MAXSAT Heuristics for Cost Optimal Planning

Lei Zhang *
State Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210093, China
zhanglei.com@gmail.com

Fahiem Bacchus†
Department of Computer Science
University of Toronto, Canada
fbacchus@cs.toronto.edu

Abstract

The cost of an optimal delete relaxed plan, known as h^+, is a powerful admissible heuristic but is in general intractable to compute. In this paper we examine the problem of computing h^+ by encoding it as a MAXSAT problem. We develop a new encoding that utilizes constraint generation to support the computation of a sequence of increasing lower bounds on h^+. We show a close connection between the computations performed by a recent approach for solving MAXSAT and a hitting set approach recently proposed for computing h^+. Using this connection we observe that our MAXSAT computation can be initialized with a set of landmarks computed by LM-cut. By judicious use of MAXSAT solving along with a technique of lazy heuristic evaluation we obtain speedups for finding optimal plans over LM-cut on a number of domains. Our approach enables the exploitation of continued progress in MAXSAT solving, and also makes it possible to consider computing or approximating heuristics that are even more informed that h^+ by, for example, adding some information about deletes back into the encoding.

Introduction

Many search heuristics for classical planning are based on the delete relaxation of the planning problem which is constructed by removing the delete effects of all actions. Among such heuristics the optimal delete relaxation heuristic, h^+, plays an fundamental role. Given a state s, $h^+(s)$ is the cost of an optimal (lowest-cost) delete relaxed plan that can transform s into a state satisfying the goal, where the cost of a plan is the sum of the costs of its actions. $h^+(s)$ is a lower bound on the true cost of achieving the goal from state s, and thus it is an **admissible heuristic** which if used within an A^* search allows us to compute cost optimal plans.

It has been shown that h^+ is a very informative heuristic (Betz and Helmert 2009). Unfortunately, in general h^+ is NP-hard to compute (Bylander 1994). In this paper we investigate computing h^+ by encoding it as a MAXSAT problem, thus allowing us to exploit ongoing advances in the area of MAXSAT solving, e.g., (Davies and Bacchus 2011; Ansótegui, Bonet, and Levy 2010; Heras, Morgado, and Marques-Silva 2011).

Our interest in h^+ is to use it as an admissible heuristic for A^*. Therefore, spending too much time on its computation can be counter productive. That is, the overall A^* search might be faster with a less-informed but cheaper to compute heuristic exploring a larger search space. Hence, our focus in this paper is to exploit techniques from MAXSAT solving to efficiently approximate h^+, thus avoiding spending too much time computing it exactly. In particular, we develop MAXSAT based techniques for computing h^+ in an anytime manner: our methods produce a sequence of successively better lower bounds to h^+ and ultimately, if given enough time, compute h^+ exactly. Since h^+ is admissible all of these lower bounds are also admissible, and we can use the best bound computed within the time available as an admissible A^* heuristic for computing optimal plans.

MAXSAT is the optimization version of SAT and since SAT has exhibited success as a technique for computing satisfying plans (Kautz and Selman 1996) it is natural to use MAXSAT directly to compute optimal plans. One approach to doing this has been described in (Robinson et al. 2010). However, this direct use of MAXSAT has not, as yet, been as effective as heuristic search (e.g., see the results of the 2008 International Planning Competition `http://ipc.icaps-conference.org`). The main difficulty with the direct approach is that the MAXSAT encoding of the original (unrelaxed) planning problem can become very large (a problem shared by the SAT approach to satisfying planning). In contrast in this work we utilize the fact that computing h^+ only requires computing optimal *delete relaxed* plans. Relaxed plans have a much simpler structure than real plans, and can be solved with a much simpler and more compact MAXSAT encoding.

This paper contain three main contributions. First, we develop a new MAXSAT encoding for computing optimal relaxed plans (from which h^+ can be computed by summing the optimal plan's action costs). The innovation in this encoding lies in its use of the technique of *constraint generation*: solutions to the initial encoding need not be legal relaxed plans as the initial encoding does not represent all of the constraints required by relaxed plans. As a result the initial encoding is particularly simple and quite compact. Once we obtain a solution to the initial encoding we can check if

*Funded by NNSF of China, Jiangsu Province, and Nanjing University.

†Funded by NSERC of Canada.

the solution represents a legal relaxed plan. If it does we can compute h^+ from it exactly. If it does not we can compute a new constraint that when added to the encoding will block this and similar illegal plans. We can then solve the problem again with this new constraint added, iterating this process until we have computed an optimal legal relaxed plan from which h^+ can be calculated. Importantly, this encoding yields a sequence of lower bounds to h^+: the solution to the current set of constraints produced at each iteration is always a lower bound on h^+ and the process converges to an exact computation of h^+.

Second, we explain a close connection between a recent approach to MAXSAT solving (Davies and Bacchus 2011) and a recently developed connection between action landmarks and h^+ (Bonet and Helmert 2010). This connection allows us to show how landmarks computed by the LM-Cut heuristic (Helmert and Domshlak 2009) can be used to seed our MAXSAT computation, ensuring that the MAXSAT computed heuristic can only improve on LM-Cut. The connection also illustrates that MAXSAT techniques can be used to generalize the methods of (Bonet and Castillo 2011; Bonet and Helmert 2010) so that landmarks for non-delete relaxed planning problems can be computed and perhaps used to compute admissible heuristics that are more powerful than h^+.

Third, we describe a technique of *lazy heuristic evaluation* that can be used to more efficiently exploit a better but more expensive to compute heuristic in A^*. Lazy heuristic evaluation is a technique where by we do not have to incur the cost of computing the more expensive heuristic at every node generated by A^* and at the same time expand no more nodes than if we had computed the heuristic at every node.

Finally, we present empirical results illustrating that reasonable performance when solving cost optimal planning problems can be obtained from our approach.

Background

Planning and Delete Relaxations. A planning problem $\Pi = \langle P, I, G, A, aCost\rangle$ consists of a set of facts P, an initial state $I \subseteq P$, a goal state $G \subseteq P$, and a set of actions A. Each action $a \in A$ is specified by the sets $\langle pre(a), add(a), del(a)\rangle$, where $pre(a)$ are the preconditions of a, $add(a)$ are the add (positive) effects of a, and $del(a)$ are the delete (negative) effects of a. Each of these sets is a subset of P. $aCost$ is a cost function that assigns each action in A a positive real valued cost bounded away from zero. The delete relaxation Π^+ of Π is the same problem but for every $a \in A$ we set $del(a) = \emptyset$. States are subsets of P.

An action a is applicable in state s if $pre(a) \subseteq s$. The result of applying action a to state s is $result(s, a) = (s \cup add(a)) \setminus del(a)$. The result of applying an action sequence to state s is defined recursively as: $result(s, \langle a_1, \cdots, a_n\rangle) = result(result(s, \langle a_1 \cdots, a_{n-1}\rangle), a_n)$. The sequence is said to be executable if each a_i is applicable, i.e., $pre(a_i) \subseteq result(s, \langle a_1, \ldots, a_{i-1}\rangle)$. A valid plan π for a state $s \subseteq P$ is an executable sequence of actions such that $G \subset result(s, \pi)$. The cost of π, $aCost(\pi)$, is the sum of the cost of its actions. The cost of a minimum cost of plan for the initial state I in the relaxed planning problem Π^+ is denoted by $h^+(\Pi^+)$.

MAXSAT. A propositional formula in CNF is a conjunction of clauses, each of which is a disjunction of literals, each of which is a propositional variable or the negation of a propositional variable. Given a CNF formula a truth assignment ρ is an assignment of true or false to all of the propositional variables in the formula.

A MAXSAT problem is specified by a CNF formula $\mathcal{F}$ along with a positive real valued weight $wt(c)$ for every clause $c \in \mathcal{F}$. Some clauses might be hard clauses, indicated by them having infinite weight. Clauses with finite weight are called soft clauses. We use $hard(\mathcal{F})$ to indicate the hard clauses of $\mathcal{F}$ and $soft(\mathcal{F})$ the soft clauses. Note that $\mathcal{F} = hard(\mathcal{F}) \cup soft(\mathcal{F})$.

We define the function $mCost$ as follows: (a) if H is a set of clauses then $mCost(H)$ is the sum of the clause weights in H; and (b) if ρ is a truth assignment to the variables in $\mathcal{F}$ then $mCost(\rho)$ is the sum of the weights of the clauses falsified by ρ. A MAXSAT solution to $\mathcal{F}$ is a truth assignment ρ to the variables of $\mathcal{F}$ with minimum cost. We denote this cost by $mincost(\mathcal{F})$. A **core** κ for a MAXSAT formula $\mathcal{F}$ is a subset of $soft(\mathcal{F})$ such that $\kappa \cup hard(\mathcal{F})$ is unsatisfiable. Given a set of cores $\mathcal{K}$ a **hitting set**, hs, of $\mathcal{K}$ is a set of soft clauses such that for all $\kappa \in \mathcal{K}$ we have that $hs \cap \kappa \neq \emptyset$.

MAXSAT Encoding

SAT encodings of unrelaxed planning problems must be concerned not only with identifying the actions that form the plan but also with how these actions must be sequenced. Hence these encodings typically use time stamped copies of the fact and action propositions. For example, propositional variables of the form $pf@t$ and $a@t$ are used to indicate that fact $p \in P$ or action $a \in A$ is true at step t of the plan.

Since we are computing h^+ and all actions in Π^+ are delete free, we can take advantage of the fact that delete relaxed plans need never repeat the same action. Thus the encoding need only have a single copy of each action variable. However, for a set of relaxed actions to form an legal (i.e., executable) plan we must ensure that there are no cyclic dependencies. For example, we cannot have that action a_1 depends on a_2 to add a precondition while at the same time a_2 depends on a_1 to add a different precondition.

Cyclic dependencies can be eliminated in various ways. For example, (Robinson et al. 2010) utilize a additional collection of causal propositions $\mathcal{K}(p_i, p_j)$ indicating that fact p_i is cause of p_j. These propositions are true if some action used fact p_i as a precondition and produced p_j as an add effect. $O(n^2)$ additional clauses are then needed to ensure that causation is transitive closed. Although the final encoding is only satisfied by legal relaxed plans, it becomes quite large and is not particularly conducive to the production of incremental bounds.

Another useful feature of relaxed plans is that given a set of delete relaxed actions it is easy to compute how they must be sequenced to form a legal plan. Utilizing this fact we eschew the complexity of ensuring that the encoding is satisfied only by legal plans. Instead we use a much simpler

encoding that can admit illegal plans, and use the approach of constraint generation to eventually converge on a legal relaxed plan. We start with an simple initial encoding that uses only action variables.

Action Based Encoding First, we define a support function sup for each fact, $sup(p)$, to be the set of actions that could add p: $sup(p) = \{a | p \in add(a)\}$. Furthermore, given a state s we define $poss\_acts(s)$ to be the set of acts that are applicable in s: $poss\_acts(s) = \{a | pre(a) \subseteq s\}$

The variables in our encoding are a single action variable a for each relaxed action $a \in A$. The encoding consists of the following clauses.

1. For each goal fact $g \in G$ such that $g \notin I$ (we do not need to achieve a goal that is already achieved in the initial state), some action supporting g must be in the plan. For each goal fact g this yields a **hard** clause of the form

$$\bigvee_{a \in sup(g)} a$$

2. For each action a and for each of its preconditions that are not already true in the initial state, i.e., $p \in pre(a) \setminus I$, if a is included in the plan then one of p's supporters must be included as well. Each such pair (a, p) yields a **hard** clause of the form

$$\neg a \vee \bigvee_{\{a' | a' \in sup(p)\}} a'$$

3. Finally, we have **soft** clauses that invoke a cost of $aCost(a)$ for each action a included in the plan. Each action a yields a soft unit clause c of the form

$$(\neg a)$$

with $mCost(c) = aCost(a)$.

To further reduce the encoding size, we can restrict the actions in the planning problems to be both reachable from initial state I and relevant to goal G. This is easy to compute.

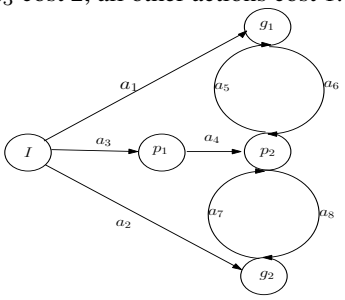
Given a relaxed planning problem Π^+, a MAXSAT solution to its ABE encoding $abe(\Pi^+)$ is a truth assignment ρ that assigns true to a minimum cost set of actions sufficient to satisfy the hard clauses (items 1 and 2). (Each action that is included in the solution invokes an added cost for ρ). The set of actions assigned true by ρ (i.e., the set of actions in the plan) might not form a legal relaxed plan. Nevertheless, we do have the following:

Proposition 1. $mincost(abe(\Pi^+)) \leq h^+(\Pi^+)$.

This holds because all relaxed plans are models of $hard(abe(\Pi^+))$: in any relaxed plans all goals and preconditions of included actions are supported either by the initial state or by other actions. Therefore, the minimum cost relaxed plan is an upper bound on the mincost truth assignment to $abe(\Pi^+)$.

Proposition 2. *If $abe(\Pi^+)$'s solution ρ is a executable relaxed plan, then ρ is an optimal relaxed plan, i.e., $mCost(\rho) = h^+(\Pi^+)$.*

Figure 1: a planning task with cyclic dependencies. a_1,a_2 have cost 3, a_3 cost 2, all other actions cost 1.



Since the returned solution ρ is a legal (executable) relaxed plan and h^+ is the minimum cost of any relaxed plans, we have $mCost(\rho) \geq h^+(\Pi^+)$. By Proposition 1, we have that $mCost(\rho) = h^+(\Pi^+)$.

The returned MAXSAT solution ρ is a truth assignment that makes some of the action variables a true. This set of true action variable specifies the set of actions A in the relaxed plan. By successively finding an action in A that can be executed in I, updating I to account for its add effects, and repeating, we can in time linear in A determine whether A is a valid relaxed plan for Π^+: A is executable if and only if all we can execute all of its actions via this process.

As mentioned above there are models of $abe(\Pi^+)$ that are not legal relaxed plans. These models correspond to action sequences that supply preconditions to each other in a cyclic manner. If the returned solution to $abe(\Pi^+)$, ρ, is one of these models we still have that $mCost(\rho) \leq h^+(\Pi^+)$, but we will not know if $mCost(\rho) = h^+(\Pi^+)$. For example, Fig. 1 shows a planning task where arrows labeled by actions indicate that the action maps a precondition to an effect, and g_1 and g_2 are the goal conditions. The solution returned by the MAXSAT solver is $\{a_5, a_6, a_7, a_8\}$ with cost 4. The optimal relaxed plan is $\{a_3, a_4, a_5, a_6\}$ with cost 5. We call these kinds of solutions inconsistent.

Definition 1. *A solution ρ to $abe(\Pi^+)$ is inconsistent if the actions made true in ρ do not form an executable plan.*

We resolve inconsistent solutions using the technique of *constraint generation*. In particular, if the returned solution is not a legal relaxed plan, we add a new clause to the encoding to block that solution (and others like it) and solve the augmented MAXSAT problem again to find another solution.

The clause we use to block inconsistent solutions is constructed as follows. Let the actions of the inconsistent solution ρ be A. First we compute the set $E(\rho, I)$ of all actions from A that can be sequentially executed from the initial state I. Note that since there are no deletes the order we execute these actions does not affect the set we compute. Let s be the state that arises from executing all of the actions in $E(\rho, I)$: $s = result(I, E(\rho, I))$. We then add a single **hard** inconsistent solution blocking clause specifying that if the plan contains all of the actions $E(\rho, I)$, then it must contain at least one more action applicable in s:

$$\bigvee_{a \in E(\rho, I)} \neg a \vee \bigvee_{a' \in poss\_acts(s)} a'$$

Simply stated, the actions of $E(\rho, I)$ are not sufficient to form a valid plan and so any valid plan containing $E(\rho, I)$ must have at least one more applicable action. It is not difficult to see that this clause must be satisfied by all legal relaxed plans.

For example, in Fig. 1, the executable set $E(\rho, I)$ of solution $\rho = \{a_5, a_6, a_7, a_8\}$ is $\emptyset$, so that $s = I$, and $poss\_acts(s) = \{a_1, a_2, a_3\}$. The blocking clause is hence $(a_1 \vee a_2 \vee a_3)$. After adding this clause, all solutions to the MAXSAT encoding are valid (and of course optimal) relaxed plans.

Alg. 1 shows the loop required to compute optimal relaxed plans using our MAXSAT encoding.

Algorithm 1 Optimal Relaxed Plan Computation

$F = abe(\Pi^+)$
while true do
  $\rho =$ maxsat_solve(F)
  compute $E(\rho, I)$
  $s = result(E(\rho, I), I)$
  if $G \subseteq s$ **then**
    return $E(\rho, I)$
  end if
  $c = \bigvee_{a \in E(\rho,I)} \neg a \vee \bigvee_{a' \in poss\_acts(s)} a'$
  $F = F \cup \{c\}$
end while

Theorem 1. *The relaxed plan π returned by Algorithm 1 is optimal, i.e., $aCost(\pi) = h^+(\Pi^+)$.*

Any π returned is a valid relaxed plan (all of its actions are executable starting at I). Furthermore, all relaxed plans satisfy the initial hard clauses of $F = abe(\Pi^+)$ along with each clause c added to F. If a relaxed plan of smaller cost than π exists it would satisfy all hard clauses of F and would falsify a lower cost set of soft clauses of F. In this case, π would not correspond to a MAXSAT solution of F.

Encoding with State. Although abe does not have any fact variables we can include them. The hard clauses of abe can then be rewritten using fact variables. The yields a new encoding $abe{+}state$ that contains more clauses than the abe encoding, but these clauses are shorter. This can sometimes improve the performance of the MAXSAT solver. The soft clauses of the $abe{+}state$ encoding are identical to the abe encoding (unit clauses incurring a cost for every true action).

MAXSAT Solving, Landmarks, and h^+

An action landmark for a planning problem is a set of actions at least one of which must be included in any plan. Recently a new analysis has shown that computing h^+ can be formulated as a hitting set problem (Bonet and Helmert 2010), where the sets to be hit are action landmarks.

In fact, (Bonet and Helmert 2010), formulated the computation of h^+ as an *implicit hitting set problem* (IHS) (Karp 2010; Chandrasekaran et al. 2011). IHS problems differ from ordinary hitting set problems in that the collection of sets to hit is not provided as input. Rather there is an oracle which when given a candidate hitting set hs either declares that hs hits all implicitly defined sets or returns an new set not hit by hs from the implicit collection of sets. (Bonet and Castillo 2011) showed how the required oracle for h^+ could be constructed generating unhit action landmarks or verifying that all landmarks of the relaxed planning problem have been hit. An advantage of this view of computing h^+ is that any set of landmarks $\mathcal{L}$ provides a lower bound for h^+ and thus an admissible heuristic for A^*. Any relaxed plan must contain contain at least one action from every landmark in $\mathcal{L}$, and hence every plan must have cost at least equal to the cost of a minimum cost hitting set of $\mathcal{L}$. Thus the cost of a minimum cost hitting set of any set of landmarks is a lower bound for h^+.

In a parallel development (Davies and Bacchus 2011) have proposed a new approach to solving MAXSAT that also exploits hitting sets. The main framework is shown in Algorithm 2.

Algorithm 2 MaxHS Algorithm (Davies and Bacchus 2011)

MaxHS($\mathcal{F}$, $\mathcal{K}$)
while TRUE **do**
  $hs =$ FindMinCostHittingSet($\mathcal{K}$)
  (sat?, κ) = sat_solve($\mathcal{F} \setminus hs$)
  if sat? **then**
    return κ
  end if
  $\mathcal{K} = \mathcal{K} \cup \{\kappa\}$
end while

This approach to solving MAXSAT is based on the idea of generating a set of cores (sets of soft clauses κ such that $\kappa \cup hard(\mathcal{F})$ is UNSAT), and finding minimum cost hitting sets of these cores. It can be initialized with a set of cores $\mathcal{K}$ (possibly empty). Its first step is to compute a minimum cost hitting set of these cores, hs, which is a set of soft clauses. Then it tests if the input CNF $\mathcal{F}$ with these clauses removed is satisfiable. If it is "sat?" the satisfying assignment returned in κ by the SAT solver is a MAXSAT solution and the cost of an optimal solution is equal to the cost of hs, the minimum cost hitting set of the current set of cores. Otherwise the SAT solver returns a new core that is un-hit by hs and that is added to the set of cores $\mathcal{K}$. It can be shown that this algorithm is sound and complete for solving MAXSAT.

If we consider our encodings abe (or $abe{+}state$) we see that all cores from these encodings are sets of unit clauses of the form $(\neg a)$ for some action variable. That is, the cores say that it cannot be the case that we can satisfy the hard clauses of abe (which specify that we have enough actions to support the goal and all needed preconditions), without making at least one of these clauses false. That is, at least one of the actions in these clauses must be true. In other words, the cores returned by the SAT solver **are landmarks**. This means that the MAXSAT encoding provides an alternative and more general way of computing action landmarks.

Proposition 3. *For an unsatisfiable core κ, $l = \{a | (\neg a) \in \kappa\}$ is an action landmark.*

Since κ is a core we have by definition that $hard(\mathcal{F}) \wedge \kappa$ is unsatisfiable. $\kappa = (\neg a_1) \wedge \ldots \wedge (\neg a_k)$ for some set of

actions $a_1, \ldots a_n$, since the only soft clauses of $\mathcal{F}$ are unit clauses of the form $(\neg a_i)$ for some action variable a_i. The unsatisfiability of $hard(\mathcal{F}) \wedge \kappa$ implies $\neg hard(\mathcal{F}) \vee a_1 \vee \ldots \vee a_k$ is a tautology, and thus $hard(\mathcal{F}) \models a_1 \vee \ldots \vee a_k$. It is not difficult to see that every valid relaxed plan of Π^+ satisfies $hard(\mathcal{F})$ (including any added hard blocking clause), so $a_1, \ldots, a_k$ must be a landmark for Π^+.

Furthermore, we observe that a hitting set of the cores computed by MaxHS is a minimal set of soft clauses $(\neg a)$ that must be falsified in order to satisfy the hard clauses. That is, this hitting set is a minimal cost set of actions that must be true, thus its cost provides a lower bound on h^+.

There are two immediate consequences to these observations. First, we can seed the MaxHS algorithm with any valid set of landmarks. In our work, we initialize MaxHS with the set of landmarks computed by LM-Cut. These landmarks are utilized in the "FindMinCostHittingSet" routine, they are not added to the MAXSAT encoding. (In the implementation of MaxHS "FindMinCostHittingSet" is computed by the integer program solver CPLEX.) Thus, in the first iteration of the algorithm we already are working with hitting sets (candidate plans) that have cost at least equal to the LM-Cut heuristic. Successive iterations can only increase the size of these candidate plans, thus improving our lower bound on h^+.

Thus, we are ensured that our MAXSAT solver will always return a solution κ whose actions cost at least as much as the value of the LM-Cut heuristic. We have also observed empirically that seeding MaxHS with the LM-Cut computed landmarks serves to speed up the solver by a factor of 4-5 (it no longer has to spend time computing these cores).

The second consequence of these observations is that if we add information about deletes to the MAXSAT encoding (which could be done, e.g., by adding time stamped actions) then we could compute landmarks for non-relaxed planning problems by way of SAT (not MAXSAT) solving. This provides a general and immediate way of computing landmarks that is not dependent on the graph cut formalization presented in (Bonet and Helmert 2010) which cannot compute landmarks that are implied by the action's delete effects.

Lower Bound Approximations of h^+ As noted above each iteration of Algorithm 1 (before we add a new blocking clause) yields a potentially improved lower bound on h^+. In particular, ρ the solution to each MAXSAT problem F specifies a set of true action variables whose cost is always a lower bound on h^+. Hence, we can stop our computation and use this lower bound before adding the next blocking clause. (Of course, if ρ specifies a legal relaxed plan we can then calculate h^+ exactly). In addition to these outer level iterations, if we use the MaxHS algorithm to solve each MAXSAT problem, we can stop at each inner level iteration were a minimum cost hitting set of the current set of cores has been computed using "FindMinCostHittingSet". Again as noted above, the cost of this minimum cost hitting set is always a lower bound on h^+. Hence, we can stop our computation after adding some bounded number of blocking clauses or after computing the minimum cost hitting set of some bounded number of cores. This yields considerable flexibility for achieving a useful tradeoff between heuristic accuracy and computation time. In our empirical results we generate lower bound approximations of h^+ by limiting the CPU time provided to the MAXSAT solver (MaxHS), and the number of blocking clauses Alg. 1 is allowed to add.

Utilizing other MAXSAT Solvers. Any other MAXSAT solver could be used in Alg. 1 rather MaxHS. These solvers could also be seeded with the landmarks computed by LM-Cut by adding these landmarks as hard clauses stating that at least one of the actions in the landmark must be true.[1]

There are, however, two practical advantages of using MaxHS in our empirical results. First, we had access to the source code so we could more conveniently terminate its execution and still obtain the latest lower bound from it. Second, in other MAXSAT solvers the LM-Cut landmarks would have to be added as clauses to the MAXSAT encoding. With solvers using the sequence of SAT approach this could potentially speed up the solver, but these solvers would still have to refute various lower cost solutions before they reached a bound equal to the LM-Cut bound. With MaxHS, on the other hand, the LM-Cut landmarks are added to the CPLEX model which immediately calculates a lower bound at least equal to the LM-Cut bound.

Lazy Heuristic Evaluation in A^*

The final technique we have developed to optimize time is to limit the use of MAXSAT solving to compute h^+ during A^* search. In particular, we utilize LM-Cut as our base heuristic. Initially, nodes of the search space are placed on the OPEN list with their heuristic value set to that computed by LM-Cut—we do not invoke the MAXSAT solver to compute a more accurate estimate of h^+ at this stage. Rather we wait until we have selected a node n from OPEN to be expanded next. If n's heuristic value was computed by MAXSAT, we expand it and add its successors to OPEN. However, if the n's heuristic value is still the LM-Cut value, we invoke Alg. 1 to compute a better heuristic value. Given that Alg. 1 is seeded with the cores (landmarks) computed by LM-Cut, this can only increase n's heuristic value. Once we finish this computation, we check to see if n should still be at the front of OPEN given its new (potentially higher) f-value. If not we place it back on OPEN, and move on to the node on OPEN that now has lowest f-value. In this way we never invoke the more expensive MAXSAT computation for nodes that never make it to the front of OPEN. It is easy to see that since all of these heuristic estimates are admissible, A^* will still find an optimal cost plan.

Experiment Results

Our experiments were run using FastDownward planner platform (Helmert 2006) and an Intel Xeon 5150 machine. Time and memory limits were set to 1800 seconds and 4GB per problem. The MaxHS solver (Davies and Bacchus 2011) we use employs MiniSAT 1.4 as its SAT solver and CPLEX to compute minimum hitting sets.

Our first set of experiments examine the accuracy of our MAXSAT computations. We ran A^* using our MAXSAT com-

[1] Thanks to the reviewers for pointing this out.

Table 1: Comparison of solved task over IPC benchmark domains.

Domain	Coverage				Nodes Expanded Relative to LM-Cut				Time (CPU sec.)			
	MS	Lm-cut	M1	M2	MS	Lm-cut	M1	M2	MS	Lm-cut	M1	M2
elevators(20)	9	16	17	**17**	230.09	1.00	0.48	0.62	706.57	254.91	2,231.50	828.16
floortile(20)	2	**6**	5	5	61.66	1.00	0.19	0.48	8.56	2.57	186.42	32.49
mystery(30)	14	**15**	14	15	83.91	1.00	0.31	0.97	214.04	191.41	1869.42	259.22
nomystery(20)	14	14	10	**16**	8.51	1.00	0.15	0.53	16.09	603.90	715.40	179.20
openstacks(20)	4	**14**	13	14	1.03	1.00	1.00	1.00	744.93	18.68	37.79	25.90
parcprinter(20)	8	13	11	**15**	576.09	1.00	0.99	1.00	105.04	50.49	557.31	372.14
pegsol(20)	0	**17**	12	17	-	1.00	0.93	1.00	-	114.48	1742.15	1359.56
sat(10)	0	2	2	**2**	-	1.00	0.02	0.02	-	993.12	707.00	528.31
satellite(36)	6	7	8	**10**	903.58	1.00	0.28	0.30	298.09	229.38	596.44	140.11
scanalyzer(20)	9	**11**	7	11	1.25	1.00	0.34	0.35	256.75	148.68	600.96	223.87
sokoban(20)	8	**20**	11	19	432.51	1.00	0.47	0.99	3301.20	1177.75	2363.33	1552.12
tidybot(20)	1	**13**	8	11	1.00	1.00	1.00	1.00	0.10	0.10	0.10	0.10
transport(20)	**6**	6	3	4	76.17	1.00	0.44	0.65	8.58	34.52	863.14	284.97
trucks(30)	5	**10**	8	8	444.09	1.00	0.24	0.64	135.04	5.77	1058.59	94.12
visitall(20)	9	10	11	**14**	1.80	1.00	0.00	0.00	246.15	335.32	429.97	199.70
woodworking(20)	6	11	10	**14**	21.76	1.00	0.34	0.35	59.74	59.12	293.13	55.08
SUM	101	185	150	**192**	2843.46	16.00	7.20	9.90	6100.88	4220.20	14652.65	6205.05

puted heuristics on the 12 problem instances shown in Table 2. We employed four configurations of resource limits for the MAXSAT solver: (0.1, 5), (1,10), (5,20), and (10,40) where (x,y) indicates that x CPU seconds were given and at most y blocking clauses could be added. Fig.2 (a) shows the percentage of node evaluations in each of these problems where the MAXSAT solver was able to compute the exact h^+ value, and 2 (b) shows the time to complete the search. The x-axis on these graphs indicate the problem instance being solved. The The graphs indicate that as we increase the resource limits, we more frequently compute h^+. However, ten seconds (10,40) is not sufficient to compute h^+ much more than 25% of the time in instances 2, 10, and 11. Table 2 also shows that as the computational resources are increased we get a higher percentage reduction in the number of expanded nodes (up to the last f-layer) as compared to LM-Cut. However, the search time can increase significantly as we provide MAXSAT with more resources as seen in Fig. (b).

Table 2 gives a brief comparison of our approach to that of (Bonet and Helmert 2010) on the same set of 12 problem instances. We compare our MAXSAT configurations with the most expensive of the heuristic estimators presented by Bonet and Helmert, namely $h_{(5,15)}$. We compare the percentage reduction over LM-Cut using the same benchmark problems. (The numbers in the LM-Cut column are the absolute number of nodes expanded). We see that $h_{(5,15)}$ tends to be more accurate than our cheapest configuration, but not as accurate as our most expensive configuration. Unfortunately, the sample set is too small to draw any clear conclusions. A more in-depth comparison is on-going work.

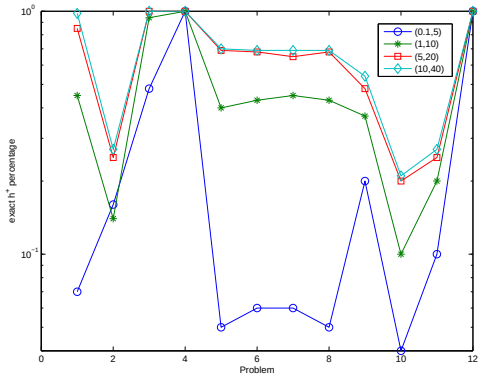
Next we conducted a more detailed evaluation of the overall performance of our planning approach on 16 different domains. 12 of them are selected from the most recent international planning competitions (IPC7). The others are domains that are considered challenging for LM-cut. The evaluation compares the performance of A^* when computing a cost optimal plan using the merge and shrink (Helmert, Haslum, and Hoffmann 2007) heuristic, the LM-cut heuristic, and our MAXSAT heuristics. M1 is standard A* search, while M2 are the results obtained when lazy heuristic evaluation of the MAXSAT heuristic is used. Both M1 and M2 use the resource bounds (0.1, 5). The number of problems solved, nodes expanded as a proportion of the nodes expanded when using the LM-cut heuristic,, and total CPU time taken (in seconds) for that problem domain (*not counting timed out instances*) is shown in Table 1. For each problem domain the number of problem instances is shown in brackets.

Table 2: Percentage of reduction of expanded nodes up to last f layer compared to $h_{(5,15)}$ in (Bonet and Helmert 2010)

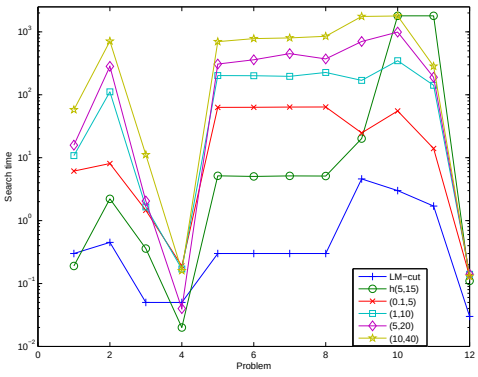
Inst.	LM-cut	$h_{(5,15)}$	(0.1,5)	(1,10)	(5,20)	(10,40)
Freecell-01	390	**100.00**	90.77	99.49	**100.00**	**100.00**
Mystery-09	30	**61.50**	3.33	10.00	33.33	40.00
Mystery-27	3	66.70	66.67	**100.00**	**100.00**	**100.00**
Mystery-28	2	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
Openstacks-01	1822	70.40	30.52	76.29	90.40	**93.08**
Openstacks-03	1805	70.50	30.53	76.01	90.47	**92.35**
Openstacks-04	1817	71.50	30.32	77.55	89.32	**92.74**
Openstacks-05	1819	69.80	31.45	74.16	88.89	**91.64**
Pipesworld-Tank.-05	129	**70.30**	15.50	41.86	57.36	61.24
Pipesworld-NoT.-06	560	68.20	15.36	49.29	67.86	**69.64**
Pipesworld-NoT.-07	56	**100.00**	7.14	23.21	60.71	64.29
Satellite-03	6	**66.67**	**66.67**	**66.67**	**66.67**	**66.67**

The best results obtained for a domain are bolded. In particular, the planner that solves the most problems in the domain is considered to be the best performer, with ties broken by the least CPU time taken.

From the table, we can see that the nodes expanded by M2 are slightly greater than M1. However, this is due to different tie breaking behavior and the difference is not significant. M1 and M2 in general expand considerably fewer nodes than LM-Cut, while merge and shrink tends to expand more. Even though the MAXSAT heuristics are computationally more expensive than LM-Cut, M2 still achieves better performance in seven domains, solving a larger total number of problems. However, on some domains it is con-



(a)



(b)

Figure 2: (a): percentage of node evaluations that compute h^+ exactly (i.e., that terminate within the given resources), (b): the overall search time of different MAXSAT configurations compared to LM-cut

siderably slower than LM-Cut. We also see that M2 (lazy heuristic evaluation) achieves a very useful performance improvement over M1.

Conclusion

In this paper, we have formulated the problem of computing h^+ as a MAXSAT problem using a novel encoding that employs constraint generation. More complex encodings could in principle capture certain delete effects of actions. In that case, the MAXSAT approach we have used would be able to produce landmarks (cores) for relaxations more accurate than the delete relaxation. Our empirical results indicate that using MAXSAT has promise for solving the optimal planning problem, and also allows us to exploit future developments in MAXSAT solver technology.

In future work we plan to investigate richer MAXSAT encodings that can capture some delete information. We also plan to investigate whether or not MAXSAT can be used directly to compute optimal plans using constraint generation style encodings.

References

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2010. A new algorithm for Weighted Partial MAXSAT. In *Proceedings of the AAAI National Conference (AAAI)*, 3–8.

Betz, C., and Helmert, M. 2009. Planning with h^+ in theory and practice. In *Proceedings of German conference on Advances in artificial intelligence*, 9–16.

Bonet, B., and Castillo, J. 2011. A complete algorithm for generating landmarks. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 329–334.

Bylander, T. 1994. The computational complexity of propositional strips planning. *Artificial Intelligence (AI)* 69:165–204.

Chandrasekaran, K.; Karp, R.; Moreno-Centeno, E.; and Vempala, S. 2011. Algorithms for implicit hitting set problems. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*, 614–629.

Davies, J., and Bacchus, F. 2011. Solving maxsat by solving a sequence of simpler sat instances. In *Principles and Practice of Constraint Programming (CP)*, volume 6876 of *Lecture Notes in Computer Science*, 225–239.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 162–169.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 176–183.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)* 26:191–246.

Heras, F.; Morgado, A.; and Marques-Silva, J. 2011. Core-guided binary search algorithms for maximum satisfiability. In *Proceedings of the AAAI National Conference (AAAI)*.

Karp, R. M. 2010. Implicit hitting set problems and multi-genome alignment. In *Combinatorial Pattern Matching*, volume 6129 of *Lecture Notes in Computer Science*, 151.

Kautz, H. A., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the AAAI National Conference (AAAI)*, 1194–1201.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Robinson, N.; Gretton, C.; Pham, D. N.; and Sattar, A. 2010. Partial weighted maxsat for optimal planning. In *Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, volume 6230 of *Lecture Notes in Computer Science*, 231–243.