State-space Planning by Integer Optimization

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

Joachim P. Walser[1]
i2 Technologies
Airway Park, Lozenberg 23
B-1932 St. Stevens Woluwe, Belgium
walser@i2.com

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 IP 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 the AI community witnessed the unexpected success of satisfiability testing as a method for solving state-space planning problems (Weld 1999). Kautz and Selman (1996) demonstrated that in certain computationally challenging domains, the approach of axiomatizing problems in propositional logic and solving them with general randomized SAT algorithms (SATPLAN) was competitive with or superior to the best specialized planning systems. The framework has been shown to be quite broad, for example encompassing both action-centered and fluent-centered representations of change, conditional and maintenance goals, causal planning (Kautz, McAllester, & Selman 1996), automatic generation of axioms from STRIPS operators (Ernst, Millstein, & Weld 1997; Kautz & Selman 1999), hierarchical task networks (Mali & Kambhampati 1998), and domain-specific knowledge (Kautz & Selman 1998b).

Despite this generality, certain limitations in the framework still prevent it from being used for many practical, real-world domains. One problem is the difficulty in dealing with resources and the associated numeric constraints. For example, you might wish to assert that a "drive" action consumes 3 units of fuel, and that a "refuel" action resets the vehicle's tank to 15 units. Numeric variables that have a very small range can be represented by a set of Boolean variables, one for each possible value, but in general this introduces too many variables. In theory one could adopt a binary encoding of numeric quantities, but this would only keep the number of variables small by introducing a huge number of clauses to encode binary arithmetic.

Perhaps a more important limitation from an applications standpoint is that the notion of optimality inherent in the SATPLAN framework may be too weak. SATPLAN allows one to minimize the *parallel length* of a solution, where several *non-interfering* actions may occur at each time step. This notion of optimality, which is also shared by the popular Graphplan system and its descendents (Blum & Furst 1995; Koehler *et al.* 1997), 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, real world planning problems usually have more complex optimality criteria, that take into account *e.g.* different costs for different types of actions, minimization of resource usage, and so forth. Furthermore, 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: or, more generally, some function of both the parallel and sequential length. The two notions of length may actually be in conflict: for example, in the logistics domain that we consider in detail below, one can construct examples where the solution with the lowest sequential length has a greater than minimum parallel length.

It is desirable, therefore, to enrich the underlying language of the SATPLAN framework while retaining its computational advantages. This paper introduces an approach to AI planning based on *integer optimization* of integer linear programs (ILP), which we call ILP-PLAN. An ILP contains both Boolean (0/1) and integer valued variables, and represents both constraints and optimization functions as linear inequalities. ILP generalizes SAT because any clause can be written as a linear inequality over 0/1 variables (*e.g.*, $(p \vee \neg q)$ becomes $p + (1 - q) \geq 1$ (Hooker 1988)). ILP is a standard tool for Operations Research, but has been

[1] This work was carried out during a visit of the second author at AT&T Shannon Labs. Copyright ©1999, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

rarely exploited in AI applications. 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 in the AMPL modeling language (Fourer, Gay, & Kernighan 1993), and solved using the branch-and-bound algorithm by a commercial mixed integer programming package (ILOG CPLEX). We will discuss various encoding schemes and their tradeoffs. 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.

2 Planning and Integer Local Search

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).[2]

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 the literature on Graphplan, SATPLAN, and other recent planning algorithms such as ASP and LRTA* (Bonet, Loerincs, & Geffner 1997). We will demonstrate that the known computational advantages of using local search on a state-based encoding (Kautz & Selman 1996) of this domain can be retained, while the integer local search framework allows us to find solutions of lower *sequential* cost. 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.

[2] CLSP is unlike the kind of AI planning discussed in this paper in that it does not involve action selection.

3 Preliminaries

The class of planning problems considered in this paper is an extension of classical bounded-length state-space planning. States assign truth values to facts, and correspond to a bounded sequence of integers. An action is a partial function over states that can be specified by a precondition, add list, and delete list of positive facts. The parallel composition of a set of actions is defined for a state if none of the actions delete a precondition or effect of another. This core semantics underlies SATPLAN and Graphplan, and can be extended to conditional effects as described in (Koehler *et al.* 1997; Anderson, Smith, & Weld 1998).

The first extension is to introduce *resource variables* that are assigned a numeric value by each state. In general, resources may be integer or real-valued. Every resource value has a global minimum and maximum value. Actions may be extended with resource preconditions and effects. Following the framework of (Koehler 1998), a resource precondition is a simple linear inequality that must hold in any states in which the action is applicable. The effect of an action may be to *consume* (decrease), *produce* (increase), or *provide* (set) the value of a resource.[3] The parallel composition of a set of actions is defined for a state i if the following holds for every resource r:

- The set is logically conflict-free, as defined above;
- If the set 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 the set satisfies the global lower bound for r;
- The value of r at i plus the sum of the producers of r in the set satisfies the global upper bound for r;
- For any action a in the set with a resource precondition for r, the value of r at i minus the sum of the consumers of r *other than* a satisfies that 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. The second extension is 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 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.

4 Operator-based Encodings

We begin by developing encodings for STRIPS operators extended with resource effects and optimization objectives. We call these encodings "operator-based", because they are based on writing constraints between variables representing

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

actions and variables representing the preconditions and effects of those actions. In the terminology for SAT encodings from Ernst, Millstein, and Weld (1997), the encodings are an extension of "regular action representations with parallel actions and explanatory frame axioms". In the second part of the paper we will consider examples that use an alternative "state-based" encoding.

Recent work by Koehler (1998) describes an extension of the IPP/Graphplan framework to handle resource constraints via annotations on STRIPS operators. We will describe conventions for translating such annotations into ILP constraints, and present the results of applying this methodology to a transportation problem ("Airplane") and solving the instance with CPLEX, a popular branch-and-bound ILP engine. Our approach is more general than that of IPP, however, in that it allows us to include explicit optimization criteria.

4.1 Encoding Conventions

The encoding consists of three parts: constraints for the logical properties of actions; for resource usage; and for optimization objectives. The logical properties of STRIPS operators are encoded by the following kinds of axioms: (i) explanatory frame-axioms (if a state-change occurs, one action that could account for it must have taken place (Haas 1987; Schubert 1989; Kautz, McAllester, & Selman 1996)); (ii) an occurring action implies its effects and preconditions; (iii) exclusiveness of logically conflicting actions; and (iv) state invariant axioms in the style of (Kautz & Selman 1998b). See the references cited above for the details of the translation into CNF; each clause can then be easily converted into a linear inequality as described earlier.

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, according to the rules described in section 3 above. 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 the resource. Let *Prod* and *Con* be the sets of producing and consuming ground operators respectively. For simplicity we will say there is exactly one providing ground operator, s, which resets r to its maximum N. We introduce a new set of 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 are:

$$r_{i+1} = r_i + k_i - \sum_{a \in Con} c_a a_i + \sum_{a \in Prod} c_a a_i \quad (1)$$

resource $0 \leq \mathit{fuel} \leq C$

fly(a, b: airport):
precondition: *at-plane*$_a$
effects: *at-plane*$_b$, $\neg$*at-plane*$_a$
fuel $-= D_{ab} \cdot F$
$\forall$passenger p: *boarded*$_p$
effects: $at_{pb}, \neg at_{pa}$

refuel:
effects: *fuel* $= C$

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

$$s_i \rightarrow \neg a_i \quad \forall a \in Prod \cup Con \quad (2)$$

$$s_i \rightarrow (r_{i+1} \geq N) \quad (3)$$

$$\neg s_i \rightarrow (k_i = 0) \quad (4)$$

$$M \leq r_i - \sum_{a \in Con} c_a a_i \quad (5)$$

$$N \geq r_i + \sum_{a \in Prod} c_a a_i \quad (6)$$

We have written these constraints as mixed logical / linear inequalities, but each can be converted to a linear inequality, as we will do in the example below. (1) propagates the value of the resource from one time step to the next. (2) makes providers exclusive of other actions that affect the resource, while (3) and (4) establish the amount of resource created by a providing action (if any). Finally (5) and (6) enforce the global bounds on the resource. Due to lack of space we omit the translation of resource preconditions; in fact, for all the examples considered in this paper they are redundant due to the global resource bounds. Finally, resource optimization constraints are simply arbitrary linear inequalities over all the variables described above.

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.

Indices	Definition
i	action step (L is the last step)
a, b ; p	airports ; passenger
Constants	**Definition**
C, F	tank capacity, fuel use per dist. unit
D_{ab}	Distance from a to b
Variables	**Definition**
f_{abi}	flight from a to b occurs in step i.
refuel_i	refuel in step i.
$\mathit{refuel_amount}_i$	provider reset variable $\in [0, C]$
fuel_i	plane's fuel level $\in [0, C]$

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

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

$$\mathit{refuel}_i \rightarrow (\mathit{fuel}_{i+1} \geq C) \quad (7)$$

$$\neg \mathit{refuel}_i \rightarrow (\mathit{refuel_amount}_i = 0) \quad (8)$$

$$\mathit{fuel}_{i+1} = \mathit{fuel}_i + \mathit{refuel_amount}_i - \sum_{a,b:a \neq b} f_{abi} D_{ab} F \quad (9)$$

$$\mathit{refuel}_i + \sum_{a,b:a \neq b} f_{abi} \leq 1 \; \forall i \quad (10)$$

(7) and (8) link the decision variables for refueling with the fuel and refuel amounts and (9) states the fuel balance. Note that (7)-(8) are directly translated to linear inequalities ($p \rightarrow (x \geq k)$ is translated to $x - pk \geq 0$, and $p \rightarrow (x = 0)$ yields $(1-p)m - m \geq 0$, where p is a binary variable, variable x has bounds $0 \leq x \leq m$ and k is a constant). (10) is a compact way of making refuel (a provider) and flying (a consumer) mutually exclusive. The optimization objective is stated as

$$\text{minimize } \mathit{fuel}_1 - \mathit{fuel}_L + \sum_i \mathit{refuel_amount}_i,$$

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 table 2 reports experimental results using CPLEX 5.0 with standard/auto parameters settings.

Aspects of an Automated Translation. In this work, we approach resource-optimal planning using integer programming by directly formulating state-based encodings in a high-level IP modeling language, AMPL (Fourer, Gay, & Kernighan 1993). This approach provides maximal flexibility for experimenting with various combinations of encodings and IP solvers. Nevertheless, a system that automatically translates resource-annotated STRIPS into IP would have several advantages, including an improved representation of the logical portion of the problem by the use of

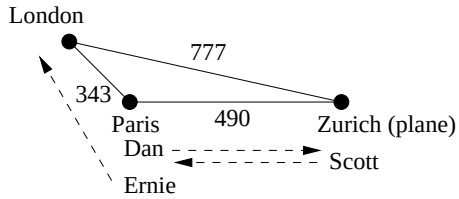

Figure 2: Scenario of initial state and travel destinations (arrows) of airplane-a. The resource-optimal plan found by CPLEX given the state-based ILP encoding has 5 steps: board Scott, fly to Paris || deplane Scott, refuel || board Dan and Ernie, fly to London || deplane Ernie, fly to Zürich || deplane Dan.

problem / steps	p/a	vars	cnstrs	min.fuel	time
airplane-a / 5	3	134	551	805	< 1s
airplane-b / 7	4	304	1774	897	54s
airplane-c / 9	5	576	4405	2096.5	895s

Table 2: Experimental results for airplane example. Columns are problem name / min. number of plan steps in a parallel plan, the number of passengers and airports (p/a), problem size in number of variables and constraints, the min. fuel consumption, and the solution time for CPLEX.

an intermediate representation such as a plan graph (Kautz & Selman 1998a), and ease in handling complex language constructs such as quantified conditional effects (Koehler *et al.* 1997; Anderson, Smith, & Weld 1998). A natural design for such a system would be a preprocessor for AMPL that handles STRIPS operators, thus allowing the user to mix the two representational levels. We are currently looking into implementation strategies.

It is important to note that many kinds of resource constraints are much more easily and naturally represented directly as linear inequalities, rather than as STRIPS annotations. For example, in the classic Missionaries and Cannibals problem, the *capacity constraint* that the cannibals never outnumber the missionaries on either shore can be written simply as

$$\sum_{m \in \mathit{missionaries}} at_{m,s,i} \geq \sum_{c \in \mathit{cannibals}} at_{c,s,i} \quad \forall i, s$$

while expressing the constraint using operator annotations is much more complex.

5 Case Study: Minimizing Plan-length

Traditionally, AI has concentrated on hard feasibility problems. In contrast, OR has put emphasis on approaches for finding near-optimal solutions to problems for which feasible solutions can be constructed easily. In both fields, there is increasing interest in finding near-optimal solutions to problems with a difficult feasibility aspect. Here, we consider a planning benchmark from the logistics domain

(Veloso 1992) that also exhibits this characteristic. The scenario is the transportation of a set of packages that involves flights and truck-drives between locations.

To model the problem, we use a variant of the state-based encodings presented in (Kautz & Selman 1996) that is extended to encode a notion of plan optimality. There are various possible criteria to optimize in this domain: (a) The total number of necessary actions, (b) the number of necessary time steps when parallel actions are allowed, (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).

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. This criteria allows us to directly compare the quality of our solutions to previous results in the literature. (This criteria appears to be generally useful for comparing planning systems, particularly for domains where it is easy to satisfy one of (a) or (b) alone.)

Since one of the best strategies to solve SATPLAN encodings is local search (Walksat), 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). Experimental results 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.

5.1 Integer Local Search Encoding

The basis for the encoding developed here is the *state-based* encoding described in Kautz and Selman (1996), since it currently provides the best representation for local search algorithms in the Walksat family. State-based encodings employ axioms that directly relate changes in fluents between adjacent states without explicit reference to actions. In theory such axiomatizations could be created from operator-based encodings by resolving away all action variables (Kautz, McAllester, & Selman 1996). However, in the current version of ILP-PLAN, the encodings were created by hand in the AMPL modeling language, a widely used specification language for linear and integer optimization.

As noted earlier, the criterion of plan optimality that we will consider in this paper is to minimize sequential plan length over plans of bounded (minimal) 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*".

It is interesting to note that the obvious alternative of using an operator-based encoding of the type described in the previous section yields encodings that are much harder to solve by local search. In fact, we were surprised to discover that the conjunction of an operator-based encoding 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).

Over-Constrained Integer Programs. To include optimization objectives into local search, the integer local search framework uses a representation introduced as over-constrained integer programs (OIPs) (Walser 1998). 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 $Ax \geq b, \quad Cx \leq d$ (*soft*), $\quad x \in D$, 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

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

wherein 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.

Augmented State-based OIP Encoding. 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. First, axioms are stated that directly relate changes in fluents between adjacent states

Indices	Definition
i, o	plan steps, objects.
a, b	locations.
v, p, t	vehicles, plane p, truck t.
Constants	**Definition**
c_l, c_f, c_d	cost of load/unload, flight, truck drive.
Action Variables	**Definition**
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 3: Parameters for the OIP encoding of 'logistics'.

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",

$$\mathit{at}_{iol} \wedge \neg \mathit{at}_{i+1,ol} \rightarrow \bigvee_{v} \mathit{in}_{i+1,ov} \quad \forall o, l, i.$$

Further, state invariant axioms are included as in the previous example. 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. Table 3 describes the indices and action variables used.

State changes are linked uni-directionally to the action variables by constraints of the type

$$\mathit{at}_{i,t} \wedge \neg \mathit{at}_{i+1,t} \rightarrow \mathit{drive_truck}_{i,t} \quad \forall t, i.$$

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.

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:

$$(\mathit{soft}) \sum_{i} c_l(\mathit{load}_{oi} + \mathit{unload}_{oi}) \leq c_l L_o$$

As noted in table 3, 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 1998). 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.[4] 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.

5.2 Experimental Results

The encodings were first simplified by AMPL's presolving algorithms 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.

[4] 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 state-based encoding			ASP functional encoding		LRTA*	ILP-PLAN augmented state-based OIP encoding			
	actions	time	m/actions	K-flips	time	actions	time	actions	actions	(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 4: 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, 'm/actions' 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.

Table 4 gives the experimental results. Note that the LRTA* and ASP algorithms are finding serial plans, and their actual parallel length is unknown but could be very high. 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".

Throughout the experiments, WSAT(OIP) was run with parameters $p_{hard} = 1.0, p_{zero} = 0.5, p_{noise} = 0.3$, and the following action costs were used: $c_f = 0.5$ (*fly_plane*), $c_d = 0.1$ (*drive_truck*), $c_l = 0.05$ (*load/unload*). We note that the computational results are relatively sensitive to the particular parameter settings. The particular setting of action costs was tuned in pre-experiments to favor short plans.

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 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); ILP-PLAN added soft constraints as described above; and LRTA* included an A*-type heuristic search function.

6 Related Work

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.

Recent work by Vossen *et al.* (1999) describes an alternative formulation of ILP encodings for planning problems (without resources or optimality conditions) that results in stronger linear relaxations. Their techniques improve the performance of branch-and-bound solvers and

are likely to be a valuable enhancement to the ILP-PLAN framework. An interesting similarity between their formulation and the one we develop for the logistics domain is that both eliminate explicit variables that represent actions (although other details differ). 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 1997) may provide the underpinnings for systems that more efficiently handle ILPs that have a large logical component.

7 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.

References

Anderson, C.; Smith, D.; and Weld, D. 1998. Conditional effects in graphplan. In *Proceedings AIPS-98*.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proc. IJCAI-95*. Montreal, Canada.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proc. AAAI-97*, 714–719.

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.

Bylander, T. 1991. Complexity results for planning. In *Proc. IJCAI-91*, 274–279. Sidney, Australia.

El-Kholy, A., and Richards, B. 1996. Temporal and resource reasoning in planning: the parcPLAN approach. In *Proc. AIPS-96*. Edinburgh.

Ernst, M.; Millstein, T.; and Weld, D. 1997. Automatic SAT-compilation of planning problems. In *Proc. IJCAI-97*. Nagoya, Japan.

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

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. 1997. Mixed logical/linear programming. *Discrete Applied Mathematics*. To appear.

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

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. AAAI-96*, 1194–1201.

Kautz, H., and Selman, B. 1998a. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the AIPS-98 Workshop on Planning as Combinatorial Search*.

Kautz, H., and Selman, B. 1998b. The role of domain-specific knowledge in the planning as satisfiability framework. In *Proceedings AIPS-98*.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. Under review.

Kautz, H.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proc. KR-96*.

Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. 1997. Extending planning graphs to an adl subset. In *Proc. 4th European Conf. on Planning*.

Koehler, J. 1998. Planning under resource constraints. In *Proceedings ECAI-98*.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *Proc. IJCAI-95*. Montreal, Canada.

Mali., A., and Kambhampati, S. 1998. Encoding htn planning in propositional logic. In *Proc. AIPS-98*. Pittsburgh, PA.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. KR-92*, 108–114. Boston.

Penberthy, J., and Weld, D. 1994. Temporal planning with continous change. In *Proc. AAAI-94*, 1010–1015.

Schubert, L. 1989. Monotonic solution of the frame problem in the situation calculus. In Kyburg, H. *et al.* eds., *Knowledge Representation and Defeasible Reasoning*. Kluwer Academic.

Selman, B.; Kautz, H.; and Cohen, B. 1994. Noise strategies for improving local search. In *Proc. AAAI-94*, 337–343.

Tate, A. 1996. Representing plans as a set of constraints - the I-N-OVA model. In *Proc. 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 *Proc. AAAI-99*.

Walser, J. 1997. Solving linear pseudo-boolean constraint problems with local search. In *Proceedings AAAI-97*.

Walser, J. 1998. *Domain-independent Local Search for Linear Integer Optimization*. Ph.D. Dissertation, Universität des Saarlandes.

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