Incremental Lower Bounds for Additive Cost Planning Problems

Patrik Haslum
Australian National University & NICTA Optimisation Research Group
firstname.lastname@anu.edu.au

Abstract

We present a novel method for computing increasing lower bounds on the cost of solving planning problems, based on repeatedly solving and strengthening the delete relaxation of the problem. Strengthening is done by compiling select con junctions into new atoms, similar to the $P^m_\star$ construction. Be cause it does not rely on search in the state space, this method does not suffer some of the weaknesses of admissible search algorithms and therefore is able to prove higher lower bounds for many problems that are too hard for optimal planners to solve, thus narrowing the gap between lower bound and cost of the best known plan, providing better assurances of plan quality.

Introduction

Many potential applications of planning require planners to produce plans of high quality, according to a metric like cost, makespan, net benefit, or other. Even when generating guaranteed optimal plans is not computationally feasible, there is a need to be able to measure, in absolute terms, how good the plans found by non-optimal planners are. Current planning technology does not offer many tools suitable for this purpose. A lower bound function (i.e., an admissible heuristic) gives an absolute assessment, since the optimal cost is known to lie between the lower bound and the cost of the best known plan. But if this gap is too large to give confidence in the quality of the solution, and further search fails to turn up a better plan, there is not much that can be done. What is needed in this situation is an incremental lower bound: a function that can produce increasingly higher admissible estimates, given more time and memory.

Given any admissible heuristic h (including the "blind" heuristic $h = 0$) such an incremental lower bound can be obtained by running A*, IDA*, or any search algorithm that approaches the optimum from below, with h for a limited time, taking the highest proven f-value. However, recent theoretical and empirical analysis (Helmert and Röger 2008; Cushing, Benton, and Kambhampati 2010; Wilt and Ruml 2011) has demonstrated that for many kinds of planning problems using search (with a less-than-perfect admissible heuristic) to prove lower bounds is not effective.

Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

We present a new approach to incremental lower bound computation, by repeatedly finding optimal solutions to relaxations of the planning problem, which are increasingly less and less relaxed. If a relaxed plan is found to be a valid plan also for the original, unrelaxed, problem, it is optimal. If not, clues from the failure of the relaxed plan are used to construct the next relaxation, in such a way that the same relaxed plan will not be found again. The relaxation considered is the standard delete relaxation, in which negative effects of actions are ignored. The delete relaxation has the property that a plan achieving atoms p and q also achieves the conjunction $\{p, q\}$. The way in which it is made less relaxed is by constructing a modified problem, in which certain conjunctions are made explicit, in the form of new atoms. This carries information about negative interactions between subgoals (achievement of conjuncts) into the delete relaxation of the modified problem. The conjunctions to encode are chosen by analysing the failure of the current relaxed plan to be valid for the real problem. This incremental lower bound function is tentatively named h^{++}.

We compare the lower bounds proven by h^{++} (within bounded time and memory) with those by A* search using a variety of admissible heuristics, including the optimal delete relaxation heuristic h^+, computed by the same method as in h^{++}, and identify some problem sets where h^{++} is clearly more effective. Although primarily intended as a method of computing lower bounds, h^{++} can also be used as a cost-optimal planner, since when it finds a plan that plan is guaranteed to be optimal. In most cases, even most of those where h^{++} proves stronger lower bounds, search is still better at finding optimal plans, but there are a few exceptions.

Related Work

The idea of incrementally refining relaxations is by no means new. It is widely used in optimisation algorithms, for example in the form of incremental generation of valid cuts in integer-linear programming (e.g. Cornuéjols 2008). An instance that is closer to planning is *counterexample-guided abstraction refinement*, in which the relaxation is an abstraction of the problem. An optimal solution (plan) for the abstract problem is checked for failures w.r.t. the parts of the original problem ignored by the current abstraction, and those parts that cause it to fail are candidates for inclusion in the next abstraction. This idea has been applied in model checking, and to planning in adversarial, probabilistic domains (Chatterjee et al. 2005), but not to lower-bounding plan cost. The bisimilar merge-and-shrink construction, described by Nissim et al. (2011), computes a perfect abstraction by iteratively merging together single-variable projections. Although this abstraction is typically too big to be built, every intermediate abstraction created in the merge-and-shrink process also yields a lower bound. It is, however, not an incremental refinement method, since the choice of the next variable to merge is not informed by flaws in the current abstraction. Yet, we can imagine incremental refinement schemes based on merge-and-shrink, either by making the merge strategy counterexample-guided, or by iterating the construction, starting with a more aggressive shrinking strategy (e.g., the greedy bisimulation proposed by Nissim et al.) and refining the shrinking strategy for the next iteration based on the failures of the current abstract plan.

The most closely related example of the idea is perhaps in the work of van den Briel et al. (2007), who formulate a relaxation of planning problems as an integer programming model of a flow problem. (The problem is further simplified by solving the LP relaxation of the IP.) It is a relaxation because certain ordering constraints, due to non-deleted action preconditions, are ignored. They use composition of state variables to refine the relaxation, though not incrementally and not guided by the current relaxations flaws.[1]

The Delete Relaxation

We adopt the standard definition of a propositional STRIPS planning problem, without negation in action preconditions or the goal (see, e.g., Ghallab, Nau, and Traverso 2004, chapter 2). To simplify the presentation, we assume that $\mathrm{del}(a) \cap \mathrm{add}(a) = \mathrm{pre}(a) \cap \mathrm{add}(a) = \emptyset$ for each action a. (This makes no difference for sequential planning.) As usual, a sequence of actions (or *plan*) *achieves* condition c from state s iff the sequence is executable in s and leads to a state where c holds. We consider the additive cost objective, where each action a has a non-negative cost, $\mathrm{cost}(a)$, and the cost of a plan is the sum of the cost of actions in it. The initial state is denoted by s_I and the goal by G.

The delete relaxation of a planning problem P, denoted P^+, is a problem exactly like P except that $\mathrm{del}(a) = \emptyset$ for each a, i.e., no action makes any atom false. The delete relaxation heuristic, $h^+(s, c)$, is defined as the minimum cost of any plan achieving c from s in the delete relaxed problem.

Let A be any set of actions in P. We denote by $R^+(A)$ the set of all atoms that are reachable in P^+, starting from the initial state, using only actions in A. We say a set of actions A is a *relaxed plan* iff the goal is relaxed reachable using only the actions in A, i.e., iff $G \subseteq R^+(A)$. An actual plan for the delete relaxed problem is of course a sequence of actions. That $G \subseteq R^+(A)$ means there is at least one sequencing of the the actions in A that reaches a goal state. When we need to distinguish a particular sequence of actions, that is valid a plan for P^+, we will call it a *sequenced relaxed plan*, or a *(relaxed) valid sequencing of A*.

We assume throughout that $G \subseteq R^+(A)$ for some set of actions A (which may be the set of all actions in P^+), i.e., that the goal is relaxed reachable. If not, the problem is unsolvable and ∞ is the highest possible lower bound.

Computing Minimum-Cost Relaxed Plans

The h^{++} lower bound procedure depends on being able to compute a minimum-cost relaxed plan, but not on how this is done. There are several methods: Recent work has shown how cost-optimal planning and search techniques can be specialised to the delete-free case (Pommerening and Helmert 2012; Gefen and Brafman 2012). Robinson et al. (2010) encode cost-optimal relaxed planning as a partial weighted MaxSAT problem. Betz and Helmert (2009) present polynomial-time algorithms for a few domains.

We have used an algorithm based on the correspondance between relaxed plans and disjunctive action landmarks, established by Bonet and Helmert (2010), and inspired by Slaney and Thiébaux's (2001) optimal planning algorithm for the Blocksworld domain. It is described in detail elsewhere in these proceedings (Haslum, Slaney, and Thiébaux 2012). A similar algorithm was recently proposed by Bonet & Castillo (2011). An important advantage of this algorithm is that it maintains at all times a lower bound on h^+, so that even if it is interrupted before finding an optimal relaxed plan, it still gives a lower bound.

The Relaxed Plan Dependency Graph

Although we have defined a relaxed plan as an unordered set of actions, there are some necessary ordering relations between actions in this set. These are captured by the *relaxed plan dependency graph*, defined below. This graph is central to the identification of flaws in a failed relaxed plan.

Let S be relaxed plan, i.e., a set of actions such that $G \subseteq R^+(S)$. We say that S is *non-redundant* if no strict subset of S is a relaxed plan. A relaxed plan can be checked for redundancy by simply testing for each action a in turn whether $G \subseteq R^+(S - \{a\})$, and made non-redundant (though not necessarily in a unique way) by greedily removing actions found to be redundant. The landmark-based algorithm can produce optimal plans containing redundant actions, although naturally only when those actions have a cost of zero. From now on, we assume all relaxed plans are non-redundant.

Definition 1 *Let S be a non-redundant relaxed plan. Construct a graph $G^{\prec}(S)$ with nodes $\{n_a \mid a \in S\} \cup \{n_G\}$, i.e., the nodes of $G^{\prec}(S)$ correspond to actions in S plus an additional node which represents the goal. With some abuse of notation, we write $\mathrm{pre}(n)$ for the precondition of node n, which is $\mathrm{pre}(a)$ for a node n_a and G for node n_G. $G^{\prec}(S)$ has an edge from n_a to n' iff $\mathrm{pre}(n') \not\subseteq R^+(S - \{a\})$. This edge is labelled with $\mathrm{pre}(n') - R^+(S - \{a\})$.*

The relaxed plan dependency graph, $\mathrm{RPDG}(S)$*, is the transitive reduction*[2] *of $G^{\prec}(S)$.*

[1] An incremental refinement method based on this relaxation has recently been developed, and used to create a quite effective cost optimal planner. (Menkes van den Briel, pers. comm.)

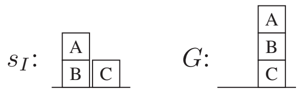
[2] The transitive reduction of a graph is the smallest edge subgraph that has the same transitive closure.

As will be shown shortly, the graph $G^{\prec}(S)$ is acyclic, and therefore its transitive reduction is unique. (It is also computable in polynomial time; cf. Aho, Garey, and Ullman 1972.) Thus, the RPDG is well-defined.

Intuitively, a path from node n to node n' in the RPDG means that the action associated with n is, directly or indirectly, necessary for $\mathrm{pre}(n')$ to be relaxed reachable. The label of an edge documents the reason for the dependency, i.e., which atoms in $\mathrm{pre}(n')$ become unreachable if an action is left out. However, the RPDG does not capture disjunctive dependencies: if several (unordered) actions in S add atom p, there may not be an edge with p in its label, and the fact that at least one of those actions is necessary to reach p will then not be visible in the graph.

If there is a path from node n to node n' in $\mathrm{RPDG}(S)$, we say that the nodes are *ordered*. Conversely, if there is no path from n to n', nor from n' to n, we say they are *unordered*. This terminology is justified by properties (2) and (3) in the theorem below.

Example We illustrate concepts with a small Blocksworld example, with initial state and goal as shown below:



An optimal delete relaxed plan for this problem is (move A B Tb), (move B Tb C). The RPDG is:

$$\begin{tikzcd}
\text{(move A B Tb)} \arrow[r, "\text{(clear B)}"] & \text{(move B Tb C)} \arrow[r, "\text{(on B C)}"] & n_G
\end{tikzcd}$$

Theorem 2
(1) The graph $G^{\prec}(S)$ in definition 1 is acyclic, and hence so is $\mathrm{RPDG}(S)$.
(2) If there is a path from n_a to n_b in $\mathrm{RPDG}(S)$, a appears before b in every valid sequencing of S.
(3) If there is no path from n_b to n_a in $\mathrm{RPDG}(S)$, there exists a valid sequencing of S in which a appears before b.
(4) Any two unordered nodes n and n' in $\mathrm{RPDG}(S)$ have a common descendant, n''.
(5) Let n_a be a node in $\mathrm{RPDG}(S)$ with an outgoing edge whose label includes p, and b an action, distinct from a, with $p \in \mathrm{add}(b)$. Then n_b is ordered after n_a.
Proof (sketch): (1) follows from (2) and non-redundancy.
(2) An edge from n_a to n_b means that $\mathrm{pre}(b) \not\subseteq R^+(S - \{a\})$. Thus, in any relaxed valid sequence b cannot be applicable unless a has been applied before. The transitive case follows by induction on path length.
(3) Let S' be all of S except a, b and their descendants. Apply all actions in S', in any valid order, then a, b, and remaining actions (in any valid order). Suppose there is $p \in \mathrm{pre}(a)$ that does not hold when a is applied. Since p cannot be initially true, there must be at least one action in S with $p \in \mathrm{add}(a)$, that is not in S' and not a descendant of a. Thus, it must be b or a descendant of b, which implies a path from n_b to n_a.
(4) By non-redundancy, every node has a path to n_G.
(5) If n_b is not ordered after n_a, there is a valid sequencing in which b appears before a. Since p is added by b, removing only a from S cannot then make p unreachable, contradicting the existence of the edge. □

Note that property (3) holds only for pairs of nodes. The reason is that the RPDG does not capture disjunctive precedences. For an example, suppose action a_1 adds preconditions of a_2 and a_3, which both add q, and q is a precondition of a_4. (a_2 and a_3 must also add some other relevant atoms, as otherwise at least one of them would be redundant.) The resulting RPDG is:

$$\begin{tikzcd}
& n_2 \arrow[rrd, dashed, bend left] & & \\
n_1 \arrow[ru, "p"] \arrow[rr, "q"] \arrow[rd, "p'"'] & & n_4 \arrow[r, dashed] & n_G \\
& n_3 \arrow[rru, dashed, bend right] & &
\end{tikzcd}$$

There are valid sequencings with a_4 before a_2 (if a_4 is preceded by a_3) and with a_4 before a_3 (if preceded by a_2), but not with a_4 before both a_2 and a_3. Thus, every valid sequencing of S is a topological sort of $\mathrm{RPDG}(S)$, but it is not necessarily the case that every topological sort of $\mathrm{RPDG}(S)$ is a valid sequencing of S. Note also that q labels an edge from n_1, even though a_1 does not add q, because q becomes relaxed unreachable without a_1.

Strengthening the Relaxation

The idea behind the incremental lower bound function h^{++} is as follows: Given a minimal-cost sequenced relaxed plan S, if it is valid also for the non-relaxed problem P, then the optimal plan cost for P equals the cost of S (and S is an optimal plan). If not, then we identify a set of "flaws", C, which are conjunctive conditions that are required but fail to hold at some point in the execution of the plan. We construct a new problem, called P^C, in which these conjunctions are explicitly represented by new atoms. This is essentially the P^m_* construction (Haslum 2009), but applied to a specific set of conjunctive conditions instead of all of size m. The new problem has the property that S is not a relaxed plan for it (or, to be precise, no sequence of representatives of each action in S is a relaxed plan). The minimum cost of a relaxed plan for the new problem is a lower bound also on the cost of solving P. There is no guarantee that it will be higher than $h^+(P)$, but since the set of non-redundant minimal-cost sequenced relaxed plans is finite, repeated application of the procedure will eventually result in a higher lower bound. In the limit it will even result in an optimal plan, unless time or memory limits halt the process.

The P^C Construction

Let $C = \{c_1, \dots, c_n\}$ be a set of (non-unit) conjunctions over the atoms in P. We construct a problem P^C, in which each of these distinguished conjunctions c_i is explicitly represented by a new atom, π_{c_i}. Actions in P^C are also modified, so that the truth of the new atoms reflects the truth of the conjunctions they represent. Each action, $\alpha_{a,X}$, in P^C is derived from an action a from the original problem P and a set $X \subseteq C$. Intuitively, $\alpha_{a,X}$ corresponds to applying a in a state where it will make true the subset X of conjunctions in C (in addition to those it necessarily makes true).

Before stating the definition of P^C, we introduce some notation. Let a be an action in P and partition C into:

$$\begin{aligned}
C^t(a) &= \{c \in C \mid c \subseteq ((\text{pre}(a) - \text{del}(a)) \cup \text{add}(a)) \text{ and } \\
&\qquad c \cap \text{add}(a) \neq \emptyset\} \\
C^f(a) &= \{c \in C \mid c \cap \text{del}(a) \neq \emptyset\} \\
C^n(a) &= \{c \in C \mid c \cap \text{del}(a) = c \cap \text{add}(a) = \emptyset\} \\
C^p(a) &= \{c \in C \mid c \cap \text{del}(a) = \emptyset, c \cap \text{add}(a) \neq \emptyset \text{ and } \\
&\qquad c \not\subseteq ((\text{pre}(a) - \text{del}(a)) \cup \text{add}(a))\}
\end{aligned}$$

(It is easy to verify that this is a partitioning of C.) This divides conjunctions in C into those that are necessarily made true ($C^t(a)$) and false ($C^f(a)$) by a, those on which a has no effect ($C^n(a)$), and those that a may or may not make true, depending on what holds in the state where a is applied ($C^p(a)$). Let $X \subseteq C^p(a)$: we say that X is *downward closed* iff for all $c \in X$ and $c' \in C^p(a)$ such that $c' \subseteq c$, $c' \in X$.

Definition 3 *P^C has all atoms of P, and for each $c \in C$ an atom π_c. π_c is initially true iff c holds in the initial state of P, and is a goal iff $c \subseteq G$ in P. For any set of atoms x in P, let $x^C = x \cup \{\pi_c \mid c \in C, c \subseteq x\}$.*

For each action a in P and for each subset X of $C^p(a)$ that is downward closed, P^C has an action $\alpha_{a,X}$ with

$$\begin{aligned}
\text{pre}(\alpha_{a,X}) &= \left(\text{pre}(a) \cup \bigcup_{c \in X} (c - \text{add}(a))\right)^C \\
\text{add}(\alpha_{a,X}) &= \text{add}(a) \cup \{\pi_c \mid c \in C^t(a) \cup X\} \\
\text{del}(\alpha_{a,X}) &= \text{del}(a) \cup \{\pi_c \mid c \in C^f(a)\}
\end{aligned}$$

and $\text{cost}(\alpha_{a,X}) = \text{cost}(a)$.

We call a the *original action* for $\alpha_{a,X}$, and the actions in P^C whose original action is a the *representatives of* a. Note that P^C always has at least one representative of each action a, namely $\alpha_{a,\emptyset}$. The size of P^C is (potentially) exponential in $|C|$, i.e., the number of conditions, but not in their size.

Theorem 4 *Given any plan for P, there is a plan for P^C made up of a representative of each action in the plan for P (and hence of equal cost).*

Proof (sketch): We choose, by induction, a representative of each action in the plan such that in each state reached when executing the plan for P^C, π_c is true whenever c is true. It is then easy to see that each action in this plan will be executable, since the precondition of an action in P^C includes π_c if and only if it includes all of c, and that the goal condition will hold at the end.

The correspondence between π_c and c holds in the initial state by definition. Assume it holds in state s and consider the state s' that results from applying the next action a in s: choose the representative of a that adds π_c for $c \in C$ that hold in s' but not in s. That such a representative exists follows from the construction of P^C. □

Theorem 4 shows admissibility: any lower bound on the cost of solving P^C is also a lower bound on the cost of any plan for P. Compared to P, P^C contains no additional information. The reason why it is nevertheless useful is that the delete relaxation of P^C may be more informative than the delete relaxation of P. In fact, if C contains a sufficient set of conjunctions, the delete relaxation of P^C captures all information about P.

Theorem 5 *There exists a set of conditions C such that $h^+(P^C) = h^\star(P)$.*

Proof (sketch): Consider the tree of all possible regressions from the goal G in P, and let C include any condition that appears in it. P^C will then have a critical path along the corresponding π_c atoms, meaning that even $h^{\max}(P^C) = h^\star(P)$, and $h^{\max}$ is a lower bound on h^+. □

If we have any additional source of information about unreachability in P, such as static mutexes or invariants, it can be used to reduce the size of P^C, and further strengthen $(P^C)^+$, by removing any action that has in its precondition or add effects an atom π_c for any condition $c \in C$ that is known to be unreachable, in P. (The correspondance shown in theorem 4 holds also with this modification, since the chosen representatives only add atoms for conditions made true by the plan.) The current implementation uses static mutexes found by h^2 (Haslum and Geffner 2000).

Flaw Extraction for a Sequenced Relaxed Plan

Consider a sequenced relaxed plan S. Deciding if it is a valid plan for the real problem, P, is easily done by simulating its execution. If the sequence is not a valid plan, then it will at some point fail, meaning that the precondition of the next action to be applied does not hold in the current state. Call this the *failed condition*, and let n_f be the corresponding node in $\text{RPDG}(S)$. Note that the failed condition may also be the goal. Let p be some unsatisfied atom in the failed condition. Since the sequence is valid in the relaxed sense, p was either true initially or added by some action preceding the point of failure. Thus, p was true in some earlier state also in the real execution of the plan, but deleted by some action taking place between that point and the point of failure. Call that action the *deleter*, and let n_d be its node in $\text{RPDG}(S)$. We call the triplet (n_f, n_d, p) a *conflict*.

A conflict gives rise to a set of *flaws*, C, which are conjunctions of atoms in P. To define the set of flaws, we need the concept of a dependency closure: Let n and n' be nodes in $\text{RPDG}(S)$, where n' is ordered after n. A *simple dependency path* is a path $n \overset{q_1}{\to} n_1 \overset{q_2}{\to} \ldots \overset{q_m}{\to} n'$ from n to n' in $\text{RPDG}(S)$, where each edge is labelled by one atom, chosen arbitrarily, from the edge label in $\text{RPDG}(S)$. (Note that whenever n' is ordered after n, a simple dependency path from n to n' exists.) A *dependency closure* from n to n' is a minimal, w.r.t. subset, union of paths, such that (1) it contains a simple dependency path from n to n', and (2) if q is the atom that labels an edge from a node n'' in the closure, and a is an action with $q \in \text{add}(a)$, where a is not the action associated with n'', then the closure contains a simple dependency path from n to the node corresponding to a. (By property (5) of theorem 2, a is ordered after n'', so such a path exists.)

Now we return to how a set of flaws is derived from a conflict. This depends on how n_d and n_f are related in the

Case 1:
$$\begin{tikzcd}
n_d \arrow[r, "q_1"'] & n_1 \arrow[r, "q_2"'] & \cdots \arrow[r, "q_{m-1}"'] & n_{m-1} \arrow[dr, "q_m"] & \\
& & & & n_f \\
& & & {} \arrow[ur, "p"'] &
\end{tikzcd}$$

Case 2:
$$\begin{tikzcd}
n_d \arrow[r, "q_1"'] & n_1 \arrow[r, "q_2"'] & \cdots \arrow[r, "q_{m-1}"'] & n_{m-1} \arrow[dr, "q_m"] & \\
& & & & n_c \\
{} \arrow[r, "p"'] & n_f \arrow[r, "q'_1"'] & n'_1 \arrow[r, "q'_2"'] & \cdots \arrow[r] & n'_{l-1} \arrow[u, "q'_l"']
\end{tikzcd}$$

Figure 1: Illustration of flaw extraction.

RPDG. We distinguish two cases:

Case 1: There is a path from n_d to n_f in $\mathrm{RPDG}(S)$. In this case, choose a dependency closure from n_d to n_f, and let L be the set of atoms that label edges in the closure. The flaw set is $\{\{p, q\} \mid q \in L\}$.

Case 2: There is no path from n_d to n_f in $\mathrm{RPDG}(S)$. There cannot be a path from n_f to n_d, since n_d appeared before n_f in the sequence, which means the nodes are unordered. Let n_c be the first of their common descendants to appear in the sequence. Choose dependency closures from n_d and n_f to n_c, and let L_d and L_f be the sets of atoms that label edges in each closure. The flaw set is $\{\{q, q'\} \mid q \in L_d, q' \in L_f \cup \{p\}, q \neq q'\}$.

The two cases are illustrated in figure 1. In the picture, each dependency closure consists of only one simple dependency path. This is normally the case: the conditions that force the inclusion of additional branches ("side paths") are quite peculiar, and do not arise in most planning problems. For an example of where it happens, see the RPDG pictured on page 3: there, a dependency closure from n_1 to n_4 must include paths to n_2 and n_3. Note that case 1 may be seen as a special case of case 2, where n_f itself is the first common descendant. (The similarity is apparent in the proof of theorem 6 below.) Atom p cannot label any edge in the dependency closure from n_d or n_f: any action in either closure appears after n_d in the sequence and p, by choice of the conflict, was true before n_d, so removing an action that comes after n_d cannot make p relaxed unreachable. Hence, each flaw also in the first case consists of two distinct atoms.

The reason for constructing the set of flaws C this way is that it is sufficient to ensure that the same sequence of actions (or, more precisely, a sequence of representatives of those actions) is not a relaxed plan for P^C. (We conjecture that it is also necessary.) This ensures progress, in the sense that each iteration of flaw extraction generates some new conjunctions, and adding these to the problem eliminates at least one relaxed plan that is not a real plan.[3]

Example Continuing the Blocksworld example, the relaxed plan fails because (move A B Tb) deletes (on A B) which is initially true and required by the goal. This is a case 1 conflict, with n_G as the failed node, which generates the flaw set $C = \{c_1 : \{\text{(on A B)}, \text{(clear B)}\}, c_2 : \{\text{(on A B)}, \text{(on B C)}\}\}$.

The same plan relaxed will not work for P^C: π_{c_2} is a goal, but the representative of (move B Tb C) that adds π_{c_2} has (on A B), and therefore π_{c_1}, in its precondition. No representative of (move A B Tb) adds π_{c_1} since the action deletes (on A B) which is part of c_1.

Theorem 6 *Let $S = a_1, \ldots, a_n$ be a non-redundant sequenced relaxed plan for P that is not a real plan for P, and let C be a set of flaws extracted as described above. No sequence $S' = \alpha_1, \ldots, \alpha_n$, where each α_i is a representative of a_i, is a relaxed plan for P^C.*

Proof: In the execution of S, we have the two nodes n_d and n_f, as described above. We examine the two cases:

Case 1 (n_f is ordered after n_d): The flaw set consists of pairs of atoms $\{p, q\}$, where p is the precondition of n_f deleted by n_d, and q belongs to a dependency closure from n_d to n_f. We show, by induction, that none of the new atoms $\pi_{\{p,q\}}$ can be made true by any choice of the sequence of representatives S' before the point where n_f appears. Since $\mathrm{pre}(n_f)$ contains both p and an atom q_m from the dependency closure, the precondition of any representative of the action associated with n_f (or the goal, if n_f is the goal node) includes $\pi_{\{p,q_m\}}$, so S' cannot be relaxed valid.

By construction, the dependency closure is a (directed) tree with n_d at the root. This follows from the requirement that the closure is minimal w.r.t. subset of edges: if there are multiple paths from some node n_i to another node n_j, one of those paths can be removed without breaking the closure. For any atom q that labels an edge in the tree, the closure contains (the node corresponding to) any action that adds q. Therefore, any such action has some precondition, q', that labels an edge into the node corresponding to the action, and this edge is also in the closure. The induction is on the depth of an edge in the tree. No representative of the action corresponding to n_d can make $\pi_{\{p,q_1\}}$ true, since the action deletes p. Consider an atom q_i that labels some edge in the closure, and an action a that adds q_i, such that a appears before n_f in the sequenced relaxed plan S. By the choice of n_d, n_f and p, a does not add p. Therefore, any representative of a that makes $\pi_{\{p,q_i\}}$ true must have p as a precondition. By the construction of the closure, a also has some precondition, q, that labels an edge into the node n_a corresponding to a, that is in the closure. (If n_a is node n_{i-1} on the path from n_d to n_f, this will be q_{i-1}. If n_a lies on one of the "side paths", it can be some other atom.) Therefore, any representative of a that makes $\pi_{\{p,q_i\}}$ true must also have as precondition $\pi_{\{p,q\}}$. By inductive assumption, $\pi_{\{p,q\}}$ is not made true by any choice of representatives of the actions that precede a in S.

Case 2 (n_d and n_f are unordered): Let n_c be the first common descendant of n_d and n_f. The precondition of n_c includes atoms q_m and q'_l, where q_m is in the dependency closure from n_d to n_c and q'_l is in the closure from n_f to n_c, and hence any representative of the associated action (or the goal, if n_c is the goal node) includes $\pi_{\{q_m,q'_l\}}$. Similar to case 1, we show that no choice of representatives of the actions preceding n_c in S will make any of the atoms corresponding to the flaws, including $\pi_{\{q_m,q'_l\}}$, true, but the proof

[3]Note that the procedure described here is slightly different from the one described in the equititled 2011 workshop paper, which was incorrect in that it did not ensure this.

now requires a nested induction.

First, note that no action that appears before n_c in S in one closure adds any atom that labels an edge in the other. Suppose n_i is a node in one closure, with an outgoing edge labelled by q_i, and that n_a is node in the other closure, such that a adds q_i. By construction, the closure containing n_i must then also include a simple dependency path to n_a, meaning that n_a is common to both closures. This contradicts the choice of n_c as the first, in S, common descendant of n_d and n_f.

Consider node n_f, and let q be an atom that labels one of its outgoing edges (this may be q'_1, if it is the edge to n'_1, or some other atom if there is a side path from n_f). As argued above, the action associated with n_f does not add any atom q_i belonging to the closure from n_d to n_c, so any representative of this action that makes $\pi_{\{q,q_i\}}$ true must have q_i as a precondition, and therefore also $\pi_{\{p,q_i\}}$. By the choice of the conflict, no action that appears between n_d and n_f in S adds p. Thus, by an induction like that in the proof for case 1, we can show that no choice of representatives of those actions will make $\pi_{\{p,q_i\}}$ true, so no representative of the action associated with n_f that makes $\pi_{\{q,q_i\}}$ true will be relaxed applicable. This establishes the base case. The induction step is again similar to that in the proof of case 1. Consider an atom q'_j that labels some edge in the closure from n_f to n_c, and an action a that adds q'_j (such that a appears before n_c). Any representative of a that makes $\pi_{\{q_i,q'_j\}}$ for some q_i in the closure from n_d to n_c must have q_i as a precondition. Action a also has some precondition, q, that labels an edge into node n_a that is in the closure from n_f to n_c. Thus, any representative of a that makes $\pi_{\{q_i,q'_j\}}$ true must also have $\pi_{\{q_i,q\}}$ as a precondition, which by inductive assumption is not made true by any choice of representatives of the actions that precede a in S. □

Flaw Extraction for Non-sequenced Plans

A non-sequenced relaxed plan is valid for the real problem iff there exists a sequencing of it that is a valid real plan. Enumerating the sequencings of a relaxed plan is straightforward: starting with an empty sequence, non-deterministically choose the next action to apply from those whose preconditions are initially true or added by previously applied actions, never applying an action more than once. Backtracking systematically over the non-deterministic choices yields all sequencings. For the purpose of determining real validity, it is only necessary to construct a (relaxed valid) sequence up to the point where it fails. Note also that since changing the order of commutative actions adjacent in a sequence will not change its relaxed or real validity, it is not necessary to backtrack over those orderings.

Extending the flaw extraction described above to non-sequenced relaxed plans is also simple, in principle: if we pick a conflict from each sequencing of the plan, the union of their sets of flaws is sufficient to rule out the same relaxed plan. In practice, it is more complicated: Each sequencing of the relaxed plan can exhibit several conflicts, each giving rise to a different set of flaws. If the choice of conflict is made carelessly, we can end up with a very large combined set of flaws, resulting in a new problem of unmanageable size. (Recall that the size of P^C may be exponential in $|C|$.) On the other hand, a conflict depends only on the relative order of a few actions in the relaxed plan, and thus often appears in many sequencings, which can all be eliminated by the same set of flaws. However, enumerating every combination of conflicts from each sequencing, and in some cases even just enumerating all sequencings, can be prohibitively computationally expensive. Thus, there is trade-off between the effort spent on analysing the relaxed plan and selecting conflicts, and the size of the resulting set of flaws and new problem.

We use a strategy that balances these objectives by searching for a small flaw set, but (over-)approximating the set of sequencings with the topological sorts of the RPDG. A *potential conflict* is a pair of nodes n_d and n_f, such that the action associated with n_d deletes some part of $\mathrm{pre}(n_f)$ and n_f is not ordered before n_d (by the RPDG). The number of potential conflicts is bounded by the squared size of the relaxed plan. A potential conflict can be avoided by either ordering n_f before n_d or placing between n_d and n_f some action that adds the deleted part of $\mathrm{pre}(n_f)$, if there is any such action in the plan. (Note that these are essentially the classical POCL threat resolutions "demotion" and "white knight".) In either case, the number of ordering constraints increases. To cover all sequencings, we pick a potential conflict, add the flaws it generates to the result, and for each way of avoiding the conflict repeat this with the additional ordering constraints created by avoiding it (taking at the end the union of the results). In each branch, the process ends when we choose a conflict that is unavoidable, i.e., where n_d is ordered before n_f and no "white knight" action can be placed between them. Each conflict is given a weight, equal to the size of the flaw set it generates, which is used as an estimate of how much it will add to the union. (This simplifies the problem by ignoring potential overlap between the flaw sets generated by different conflicts.) We use a backtracking search over the choice of potential conflicts, with bounding and a heuristic ordering, to minimise the total weight of the final flaw set.

Incremental Strengthening

h^{++} iterates finding a relaxed plan, extracting a set of flaws, and adding new atoms representing them to the problem, until the relaxed plan is also a real plan (or we run out of patience or memory). In the process, we construct a sequence of problems, P, P^{C_1}, $(P^{C_1})^{C_2}$, etc. Flaw extraction as described above generates only binary conjunctions, but from the second iteration on, the atoms in such a pair may themselves represent conjunctions, and the pair thus represent a larger conjunction.

Constructing the problem naively according to defintion 3, however, leads to a loss of information, because the fact that conjunctions in $C_1 \cup \ldots \cup C_{k-1}$ are already represented by atoms is not taken into account when building $(\cdots(P^{C_1})^{\cdots})^{C_k}$. To see how, consider the following example: Suppose P contains an action a with $\mathrm{pre}(a) = \emptyset$ and $\mathrm{add}(a) = \{p\}$. Let $C_1 = \{\{p,q\},\{q,r\}\}$:

P^{C_1} will have two representatives of a, $\alpha_{a,\emptyset}$ (identical to a) and $\alpha_{a,\{\{p,q\}\}}$, with $\text{pre}(\alpha_{a,\{\{p,q\}\}}) = \{q\}$ and $\text{add}(\alpha_{a,\{\{p,q\}\}}) = \{p, \pi_{\{p,q\}}\}$. Now let $C_2 = \{\{p,r\}\}$, and construct $(P^{C_1})^{C_2}$. In this problem, $\alpha_{a,\{\{p,q\}\}}$ will have a representative that adds $\pi_{\{p,r\}}$, and its precondition will include atoms q and r both, but not $\pi_{\{q,r\}}$, since $\{q,r\}$ belongs to C_1 and not C_2.

To overcome this problem, we modify the construction slightly, parameterising it with two sets of conjunctions: one set E that is already represented by existing atoms in P, and one set C for which new atoms are to be created. The definition of $P^{E,C}$ is identical to definition 3 except that

$$\text{pre}(\alpha_{a,X}) = \left(\text{pre}(a) \cup \bigcup_{c \in X} (c - \text{add}(a))\right)^{E \cup C},$$

i.e., the atoms for all conjunctions in $E \cup C$ are considered for the preconditions of new action representatives. With this in hand, each problem created by iterating the process of flaw extraction and remedy is constructed as $P^{E_k, C_k - E_k}$, where $E_k = C_1 \cup \ldots \cup C_{k-1}$. In other words, all conjunctions added in previous iterations are used to form action preconditions, while new atoms are added only for the new flaws. It is easy to show that this preserves admissibility:

Theorem 7 *Let $C_1, \ldots, C_k$ be sets of conjunctions, $E_i = C_1 \cup \ldots \cup C_i$, and define the series of problems $P_{i+1} = P_i^{E_i, C_{i+1}}$ (with $P_0 = P$). Given any plan for P, there is a plan for P_k of equal cost.*

Proof (sketch): By induction on i. From a plan for P_i, a plan for P_{i+1} that consists of one representative of each action in the plan for P_i is constructed in exactly the same way as in the proof of theorem 4. □

Evaluation

We compare the lower bounds proven by h^{++} (within 1h CPU time and 3Gb memory limits) with those proven by A*, using the four admissible heuristics that make up the (optimal) FastDownward StoneSoup portfolio planner (Helmert et al. 2011) and the optimal delete relaxation heuristic h^+, computed by the same method as in h^{++}. Note that we do not use the portfolio planner itself: search with each of the four heuristics is run to the full 1h CPU limit, and we take as the result the highest lower bound proven (resp. any solution found) by any one of them.

The comparison is made across problem sets from IPC 1998 2011, the 3-operators version of Blocksworld, and two domains encoding applications of discrete event diagnosis as planning problems (Haslum and Grastien 2011). The Logistics (2000) and Blocksworld problem sets include the larger instances intended for planners with hand-crafted control knowledge. The IPC 2008 and 2011 problem sets are from the sequential satisificing track.

Table 1 summarises the results. For each domain, it shows the number of problems solved (optimally), and the number of problems on which each method proves a lower bound matching (=) or strictly greater than (>) the best of the others. Column "A*" represents the best, per instance, of four

Unit Cost Domains

	#	Solved			Max Lower Bound					
		h^{++}	A*	A*/h^+	h^{++}		A*		A*/h^+	
					=	>	=	>	=	>
Airport	50	24	27	25	50	13	35	0	37	0
Blocksworld										
3 ops	101	14	27	20	70	15	55	29	60	2
4 ops	101	8	28	25	8	8	101	93	26	0
Depots	22	1	7	5	17	11	9	5	7	0
Driverlog	20	6	14	8	9	2	18	11	8	0
Freecell	60	5	45	6	29	0	60	31	14	0
Gripper	20	1	19	4	1	0	20	19	4	0
Logistics ('00)	80	15	20	15	52	32	47	28	16	0
Logistics ('98)	35	4	6	4	17	8	25	18	7	0
Miconic	150	17	142	58	144	0	150	6	121	0
MPrime	35	21	19	17	34	5	24	1	25	0
Mystery	30	13	17	12	19	2	28	11	14	0
Pathways	30	4	5	4	30	21	5	0	8	0
PSR	50	22	50	48	23	0	50	27	48	0
Rovers ('06)	40	4	8	5	19	10	29	21	8	0
Satellite ('04)	36	3	7	5	8	0	35	28	6	0
Schedule	500	53	48	40	488	376	119	12	69	0
Storage	30	6	15	12	7	0	30	23	13	0
TPP	30	6	7	6	23	13	15	5	9	2

Action Cost Domains

	#	Solved			Max Lower Bound					
		h^{++}	A*	A*/h^+	h^{++}		A*		A*/h^+	
					=	>	=	>	=	>
Barman	20	0	0	0	9	0	13	11	8	0
Cybersec	30	3	4	3	28	9	19	1	18	0
Elevators	30	0	2	2	24	19	4	1	9	3
Floortile	20	0	6	0	0	0	20	20	0	0
Openstacks	30	0	12	8	0	0	30	22	8	0
ParcPrinter	30	27	18	20	30	8	18	0	20	0
PegSol	30	3	29	26	4	0	30	4	26	0
Scanalyzer	30	6	17	5	21	6	20	9	10	0
Sokoban	30	1	26	8	2	0	30	22	8	0
Transport	30	3	6	3	5	2	28	25	3	0
Visitall	20	0	0	0	5	1	19	15	4	0
Woodworking	30	9	7	8	25	15	12	4	10	1
APPN	35	33	13	20	35	12	22	0	23	0
WITAS	196	22	132	95	55	0	185	55	141	11

Table 1: Summary of results.

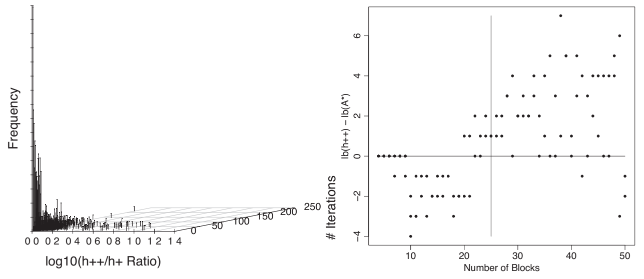


Figure 2: Left: Distribution of h^{++}/h^+ ratio to h^{++} iterations, over problems where at least one iteration completed. Right: 3-ops Blocksworld problems, size vs. relative strength of h^{++} and A* lower bounds. Lines show axis medians.

A* runs with different heuristics, and column "$\mathrm{A}^\star/h^+$" is the result for A* search using the h^+ heuristic. Where the computation of a first relaxed plan did not finish within the time limit, the result of h^{++} and $\mathrm{A}^\star/h^+$ is the highest lower bound proven by the h^+ solver.

For finding plans, A* search is clearly better most of the time, with ParcPrinter the only stand-out exception. However, at proving lower bounds on problems that cannot be solved optimally by any method, h^{++} is better in many cases. The relative strength of lower bounds computed by h^{++} and A* search for a given problem depends on several factors: the discrepancy between the optimal delete relaxed and real plans, and how much of that discrepancy is removed by each flaw repair; the efficiency of the h^+ solver; and how easy or hard the problem is for A* search. The part that each of these factors plays varies between domains, but also with characteristics of the problem instances.

Comparing the results of A* and $\mathrm{A}^\star/h^+$ in the problem domains where h^{++} outperforms A* shows that this is not because the four admissible heuristics used are too weak. (The LM-Cut heuristic is often very close to h^+, averaging only 3.5% relative error across the initial states of unit-cost problems where both are known.) Thus, it is using search to raise the lower bound above that given by the initial heuristic value that is not effective on these problems.

Across all problems, there is a statistically significant positive correlation between the number of iterations (after the first) and the improvement that h^{++} makes over h^+ (66% of problems with a higher-than-median number of iterations also show an above-median h^{++}/h^+ ratio), even slightly stronger (68%) across problems not solved by h^{++}. However, there is no easily discernible relationship between the two (see left half of figure 2). It also varies greatly between domains, since the number of iterations required to raise the lower bound grows with the number of distinct minimum-cost relaxed plans. A problem feature that can lead to many alternative relaxed plans is functionally equivalent objects (e.g., multiple vehicles with the same roadmap in transportation problems). This partly explains the performance difference between the Logistics 1998 and 2000 problem sets, which scale up differently in this respect, though the negative impact this has on the h^+ solver plays a bigger part. There is also a positive correlation between iterations and the difference between h^{++} and A* lower bounds over problems without zero-cost actions, but which does not hold across the set of problems with zero-cost actions.

Since an optimal relaxed plan is computed in each iteration, the time consumed by the h^+ solver strongly influences the effectiveness of h^{++}: in 93.8% of instances where h^{++} does not find a plan, it runs out of time computing relaxed plans; only in 6.2% it runs out of memory by generating too large flaw sets. The time for flaw extraction never exceeds a second.

The right half of figure 2 plots the difference between lower bounds by h^{++} and A* against the number of blocks in Blocksworld (3-ops) problems. Here, there is a clear positive correlation, i.e., the larger the problem, the greater the advantage of h^{++}. A similar correlation with the number of (semi-independent) goals can be found in, e.g. the Logistics and Elevators domains.

Conclusions

Strong lower bounds are essential for proving the quality of plans, and thus for inspiring confidence in planners, which is important for the acceptance of automated planning in many applications. Cushing et al. (2010) observe that finding plans and proving their optimality (or near-optimality) are fundamentally different problems. The logical conclusion is that they should be approached by different methods. Search is often a very effective means for finding plans, but not always as effective at proving lower bounds on their cost.

Solving increasingly less and less relaxed problem relaxations is an attractive approach to constructing incremental lower bound functions, but has not been much explored in planning. We presented one realisation of this idea, based on the common delete relaxation, and showed that it is in many cases able to prove stronger lower bounds than A* search, even A* search using h^+, the optimal cost of the delete relaxed problem, as the heuristic.

To characterise more precisely the effectiveness of h^{++} as a means of computing lower bounds, compared to that of A* search, is difficult, since both are influenced by many factors. A conclusion from the experiments is that it is intimately linked with the efficiency of the method used to compute relaxed plans. The h^+ solver we have used is particularly weak in some domains. Fortunately, there now exist several other methods for optimal delete relaxed planning (Pommerening and Helmert 2012; Gefen and Brafman 2012), and any one of them can be used within h^{++}.

In fact, the process of iterative flaw extraction and remedy does not even depend on the relaxed plan having minimal cost; that serves only to ensure that the cost of the relaxed plan is a lower bound on real plan cost. Thus, the same approach may conceivably be taken also to deriving an incremental lower bound function for a different cost measure, such as makespan. Interestingly, finding a relaxed plan with minimum makespan is tractable (as it is given by $h^{\max}$).

The ultimate aim is of course to produce plans of assured high quality. In many cases, even with the higher lower bounds we obtain with h^{++}, the remaining gap to the best known plan remains large. Thus, an important question is how to make effective use of an incremental lower bound function like h^{++} in bounded suboptimal search algorithms, which normally rely on node expansion to improve lower bounds above initial heuristic estimates (e.g. Thayer and Ruml 2011), and thus suffer the same weaknesses as A* in this respect.

Acknowledgements My thanks to Sylvie Thiébaux, John Slaney, Joerg Hoffmann, Malte Helmert and Avitan Gefen for helpful comments on this paper, and on the workshop paper that preceded it. This work was supported by the Australian Research Council discovery project DP0985532 "Exploiting Structure in AI Planning". NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

Aho, A.; Garey, M.; and Ullman, J. 1972. The transitive reduction of a directed graph. *SIAM Journal on Computing* 1(2):131 137.

Betz, C., and Helmert, M. 2009. Planning with h^+ in theory and practice. In *Proc. of the ICAPS'09 Workshop on Heuristics for Domain-Independent Planning*.

Bonet, B., and Castillo, J. 2011. A complete algorithm for generating landmarks. In *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, 315 318.

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *Proc. 19th European Conference on Artificial Intelligence (ECAI'10)*, 329 334.

Chatterjee, K.; Henzinger, T.; Jhala, R.; and Majumdar, R. 2005. Counterexample-guided planning. In *Proc. 21st International Conference on Uncertainty in Artificial Intelligence (UAI'05)*, 104 111.

Cornuéjols, G. 2008. Valid inequalities for mixed integer linear programs. *Mathematical Programming* 112(1):3 44. `http://dx.doi.org/10.1007/s10107-006-0086-0`.

Cushing, W.; Benton, J.; and Kambhampati, S. 2010. Cost-based search considered harmful. In *Proc. Symposium on Combinatorial Search (SoCS'10)*.

Gefen, A., and Brafman, R. 2012. Pruning methods for optimal delete-free planning. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers. ISBN: 1-55860-856-7.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proc. 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'00)*, 140 149. AAAI Press.

Haslum, P., and Grastien, A. 2011. Diagnosis as planning: Two case studies. In *ICAPS'11 Scheduling and Planning Applications Workshop*.

Haslum, P.; Slaney, J.; and Thiébaux, S. 2012. Minimal landmarks for optimal delete-free planning. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*.

Haslum, P. 2009. $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. In *Proc. of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *Proc. 23nd AAAI Conference on Artificial Intelligence (AAAI'08)*, 944 949.

Helmert, M.; Röger, G.; Seipp, J.; Karpas, E.; Hoffmann, J.; Keyder, E.; Nissim, R.; Richter, S.; and Westphal, M. 2011. Fast downward stone soup (planner abstract). In *7th International Planning Competition (IPC 2011), Deterministic Part*. `http://www.plg.inf.uc3m.es/ipc2011-deterministic`.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *Proc. of the 22nd International Joint Conference on AI (IJCAI'11)*, 1983 1990.

Pommerening, F., and Helmert, M. 2012. Optimal planning for delete-free tasks with incremental LM-cut. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*.

Robinson, N.; Gretton, C.; Pham, D.; and Sattar, A. 2010. Partial weighted MaxSAT for optimal planning. In *Proc. 11th Pacific Rim International Conference on AI (PRICAI'10)*.

Slaney, J., and Thiebaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125. `http://users.cecs.anu.edu.au/~jks/bw.html`.

Thayer, J., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proc. 22nd International Conference on Artificial Intelligence (IJCAI'11)*, 674 679.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In *Proc. 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, 651 665.

Wilt, C., and Ruml, W. 2011. Cost-based heuristic search is sensitive to the ratio of operator costs. In *Proc. Symposium on Combinatorial Search (SoCS'11)*.