Efficient Encoding of Cost Optimal Delete-Free Planning as SAT

Masood Feyzbakhsh Rankooh and Jussi Rintanen

Department of Computer Science, Aalto University, Helsinki, Finland
jussi.rintanen@aalto.fi, masood.feyzbakhshrankooh@aalto.fi

Abstract

We introduce a novel method for encoding cost optimal delete-free STRIPS Planning as SAT. Our method is based on representing relaxed plans as partial functions from the set of propositions to the set of actions. This function can map any proposition to a unique action that adds the proposition during execution of the relaxed plan. We show that a relaxed plan can be produced by maintaining acyclicity in the graph of all causal relations among propositions, represented by the mentioned partial function. We also show that by efficient encoding of action cost propagation and enforcing a series of upper bounds on the total costs of the output plan, an optimal plan can effectively be produced for a given delete-free STRIPS problem. Our empirical results indicate that this method is quite competitive with the state of the art, demonstrating a better coverage compared to that of competing methods on standard STRIPS planning benchmark problems.

Introduction

Delete-free planning problems are those for which every action can add propositions but not delete them. Every planning problem can be relaxed into a delete-free problem in linear time. The optimal cost of the solution for delete-free relaxation of a planning problem, denoted by h^+ is a lower bound of the optimal cost of the original problem. Computing h^+, however, is NP-equivalent (Bylander 1994), and also hard to approximate (Betz and Helmert 2009).

Solving delete-free planning problems is important for several reasons. Many admissible heuristics functions have been introduced to compute lower bounds for h^+. Examples are h^{max} heuristic (Bonet and Geffner 2001), LM-cut heuristic (Helmert and Domshlak 2009), set-additive heuristic (Keyder and Geffner 2008), and cost-sharing approximations of h^{max} (Mirkis and Domshlak 2007). This accentuates the need for computing the exact value of h^+, as it is a measure for informativeness of such heuristic functions.

Moreover, efficient solving of delete-free planning problems is inherently important, as there exist delete-free planning tasks that are of interest for AI community. Examples of such problems are the minimal seed-set problem (Gefen and Brafman 2011), and the problem of determining join orders in relational database query plan generation (Robinson, McIlraith, and Toman 2014).

Another reason for importance of efficient delete-free planning emerges from the fact that optimal plans for general planning problems can be produced by iterative solving and reformulating delete-free planning tasks (Haslum 2012). This can be done by repeatedly finding optimal plans for delete-free relaxations of the planning problem, while reducing the relaxation by reformulating the problem in each iteration, until the optimal plan for the delete-free problem is in fact an optimal plan for the original problem.

The current state-of-the-art methods for finding exact value of h^+ are based on Integer and Linear Programming (IP/LP) (Haslum, Slaney, and Thiébaux 2012; Imai and Fukunaga 2015; Castro et al. 2020). Because propositional satisfiability has a limited expressive power compared to that of LP and IP, using it for computing h^+ might seem counter-intuitive. In our view, however, there are compelling reasons to the contrary.

Representing and reasoning about causality, which is a sizable part of knowledge representation and inference needed to compute h^+, is inherently propositional. Therefore, propositional satisfiability is a well-suited platform to perform such tasks. Nevertheless, the main tentative weakness of satisfiability is representing and propagating information about action costs.

In this work, we show that by using succinct representation of causality, and also careful encoding of action cost propagation, one can efficiently compute h^+ using propositional satisfiability. Our empirical results indicate that our method is quite competitive with the state of the art, demonstrating a better coverage than LP/IP based methods on conventional STRIPS planning problems.

Preliminaries and Background

We formalize a STRIPS planning problem as a 5-tuple $\Pi = (P, I, A, G, cost)$ where P is a finite set of Boolean state variables, also called *propositions*. I, the initial state, and G, the set of goal conditions, are subsets of P. A is a finite set of actions. Each member a of A is a triple $(pre(a), add(a), del(a))$, where $pre(a)$, $add(a)$ and $del(a)$ are sets of propositions denoting the set of preconditions, positive effects, and negative effects of a, which are the propositions that a requires, adds, and deletes, respectively.

The cost function $cost$ maps every members of A to a non-negative integer.

When states are represented as subsets of P, the successor $s' = exec_a(s)$ of s with respect to action $a \in A$ is defined if $pre(a) \subseteq s$, and $s' = (s \backslash del(a)) \cup add(a)$. An action sequence $a_1, ..., a_n$ is executable in state s if $exec_{a_1,...,a_n}(s) = exec_{a_n}(exec_{a_{n-1}}(...exec_{a_2}(exec_{a_1}(s))))$ is defined. A plan for Π is a sequence π of actions from A such that $G \subseteq exec_\pi(I)$. The cost of plan $\pi = \langle a_1, \ldots, a_n \rangle$ for Π, denoted by $cost(\pi)$, is defined by $\Sigma_{i=1,...,n} cost(a_i)$. An optimal plan for Π is a plan with minimal cost for Π.

For any STRIPS planning problem $\Pi = (P, I, A, G, cost)$, the delete relaxation (Bonet and Geffner 2001) $\Pi^+ = (P, I, A^+, G, cost)$ is defined, where A^+ is produced from A by replacing the set of negative effects of each member of A with the empty set. A plan for Π^+ is called a relaxed plan for Π. The optimal cost of Π^+ is denoted by $h^+(\Pi)$. If there is no relaxed plan for Π, we set $h^+(\Pi)$ to ∞.

Related Works

Encoding h^+ using constraint programming has been extensively studied. Most of the research in this field has been focusing on finding lower bounds for h^+ using Linear Programming. Examples are getting information from abstraction heuristics (van den Briel et al. 2007), using linear programming to derive heuristic estimates from landmarks for classical planning (Karpas and Domshlak 2009) and numeric planning (Scala et al. 2017), the state-equation heuristic (Bonet 2013), and post-hoc optimization heuristics (Pommerening, Röger, and Helmert 2013). A unified formulation for combining Integer and Linear Programming approaches has also been introduced (Pommerening et al. 2014).

Finding the exact value of h^+ has also been done using Integer and Linear Programming. Most notably, Haslum, Slaney, and Thiébaux (2012) find the value of h^+ by using set-inclusion minimal disjunctive landmarks and solving the LP relaxation of the integer programming formulation of a hitting set problem. Imai and Fukunaga (2015) compute the value of h^+ by solving a Mixed Integer and Linear Programming (MILP) formulation of delete-free planning problems. Another notable work is computing h^+ by using relaxed Decision Diagram based heuristics (Castro et al. 2020).

Currently, the MILP formulation mentioned above is considered to be the state-of-the-art method for optimally solving delete-free problems, though the work introduced in (Haslum, Slaney, and Thiébaux 2012) shows a somewhat complementary coverage.

Causal Relaxed Plan Representations

In this section, we provide an alternative representation of relaxed plans that corresponds to our SAT encoding of delete-free STRIPS problems. We only consider STRIPS planning problems for which the value of h^+ is finite. This assumption is not restrictive as $h^+(\Pi) = \infty$ is decidable in polynomial time (Hoffmann and Nebel 2001).

Definition 1 (Minimal Relaxed Plans)**.** *Let $\Pi = (P, A, I, G, cost)$ be a STRIPS planning problem and $\pi = \langle a_1, \ldots, a_n \rangle$ be a relaxed plan for Π. We say π is a minimal relaxed plan for Π iff each action a_i in π adds at least one proposition that is neither in I, nor in the positive effects of any other action a_j in π for $j < i$.*

Clearly, every optimal relaxed plan for Π is also a minimal relaxed plan for Π.

Let f be a partial function from X to Y. The *domain* of f is the set of members of X for which f is defined. The *range* of f, also $Range(f)$, is the set of members of Y to which some member of X is mapped by f.

Definition 2 (Causal Relaxed Plan Representations)**.** *Let $\Pi = (P, A, I, G, cost)$ be a STRIPS planning problem. Assume that f is a partial function from $P \backslash I$ to A such that 1) if $f(p) = a$ then a adds p; 2) if $f(p) = a$ then for every precondition q of a, either $q \in I$ or f is defined for q; 3) for every $p \in G$, either $p \in I$ or f is defined for p. We call any partial function with conditions 1 to 3 a causal partial function for Π. Let $G_f = (P, E_f)$ be a graph for which there is an edge in E_f from q to p iff for some a, $f(p) = a$ and q is a precondition of a. We say f is a causal relaxed plan representation for Π iff G_f is acyclic. The cost of f, denoted by $cost(f)$, is defined by the total costs of all actions in the range of f, i.e., those actions to which some proposition is mapped by f .*

We call the above-mentioned representation *causal*, because for each p, $f(p)$ is intended to be equal to an action that is the cause of p becoming *true*. Condition 1 of Definition 2 is thereby necessary. Condition 2 guarantees that if action a is the cause of p becoming *true*, then the preconditions of a need to have causes. Condition 3 provides that all goals must have causes. The acyclicity of G_f is required to avoid causal cycles.

We now prove that for every STRIPS planning problem Π, minimum cost over all causal relaxed plan representations is equal to $h^+(\Pi)$.

Theorem 1. *For every STRIPS planning problem $\Pi = (P, A, I, G, cost)$, there exist a causal relaxed plan representation f for Π such that $cost(f) = h^+(\Pi)$.*

Proof. Let $\pi = \langle a_1, \ldots, a_n \rangle$ be an optimal relaxed plan for Π. We construct partial function f from $P \backslash I$ to A. Let $f(p) = a_i$ iff a_i adds p, $p \notin I$, and for every $j < i$, a_j does not add p. Clearly, f is well-defined. Assume that for some $p \in P$, $f(p) = a_i$. For every precondition q of a_i if $q \notin I$, then there must exist $j < i$ such that a_j adds q, and thus, f is defined for q. Similar discussion shows that for every $p \in G$, either $p \in I$ or f is defined for p. Let G_f be the graph constructed as described in Definition 2. Members of I cannot be in any cycles as they have no incoming arcs. We define function g on the domain of f, such that $g(p) = i$ iff $f(p) = a_i$. It is easy to check that g is monotonically increasing along the nodes of any path in G_f. Therefore, G_f is acyclic. We conclude that f is a causal relaxed plan representation for Π. Since π is a minimal relaxed plan for Π, $Range(f)$ is equal to $\{a_1, ..., a_n\}$. Thus, $cost(f) = cost(\pi)$. □

Theorem 2. *Let $\Pi = (P, A, I, G, cost)$ be a STRIPS planning problem and f be a causal relaxed plan representation for Π. Then $cost(f)$ is an upper bound of $h^+(\Pi)$.*

Proof. Since G_f described in Definition 2 is acyclic, we can order members of the domain of f by the topological sorting according to G_f. Let $p_1, ..., p_n$ be such an order. We now order members of $Range(f)$. For each action a in $Range(f)$, let $O(a) = argmin_i\{p_i | f(p_i) = a\}$. One clear conclusion is that if $f(p_i) = a$, then $O(a) \leq i$. Let $a_1, ..., a_m$ be the increasing ordering of members of $Range(f)$ according to O. This ordering is well-defined because by definition, f does not map any proposition to different actions. We show that $\pi = \langle a_1, ..., a_m \rangle$ is valid relaxed plan for Π. For k such that $1 \leq k \leq m$, assume that $O(a_k) = i$, and therefore p_i is an add effect of a_k. If $p_j \in pre(a_k) \backslash I$, then there is an arc in G_f from p_j to p_i, and we have: $j < i$. According to Definition 2, there must exist a_l such that $f(p_j) = a_l$. Therefore, $O(a_l) \leq j < i = O(a_k)$, and thus $l < k$. We conclude that every precondition of a_k must be provided either in the initial state or by another action ordered before a_k in π. Furthermore, According to Definition 2, every member of G must be added by some action in the range of f. Therefore, π is a relaxed plan for Π. Since $cost(f) = cost(\pi)$, we conclude that $cost(f)$ is an upper bound of $h^+(\Pi)$. □

Theorem 1 shows that for every STRIPS planning problem Π, there exists a causal relaxed plan representation f such that $cost(f) = h^+(\Pi)$. On the other hand, Theorem 2 shows that cost of no causal relaxed plan representation for Π can be lower than $h^+(\Pi)$. In other words, finding the minimum cost over all possible causal relaxed plan representations is equivalent of finding h^+. The proof of Theorem 2 is also constructive: once a causal relaxed plan representations has been found, the optimal plan can be extracted in polynomial time from it by the method described in proof of Theorem 2.

SAT Encoding of Causal Relaxed Plan Representations

We now introduce propositional variables and formulas needed to encode the causal partial function of Definition 2. We then explain our choice of the method used here for guaranteeing acyclicity of G_f. This method is based of the work done in (Rankooh and Rintanen 2022), which leverages low directed elimination widths (Hunter and Kreutzer 2007) of the underlying graph of our SAT formula, i.e., G_f.

Encoding of Causal Partial Functions

Let $\Pi = (P, A, I, G, cost)$ be a STRIPS planning problem. Without loss of generality assume that all members of I have been removed from P, preconditions and effects of all actions, and G. In order to encode causal partial function f of Definition 2 we use the following propositional variables:

- for each $p \in P$, f_p indicates whether f is defined for p.
- for each $a \in A$ and $p \in add(a)$, $f_{p,a}$ indicates whether $f(p) = a$.

Conjunction of the following formulas encodes f:

$$\bigwedge_{p \in P} (f_p \leftrightarrow \bigvee_{p \in add(a)} f_{p,a}) \tag{1}$$

$$\bigwedge_{a,a' \in A, p \in add(a) \cap add(a'), a \neq a'} f_{p,a} \rightarrow \neg f_{p,a'} \tag{2}$$

$$\bigwedge_{a \in A, p \in add(a), q \in pre(a)} f_{p,a} \rightarrow f_q \tag{3}$$

$$\bigwedge_{p \in G} f_p \tag{4}$$

Formulas (1) to (4) are straightforward translation of conditions in Definition 2. Formulas (1) and (2) guarantee that f is a partial function. Formulas (3) and (4) ensure conditions (2) and (3) of Definition 2, respectively.

Encoding of Acyclicity

According to Definition 2, for any given causal partial function f, the graph G_f is needed to be acyclic in order to achieve a causal relaxed plan representation. Assume that $P = \{p_1, ..., p_{|P|}\}$. To encode the edges of G_f, we use propositional variable $e_{i,j}$ to indicate whether there is an edge in G_f from p_i to p_j. Formula (5) below encodes G_f:

$$\bigwedge_{a \in A, p_i \in pre(a), p_j \in add(a)} f_{p_j,a} \rightarrow e_{i,j} \tag{5}$$

We now need to encode checking the acyclicity for G_f. This can be done simply by enforcing transitive closure on G_f using formulas (6) to (8) below (Gebser, Janhunen, and Rintanen 2014), where propositions $t_{i,j}$ indicate whether there is an edge in the transitive closure of G_f from p_i to p_j

$$\bigwedge_{(v_i,v_j) \in E_f} e_{i,j} \rightarrow t_{i,j} \tag{6}$$

$$\bigwedge_{(v_i,v_j) \in E_f, p_k \in P} e_{i,j} \wedge t_{j,k} \rightarrow t_{i,k} \tag{7}$$

$$\bigwedge_{(v_i,v_j) \in E_f} e_{i,j} \rightarrow \neg t_{j,i} \tag{8}$$

However, as it will be shown in our Empirical Results section, encoding acyclicity can be done in a more efficient way by using vertex elimination graphs (Rose and Tarjan 1975) instead of transitive closure graphs. We now review the encoding of acyclicity for SAT formulas with underlying graphs using vertex elimination method, which has previously been introduced in (Rankooh and Rintanen 2022).

Propositional Formulas with Underlying Digraphs Assume that ϕ is a propositional formula over the set of propositional variables P. Let $G = (V, E)$ be a digraph and g be a partial function from P to E. Then ϕ is a propositional formula with underlying digraph G with respect to g. In this work, we avoid explicit definition of g by denoting the proposition in ϕ that is mapped by g to $(s, t) \in E$ by $e_{s,t}$. If there exists a model $\mathcal{M}$ for ϕ, we can construct $G_{\mathcal{M}} = (V, E_{\mathcal{M}})$, the underlying graph of $\mathcal{M}$, where $E_{\mathcal{M}} = \{(s, t) | \mathcal{M}(e_{s,t}) = true\}$.

It should be clear that the conjunction of formulas (1) to (5) is a propositional formula with underlying graph G_f. We now explain how cycles can be avoided in the underlying graph.

Vertex Elimination Graphs Vertex elimination graph has originally been introduced in (Rose and Tarjan 1975). Let $G = (V, E)$ be a directed graph, $G^+ = (V, E^+)$ be the transitive closure of G, and $O = v_1, ..., v_{|V|}$ be an arbitrary ordering of members of V. According to ordering O, we produce a sequence of graphs $G_0 = G, ..., G_{|V|}$ by eliminating vertices of G. For each $i > 0$, G_i is produced from G_{i-1}, by eliminating v_i, and adding edges from all its in-neighbors to all its out-neighbors. Formally, $G_i = (V_i, E_i)$ is produced from $G_{i-1} = (V_{i-1}, E_{i-1})$ so that $V_i = V_{i-1} \backslash \{v_i\}$, and $E_i = E_{i-1} \backslash (\{(v_j, v_i) | (v_j, v_i) \in E_{i-1}\} \cup \{(v_i, v_k) | (v_i, v_k) \in E_{i-1}\}) \bigcup D_i$, where $D_i = \{(v_j, v_k) | (v_j, v_i) \in E_{i-1}, (v_i, v_k) \in E_{i-1}, j \neq k\}$. $G^* = (V, E^*)$ is the vertex elimination graph of G according to elimination ordering O, where $E^* = \bigcup_{i=1,...,|V|} E_i$.

The directed elimination width (Hunter and Kreutzer 2007) of ordering O for graph G is the maximum over the number of neighbors of v_i in G_i for $i = 1, ...|V|$. The directed elimination width of G is the minimum width over all directed elimination orderings for G.

It has been shown that finding the ordering that produces the minimum width for a given digraph is NP-complete (Rose and Tarjan 1975). However, there are ad-hoc methods for producing orderings with low directed elimination width in practice. An examples is *minimum degree* heuristic, which chooses v_i with the minimum degree from G_{i-1}.

Guaranteeing Acyclicity Using Vertex Elimination We here review the work done in (Rankooh and Rintanen 2022) that employs vertex elimination for explicit encoding of acyclicity for propositional formulas with underlying digraphs. Assume that ϕ is a propositional formula over the set X of variables, with underlying graph $G = (V, E)$. Let O be an elimination ordering for G, $G^* = (V, E^*)$ be the vertex elimination graph of G according to O, and δ be the directed elimination width of O for G. Let Δ be the set of all triangles produced by elimination ordering O for graph G. Members of Δ are all ordered triples (v_i, v_j, v_k) such that (v_i, v_k) is added by eliminating v_j. Also if model $\mathcal{M}$ satisfies ϕ, let $G_{\mathcal{M}}$ be the underlying graph of $\mathcal{M}$. The encoding of acyclicity for ϕ using vertex elimination according to O, denoted by ϕ_G^{acycl}, is produced by conjunction of formulas (9) to (11). Note that propositional variable $e'_{i,j}$ indicates whether there is an arc from v_i to v_j in G^*.

$$\bigwedge_{(v_i,v_j)\in E} e_{i,j} \rightarrow e'_{i,j} \tag{9}$$

$$\bigwedge_{(v_i,v_j)\in E^*,(v_j,v_i)\in E^*,i<j} e'_{i,j} \rightarrow \neg e'_{j,i} \tag{10}$$

$$\bigwedge_{(v_i,v_j,v_k)\in \Delta} (e'_{i,j} \wedge e'_{j,k}) \rightarrow e'_{i,k} \tag{11}$$

It has been shown that ϕ has a model $\mathcal{M}$ such that $G_{\mathcal{M}}$ is acyclic if and only if $\phi \wedge \phi_G^{acycl}$ is satisfiable by an extension of $\mathcal{M}$ (Rankooh and Rintanen 2022). The number of Boolean variables used in ϕ_G^{acycl} is $\mathcal{O}(\delta|V|) \subseteq \mathcal{O}(|V|^2)$, and the number of clauses is $\mathcal{O}(\delta^2|V|) \subseteq \mathcal{O}(|V|^3)$.

It has also been shown that for formulas with underlying graphs, for the cases with low directed elimination widths, using vertex elimination for guaranteeing acyclicity can result in considerably higher performance compared to that of alternative methods such as encoding transitive closure and using GraphSAT (Gebser, Janhunen, and Rintanen 2014) as the solver (Rankooh and Rintanen 2022). Our experiments confirm that δ is low for a great majority of the underlying graphs for our encodings of causal delete-free representations for STRIPS planning problems, which justifies our choice of the method used of cycle prevention for G_f.

Enforcing Upper Bound on the Cost of Causal Delete-Free Representations

The conjunction of formulas (1) to (5) and $\phi_{G_f}^{acycl}$ encodes a valid causal delete-free representation for any given STRIPS planning problem. To minimize the cost of the produced causal delete-free representation, we encode the enforcement of a given upper bound on the total cost of all chosen actions. We show that this can be done in two different ways: propagating costs of chosen actions 1) along all actions, and 2) along all propositions. Both these methods do the propagation sequentially. We leave investigating more elaborated ways of cost propagation such as tree-based methods (Anbulagan and Grastien 2009) for our future research.

Once encoding of an upper bound is established, by Theorems 1 and 2, if a formula with upper bound u is unsatisfiable then $u + 1$ is a lower bound of h^+. Similarly, if a formula with upper bound u is satisfiable, then u is an upper bound of h^+. By finding value u such that u is both an upper bound and a lower bound of h^+, we prove that h^+ is equal to u.

Propagating Cost through Actions

Let $a_1, ..., a_{|A|}$ be an arbitrary ordering of the actions of a STRIPS planning problem $\Pi = (P, A, I, G, cost)$. Also let u be the upper bound that we want to enforce on the cost of produced causal delete-free representation. Consider the following propositional variables:

- a_i for $a_i \in A$ to indicate action a_i has been chosen to be included in the produced relaxed plan.
- $c_{i,j}$ for $1 \leq i \leq |A| + 1$ and $j = 0, ..., u + 1$ to indicate that j is a lower bound of the total cost of all chosen actions with indices lower that i.

Consider the following formulas:

$$c_{1,0} \tag{12}$$

$$\bigwedge_{a_i\in A} (a_i \leftrightarrow \bigvee_{p\in add(a_i)} f_{p,a_i}) \tag{13}$$

$$\bigwedge_{1\leq i\leq |A|,0\leq j\leq u} c_{i,j} \rightarrow c_{i+1,j} \tag{14}$$

$$\bigwedge_{1\leq i\leq |A|,0\leq j\leq u+1} c_{i,j} \wedge a_i \rightarrow c_{i+1,min(j+cost(a_i),u+1)} \tag{15}$$

$$\neg c_{|A|+1,u+1} \tag{16}$$

Formula (12) provides the initial value of zero for the cost. Formula (13) ensures that an action is included in the relaxed plan iff it is a cause of some proposition. Formula (14) guarantees the propagation of the found lower bounds through all actions. Formula (15) updates the lower bound in the case that an action has been chosen to be included in the relaxed plan. Finally, formula (16) ensures that the found lower bound is not greater than u, the supposed upper bound of the total cost of all actions.

One important observation is that it is not necessary to use all propositional variables of the form $c_{i,j}$ explained above to encode cost propagations. The total cost of all chosen actions with indices lower that i may not take all values from 1 to $u+1$. One simple example is when the GCD of the costs of all actions is $m > 1$. In this case the propagated values only need to be divisible by m. As another example, note that a superset of the set of feasible values for total cost of all chosen actions with indices lower that i when $i = 3$ is $\{0, cost(a_1), cost(a_2), cost(a_1) + cost(a_2)\}$. We can easily compute a superset of the set of feasible values for total cost of all chosen actions with indices lower that i for $i \leq |A|+1$ using a dynamic programming implementation of the following recursive function with $S(1) = \{0\}$:

$$\begin{aligned} S(i) =& \{c | c \in S(i-1)\} \cup \\ & \{min(c + cost(a_{i-1}), u+1) | c \in S(i-1)\} \end{aligned} \tag{17}$$

The variable $c_{i,j}$ (and also the formulas including $c_{i,j}$) can be removed from our encoding if $j \notin S(i)$.

Propagating Cost through Propositions

Although the method described above for propagating costs through actions is enough to encode causal relaxed plan representations, we show that there are situations in which propositions may carry additional information as to the cost of potentially produced relaxed plan. These pieces of information could be utilized to prune the search space for the SAT solver.

Consider a case that the SAT solver has decided to set f_p to *true*. Assume that every action that adds p adds no other propositions. In this case, even before choosing any action that adds p, we know that the cost of the potentially produced relaxed plan must be increased at least by the minimum costs of all actions that add p. We can also generalize this idea for obtaining more information about to total cost.

Definition 3 (Costs of Propositions)**.** *Let f be a causal relaxed plan representation for $\Pi = (P = \{p_1, ..., p_{|P|}\}, A, I, G, cost)$. We define a cost function for propositions in P. Let $cost(p_i)$ be equal to $cost(f(p_i))$, if $f(p_i)$ is defined, and for every $j < i$, $f(p_j)$ is either undefined or not equal to $f(p_i)$. We let $cost(p_i)$ be equal to 0, otherwise. Furthermore, for each i we define $l(p_i)$ as a lower bound of $cost(p_i)$. If $f(p_i)$ is undefined or 0, we set $l(p_i)$ to 0. Otherwise, we set $l(p_i)$ to the minimum of costs of all actions in A that add p_i .*

It should be clear that for every i, $l(p_i) \leq cost(p_i)$.

Theorem 3. $cost(f) = \sum_{i=1}^{|P|} cost(p_i)$.

Proof. Let M a mapping from $\{p_i \in P | cost(p_i) \neq 0\}$ to the range of f, defined by $M(p_i) = f(p_i)$. M is well-defined because by Definition 3, f must be defined for members of P with non-zero cost. We show that M is a bijection. For a in the range of f, the set $\{p_i | f(p_i) = a\}$ is non-empty. Let p_j be the member with the lowest index from this set. By Definition 3, $M(p_j) = a$, and thus M is a bijection. Moreover, we have $cost(M^{-1}(a)) = cost(p_j) = cost(a)$, where M^{-1} is the reverse mapping of M. Therefore:

$$\begin{aligned} cost(f) &= \sum_{a \in Range(f)} cost(a) \\ &= \sum_{a \in Range(f)} cost(M^{-1}(a)) \\ &= \sum_{i=1}^{|P|} cost(p_i) \end{aligned} \tag{18}$$

□

Theorem 3 shows that propagation of $cost(p)$ is enough for a sound and complete encoding of f. However, we encode propagation of cost through propositions $p_1, ...p_{|P|}$ using both $l(p_i)$ and $cost(p_i)$ for each i. The reason for encoding $l(p_i)$ is that in some situations, it does not need the cause of p_i to be determined before passing the added cost to p_{i+1}. Although, as it was discussed above, this added cost could be lower than $cost(p_i)$, its propagation can cause earlier pruning of the search space of the SAT solver.

Let $p_1, ..., p_{|P|}$ be an arbitrary ordering of the propositions of a STRIPS planning problem $\Pi = (P, A, I, G, cost)$. Let P' be the subset of P, such that $p \in P'$ iff every action that adds p adds no other proposition. Also, let u be the upper bound that we want to enforce on the cost of produced causal delete-free representation. We use propositions $c'_{i,j}$ for $1 \leq i \leq |P| + 1$ and $j = 0, ..., u+1$ to indicate that j is a lower bound of the total cost of all propositions with indices lower that i. Consider the following formulas:

$$c'_{1,0} \tag{19}$$

$$\bigwedge_{a \in A, p_i \in add(a), 0 \leq k \leq u} f_{p_i,a} \wedge \bigwedge_{p_j \in add(a), j < i} \neg f_{p_j,a} \wedge c'_{i,k} \rightarrow c'_{i+1,min(k+cost(a),u+1)} \tag{20}$$

$$\bigwedge_{1 \leq i \leq |P|, 0 \leq j \leq u+1} c'_{i,j} \rightarrow c'_{i+1,j} \tag{21}$$

$$\bigwedge_{p_i \in P', 0 \leq j \leq u} c'_{i,j} \wedge f_{p_i} \rightarrow c'_{i+1,min(j+l(p_i),u+1)} \tag{22}$$

$$\neg c'_{|P|+1,u+1} \tag{23}$$

Formula (19) provides the initial value of zero for the costs. Formula (20) is used to pass the updated lower bound to p_{i+1} in the case that cost of p_i is non-zero according to

Definition 3. Formula (21) guarantees the propagation of the found lower bounds through all propositions. Formula (22) passes the updated lower bound to p_{i+1} by adding $l(p_i)$ to the currently found lower bound in the case that p_i is added only by actions that add no other propositions. Finally, formula (16) ensures that the found lower bound is not greater than u, the supposed upper bound of the total cost of all propositions.

As in the case of cost propagation through actions, we can easily compute a superset of the set of feasible values for total cost of all propositions with indices lower that i for $i \leq |P| + 1$ using the following recursive function with $S'(1) = \{0\}$:

$$S'(i) = \{c | c \in S'(i-1)\} \cup \{min(c + cost(a), u + 1) | \; c \in S'(i-1), p_{i-1} \in add(a)\} \quad (24)$$

The variable $c'_{i,j}$ (and also the formulas including $c'_{i,j}$) can be removed from our encoding if $j \notin S'(i)$.

Empirical Results

We have implemented our SAT-based encoding inside Madagascar planner (Rintanen 2012). All experiments have been run on a cluster of Linux machines, using a timeout of 1800 seconds per problem, and a memory limit of 64 GB. In versions that vertex elimination based method is used, for determining the order of vertex elimination, we have implemented the *minimum degree* heuristic, i.e., eliminating a vertex with minimal total number of incoming and outgoing edges in the graph produced after the elimination of previously eliminated vertices. In all cases we have used *Kissat* (Biere et al. 2020) as the SAT solver.

For producing the optimal cost for the given problem, we perform a search mechanism similar to binary search method. First, for $u = 1, 2, 4, 8, ..$ formulas with upper bound u on the total cost are produced and given sequentially to the SAT solver until a satisfiable formula is found. Let l be the the upper bound for last unsatisfiable formula found in this way. Then $(l, u]$ is the range of all possible values for the optimal cost. We can narrow down this range by checking the satisfiability of a formula with upper bound $m = (u + l)/2$. The new range will be $(l, m]$ if the formula with upper bound m is satisfiable, and $(m, u]$ otherwise. We continue this scheme until value k is found such that formula with upper bound k and $k - 1$ are satisfiable and unsatisfiable, respectively.

All of our implemented versions use some of the preprocessing methods presented in (Imai and Fukunaga 2015). The used preprocessing methods are 1) finding *fact landmarks* (Gefen and Brafman 2011) and adding them to the goal conditions; 2) doing action relevance analysis and removing non-relevant actions; and 3) dominated action elimination. We perform landmark extraction, relevance analysis, and action domination reasoning exactly as they are described in (Imai and Fukunaga 2015).

As our benchmark problem set, we have used the delete-free versions of all STRIPS planning problem sets found in the *planning repository*[1]. From IPC domains, domains from both satisficing and optimal tracks have been considered. In total, 2212 problem instances from 84 problem sets have been used for comparison.

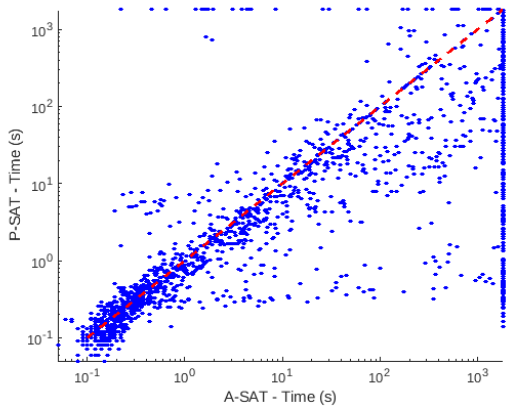

Figure 1: P-SAT vs A-SAT runtime on STRIPS planning problems

The Impact of Cost Propagation Method

We first show that the method used for cost propagation can have a considerable impact on the efficiency of our SAT-based encoding. We have implemented three different solvers: A-SAT by propagating costs only through actions, P-SAT by propagating cost only through propositions, and AP-SAT by propagating costs through actions and propositions. Figures 1 and 2 show the comparison of the time used for solving benchmark instances for the mentioned three versions. For all versions vertex elimination based method has been used for guaranteeing acyclicity in the underlying graphs. Although we observed the same patterns in the results when transitive closure method was used for acyclicity checking, these results have not been presented here for the sake of brevity.

Our results show that P-SAT produces the best results among these three solvers. In total 1614, 1875, and 1808 instances were solved by A-SAT, P-SAT, and AP-SAT versions, respectively. The better performance of P-SAT comes partly from the fact that for many planning problems, the number of propositions is considerably smaller than number of actions. This causes the encoding of cost propagation through proposition to produce SAT formulas with smaller number of variables and clauses, which in turn, results in better performance of the SAT solver.

For the rest of our empirical results, we only use the P-SAT version as it generally outperforms the other versions.

The Impact of Acyclicity Checking Method

We have implemented both vertex elimination and transitive closure based methods for guaranteeing acyclicity for the underlying graphs of our produced formulas. Figure 3

[1]https://github.com/AI-Planning/classical-domains

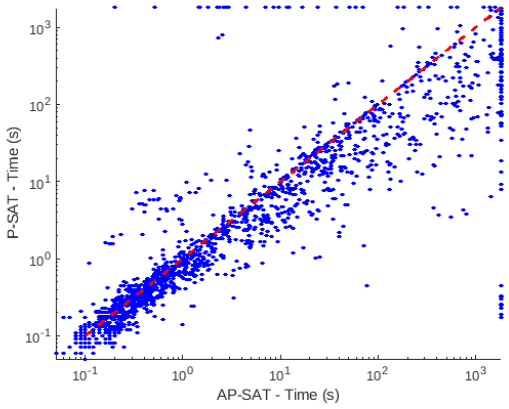

Figure 2: P-SAT vs AP-SAT runtime on STRIPS planning problems

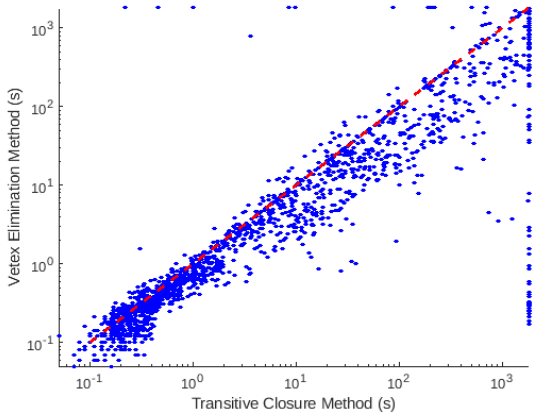

Figure 3: P-SAT with vertex elimination vs P-SAT with transitive closure

shows the comparison of the time used for solving benchmark instances for these methods when implemented inside P-SAT version of our solver.

As it can be seen in Figure 3, using vertex elimination method for guaranteeing acyclicity in the underlying graph has a strong impact on the efficiency of our solver. This is mainly because the directed elimination widths of the underlying graphs are often considerably smaller than the number of propositions. Since variable elimination method uses $\mathcal{O}(\delta|V|)$ propositions and $\mathcal{O}(\delta^2|V|)$ clauses, where δ is the directed elimination width of the elimination order, when δ is low, it can produce more compact formulas in comparison to the transitive closure method that uses $\mathcal{O}(|V|^2)$ variables and $\mathcal{O}(|V||E|)$ clauses. In total 1799 instances were solved by transitive closure method, 76 instances less than the 1875 instances solved by the vertex elimination method.

Comparison with State-of-the-Art Methods

We have compared our method with the HST (Haslum, Slaney, and Thiébaux 2012) and IF (Imai and Fukunaga 2015) methods. Although the Decision Diagram based method (Castro et al. 2020) is more recent than HST and IF, its results are strongly dominated by the other methods and therefore not presented here.

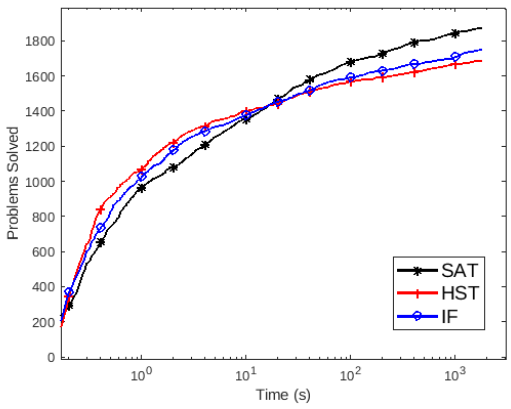

Figure 4: Cumulative number of problems solved by HST, IF, and the SAT-based method

Since the original implementation of IF is not publicly accessible, we have used the implementation of IF that is included in HST solver. As mentioned before, both HST and IF rely on a Linear Programming optimizer. As the optimizer for these methods we have used IBM ILOG CPLEX Optimization Studio 20.1[2].

Figure 4 shows the comparison of the cumulative number of problems solved within the time period of 0 to 1800 seconds. In total, our SAT-based method, HST, and IF solve 1875, 1689, 1751 problems, respectively. As can be seen in Figure 4, if the time limit is greater than approximately 17 seconds, our method outperforms both HST and IF.

Our results also demonstrate complementary solving power of our method and the competing methods. There are 240 problems solved by our method and not by IF, and 116 problems vice versa. Moreover, 296 problems are solved by our method and not by HST, and 110 problems vice versa. This complementarity was expected, as our SAT-based approach is inherently different from the Linear/Integer Programming approach used in both HST and IF.

Conclusion

We introduced a novel representation of relaxed plans for a given STRIPS planning problem, and proved that minimizing the cost over our new representations is equivalent to finding the optimal cost of relaxed plans for the original problem. We also provided an encoding method to translate our representation to SAT, as well as formulas needed for minimizing the cost of found solutions. According to our empirical results, our method is competitive with the state of the art, and shows better coverage in comparison to competing methods for benchmark STRIPS planning problems.

[2]https://www.ibm.com/products/ilog-cplex-optimization-studio

Acknowledgements

We would like to thank Patrik Haslum for assistance with his code for computing h^+.

References

Anbulagan; and Grastien, A. 2009. Importance of Variables Semantic in CNF Encoding of Cardinality Constraints. In Bulitko, V.; and Beck, J. C., eds., *Eighth Symposium on Abstraction, Reformulation, and Approximation, SARA 2009, Lake Arrowhead, California, USA, 8-10 August 2009*. AAAI.

Betz, C.; and Helmert, M. 2009. Planning with h^+ in theory and practice. In *KI 2009: Advances in Artificial Intelligence, 32nd Annual German Conference on AI, Paderborn, Germany, September 15-18, 2009. Proceedings*, volume 5803 of *Lecture Notes in Computer Science*, 9–16. Springer-Verlag.

Biere, A.; Fazekas, K.; Fleury, M.; and Heisinger, M. 2020. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020. In *Proceedings of SAT COMPETITION 2020*, 50–54.

Bonet, B. 2013. An admissible heuristic for SAS+ planning obtained from the state equation. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 2268–2274. IJCAI/AAAI.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1-2): 5–33.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2): 165–204.

Castro, M. P.; Piacentini, C.; Ciré, A. A.; and Beck, J. C. 2020. Solving delete free planning with relaxed decision diagram based heuristics. *Journal of Artificial Intelligence Research*, 67: 607–651.

Gebser, M.; Janhunen, T.; and Rintanen, J. 2014. SAT modulo graphs: acyclicity. In *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014*, volume 8761 of *Lecture Notes in Computer Science*, 137–151. Springer-Verlag.

Gefen, A.; and Brafman, R. I. 2011. The minimal seed set problem. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011*. AAAI Press.

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*. AAAI Press.

Haslum, P.; Slaney, J. K.; and Thiébaux, S. 2012. Minimal landmarks for optimal delete-free planning. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*. AAAI Press.

Helmert, M.; and Domshlak, C. 2009. Landmarks, critical paths and abstractions: what's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009*. AAAI Press.

Hoffmann, J.; and Nebel, B. 2001. The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Hunter, P.; and Kreutzer, S. 2007. Digraph measures: Kelly decompositions, games, and orderings. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007*, 637–644. SIAM.

Imai, T.; and Fukunaga, A. 2015. On a practical, integer-linear programming model for delete-free tasks and its use as a heuristic for cost-optimal planning. *Journal of Artificial Intelligence Research*, 54: 631–677.

Karpas, E.; and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 1728–1733.

Keyder, E.; and Geffner, H. 2008. Heuristics for Planning with Action Costs Revisited. In *ECAI 2008 - 18th European Conference on Artificial Intelligence*, 588–592. IOS Press.

Mirkis, V.; and Domshlak, C. 2007. Cost-sharing approximations for h+. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007*, 240–247. AAAI Press.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 2357–2364. IJCAI/AAAI.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014*. AAAI Press.

Rankooh, M. F.; and Rintanen, J. 2022. Propositional encodings of acyclicity and reachability by using vertex elimination. In *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence*. AAAI Press.

Rintanen, J. 2012. Planning as satisfiability: heuristics. *Artificial Intelligence*, 193: 45–86.

Robinson, N.; McIlraith, S. A.; and Toman, D. 2014. Cost-based query optimization via AI planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2344–2351. AAAI Press.

Rose, D. J.; and Tarjan, R. E. 1975. Algorithmic aspects of vertex elimination. In *Proceedings of the 7th Annual ACM Symposium on Theory of Computing, May 5-7, 1975, Albuquerque, New Mexico, USA*, 245–254. ACM.

Scala, E.; Haslum, P.; Magazzeni, D.; and Thiébaux, S. 2017. Landmarks for numeric planning problems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, 4384–4390. ijcai.org.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, 651–665. Springer-Verlag.