A Linear Programming Heuristic for Optimal Planning

Tom Bylander
Division of Computer Science
The University of Texas at San Antonio
San Antonio, Texas 78249
bylander@cs.utsa.edu

Abstract

I introduce a new search heuristic for propositional STRIPS planning that is based on transforming planning instances to linear programming instances. The linear programming heuristic is admissible for finding minimum length plans and can be used by partial-order planning algorithms. This heuristic appears to be the first non-trivial admissible heuristic for partial-order planning. An empirical study compares Lplan, a partial-order planner incorporating the heuristic, to Graphplan, Satplan, and UCPOP on the tower of Hanoi domain, random blocks-world instances, and random planning instances. Graphplan is far faster in the study than the other algorithms. Lplan is often slower because the heuristic is time-consuming, but Lplan shows promise because it often performs a small search.

Introduction

Planning is the problem of finding a combination of actions that achieves a goal (Allen, Hendler, & Tate 1990; Hendler, Tate, & Drummond 1990). So far, there is limited success in general-purpose planning, in large part due to two factors. One factor is the computational complexity of planning, e.g., propositional STRIPS planning is PSPACE-complete except for very severe restrictions (Bäckström & Nebel 1995; Bylander 1994; Erol, Nau, & Subrahmanian 1995). The other factor is that planning formalisms make very strong representational assumptions, e.g., the STRIPS formalism assumes that actions are deterministic, and that an agent knows all relevant aspects of the environment (Dean & Wellman 1991).

Work within planning research can be viewed as attempting to find a suitable compromise or deviation from the known formalisms and algorithms. This paper introduces a new heuristic for propositional STRIPS planning based on converting planning instances to linear programming instances.

More precisely, an incomplete partial-order plan with a constraint on the length (number of operators) of the plan is translated to a 0-1 integer programming instance, and linear programming is used to determine if the relaxed instance has a solution. The relaxed instance does not require an integer solution, but retains the constraint that the values range from 0 to 1. If the relaxed instance has no solution, then the 0-1 integer programming instance has no solution, which implies that the partial-order plan cannot be refined to a plan with the given length. The heuristic value for an incomplete partial-order plan is the smallest length that results in a solution for the corresponding relaxed instance.

This heuristic is admissible for finding the shortest plan using partial-order planning with propositional STRIPS operators. To my knowledge, this is the first non-trivial admissible heuristic for partial-order planning. Heuristics discussed in the literature sacrifice optimality for efficiency (Ephrati, Pollack, & Milshtein 1996; Gerevini & Schubert 1996; Mcdermott 1996).

Furthermore, because linear programming variables are not restricted to binary values, I believe that this approach has great potential for incorporating utility, probabilistic operators, and lack of knowledge about the environment. However, the results in this paper are restricted to propositional STRIPS planning. Future research will consider these more general problems.

I compare Lplan, a planner using the linear programming heuristic, to Graphplan (Blum & Furst 1995), Satplan (Kautz & Selman 1996), and UCPOP (Penburthy & Weld 1992) on three domains: tower of Hanoi, random blocks world instances (Slaney & Thiébaux 1996), and random planning instances (Bylander 1996). Given a set of propositional STRIPS operators, an initial state, and a conjunctive goal, Lplan returns optimal plans (minimum number of operators). The comparison uses the other algorithms to solve the same problem. The operators for Graphplan were modified so that only one operator per time step is allowed. A linear encoding (Kautz & Selman 1992) was used for Satplan. The best-first search heuristic function for UCPOP returned the length of the partial plan. Overall, Graphplan was much faster than the other algorithms.

Lplan was often slower than the other algorithms primarily due to the time to evaluate the linear programming heuristic. Using high-quality commercial software (CPLEX) on a fast machine (Sun UltraSparc), each heuristic value required around a second of CPU time. Often, Lplan expanded a small number of nodes. Future research will attempt to find more efficient versions of this heuristic.

The remainder of this paper is organized as follows. First, I describe propositional STRIPS planning and how planning instances can be translated to linear programming instances. Next, I discuss how additional elements of partial-order planning can be translated to linear programming. Finally, I compare Lplan's performance to Graphplan, Satplan, and UCPOP on three domains. Space limitations prevent a more complete description of many topics.

Planning and Linear Programming

In this section, I define propositional STRIPS planning and show how a simple instance can be translated to a linear programming problem.

Propositional STRIPS Planning

An instance of *propositional STRIPS planning* is specified by a tuple $\langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where: $\mathcal{P}$ is a finite set of ground atomic formulas, called the *conditions*; $\mathcal{O}$ is a finite set of *operators*, where each operator consists of preconditions and postconditions, both of which are satisfiable conjunctions of positive and negative conditions; $\mathcal{I} \subseteq \mathcal{P}$ is the *initial state*; and $\mathcal{G}$, the *goals*, is a satisfiable conjunction of positive and negative conditions.

That is, $\mathcal{P}$ is the set of conditions that are relevant. Any state, e.g., the initial state $\mathcal{I}$, can be specified by a subset $S \subseteq \mathcal{P}$, indicating that $p \in \mathcal{P}$ is true in that state if $p \in S$, and false otherwise. $\mathcal{O}$ is the set of the operators that can change one state to another; allowing negative preconditions is a minor deviation from standard STRIPS (Fikes & Nilsson 1971). $S \subseteq \mathcal{P}$ is a *goal state* if the goals $\mathcal{G}$ is true of S.

An operator may be applied to a state if its preconditions are true of that state. If an operator is applied to a state, then the only changes in the new state from the old state is that the operator's postconditions become true.

For example, one propositional operator in the blocksworld is to move block A from on top of block B to on top of block C. This can be formulated as:

$$clear(A) \wedge on(A,B) \wedge clear(C) \Rightarrow$$
$$\neg on(A,B) \wedge on(A,C) \wedge clear(B) \wedge \neg clear(C)$$

where the $\Rightarrow$ separates the preconditions on the left from the postconditions on the right.

Propositional planning is PSPACE-complete in general (Bäckström & Nebel 1995; Bylander 1994; Erol, Nau, & Subrahmanian 1995) (NP-complete if plans are limited to a polynomial number of steps), so it appears very unlikely that there is a polynomial-time algorithm for planning. However, several recent results suggest that there might be algorithms that work well on average over a variety of planning domains. Graphplan (Blum & Furst 1995) and Satplan (Kautz & Selman 1996) are two new algorithms that have been empirically shown to be efficient on many instances. Work on partial-order planning algorithms has led to heuristics that lead to good empirical results (Gerevini & Schubert 1996). In addition, a theoretical analysis shows that certain kinds of random planning instances are generally easy on average (Bylander 1996).

From Planning to Linear Programming

A linear programming instance is defined by a set of variables, an objective function (a linear function of the variables), and a set of linear constraints (linear inequalities and equalities) on the variables. A feasible solution satisfies the linear constraints. An optimal solution satisfies the linear constraints and maximizes the objective function (Hillier & Leiberman 1974) (switching to minimization is trivial).

The translation from planning instances is based on restricting the length of the plan and mapping conditions and operators at each time point to variables, e.g., truth values for conditions are mapped to 0 and 1, an operator at a given time point maps to a value of 1 if it is applied at that time point, otherwise 0. If p is a condition, a variable is needed for each time point from 0 (the initial state) to time point l (the final state). These variables are denoted by $p(0), \ldots, p(l)$. If op is an operator, variables are needed for 0 to $l-1$; these are denoted by $op(0), \ldots, op(l-1)$.

The objective function is maximized if the goal is true at the last time point. The linear constraints encode the pre- and postconditions of each operator at each time step as well as the frame axioms.

For example, consider the following planning instance.

$\mathcal{P} = \{a, b, c, d\}$

$\mathcal{O} = \{a \wedge b \Rightarrow c \wedge \neg d, \;\; b \wedge c \Rightarrow a \wedge d\}$. Call the two operators $op1$ and $op2$, respectively.

$\mathcal{I} = \{a, b\}$

$\mathcal{G} = c \wedge d$

The goal can be achieved in two time steps by first applying $op1$, which leads to the state $\{a, b, c\}$, followed by applying $op2$, which results in the goal state $\{a, b, c, d\}$.

Assuming that a plan of length 2 is called for, the objective function is realized as follows:

maximize $c(2) + d(2)$

When both c and d are true at time point 2, this corresponds to $c(2) = 1$ and $d(2) = 1$.

The initial state is specified by assigning values to conditions at time point 0.

$$\begin{array}{lcl@{\qquad}lcl} a(0) & = & 1 & b(0) & = & 1 \\ c(0) & = & 0 & d(0) & = & 0 \end{array}$$

At most one operator is allowed at each time point. This is specified by:

$$1 \geq op1(0) + op2(0) \qquad 1 \geq op1(1) + op2(1)$$

An operator cannot be applied unless its preconditions are true. Here are the inequalities for time point 0.

$$\begin{array}{lcl} a(0) & \geq & op1(0) \\ b(0) & \geq & op1(0) + op2(0) \\ c(0) & \geq & op2(0) \end{array}$$

One inequality can represent the fact that b is a precondition for both $op1$ and $op2$.

The truth value of a condition will remain the same unless an operator makes it true or false. Here are the (not yet correct) equalities for the transition from time point 0 to 1.

$$\begin{aligned}
a(1) &= a(0) + op2(0)\\
b(1) &= b(0)\\
c(1) &= c(0) + op1(0)\\
d(1) &= d(0) + op2(0) - op1(0)
\end{aligned}$$

The problem with these equalities is that a postcondition of an operator might already be true (unless the preconditions specify otherwise). This problem is resolved with additional variables that represent these possibilities. The equalities above are changed to:

$$\begin{aligned}
a(1) &= a(0) + op2(0) - op2_a(0)\\
b(1) &= b(0)\\
c(1) &= c(0) + op1(0) - op2_c(0)\\
d(1) &= d(0) + op2(0) - op2_d(0) - op1(0) + op1_d(0)
\end{aligned}$$

When an additional variable equal to 1, it represents the situation where the corresponding postcondition is already true before the operator is applied. For example, if d is already true before $op2$ is applied at time 0, then this is represented by $op2_d(0) = 1$. Similarly, if $\neg d$ is already true before $op1$ is applied at time 0, then this is represented by $op1_d(0) = 1$.

Also, each additional variable must not be larger than the corresponding operator variable:

$$\begin{aligned}
op2_a(0) &\leq op2(0) & op1_d(0) &\leq op1(0)\\
op1_c(0) &\leq op1(0) & op2_d(0) &\leq op2(0)
\end{aligned}$$

Finally, the precondition inequalities need to be modified and extended:

$$\begin{aligned}
a(0) &\geq op1(0) + op2_a(0)\\
1 - a(0) &\geq op2(0) - op2_a(0)\\
b(0) &\geq op1(0) + op2(0)\\
c(0) &\geq op2(0) + op1_c(0)\\
1 - c(0) &\geq op1(0) - op1_c(0)\\
d(0) &\geq op2_d(0) + op1(0) - op1_d(0)\\
1 - d(0) &\geq op1_d(0) + op2(0) - op2_d(0)
\end{aligned}$$

These are needed so that an additional variable is not larger than the corresponding condition variable. For example, $op2_a(0) = 1$ represents the situation where a is true before $op2$ is applied at time 0, so $op2_a(0)$ cannot exceed $a(0)$. Similarly, $op2(0) - op2_a(0) = 1$ represents the situation where a is false before $op2$ is applied at time 0, so $op2(0) - op2_a(0)$ cannot exceed $1 - a(0)$.

Often, negative preconditions are added to operators if possible to avoid the creation of additional variables. This increases the efficiency of the linear programming heuristic.

Finally, all variables must be constrained to lie between 0 and 1, inclusive. The integer programming instance requires that each variable be either 0 or 1; the relaxed linear programming instance allows any value between 0 and 1.

An Admissible Heuristic

When a planning instance is translated to a set of linear constraints as done above, then the planning instance has a solution plan with a given length or less if and only if the corresponding integer programming instance has a "solution" (meaning that the objective function attains a value corresponding to all goals being true). This translates one NP-hard problem into another, which by itself is not much progress. The hope is that a solution for the easier relaxed instance corresponds to a solution for the planning instance.

If the relaxed instance with plan length l has a solution, but not the relaxed instance with plan length $l - 1$, then the minimum length plan must be at least l steps long. l cannot be an overestimate because there is a integer programming solution for the true length $l*$, which implies a relaxed solution for plan length $l*$ as well. Thus, relaxed instances with varying plan lengths can be used as an admissible heuristic. However, a solution to a relaxed instance does not even imply that the planning instance is solvable.

For the simple planning instance above, it turns out that there is no solution to the relaxed instance with plan length 1, and the only solution with plan length 2 is the integer solution with $op1(0) = 1$ and $op2(1) = 1$. In general though, one might not be so lucky.

Suppose that the planning instance has the additional operators $op3 \equiv a \Rightarrow \neg b \wedge c$ and $op4 \equiv b \Rightarrow \neg a \wedge d$. Because $op3$ and $op4$ conflict with each other, the goal cannot be reached using just $op3$ and $op4$. However, the relaxed instance with these additional operators permits the noninteger solution $op3(0) = op4(0) = op3(1) = op4(1) = 0.5$. $op3(0) = 0.5$ leads to $b(1) = 0.5$, which allows $op4(1) = 0.5$. Similarly, $op4(0) = 0.5$ results in $a(1) = 0.5$, which allows $op3(1) = 0.5$. The goals c and d are "achieved" halfway at the first time point, and fully at the second time point.

This "superposition" of operators in relaxed solutions is a major reason why the heuristic can lead to poor results. As described below in the empirical results , the tower of Hanoi domain is pathologic in this regard.

Partial-Order Planning

The above translation from planning instances to linear programming instances can be used fairly directly as an admissible heuristic for searching in the space of world states. For example, for forward progression, the only change is that the initial state changes from a parent state to a child state.

However, for partial-order planning, i.e., search in the space of partially-ordered operators, additional linear inequalities are needed to represent ordering constraints. There is also the problem of plan refinement (Kambhampati, Knoblock, & Yang 1995). See (Weld 1994) for an introduction to partial-order planning.

Ordering constraints are mapped to linear constraints as follows. Suppose steps $s1$ and $s2$ are instances of operators $op1$ and $op2$, respectively. Suppose that step $s1$ is ordered before step $s2$. With plan length l, variables $s1(0), \ldots, s1(l - 1), s2(0), \ldots, s2(l - 1)$ are created. $s1(i) = 1$ means that $s1$ is applied before or at time point i. All variables are restricted to the $[0, 1]$ range. This leads to:

$$
\begin{aligned}
s1(0) &\leq op1(0) \\
s2(0) &\leq op2(0) \\
s1(l-1) &= 1 \\
s2(l-1) &= 1 \\
s1(i) - s1(i-1) &\leq op1(i) \quad \text{for } 1 \leq i \leq l-1 \\
s2(i) - s2(i-1) &\leq op2(i) \quad \text{for } 1 \leq i \leq l-1 \\
s2(0) &= 0 \\
s2(i) &\leq s1(i-1) \quad \text{for } 1 \leq i \leq l-1
\end{aligned}
$$

The first four constraints are boundary conditions. The fifth and sixth constraints relate the values of the step variables to the operator variables. The last two constraints represent the ordering; $s2(0)$ must be 0, and $s2(i)$ cannot be higher than $s1(i-1)$. If an operator is used for more than one step, then the first, second, fifth, and sixth constraints above should sum all the appropriate step variables.

Empirical Study

I implemented a partial-order causal-link propositional planner based on the POP algorithm in (Weld 1994). During the search, the linear programming heuristic is evaluated as needed for each node. That is, the current node is known to have a heuristic value h, and all other open nodes are only known to be $\geq h$ or $\geq h+1$.

Flaw selection was done in two parts. First, flaws are found by (temporarily) linearizing the steps, attempting to minimize the number of steps with false preconditions. If this resulted in a solution plan, there is no need to explicitly add causal links for any remaining open conditions. Second, the flaw is selected from those found based on a simplified form of the LCFR strategy (Joslin & Pollack 1994). Threats were given priority over open conditions. Open conditions with the fewest available operators were preferred. The planner is implemented in Lucid Common Lisp.

Although linear programming is polynomial (Khachiyan 1979; Karmarkar 1984), solving a linear programming instance requires considerable time with currently known algorithms. I used the CPLEX Barrier Solver linear programming algorithm (CPLEX 1995).

Algorithms

I compared the implemented algorithm, called Lplan (Linear Programming Planning), to Graphplan (Blum & Furst 1995), Satplan (Kautz & Selman 1996), and UCPOP (Penburthy & Weld 1992). The intention of this empirical study was to make this comparison based on finding optimal plans (minimum number of operators) for propositional STRIPS planning instances. The other algorithms were applied accordingly.

Graphplan is based on searching a data structure called the planning graph for a solution plan. In general, Graphplan allows multiple operators at each time point as long as there is no conflict between the operators. However, this might result in a plan with a non-optimal number of operators, so for the study, the operators were preprocessed so that one operator is allowed per time point. Additional preprocessing was necessary if an instance had negative preconditions in its operators. Other parameters of Graphplan were maintained at default values.

	Lplan		Graphplan	Satplan	UCPOP	
Towers	Visits	Time	Time	Time	Visits	Time
2	4	0.42	0.02	0.02	39	0.03
3	12	3.97	0.07	5.44	5246	13.20
4	?	?	1.21	?	?	?
5	?	?	15.50	?	?	?
6	?	?	251.00	?	?	?

Table 1: Performance of the planning algorithms on the tower of Hanoi domain

Satplan is based on converting planning instances to propositional formulas, and then performing a randomized search for a satisfying assignment. Because the intent is to study propositional planning in general, the input to Satplan is based on a linear encoding of the operators (Kautz & Selman 1992). It should be noted, though, that for a specific domain, a suitably encoded domain theory can increase Satplan's efficiency (Kautz & Selman 1996), The default parameters of the randomized search algorithm were used as is except for adjusting the number of "tries," i.e., the number of times a search is attempted starting from a random assignment to the propositional formula. The running time of Satplan shown below does not include preprocessing of propositional formulas. Satplan was only run for the optimal plan length.

UCPOP is a causal-link partial-order planner that uses general action schemas. Nevertheless, propositional operators were input to UCPOP; this did not appear to decrease efficiency. To ensure that UCPOP finds optimal plans, the search heuristic is the number of plan steps, which leads to a uniform cost search. ZLIFO was turned on in this study.

Tower of Hanoi

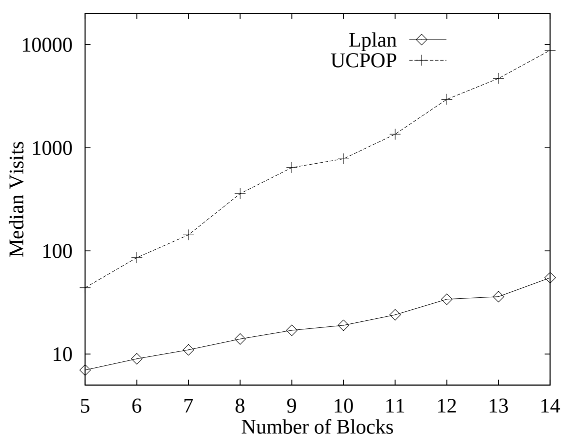
For the tower of Hanoi domain, the operators were encoded in the following manner:

$$
\begin{aligned}
& on(d3,t1) \wedge \neg on(d3,t2) \wedge on(d1,t3) \wedge on(d2,t3) \\
& \Rightarrow on(d3,t2) \wedge \neg on(d3,t1)
\end{aligned}
$$

where $d3$ refers to disk 3 (disk 1 is the smallest) and $t1$ refers to tower 1. Note that the operator has a negative precondition $\neg on(d3,t2)$. Otherwise, Lplan must create additional variables because it does not know whether or not $on(d3,t2)$ is true before the operator is applied. Negative preconditions were omitted for UCPOP.

Table 1 gives the results for the tower of Hanoi domain from 2 to 6 disks. Visits is the number of nodes visited. Time is given in seconds. As can be seen, Graphplan was able to find solutions up to 6 disks (plan length = 63). Lplan, Satplan, and UCPOP were unable to find a solution to the 4 disk instance. Lplan and UCPOP were halted after $100,000$ node visits. Satplan was halted after $100,000$ tries.

The difficulty for Lplan is that the linear programming heuristic is pathologically bad for this domain. For $n \geq 3$ disks, the heuristic gives a value of $2n$ for the distance from the initial state to the goal state. However, the shortest plan requires $2^n - 1$ steps, so the heuristic is exponentially bad



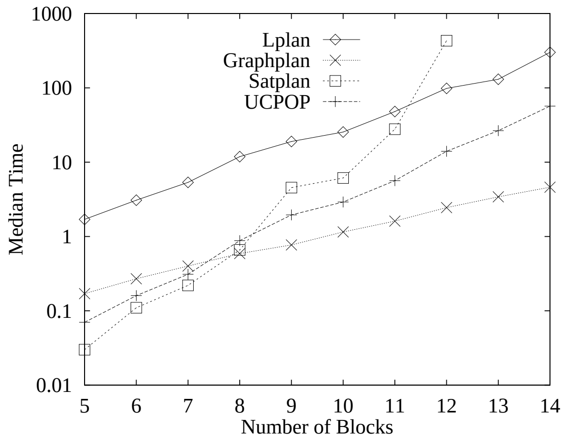


Figure 1: Empirical results for random blocks-world instances. The first graph is for Lplan and UCPOP only.

for this domain. The heuristic gives a low value because the relaxed instance allows a disk to be halfway on a tower, and disks that are halfway on can be freely moved past each other.

Random Blocks-World Problems

For the blocks-world domain, random blocks-world instances (Slaney & Thiébaux 1996) are generated for 5 blocks up to 14 blocks. A random block-world instance for a given number of blocks selects a random initial state and goal state from a uniform distribution of possible blocks-world states.

The number of operators in each instance was reduced. E.g., if block A is initially on block B, and the goal state has A on C, then only three operators are needed to move A: move A from B to the table, move A from the table to C, and move A from B to C. This reduces the number of operators for n blocks from $O(n^3)$ to $3n$ or less. Except for UCPOP, the operators included a negative precondition corresponding to each positive postcondition.

100 instances were generated for each number of blocks. Figure 1 illustrates the results. The first graph shows the median number of nodes visited by Lplan and UCPOP. The second graph shows the median running time for all four algorithms. The y-axes are logarithmically scaled. Lplan usually visits a very small portion of the search space, but uses significant time per node visit. Of all the algorithms, Graphplan is clearly much faster and appears to have the smallest exponential growth (the slope of its line is lowest). However, Lplan's exponential growth appears to be lower than Satplan and UCPOP.

Random Planning Instances

Random planning instances are generated by randomly generating operators (Bylander 1996). The advantage of random planning instances is that it is not possible to take unfair advantage of domain-specific characteristics. A possible disadvantage is that determining solvability of these instances is known to be easy on average if there are few or many operators. It is not known whether finding optimal plans for random planning instances is hard on average.

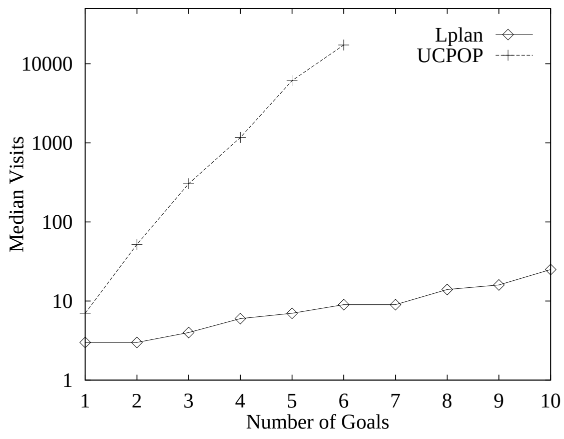
The experiment used 10 conditions, 2 preconditions/operator, and 2 postconditions/operator. The pre- and postconditions are randomly selected and randomly negated. Initial states and goals are randomly generated. No goal is true in the initial state. The number of goals were varied from 1 to 10. The number of operators were varied from 1 to 50. 10 instances were generated for each number of goals and operators, leading to 5000 instances. 2360 of these instances had solution plans. The empirical results are specific to solvable instances.

Figure 2 shows the results with respect to the number of goals. The first graph shows the median number of nodes visited by Lplan and UCPOP. The second graph shows the median running time for each algorithm. The y-axes are logarithmically scaled. Again, Lplan usually visits a very small portion of the search space. Lplan outperforms UCPOP when there 5 or more goals. The median running time of Satplan was faster than Graphplan, but this would not be true if Satplan's preprocessing time were included. Lplan appears to have larger exponential growth than Graphplan and Satplan. In the future, I intend to obtain results for larger random planning instances.

Conclusion

I have presented a new search heuristic for propositional planning, which, to my knowledge, is the first non-trivial admissible heuristic applicable to partial-order planning. The heuristic is based on transforming planning instances to integer programming instances, and solving the relaxed linear programming instance. Empirical results show that Lplan, a partial-order planner employing this heuristic, often visits a small number of nodes, but that evaluating the heuristic is often time-consuming when compared to the performance of Graphplan, Satplan, and UCPOP.

Future research will focus on more efficient use of the linear programming heuristic and more efficient versions of the heuristic.



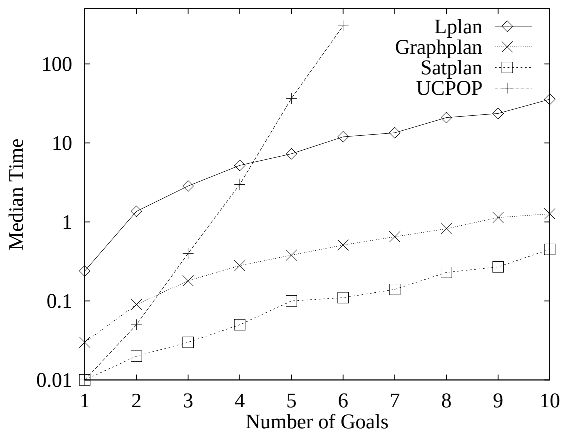


Figure 2: Empirical results for random planning instances. The first graph is for Lplan and UCPOP only.

Acknowledgments

I thank Subbarao Kamphampati for helpful comments. I thank several people for making their software available: Avrim Blum and Merrick Furst for Graphplan, Henry Kautz and Bart Selman for Satplan, and Daniel Weld et al. for UCPOP.

References

Allen, J.; Hendler, J.; and Tate, A., eds. 1990. *Readings in Planning*. San Mateo, California: Morgan Kaufmann.

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

Blum, A. L., and Furst, M. L. 1995. Fast planning through planning graph analysis. In *Proc. Fourteenth Int. Joint Conf. on Artificial Intellgence*, 1636–1642.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69:161–204.

Bylander, T. 1996. A probabilistic analysis of propositional STRIPS planning. *Artificial Intelligence* 81:241–270.

CPLEX Optimization, Inc., Incline Village, Nevada. 1995. *Using the CPLEX Callable Library*.

Dean, T. L., and Wellman, M. P. 1991. *Planning and Control*. San Mateo, California: Morgan Kaufmann.

Ephrati, E.; Pollack, M. E.; and Milshtein, M. 1996. A cost directed planner: Preliminary report. In *Proc. Thirteenth National Conf. on Artificial Intelligence*, 1223–1228.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, decidability, and undecidability results for domain-independent planning. *Artificial Intelligence* 76:75–88.

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

Gerevini, A., and Schubert, L. 1996. Accelerating partial-order planners: Some techniques for effective search control and planning. *J. Artificial Intelligence Research* 5:95–137.

Hendler, J.; Tate, A.; and Drummond, M. 1990. AI planning: Systems and techniques. *AI Magazine* 11(2):61–77.

Hillier, F. S., and Leiberman, G. J. 1974. *Operations Research*. San Francisco: Holden-Day.

Joslin, D., and Pollack, M. E. 1994. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proc. Twelfth National Conf. on Artificial Intelligence*, 1004–1009.

Kambhampati, S.; Knoblock, C. A.; and Yang, Q. 1995. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence* 76:167–238.

Karmarkar, N. 1984. A new polynomial time algorithm for linear programming. *Combinatorica* 4:373–395.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proc. Tenth European Conf. on Artificial Intelligence*, 359–363.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. Thirteenth National Conf. on Artificial Intelligence*, 1194–1201.

Khachiyan, L. G. 1979. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady* 20:191–194.

Mcdermott, D. 1996. A heuristic estimator for means-ends analysis in planning. In *Proc. Third Int. Conf. on Artificial Intelligence Planning Systems*, 142–149.

Penburthy, J. S., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planning for ADL. In *Proc. Third Int. Conf. on Principles of Knowledge Representation and Reasoning*, 103–114.

Slaney, J., and Thiébaux, S. 1996. Linear time near-optimal planning in the blocks world. In *Proc. Thirteenth National Conf. on Artificial Intelligence*, 1208–1214.

Weld, D. S. 1994. An introduction to least-commitment planning. *AI Magazine* 15(4):27–61.