bound on the optimal value, and actually decreases as the problem size increases. Because of this weak bound, the linear relaxation cannot be effectively used to prune the branch and bound search tree. Thus, while the encodings provided in this section provide a general basis for

[4]A particular issue that requires investigation is the conditions under which the relaxed problem yields an integer solution. It is clear from the form of the encoding that fluent variables can be safely relaxed. For the airplane example integer solutions were also found when the action variables were relaxed, but this may not occur in general.

transforming extended STRIPS into ILP, it is clear that there is room for improving the encodings to enhance the performance of branch and bound solvers on large instances. As noted above, a promising direction would be to combine our work on resources with Vossen *et al.*'s translation of the logical (non-resource) component.

2.4 Quantified Conditional Effects

An important aspect of a modeling language for complex planning domains is its expressivity. In particular, quantification and conditional effects in the style of ADL (Pednault 1989) are interesting expressive features, and several authors have described ways to incorporate them into Graphplan (Anderson, Smith, & Weld 1998; Koehler *et al.* 1997; Gazen & Knoblock 1994). Here, some aspects of a translation of conditional effects in ILP-PLAN will be sketched, in particular *quantified* conditional effects, to be able to model the subsequent examples. Fortunately, some of the complexities that arise with Graphplan are avoided in ILP-PLAN, since they relate to the handling of mutex conditions which Graphplan uses.

To motivate the problem of quantification and describe its solution for ILP-PLAN, we continue with our discussion of the ILP-PLAN representation of the airplane problem (Koehler 1998). In this model, a "boarding" action at an airport causes all passengers at the airport to obtain *boarded* status and lose their status as *checked-in* passenger (note that in the example, there is only one plane). The complete rule for boarding is:

action: *board*(a: airport):
  precondition: $\textit{at-plane}_a$
  effect: $\forall \textit{passenger}\ p$: $\textit{checked-in}_p \wedge \textit{at}_{p,a}$
    effect: $\textit{boarded}_p, \neg\textit{checked-in}_p$.

Aside from the usual translation of the implied preconditions "$\textit{board}_{a,i} \rightarrow \textit{at-plane}_{a,i}$", the ILP translation needs to assert the implied *conditional* effects:

$$\begin{aligned}\forall i, p, a: \quad & (\textit{board}_{a,i} \wedge \textit{checked-in}_{p,i} \wedge \textit{at}_{p,a,i}) \rightarrow \\ & (\textit{boarded}_{p,i+1} \wedge \neg\textit{checked-in}_{p,i+1})\end{aligned} \tag{20}$$

Further, explanatory frame axioms need to be generated as ILP constraints. For instance, the frame axiom for *boarded* requires the introduction of a disjunction over all airports to express that at *some* airport, boarding has occurred, and the *conditional* preconditions hold:

$$\begin{aligned}\forall i, p: \ & (\neg\textit{boarded}_{p,i} \wedge \textit{boarded}_{p,i+1}) \rightarrow \\ & \bigvee_{\textit{airport}\, a} (\textit{board}_{a,i} \wedge \textit{checked-in}_{p,i} \wedge \textit{at}_{p,a,i}).\end{aligned} \tag{21}$$

Note that here the disjunction binds a, which is otherwise unbound in the *boarded* predicate. In general, to convert (21) to a set of linear inequalities, we can introduce auxiliary variables $\textit{aux}_{p,a,i}$ (meaning "p is boarded at a") and assert that one of those needs to be true for every p that gets boarded. This yields the following clauses which are equivalent to (21):

$$\forall i,p:\ (\neg \mathit{boarded}_{p,i} \land \mathit{boarded}_{p,i+1}) \quad \rightarrow \quad \bigvee_{\text{airport } a} \mathit{aux}_{p,a,i} \tag{22}$$

$$\forall i,p,a:\ \mathit{aux}_{p,a,i} \quad \rightarrow \quad (\mathit{board}_{a,i} \land \mathit{checked\text{-}in}_{p,i} \land \mathit{at}_{p,a,i}) \tag{23}$$

Intuitively, this means that for every $\mathit{aux}_{p,a,i}$ equal to true, all conditional preconditions need to be true. All clauses are now equivalent to linear inequalities as can be seen by considering the equivalence of (23) to:

$$\forall i,p,a:\ \mathit{board}_{a,i} + \mathit{checked\text{-}in}_{p,i} + \mathit{at}_{p,a,i} - 3\mathit{aux}_{p,a,i} \geq 0 \tag{24}$$

This technique of introducing auxiliary variables to properly handle variable binding provides a general solution for encoding conditional effects in ILP-PLAN.

3 ILP Translations of SATPLAN Problems

In the Section 2.1 we described how STRIPS problems can be translated to propositional logic. An alternative approach is to directly model planning domains in propositional logic — as was done in the original work on the SATPLAN system (Kautz & Selman 1992) — instead of starting with a STRIPS formulation. One advantage of the direct approach is that it makes it easy to express complex intermediate goals and domain-specific control knowledge in axiomatic form (Kautz & Selman 1998; Mali & Kambhampati 1998). Another advantage is in experimenting with different encoding schemata, including action-based encodings (similar to the STRIPS conventions treated above), causal encodings (Kautz, McAllester, & Selman 1996), and in particular, state-based encodings (Kautz & Selman 1996). We call an encoding *state-based* if it only uses variables to represent fluents, not actions. The axioms in a state-based encoding directly relate changes in fluents between adjacent states or place constraints on the relationships between fluents within a state. For certain challenging planning domains the solution times for direct state-based SAT encodings are better by an order of magnitude or more than for either STRIPS translations or non-SAT approaches.

Despite the power and generality of the SATPLAN framework, it lacks the ability to express the kind of resource constraints and optimality conditions needed for many realistic domains. In this section we will focus on strengthening SATPLAN's optimization abilities. SATPLAN allows one to minimize the *parallel length* of a solution. This notion of optimality (shared by Graphplan, as noted above) is an advance over planning frameworks that treat all feasible solutions indifferently: for many popular test domains finding a shortest solution is at least NP-hard, while finding a feasible solution can be done in linear time (Bylander 1991). However, even if all actions have the same cost, one may wish to minimize the number of *actions* in a solution, that is, the *sequential length* rather than the parallel length. The two notions of length may actually be in conflict, where the parallel-minimum solution has non-minimum sequential length.

We will investigate the issue of improving solution quality by moving from a SAT to an ILP encoding in the context of a popular benchmark, the logistics planning domain (Veloso 1992). The scenario is the transportation of a set of packages that involves flights and truck-drives between locations. There are various possible criteria to optimize in this domain: (a) The sequential length, (b) the parallel length, (c) some function of the sequential and parallel lengths, and (d) yet more realistic measures of plan quality, *e.g.* including specific action costs for the different action types (flying an airplane is typically more expensive than driving a truck or loading a packet), or finally (e) timing aspects.

Previous approaches to the logistics domain include finding parallel optimal solutions (criterion b) using SATPLAN (Kautz & Selman 1996); more recently, Bonet, Loerincs and Geffner (1997) presented a method that found better serial optimal solutions (criterion a) using LRTA*, however at high computational cost for near-optimal solutions. We will concentrate on criteria (c), where our goal is find solutions of minimal sequential length *among* all solutions of minimal parallel length. (However the encoding we use also makes it easy to assign different costs to different types of actions if desired.)

To model the problem, we will use an ILP variant of the state-based encodings presented in (Kautz & Selman 1996) that is extended to encode a notion of plan optimality.

3.1 Local Search Algorithms

The best known algorithm for solving state-based SATPLAN encodings of the logistic domain is Walksat (Selman, Kautz, & Cohen 1994). Walksat performs local search over the space of (complete) variable truth assignments. At each step it "flips" (reverses) the truth-value assigned to some variable which appears in an unsatisfied clause, with preference given to choices that would increase the number of satisfied clauses. On some domains Walksat greatly outperforms traditional backtrack-style satisfiability algorithms (Selman & Kautz 1993).

We employ a similar strategy for optimization encodings in ILP-PLAN. The ILP-PLAN approach to the domain casts the problem in integer constraints and solves it using integer local search, WSAT(OIP). The experimental results reported in the following demonstrate that ILP-PLAN can find plans with fewer actions than SATPLAN. In comparison with LRTA* it finds plans with the same number of actions or fewer at reduced computational cost. In contrast to all previous approaches to this domain, it allows for stating planning objectives explicitly and opens up the way for even more practical criteria of plan-optimality.

To include optimization objectives into local search, the integer local search framework uses a representation introduced as *over-constrained integer programs (OIPs)* (Walser 1999). OIP formulates optimization criteria by means of soft inequality constraints over bounded integer variables and can be reduced to ILP. An OIP consists of hard and soft inequality constraints, wherein the optimization objectives are represented by the soft constraints. If all inequalities are linear, the OIP problem can be formulated in matrix notation as

$$\begin{aligned} Ax &\geq b \\ Cx &\leq d \qquad (\textit{soft}) \\ x &\in D \end{aligned}$$

where A and C are $m \times n$-matrices, b, d are m-vectors, and $x = (x_1, \ldots, x_n)$ is the variable vector, ranging over positive finite domains $x_i \in D_i$. A variable assignment that satisfies all hard constraints is called a *feasible solution*. Given a tuple (A, b, C, d, D), the OIP minimization problem is

$$\min \; \{ \, ||Cx - d|| \; : \; Ax \geq b, \; x \in D \}$$

where the objective is to find a feasible solution with minimal soft constraint violation, $||v|| := \sum_i \max(0, v_i)$. The contribution of each violated soft constraint to the overall objective is thus its degree of violation. The WSAT(OIP) algorithm then performs local search over the space of variable assignments, where in each step it increments or decrements the value of some variable that appears in a violated constraint.

3.2 Augmented State-Based Encodings

The criterion of plan optimality that we will consider in this section is to minimize sequential plan length over plans of bounded (minimum) parallel length. To formulate this, the scheme is augmented by action variables, and optimization (soft) constraints are used to formulate the objective function. However, instead of adding the full descriptive set of action variables and requiring state/action consistency, a much smaller *reduced* set of action variables is used. We will refer to this encoding scheme as an "*augmented state-based encoding*".[5] It is interesting to note that the obvious alternative of using an encoding that uses of the type described in Sec. 2 yields encodings that are much harder to solve by local search. In fact, we were surprised to discover that the conjunction of an encoding using explicit action variables with a state-based encoding also is problematic for local search. An open question we are currently investigating is *why* the inclusion of a full (unreduced) set of action variables and corresponding axioms in this domain slows down the search; an understanding of this issue may help us devise more robust heuristics for local search that are immune to the effect. We currently hypothesize that the underlying problem is that it is costly for local search to maintain consistency between the settings of an action variable and those for its preconditions and effects (a inference step that is, by contrast, trivial for *systematic* inference engines).

As before, AMPL was used as a modeling language, and we consider the logical and optimization constraints in turn.

[5]A solution to the original planning problem can be extracted from a model of the state-based encoding by a post-processing phase which determines the set of actions whose preconditions and postconditions are true in the model, which can be done in linear time. This entire set of actions can be taken to be the plan under the condition (which holds in the logistics domain) that mutually exclusive actions have mutually exclusive preconditions or effects (taking into consideration the state invariant axioms).

Indices	Definition
i	plan steps.
o	objects.
a, b	locations.
$v;\ p, t$	vehicles v: plane p, truck t.
Constants	Definition
c_l, c_f, c_d	cost of load/unload, flight, truck drive.
Variables	Definition
at_{vai}	vehicle v is at a in step i.
at_{oai}	object o is at a in step i.
in_{ovi}	object o is in vehicle v in step i.
load_{oi}	load o in step i.
unload_{oi}	unload o in step i.
$\mathit{drive\_truck}_{ti}$	drive truck t in step i.
$\mathit{fly\_plane}_{pi}$	fly plane p in step i.

Table 4: Parameters for the OIP encoding of 'logistics'.

Feasibility Constraints The logical part of the OIP encoding for the domain is the direct translation of the CNF encoding with parallel (non-conflicting) actions used in SATPLAN. Table 4 describes the indices and action variables used. First, axioms are stated that directly relate changes in fluents between adjacent states without reference to action variables, as described in (Kautz & Selman 1996). An example in the logistics domain is "objects stay in place or are loaded":

$$\mathit{at}_{oli} \rightarrow \mathit{at}_{ol,i+1} \vee \bigvee_{v} \mathit{in}_{ov,i+1} \quad \forall o, l, i. \tag{25}$$

Further, *state invariant axioms* are included that describe the relationships between fluents within a state. For example, these include "a truck is always at one location":

$$\sum_{l} \mathit{at}_{tli} = 1 \quad \forall t, i. \tag{26}$$

In order to allow us to count the number of actions in a plan, we introduce a small number of reduced action variables, as mentioned above. These variables are used to help direct the search for *optimal* solutions, but not to constrain the set of *feasible* solutions. A reduced variable stands for the occurrence of any of a set of mutually exclusive actions. For example, we introduce the variable $\mathit{drive\_truck}_{ti}$, meaning "truck t is driven (from somewhere to somewhere else) at time i", in place of the set of variables $\{\mathit{drive\_truck}_{tabi}\}$ for all locations a and b. Similarly, the variable load_{oi} stands for "package o is loaded (onto some vehicle)", in place of the set of actions $\{\mathit{load}_{ovi}\}$ for loading o onto particular vehicles, because a package can only be loaded into one vehicle at a time.

State changes are linked uni-directionally to the action variables by constraints of the type of *implied reduced actions*, *e.g.*

$$\mathit{at}_{t,a,i} \wedge \neg \mathit{at}_{t,a,i+1} \rightarrow \mathit{drive\_truck}_{t,i} \ \ \forall t, a, i. \tag{27}$$

We do not include implications in the opposite direction, that would assert that an action implies its effects and preconditions. Encoding bi-directional consistency would require full action specification and thus degrade performance for local search as mentioned above.

Optimization Constraints To optimize sequential plan length, all action variables appear in the minimization objective, weighted by cost coefficients, and formulated using soft constraints. There are many ways to write down this function; for example, one could write a *single* constraint that simply summed all the action variables. Alternatively, one could write a constraint for each time step: minimize the sum of the actions at time 1, then also at time 2, and so on. We obtained the best performance in this domain by encoding a separate soft constraint for each *object* in the domain, that is, each package, truck, or airplane. For example, for each package o there is a soft constraint that minimizes the number of times the package is loaded or unloaded:

$$(\textit{soft}) \sum_i c_l(\textit{load}_{oi} + \textit{unload}_{oi}) \leq c_l L_o \tag{28}$$

As noted in Table 4, the c_l represents a cost factor for a load or unload. L_o represents a valid lower bound on the number of load/unload actions required to transport object o. L_o could be chosen as zero, but local search performance can be improved by making such bounds as tight (large) as possible (Walser 1999). It is possible to determine such tight lower bounds by *static analysis* of the problem domain. For example, in this logistics domain at least 6 load/unload actions are required for any object whose initial and goal locations are at non-airport locations in different cities.[6] In a similar fashion one can write a separate soft constraint for each truck (minimizing driving) and each airplane (minimizing flying). We did not attempt static analysis for these constraints, and simply took the right-hand sides of the soft constraints to be 0.

In summary, the representation consists of constraints for (i) state-transition consistency, (ii) state invariant axioms, (iii) implied reduced actions, and (iv) optimization criteria.

Post-optimization. To construct the full set of actions from a consistent solution encoded by fluent variables, a post-optimization stage is applied. In an AMPL control script, after a solution has been reached, all fluent variables are fixed at their current values. Subsequently, the full (bi-directional) set of state/action consistency constraints is posted, and the system is re-optimized according to the same minimization function as before, this time using ILP branch-and-bound. This post-optimization process yields the actual plan encoded by the action variables, and is a simple yet general strategy to derive valid plans.

[6]We obtained the L_o values for the problem instances in this study by simple inspection; a formal development of the static analysis necessary to derive lower bounds is beyond the scope of this paper.

problem/steps	GRAPHPLAN		SATPLAN			ASP		LRTA*	ILP-PLAN			
			state-based encoding			functional encoding			augmented state-based OIP encoding			
	acts	time	acts	K-flips	time	acts	time	acts	acts	(f-d)	K-flips	time
log-a / 11	54	5942s	63	149	2.7s	57	34s	54	54	(6-6)	330	27s
log-a / 11									53	(5-6)	1,795	141s
log-a / 11									52	(4-6)	41,104	3,178s
log-a* / 13									51	(3-6)	4,938	401s
log-b / 13	47	2538s	68	93	0.7s	51	29s	42	42	(4-8)	3,478	340s
log-c / 13	–	–	72	161	1.4s	61	53s	52	52	(6-8)	2,466	274s
log-d / 14			86	1,425	13.3s				71	(7-15)	1,416	224s
log-d / 14									68	(7-15)	15,171	2,402s

Table 5: Performance of different planning systems. The columns are: number of sequential actions and runtime. A blank space indicates that no attempt was made at solving the problem. A dash (–) indicates that the problem could not be solved due to memory limitations. For SATPLAN, 'acts' reflects the mean number of actions found in 1,000 runs. The (f-d) column reflects the actual plan quality in number of required flights and truck drives. Note that the LRTA* and ASP algorithms are finding serial plans only. Solution times of LRTA* were not published. Results for SATPLAN and ILP-PLAN run on a 194 MHz R10000 SGI Challenge. ASP and LRTA* were reported for an IBM RS/6000 C10 with 100 MHz PowerPC 601 processor.

3.3 Experimental Results

The encodings were first simplified by AMPL's presolving algorithms (Fourer & Gay 1994) and subsequently solved by integer local search, WSAT(OIP). We were not able to solve the described encoding using integer programming branch-and-bound techniques, although such techniques are potentially applicable. To ensure that every ILP-PLAN solution meets the given plan length requirement, the value of the objective function was read off from the solutions and subsequently used as a lower bound on the objective function; the WSAT(OIP) search was then terminated upon reaching the bound. The SATPLAN numbers stem from evaluating the sequential plan length found in the solutions without encoding any planning objective.

Table 5 gives the experimental results. The parallel solution length were given as input to Graphplan, SATPLAN, and ILP-PLAN; ASP and LRTA* did not take such as parameter. The parallel lengths, except for the line labeled "log-a*", were the optimal (smallest) values. The LRTA* and ASP algorithms find serial plans, and their actual parallel length is unknown but may be much greater. Note that the solutions found by ILP-PLAN are as good or better than the solutions found by the other solvers by both the parallel and sequential length criteria. Solution times of LRTA* were not published, but it was noted that the algorithm did not converge after 500 trials (Bonet, Loerincs, & Geffner 1997). The state-based OIP encodings of ILP-PLAN were solved by WSAT(OIP) and averaged over 20 runs. Also note that SATPLAN times are for running the Walksat solver only, and do not include generating the wff (which requires approximately 1 minute on the test machines).

In addition to just sequential plan length, a more meaningful measure is given in the (f-d) column. It gives the actual number of flights and truck-drives in the solutions (all other actions are load/unload). It is clear that those numbers represent the most realistic quality measure. In general, of course one cannot infer the (f-d) value from the number of actions; however, for log-a we never observed a 54 action plan with 4 flights rather than 6. Also for log-a, we observe that it is possible to further reduce the number of actions to 51 (only 3 flights by making a circular plane trip around the airports with a single plane). This exemplifies that there is a tradeoff between short parallel and resource-optimal plans because it can be shown that a circular trip requires at least 13 steps. To compute the solution, we aided ILP-PLAN by removing one plane from the encoding labeled "log-a*/13".

The setting of action costs was $c_f = 0.5$ (flying), $c_d = 0.1$ (driving), and $c_l = 0.05$ (loading), and was tuned in pre-experiments to favor short plans. For completeness, we give the used parameter settings for WSAT(OIP), but refer the reader to (Walser 1999) for further details. Throughout the experiments, WSAT(OIP) was run with parameters $p_{hard} = 1.0$, $p_{zero} = 0.5$, $p_{noise} = 0.3$. The computational results are relatively sensitive to the particular parameter settings.

In interpreting these results it is important to note that we are comparing not just algorithms, but algorithms together with representations. Indeed, the same algorithm can yield quite different results when the same problem is encoded in different ways (Bonet, Loerincs, & Geffner 1997). In particular, only Graphplan and LRTA* took as input a "bare" STRIPS representation of the problem domain; for each of the others, the form of the input was tailored to the system and incorporated some degree of what is considered domain-specific knowledge. For example, SATPLAN included state-invariant axioms (*e.g.,* a package is only at one location), and ILP-PLAN added soft constraints as described above. The Graphplan and SATPLAN systems attempt to optimize parallel length only. ASP and LRTA* attempt to optimize sequential length only.

4 Related Work

As previously noted, the work on ILP-PLAN was partially inspired by that of Koehler (1998) on extending Graphplan to handle resource constraints. Unlike ILP-PLAN, however, that system handled resource usage strictly by annotations on STRIPS operators, and did not include objective functions. The ZENO planner (Penberthy & Weld 1994) included a rich language that could express complex resource constraints, although it too lacked explicit optimization functions. It also differed from ILP-PLAN in that the underlying planner was a least commitment, regression planner, and the architecture involved a collection of specialized routines to handle different kinds of constraints (including a linear programming subroutine), rather than a single technique (as in ILP-PLAN) like local search or branch-and-bound. Other planners that extended nonlinear planning to include metric constraints in constraint programming type frameworks include O-PLAN (Tate 1996) and parcPLAN (El-Kholy & Richards 1996). The IxTeT planner (Laborie & Ghallab 1995) is a least-commitment planner notable for using an efficient graph-based algorithm for detecting resource conflicts between parallel actions. The Remote Agent Planner (Muscettola *et al.* 1998) incorporates resources and temporal information, however it does not do any optimization.

The work cited above by Vossen *et al*. (1999) on an alternative formulation of ILP encodings for planning problems does not deal with resources or optimality conditions. Their formulation eliminates fluent variables instead of action variables. However, they do introduce auxiliary variables that strongly resemble the reduced action variables discussed above. Independent work by Bockmayr and Dimopoulos (1998) develops ILP encodings for planning where the linear relaxation gives guidance to the branch-and-bound strategy by including (i) an objective function that maximizes the number of goals achieved, and (ii) a domain-specific strengthening of the linear relaxation which they show to be effective for the blocks world domain, but not strong enough in the logistics domain.

Finally, while the ILP solvers used in our work so far (CPLEX and WSAT(OIP)) require all constraints to take the form of linear inequalities, recent work on *mixed logical/linear programming* (Hooker & Osorio 1999) may provide the underpinnings for systems that more efficiently handle ILPs that have a large logical component. In this vein (Wolfman & Weld 2000) have recently developed an IP encoding for plans with resource constraints that separates the logical (integer) constraints from the linear resource constraints. This encoding allows them to take advantage of a specialized search engine called LPSAT that combines the Davis-Putnam SAT procedure with an LP solver.

5 Conclusions

We have described ILP-PLAN, a new framework for solving AI planning problems under resource constraints and optimization objectives. By casting AI planning as integer programming, ILP-PLAN allows for constraints and objective functions over resource usage, action costs, or regular fluents. Using ILP as the base representation, it brings together threads in AI planning and integer optimization and extends previous frameworks (SATPLAN) to new practical planning *optimization* domains. The conceptual approach of ILP-PLAN integrates STRIPS style operator descriptions with linear inequality constraints. This can be seen as making ILP machinery applicable to AI planning, or conversely, as adding a new representational layer on top of linear inequalities. Like SATPLAN, ILP-PLAN is flexible with respect to inference methods and can be used in conjunction with both systematic and local search algorithms for integer optimization. We have demonstrated that two challenging planning problems can be solved effectively using a traditional ILP branch-and-bound solver and a new strategy for integer local search. For a set of hard benchmark logistics planning problems, ILP-PLAN can find better quality solutions than had been found by any earlier system.

6 Acknowledgements

We would like to thank the valuable feedback provided by the editor and anonymous reviewers of this special issue.

References

Anderson, C.; Smith, D.; and Weld, D. 1998. Conditional effects in graphplan. In *Proceedings Fourth International Conference on AI Planning Systems (AIPS-98)*. Pittsburgh, PA.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*. Montreal, Canada.

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

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proceedings Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 714–719. Providence, RI.

Bylander, T. 1991. Complexity results for planning. In *Proceedings Twelveth International Joint Conference on Artificial Intelligence (IJCAI-91)*, 274–279. Sidney, Australia.

El-Kholy, A., and Richards, B. 1996. Temporal and resource reasoning in planning: the parcPLAN approach. In *Proceedings Third International Conference on AI Planning Systems (AIPS-96)*. Edinburgh.

Ernst, M.; Millstein, T.; and Weld, D. 1997. Automatic SAT-compilation of planning problems. In *Proceedings Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*. Nagoya, Japan.

Fikes, R., and Nilsson, N. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3/4).

Fourer, R., and Gay, D. M. 1994. Large scale optimization: State of the art. In Hager, W.; Hearn, D.; and Pardalos, P., eds., *Experience with a Primal Presolve Algorithm*. Kluwer Academic Publishers. 135–154.

Fourer, R.; Gay, D. M.; and Kernighan, B. W. 1993. *AMPL, A Modeling Language for Mathematical Programming*. Boyd & Fraser publishing Company.

Gazen, B., and Knoblock, C. 1994. Combinding the expressivity of UCPOP with the efficiency of Graphplan. In Steel, S., ed., *Proceedings Fourth European Conf. on Planning*. Springer. vol. 1248 of LNAI.

Giunchiglia, E.; Massarotto, A.; and Sebastiani, R. 1998. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In *Proceedings Fifteenth National Conference on Artificial Intelligence (AAAI-98)*.

Green, C. 1969. Application of theorem proving to problem solving. In Bajcsy, R., ed., *Proceedings International Joint Conference on Artificial Intelligence (IJCAI-69)*. Washington, D.C.

Haas, A. 1987. The case for domain-specific frame axioms. In Brown, F., and Lawrence, K., eds., *The Frame Problem in Artificial Intelligence, Proceedings of the 1987 Workshop*. Morgan Kaufmann. Los Altos, CA.

Hooker, J., and Osorio, M. 1999. Mixed logical/linear programming. *Discrete Applied Mathematics* 96-97:395–442.

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

Kambhampati, S.; Lambrecht, E.; and Parker, E. 1997. Understanding and extending graphplan. In Steel, S., ed., *Proceedings Fourth European Conference on Planning*. Springer. vol. 1248 of LNAI.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proceedings Tenth European Conference on Artificial Intelligence (ECAI '92).* Vienna, Austria.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1194–1201. Portland, OR.

Kautz, H., and Selman, B. 1998. The role of domain-specific knowledge in the planning as satisfiability framework. In *Proceedings Fourth International Conference on AI Planning Systems (AIPS-98).* Pittsburgh, PA.

Kautz, H.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96).* Cambridge, MA.

Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. 1997. Extending planning graphs to an adl subset. In Steel, S., ed., *Proceedings Fourth European Conference on Planning*. Springer. vol. 1248 of LNAI.

Koehler, J. 1998. Planning under resource constraints. In *Proceedings Thirteenth European Conference on Artificial Intelligence (ECAI-98).* Brighton, England.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *Proceedings Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95).* Montreal, Canada.

Mali, A., and Kambhampati, S. 1998. Encoding htn planning in propositional logic. In *Proceedings Fourth International Conference on AI Planning Systems (AIPS-98).* Pittsburgh, PA.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence* 103(1/2).

Pednault, E. 1989. ADL: Exploring the middle ground between strips and the situation calculus. In *Proceedings First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, 324–332. Toronto, Canada.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning, (KR'92)*, 108–114. Boston, MA.

Penberthy, J., and Weld, D. 1994. Temporal planning with continous change. In *Proceedings Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1010–1015. Seattle, WA.

Schubert, L. 1989. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In Kyburg, H.; Loui, R.; and Carlson, G., eds., *Knowledge Representation and Defeasible Reasoning*. Boston: Kluwer Academic Publishers.

Selman, B., and Kautz, H. 1993. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*. Chambery, France.

Selman, B.; Kautz, H.; and Cohen, B. 1994. Noise strategies for improving local search. In *Proceedings Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 337–343. Seattle, WA.

Tate, A. 1996. Representing plans as a set of constraints - the I-N-OVA model. In *Proceedings Third International Conference on AI Planning Systems (AIPS-96)*. Edinburgh.

Veloso, M. 1992. *Learning by analogical reasoning in general problem solving*. Ph.D. Dissertation, CMU.

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the use of integer programming models in ai planning. In *Proceedings Sixteenth National Conference on Artificial Intelligence (AAAI-99)*. Orlando, FL.

Walser, J. 1997. Solving linear pseudo-boolean constraint problems with local search. In *Proceedings Fourteenth National Conference on Artificial Intelligence (AAAI-97)*. Providence, RI.

Walser, J. P. 1999. *Integer Optimization by Local Search, A Domain-Independent Approach*. Springer Lecture Notes in Artificial Intelligence.

Weld, D. 1999. Recent advances in AI planning. *AI Magazine* 20(2).

Wolfman, S. A., and Weld, D. S. 2000. Combining linear programming and satisfiability solving for resource planning. *Knowledge Engineering Review* (this issue).