An LP-Based Heuristic for Optimal Planning

Menkes van den Briel[1], J. Benton[2], Subbarao Kambhampati[2], and Thomas Vossen[3]

[1] Arizona State University, Department of Industrial Engineering,
[2] Department of Computer Science and Engineering,
Tempe AZ, 85287, USA
{menkes,j.benton,rao}@asu.edu
[3] University of Colorado, Leeds School of Business,
Boulder CO, 80309, USA
vossen@colorado.edu

Abstract. One of the most successful approaches in automated planning is to use heuristic state-space search. A popular heuristic that is used by a number of state-space planners is based on relaxing the planning task by ignoring the delete effects of the actions. In several planning domains, however, this relaxation produces rather weak estimates to guide search effectively. We present a relaxation using (integer) linear programming that respects delete effects but ignores action ordering, which in a number of problems provides better distance estimates. Moreover, our approach can be used as an admissible heuristic for optimal planning.

Keywords: Automated planning, improving admissible heuristics, optimal relaxed planning

1 Introduction

Many heuristics that are used to guide heuristic state-space search planners are based on constructing a relaxation of the original planning problem that is easier to solve. The idea is to use the solution to the relaxed problem to guide search for the solution to the original problem. A popular relaxation that has been implemented by several planning systems, including UNPOP [17,18], HSP [4,5], and FF [15], involves using *relaxed actions* in which the delete effects of the original actions are ignored.

For example, FF estimates the distance between an intermediate state and the goals by creating a planning graph [3] using relaxed actions. From this graph, FF extracts in polynomial time a relaxed plan whose corresponding plan length is used as an inadmissible, but effective, distance estimate. One can transform this approach into an admissible heuristic by finding the optimal relaxed plan, also referred to as h^+ [14], but computing such a plan is NP-Complete [8]. In order to extract the optimal relaxed plan one must extend the relaxed planning graph to level off [3] so that all reachable actions can be considered.

Although ignoring delete effects turns out to be quite effective for many planning domains, there are some obvious weaknesses with FF's relaxed plan heuristic. For example, in a relaxed plan no atom changes more than once, if an

C. Bessiere (Ed.): CP 2007, LNCS 4741, pp. 651–665, 2007.

atom becomes true it remains true as it is never deleted. However, in a plan corresponding to the original problem an atom may be added and deleted several times. In order to improve the quality of the relaxed plan we consider a relaxation based on *relaxed orderings*.

In particular, we view a planning problem as a set of interacting network flow problems. Given a planning domain where states are defined in terms of n boolean or multi-valued state variables (i.e. fluents), we view each fluent as a separate flow problem, where nodes correspond to the values of the fluent, and arcs correspond to the transitions between these values. While network flow problems are computationally easy, what makes this flow problem hard is that the flows are "coupled" as actions can cause transitions in multiple fluents.

We set up an IP formulation where the variables correspond to the number times each action is executed in the solution plan. The objective is to minimize the number of actions, and the constraints ensure that each pre-condition is supported. However, the constraints do not ensure that pre-conditions are supported in a correct ordering. Specifically, for pre-conditions that are deleted we setup balance of flow constraints. That is, if there are m action instances in the plan that cause a transition from value f of fluent c, then there must be m action instances in the plan that cause a transition to value f of c. Moreover, for pre-conditions that are not deleted we simply require that they must be supported.

The relaxation that we pursue in this paper is that we are not concerned about the specific positions of where the actions occur in the plan. This can, to some extent, be thought of as ignoring the ordering constraints that ensure that the actions can be linearized into a feasible plan. An attractive aspect of our formulation is that it is not dependent on the length of the plan. Previous integer programming-based formulations for planning, such as [7,9,21], use a step-based encoding. In a step-based encoding the idea is to set up a formulation for a given plan length and increment it if no solution can be found. The problem with such an encoding is that it may become impractically large, even for medium sized planning tasks. In a step-based encoding, if l steps are needed to solve a planning problem then l variables are introduced for each action, whereas in our formulation we require only a single variable for each action.

The estimate on the number of actions in the solution plan, as computed by our IP formulation, provides a lower bound on the optimal (minimum number of actions) plan. However, since solving an IP formulation is known to be computationally intractable, we use the linear programming (LP) relaxation which can be solved in polynomial time. We will see that this double relaxation is still competitive with other admissible heuristics after we add non-standard and strong valid inequalities to the formulation. In particular, We show that the value of our LP-relaxation with added inequalities, gives very good distance estimates and in some problem instances even provides the optimal distance estimate.

While the current paper focuses on admissible heuristics for optimal sequential planning, the flow-based formulation can be easily extended to deal with more general planning problems, including cost-based planning and over-subscription

planning. In fact, a generalization of this heuristic leads to state-of-the-art performance in oversubscription planning with non-uniform actions costs, goal utilities as well as dependencies between goal utilities [2].

This paper is organized as follows. In Section 2 we describe our action selection formulation and describe a number of helpful constraints that exploit domain structure. Section 3 reports some experimental results and related work is described in Section 4. In Section 5 we summarize our main conclusions and describe some avenues for future work.

2 Action Selection Formulation

We find it useful to do the development of our relaxation in terms of multi-valued fluents (of which the boolean fluents are a special case). As such, we will use the SAS+ formalism [1] rather than the usual STRIPS/ADL one as the background for our development. SAS+ is a planning formalism that defines actions by their prevail-conditions and effects. Prevail-conditions describe which variables must propagate a certain value during the execution of the action and effects describe the pre- and post-conditions of the action.

To make the connection between planning and network flows more straightforward, we will restrict our attention to a subclass of SAS+ where each action that has an post-condition on a fluent also has a pre-condition on that fluent. We emphasize that this restriction is made for ease of exposition and can be easily removed; indeed our work in [2] avoids making this restriction.

2.1 Notation

We define a SAS+ planning task as a tuple $\Pi = \langle C, A, s_0, s_* \rangle$, where

- $C = \{c_1, ..., c_n\}$ is a finite set of state variables, where each state variable $c \in C$ has an associated domain V_c and an implicitly defined extended domain $V_c^+ = V_c \cup \{u\}$, where u denotes the *undefined value*. For each state variable $c \in C$, $s[c]$ denotes the value of c in state s. The value of c is said to be *defined* in state s if and only if $s[c] \neq u$. The total state space $S = V_{c_1} \times ... \times V_{c_n}$ and the partial state space $S^+ = V_{c_1}^+ \times ... \times V_{c_n}^+$ are implicitly defined.
- A is a finite set of actions of the form $\langle pre, post, prev \rangle$, where pre denotes the pre-conditions, $post$ denotes the post-conditions, and $prev$ denotes the prevail-conditions. For each action $a \in A$, $pre[c], post[c]$ and $prev[c]$ denotes the respective conditions on state variable c. The following two restrictions are imposed on all actions: (1) Once the value of a state variable is defined, it can never become undefined. Hence, for all $c \in C$, if $pre[c] \neq u$ then $pre[c] \neq post[c] \neq u$; (2) A prevail- and post-condition of an action can never define a value on the same state variable. Hence, for all $c \in C$, either $post[c] = u$ or $prev[c] = u$ or both.
- $s_0 \in S$ denotes the initial state and $s_* \in S^+$ denotes the goal state. We say that state s is *satisfied* by state t if and only if for all $c \in C$ we have $s[c] = u$ or $s[c] = t[c]$. This implies that if $s_*[c] = u$ for state variable c, then any defined value $f \in V_c$ satisfies the goal for c.

While SAS+ planning allows the initial state, the goal state and the pre-conditions of an action to be partial, we assume that s_0 is a total state and that all pre-conditions are defined for all state variables on which the action has post-conditions (i.e. $pre[c] = u$ if and only if $post[c] = u$). The assumption that s_0 is a total state is common practice in automated planning. However, the assumption that all pre-conditions are defined is quite strong, therefore, we will briefly discuss a way to relax this second assumption in Section 5.

An important construct that we use in our action selection formulation is the so-called *domain transition graph* [13]. A domain transition graph is a graph representation of a state variable and shows the possible ways in which values can change. Specifically, the domain transition graph DTG_c of state variable c is a labeled directed graph with nodes for each value $f \in V_c$. DTG_c contains a labeled arc (f_1, f_2) if and only if there exists an action a with $pre[c] = f_1$ and $post[c] = f_2$ or $pre[c] = u$ and $post[c] = f_2$. The arc is labeled by the set of actions with corresponding pre- and post-conditions. For each arc (f_1, f_2) with label a in DTG_c we say that there is a *transition* from f_1 to f_2 and that action a has an *effect* in c.

We use the following notation.

- $DTG_c = (V_c, E_c)$: is a directed domain transition graph for every $c \in C$
- V_c: is the set of possible values for each state variable $c \in C$
- E_c: is the set of possible transitions for each state variable $c \in C$
- $V_c^a \subseteq V_c$ represents the prevail condition of action a in c
- $E_c^a \subseteq E_c$ represents the effect of action a in c
- $A_c^E := \{a \in A : |E_c^a| > 0\}$ represents the actions that have an effect in c, and $A_c^E(e)$ represents the actions that have the effect e in c
- $A_c^V := \{a \in A : |V_c^a| > 0\}$ represents the actions that have a prevail condition in c, and $A_c^V(f)$ represents the actions that have the prevail condition f in c
- $V_c^+(f)$: to denote the in-arcs of node f in the domain transition graph G_c;
- $V_c^-(f)$: to denote the out-arcs of node f in the domain transition graph G_c;

Moreover, we define the *composition* of two state variables, which is related to the parallel composition of automata [10], as follows.

Definition 1. *(Composition) Given the domain transition graph of two state variables c_1, c_2, the composition of DTG_{c_1} and DTG_{c_2} is the domain transition graph $DTG_{c_1||c_2} = (V_{c_1||c_2}, E_{c_1||c_2})$ where*

- $V_{c_1||c_2} = V_{c_1} \times V_{c_2}$
- $((f_1, g_1), (f_2, g_2)) \in E_{c_1||c_2}$ *if* $f_1, f_2 \in V_{c_1}$, $g_1, g_2 \in V_{c_2}$ *and there exists an action* $a \in A$ *such that one of the following conditions hold.*
  - $pre[c_1] = f_1$, $post[c_1] = f_2$, *and* $pre[c_2] = g_1$, $post[c_2] = g_2$
  - $pre[c_1] = f_1$, $post[c_1] = f_2$, *and* $prev[c_2] = g_1$, $g_1 = g_2$
  - $pre[c_1] = f_1$, $post[c_1] = f_2$, *and* $g_1 = g_2$

We say that $DTG_{c_1||c_2}$ is the composed domain transition graph of DTG_{c_1} and DTG_{c_2}.

Example. Consider the set of actions $A = \{a, b, c, d\}$ and set of state variables $C = \{c_1, c_2\}$ whose domain transition graphs have $V_{c_1} = \{f_1, f_2, f_3\}$, $V_{c_2} = \{g_1, g_2\}$ as the possible values, and $E_{c_1} = \{(f_1, f_3), (f_3, f_2), (f_2, f_1)\}$, $E_{c_2} = \{(g_1, g_2), (g_2, g_1)\}$ as the possible transitions as shown in Figure 1. Moreover, $A^E_{c_1} = \{a, b, c\}$, $A^E_{c_2} = \{b, d\}$ are the actions that have an effect in c_1 and c_2 respectively, and $A^V_{c_1} = \emptyset$, $A^V_{c_2} = \{a\}$ are the actions that have a prevail condition in c_1 and c_2 respectively. The effect and prevail condition of action a are represented by $E^a_{c_1} = (f_1, f_3)$ and $V^a_{c_2} = g_1$ respectively and the set of in-arcs for node g_1 is given by $V^+_{c_2}(g_1) = \{(g_2, g_1)\}$. Note that, since prevail conditions do not change the value of a state variable, we do not consider them to be transitions. The common actions in the composed domain transition graph, that is, actions in $A^E_{c_1} \cap A^E_{c_2}$ can only be executed simultaneously in the two domain transition graphs. Hence, in the composition the two domain transition graphs are synchronized on the common actions. The other actions, those in $A^E_{c_1} \backslash A^E_{c_2} \cup A^E_{c_2} \backslash A^E_{c_1}$, are not subject to such a restriction and can be executed whenever possible.

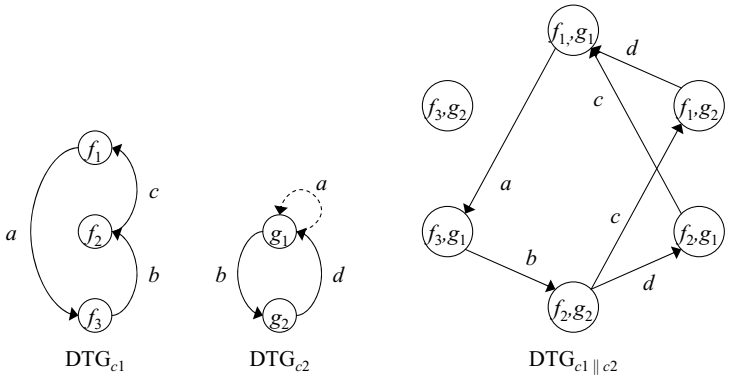

Fig. 1. Two domain transition graphs and their composition

2.2 Formulation

Our action selection formulation models each domain transition graph in the planning domain as an appropriately defined network flow problem. Interactions between the state variables, which are introduced by the pre-, post-, and prevail-conditions of the actions, are modeled as side constraints on the network flow problems. The variables in our formulation indicate how many times an action is executed, and the constraints ensure that all the action pre-, post-, and prevail-conditions must be respected. Because we ignore action ordering, we are solving a relaxation on the original planning problem. Our relaxation, however, is quite different from the more popular relaxation that ignores the delete effects of the actions.

Variables. We define two types of variables. We create one variable for each ground action and one for each state variable value. The action variables indicate

how many times each action is executed and the variables representing state variable values indicate which values are achieved at the end of the solution plan. The variables are defined as follows.

- $x_a \in \mathbb{Z}^+$, for $a \in A$; $x_a \geq 0$ is equal to the number of times action a is executed.
- $y_{c,f} \in \{0,1\}$, for $c \in C$, $f \in V_c$; $y_{c,f}$ is equal to 1 if the value f in state variable c is achieved at the end of the solution plan, and 0 otherwise.

Objective function. The objective function that we use minimizes the number of actions. Note, however, that we can deal with action costs by simply multiplying each action variable with a cost parameter c_a. Goal utilities can be dealt with by including the summation $\sum_{c \in C, f \in V_c : f = s_*[c]} u_{c,f} y_{c,f}$, where $u_{c,f}$ denotes the utility parameter for each goal.

$$\sum_{a \in A} x_a \tag{1}$$

Constraints. We define three types of constraints. Goal constraints ensure that the goals in the planning task are achieved. This is done by fixing the variables corresponding to goal values to one. Effect implication constraints define the network flow problems of the state variables. These constraints ensure that the effect of each action (i.e. transition in the domain transition graph) is supported by the effect of some other action. That is, one may execute an action that deletes a certain value if and only if one executes an action that adds that value. These constraints also ensure that all goals are supported. The prevail condition implication constraints ensure that the prevail conditions of an action must be supported by the effect of some other action. The M in these constraints denotes a large constant and allows actions with prevail conditions to be executed multiple times as long as their prevail condition is supported at least once. Note that, the initial state automatically adds the values that are present in the initial state.

- Goal constraints for all $c \in C$, $f \in V_c$: $f = s_*[c]$

$$y_{c,f} = 1 \tag{2}$$

- Effect implication constraints for all $c \in C$, $f \in V_c$

$$\sum_{e \in V_c^+(f) : b \in A_c^E(e)} x_b + 1\{\text{if } f = s_0[c]\} = \sum_{e \in V_c^-(f) : a \in A_c^E(e)} x_a + y_{c,f} \tag{3}$$

- Prevail condition implication constraints for all $c \in C$, $f \in V_c$: $a \in A_c^V(f)$

$$\sum_{e \in V_c^+(f) : b \in A_c^E(e)} x_b + 1\{\text{if } f = s_0[c]\} \geq x_a / M \tag{4}$$

One great advantage of this IP formulation over other step-based encodings is its size. The action selection formulation requires only one variable per action, whereas a step-based encoding requires one variable per action for each plan step. In a step-based encoding, if l steps are needed to solve a planning problem then l variables are introduced for each action.

Note that any feasible plan satisfies the above constraints. In a feasible plan all goals are satisfied, which is expressed by the constraints (2). In addition, in a feasible plan an action is executable if and only if its pre-conditions and prevail conditions are supported, which is expressed by constraints (3) and (4). Since any feasible will satisfy the constraints above, the formulation provides a relaxation to the original planning problem. Hence, an optimal solution to this formulation provides a bound (i.e. an admissible heuristic) on the optimal solution of the original planning problem.

2.3 Adding Constraints by Exploiting Domain Structure

We can substantially improve the quality of LP-relaxation of the action selection formulation by exploiting domain structure in the planning problem. In order to automatically detect domain structure in a planning problem we use the so-called *causal graph* [22]. The causal graph is defined as a directed graph with nodes for each state variable and directed arcs from source variables to sink variables if changes in the sink variable have conditions in the source variable. In other words, there is an arc in the causal graph if there exists an action that has an effect in the source variable and an effect or prevail condition in the source variable. We differentiate between two types of arcs by creating an effect causal graph and a prevail causal graph as follows ([16] use labeled arcs to make the same distinction).

Definition 2. *(Effect causal graph) Given a planning task $\Pi = \langle C, A, s_0, s_* \rangle$, the effect causal graph $G_\Pi^{effect} = (V, E^{effect})$ is an undirected graph whose vertices correspond to the state variables of the planning task. G_Π^{effect} contains an edge (c_1, c_2) if and only if there exists an action a that has an effect in c_1 and an effect in c_2.*

Definition 3. *(Prevail causal graph) Given a planning task $\Pi = \langle C, A, s_0, s_* \rangle$, the prevail causal graph $G_\Pi^{prevail} = (V, E^{prevail})$ is a directed graph whose nodes correspond to the state variables of the planning task. $G_{prevail_\Pi}$ contains a directed arc (c_1, c_2) if and only if there exists an action a that has a prevail condition in c_1 and an effect in c_2.*

By analyzing the effect causal graph, the prevail causal graph, and the domain transition graphs of the state variables, we are able to tighten the constraints of the integer programming formulation and improve the value of the corresponding LP relaxation. In particular, we add constraints to our formulation if certain (global) causal structure and (local) action substructures are present in the causal graphs and domain transition graphs respectively.

Example. The effect causal graph and prevail causal graph corresponding to the example shown in Figure 1 is given by Figure 2. Since action b has an effect in state variables c_1 and c_2 there is an edge (c_1, c_2) in G_{Π}^{effect}. Similarly, since action a has an effect in c_1 and a prevail condition in c_2 there is an arc (c_2, c_1) in $G_{\Pi}^{prevail}$.

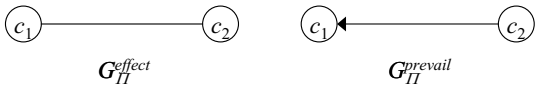

Fig. 2. The effect causal graph and prevail causal graph corresponding to Figure 1

Type 1 Domain Structure Constraints. The first set of domain structure constraints that we add to the action selection formulation deals with cycles in the causal graph. Causal cycles are undesirable as they describe a two-way dependency between state variables. That is, changes in a state variable c_1 will depend on conditions in a state variable c_2, and vice versa. It is possible that causal cycles involve more than two state variables, but we only consider 2-cycles (i.e. cycles of length two). A causal 2-cycle may appear in the effect causal graph and in the prevail causal graph. Note that, since the effect causal graph is undirected, any edge in the effect causal graph corresponds to a 2-cycle.

For every 2-cycle involving state variables c_1 and c_2 we create the composition $DTG_{c_1||c_2}$ if the following conditions hold.

- For all $a \in A_{c_1}^E$ we have $a \in (A_{c_2}^E \cup A_{c_2}^V)$
- For all $a \in A_{c_2}^E$ we have $a \in (A_{c_1}^E \cup A_{c_1}^V)$

In other words, for every action a that has an effect in state variable c_1 (c_2) we have that a has an effect or prevail condition in state variable c_2 (c_1). This condition will restrict the composition to provide a complete synchronization of the two domain transition graphs. Now, for each composed domain transition graph that is created we define an appropriately defined network flow problem. The corresponding constraints ensure that the two-way dependencies between state variables c_1 and c_2 are respected.

- Type 1 domain constraints for all $c_1, c_2 \in C$ such that $DTG_{c_1||c_2}$ is defined and $f \in V_{c_1}$, $g \in V_{c_2}$

$$\sum_{e \in V^+_{c_1||c_2}(f,g): b \in A^E_{c_1||c_2}(e)} x_b + 1\{\text{if } f = s_0[c_1] \wedge g = s_0[c_2]\} = \sum_{e \in V^-_{c_1||c_2}(f,g): a \in A^E_{c_1||c_2}(e)} x_a \tag{5}$$

Example. In order to provide some intuition as to why these constraints are important and help improve the action selection formulation, consider the following

scenario. Assume we are given the set of actions $A = \{a, b\}$ and the set of state variables $C = \{c_1, c_2\}$, such that $E^a_{c_1} = (f_1, f_2)$, $E^a_{c_2} = (g_2, g_3)$, $E^b_{c_1} = (f_2, f_3)$, and $E^b_{c_2} = (g_1, g_2)$. The effect implication constraint (3) allows the effect of action a support the effect of action b in c_1, and it allows the effect of action b support the effect of action a in c_2. However, in the composed domain transition graph, it is clear that neither action a or b can support each other. Hence, if actions a and b are selected in the solution plan, then the solution plan must include one or more other actions in order to satisfy the network flow constraints in the composed domain transition graph.

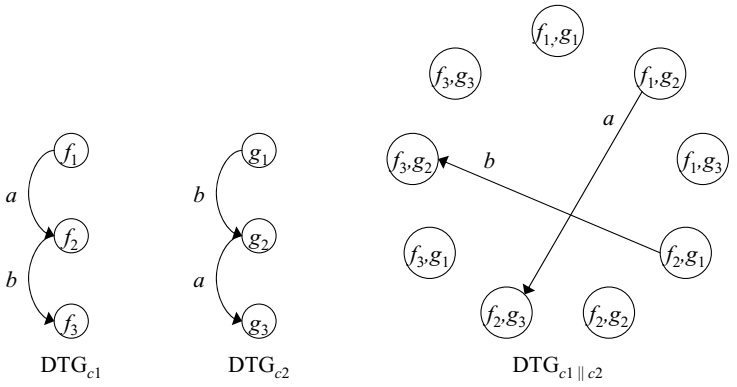

Fig. 3. Actions a and b can both support each other in either DTG_{c_1} and DTG_{c_2}, but not in $DTG_{c_1||c_2}$

Type 2 Domain Structure Constraints. The second set of domain structure constraints that we add to the action selection formulation deals with the structure given in Figure 4. That is, we have an arc (c_1, c_2) in the prevail causal graph that is not in a 2-cycle. In addition, we have a pair of actions a and b that have different prevail conditions in c_1 and different effects in c_2, such that a supports b in c_2.

Since actions a and b are mutex (action b deletes a post-condition of action a) they cannot be executed in parallel. Therefore, in a solution plan we must have that either a is executed before b, or that b is executed before a. If the solution plan executes a before b, then the network flow problem corresponding to state variable c_1 must have flow out of f_1. On the other hand, if b is executed before a, then the network flow problem corresponding to state variable c_2 must have flow out of g_3. These flow conditions may seem rather obvious, they are ignored by the action selection formulation. Therefore, we add the following constraints to our formulation to ensure that the flow conditions with respect to the domain structure given in Figure 4 are satisfied.

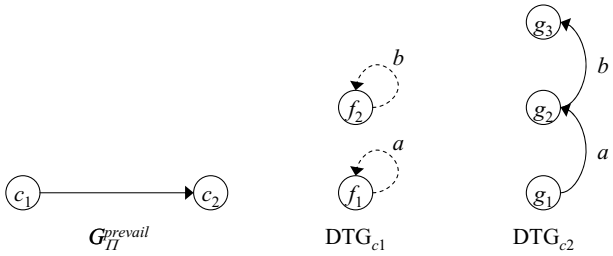

Fig. 4. Domain structure for type 2 domain structure constraints

– Type 2 domain constraints for all $c_1, c_2 \in C$ such that $(c_1, c_2) \in E^{prevail}$, $(c_2, c_1) \notin E^{prevail}$ and $f_1, f_2 \in V_{c_1}$, $g_1, g_2, g_3 \in V_{c_2}$, $e_1 \in V^+_{c_2}(g_2)$, $e_2 \in V^-_{c_2}(g_2)$: $a \in A^E_{c_2}(e_1)$, $b \in A^V_{c_2}(e_2)$, $g_3 = head(e_2)$

$$x_a + x_b - 1 \leq \sum_{e \in V^-_{c_1}(f_1) : a' \in A^E_c(e)} x_{a'} + \sum_{e \in V^-_{c_2}(g_3) : b' \in A^E_c(e)} x_{b'} \qquad (6)$$

3 Experimental Results

In this section, we give a general idea of the distance estimates that both admissible and inadmissible heuristics provide on a set of planning benchmarks from the international planning competitions (IPCs). In particular, we will compare the results of our action selection formulation (with and without the domain structure constraints) with four distance estimates: (1) the admissible distance estimate given by a step based formulation that is very similar to Lplan [9], (2) the admissible distance estimate h^+, which represents the length of the optimal relaxed plan in which the delete effects of the actions are ignored [14], (3) the inadmissible distance estimate h^{FF}, which represents the relaxed plan heuristic of the FF planner [15], and (4) the optimal distance estimate given by Satplanner [19] using the -opt flag.

We use Logistics and Freecell from IPC2, Driverlog and Zenotravel from IPC3, and TPP from IPC5. In addition, we included a few results on well known Blocksworld problems. We focus on these domains mainly because we assume that all pre-conditions are defined. There are several planning domains where this assumption does not hold which limits our experimentation. In Section 5, however, we briefly discuss how we can relax this assumption. All our tests were performed on a 2.67GHz Linux machine with 1GB of memory using a 15 minute timeout. The heuristics that use linear programming were solved using ILOG CPLEX 10.1 [11], a commercial LP/IP solver.

Table 1 summarizes the results. The results represent the distance estimate in terms of the number of actions from the initial state to the goals. LP and LP$^-$ shows the results of our action selection formulation with and without the domain structure constraints respectively. Lplan shows the results of a formulation that is very similar Lplan (the actual planner is not publicly available). They were

Table 1. Distance estimates from the initial state to the goal (values shown shown in bold equal the optimal distance). A dash '-' indicates a timeout of 15 minutes, and a star '*' indicates that the value was rounded to the nearest decimal.

Problem	LP	LP$^-$	Lplan	h^+	h^{FF}	Optimal
logistics4-0	**20**	16.0*	17	19	19	20
logistics4-1	**19**	14.0*	15	17	17	19
logistics4-2	**15**	10.0*	11	13	13	15
logistics5-1	**17**	12.0*	13	15	15	17
logistics5-2	**8**	6.0*	7	**8**	**8**	8
logistics6-1	**14**	10.0*	11	13	13	14
logistics6-9	**24**	18.0*	19	21	21	24
logistics12-0	42	32.0*	33	39	39	-
logistics15-1	67	54.0*	-	63	66	-
freecell2-1	**9**	**9**	**9**	**9**	**9**	9
freecell2-2	**8**	**8**	**8**	**8**	**8**	8
freecell2-3	**8**	**8**	**8**	**8**	9	8
freecell2-4	**8**	**8**	**8**	**8**	9	8
freecell2-5	**9**	**9**	**9**	**9**	**9**	9
freecell3-5	12	12	13	13	14	-
freecell13-3	55	55	-	-	95	-
freecell13-4	54	54	-	-	94	-
freecell13-5	52	52	-	-	94	-
driverlog1	**7**	3.0*	**7**	6	8	7
driverlog2	**19**	12.0*	13	14	15	19
driverlog3	11	8.0*	9	11	11	12
driverlog4	15.5*	11.0*	12	12	15	16
driverlog6	**11**	8.0*	9	10	10	11
driverlog7	**13**	11.0*	12	12	15	13
driverlog13	24	15.0*	16	21	26	-
driverlog19	96.6*	60.0*	-	89	93	-
driverlog20	89.5*	60.0*	-	84	106	-
zenotravel1	**1**	**1**	**1**	**1**	**1**	1
zenotravel2	**6**	3.0*	5	4	4	6
zenotravel3	**6**	4.0*	5	5	5	6
zenotravel4	**8**	5.0*	6	6	6	8
zenotravel5	**11**	8.0*	9	**11**	**11**	11
zenotravel6	**11**	8.0*	9	11	13	11
zenotravel13	24	18.0*	19	23	23	-
zenotravel19	66.2*	46.0*	-	62	63	-
zenotravel20	68.3*	50.0*	-	-	69	-
tpp01	**5**	3.0*	**5**	4	4	5
tpp02	**8**	6.0*	7	7	7	8
tpp03	**11**	9.0*	10	10	10	11
tpp04	**14**	12.0*	13	13	13	14
tpp05	**19**	15.0*	17	17	17	19
tpp06	25	21.0*	23	21	21	-
tpp28	-	150.0*	-	-	88	-
tpp29	-	-	-	-	104	-
tpp30	-	174.0*	-	-	101	-
bw-sussman	4	4	**6**	5	5	6
bw-12step	4	4	8	4	7	12
bw-large-a	**12**	**12**	**12**	**12**	**12**	12
bw-large-b	16	16	**18**	16	16	18

obtained by running a step-based encoding that finds minimum length plans. The values in this column represent the plan length at which the LP-relaxation returns a feasible solution. h^+ shows the length of the optimal relaxed plan and h^{FF} shows the length of FF's extracted relaxed plan. Finally, Optimal shows the results of Satplanner using the -opt flag, which returns the minimum length plan. Note that, Satplanner does not solve a relaxation, but the actual planning problem, so the values in this column represent the optimal distance estimate.

When comparing the results of LP with Lplan and h^+, we see that in many problem instances LP provides better distance estimates. However, there are a few instances in the Freecell and Blocksworld domains in which both Lplan and h^+ provide better estimates. The action selection formulation does clearly outperform both Lplan and h^+ in terms of scalability and time to solve the LP-relaxation. Lplan, generally takes the most time to solve each problem instance as it spends time finding a solution on a plan length for which no feasible solution exists. Both Lplan and h^+ fail to solve several of the large instances within the 15 minutes timeout. The LP-relaxation of the action selection formulation typically solves all small and some medium sized problem instances in less than one second, but the largest problem instances take several minutes to solve and on the largest TPP problem instances it times out at 15 minutes.

When we compare the results of LP with h^{FF} we are comparing the difference between an admissible and an inadmissible heuristic. The heuristic computation of FF's relaxed plan is very fast as it solves most problem instances in a fraction of a second. However, the distance estimate it provides is not admissible as it can overestimate the minimum distance between two states (see for example, driverlog7 and zenotravel6). The results that our action selection formulation provides are admissible and thus can be used in an optimal search algorithm. Moreover, in some problem instances the quality of our distance estimate is outstanding. For example, in the Logistics, Driverlog, and Zenotravel domains, the distance estimate given by LP equals the optimal distance in all problem instances for which Satplanner found the optimal solution.

Finally, when comparing the results of LP with LP^- we see that the domain structure constraints help improve the value of the LP-relaxation in many problem instances except in instances in the Freecell and Blocksworld domains. Both these domains seem to have domain structure that we have not captured yet in our constraints. While this may seem like a problem, we rather take this as a challenge, as we believe that more domain structure can be exploited.

4 Related Work

Admissible heuristics for optimal planning (such as, minimize the number of actions in the solution plan or minimize the cost of the actions in the solution plan) are very scarce and often provide poor distance estimates. On the other hand, inadmissible heuristics are plentiful and have shown to be very effective in solving automated planning problems. HSPr* [12] is one of the few approaches that describes admissible heuristics for planning. HSPr* creates an appropriately

defined shortest-path problem to estimate the distance between two states. Our work differs from HSPr* as we use linear programming to solve a relaxation of the original planning problem.

The use of linear programming as an admissible heuristic for optimal planning was introduced by the Lplan planning system [9]. However, the idea was never incorporated in other planning systems due to poor performance results. Lplan sets up an IP formulation for step-based planning. Thus, when the LP-relaxation of the IP has a solution for plan length l, but not for plan length $l-1$, then the minimum length plan must be at least l steps long. The drawbacks with the LP-relaxation to a step-based encoding is that goal achievement can be accumulated over different plan steps. In general, the quality of the LP-relaxation of an IP formulation depends on how the problem is formulated ([21] describe the importance of developing strong IP formulations in automated planning).

5 Conclusions

We described an integer programming formulation whose LP-relaxation can be used as an admissible heuristic for optimal planning, including planning problems that involve costs and utilities. In fact, in ongoing work we have successfully incorporated our LP-based heuristic in a search algorithm that solves oversubscription planning problems [2].

Our action selection formulation and the heuristic it provides differs in two ways from other formulations that have been used in planning: (1) we do not use a step based encoding, and so, do not have to deal with a bound on the plan length in the IP formulation, and (2) we ignore action ordering, which provides a rather different view on relaxed planning than the more popular approach that ignores the delete effects of the actions.

The experimental results show that the action selection formulation oftentimes provides better distance estimates than a step-based encoding that is similar to Lplan [9]. It outperforms this step-encoding with respect to scalability and solution time, making it a viable distance estimate for a heuristic state-space planner. Moreover, in most problem instances it outperforms h^+ [14], which provides the optimal relaxed plan length when delete effects are ignored. Hence, the relaxation based on ignoring action orderings seems to be stronger than the relaxation based on ignoring delete effects.

Unlike most admissible heuristics that have been described in the planning literature, we can use the action selection formulation to provide an admissible distance estimate for various optimization problems in planning, including but not limited to, minimizing the number of actions, minimizing the cost of actions, maximizing the number of goals, and maximizing the goal utilities. There are several interesting directions that we like to explore in future work.

First, we would like to relax the assumption that all preconditions are defined. This would allow us to create a general action selection formulation and tackle a much broader range of planning domains. We simply need to replace the current action variables with variables that represent the action effects the action prevail

conditions. In case an action has one or more undefined pre-conditions we create one effect variable for each possible value that the pre-condition may take and introduce an extra prevail variable as well. We have a preliminary implementation of this general action selection formulation [2], but have not yet extended it with the domain structure constraints.

Second, it would be interesting to analyze planning domains more carefully and see if there are more domain structures that we can exploit. We already have encountered two domains, namely Freecell and Blocksworld, that may suggest that other domain structures could be discovered.

Acknowledgements. This research is supported in part by the NSF grant IIS308139, the ONR grant N000140610058, and by the Lockheed Martin subcontract TT0687680 to Arizona State University as part of the DARPA integrated learning program.

References

1. Bäckström, C., Nebel, B.: Complexity results for SAS+ planning. Computational Intelligence 11(4), 625–655 (1995)
2. Benton, J., van den Briel, M.H.L., Kambhampati, S.: A hybrid linear programming and relaxed plan heuristic for partial satisfaction planning problems. In: Proceedings of the 17th International Conference on Automated Planning and Scheduling (to appear 2007)
3. Blum, A., Furst, M.: Fast planning through planning graph analysis. In: Proceedings of the 14th International Joint Conference on Artificial Inteligence, pp. 1636–1642 (1995)
4. Bonet, B., Loerincs, G., Geffner, H.: A fast and robust action selection mechanism for planning. In: Proceedings of the 14th National Conference on Artificial Intelligence, pp. 714–719 (1997)
5. Bonet, B., Geffner, H.: Planning as heuristic search. Aritificial Intelligence 129(1), 5–33 (2001)
6. Botea, A., Müller, M., Schaeffer, J.: Fast planning with iterative macros. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 1828–1833 (2007)
7. van den Briel, M.H.L., Kambhampati, S., Vossen, T.: Reviving integer programming Approaches for AI planning: A branch-and-cut framework. In: Proceedings of the 15th International Conference on Automated Planning and Scheduling, pp. 310–319 (2005)
8. Bylander, T.: The computational complexity of propositional STRIPS planning. Artificial Intelligence 26(1-2), 165–204 (1995)
9. Bylander, T.: A linear programming heuristic for optimal planning. In: Proceedings of the 14th National Conference on Artificial Intelligence, pp. 694–699 (1997)
10. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Kluwer Academic Publishers, Dordrecht (1999)
11. ILOG Inc.: ILOG CPLEX 8.0 user's manual. Mountain View, CA (2002)
12. Haslum, P., Geffner, H.: Admissible heuristics for optimal planning. In: Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (2000)

13. Helmert, M.: The Fast Downward planning system. Journal of Artificial Intelligence Research 26, 191–246 (2006)
14. Hoffmann, J.: Where "ignoring delete lists" works: Local search topology in planning benchmarks. Journal of Artificial Intelligence Research 24, 685–758 (2005)
15. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research 14, 253–302 (2001)
16. Jonsson, P., Bäckström, C.: Tractable plan existence does not imply tractable plan generation. Annals of Mathematics and Artificial Intelligence 22(3), 281–296 (1998)
17. McDermott, D.: A heuristic estimator for means-ends analysis in planning. In: Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems, pp. 142–149 (1996)
18. McDermott, D.: Using regression-match graphs to control search in planning. Artificial Intelligence 109(1-2), 111–159 (1999)
19. Rintanen, J., Heljanko, K., Niemelä, I.: Planning as satisfiability: parallel plans and algorithms for plan search. Albert-Ludwigs-Universität Freiburg, Institut für Informatik, Technical report 216 (2005)
20. Vidal, V.: A lookahead strategy for heuristic search planning. In: Proceedings of the 14th International Conference on Automated Planning and Scheduling, pp. 150–159 (2004)
21. Vossen, T., Ball, B., Lotem, A., Nau, D.S.: On the use of integer programming models in AI planning. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, pp. 304–309 (1999)
22. Williams, B.C., Nayak, P.P.: A reactive planner for a model-based executive. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence, pp. 1178–1195 (1997)