Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)
Sister Conferences Best Papers Track

Capturing (Optimal) Relaxed Plans with Stable and Supported Models of Logic Programs (Extended Abstract)*

Masood Feyzbakhsh Rankooh and **Tomi Janhunen**
Tampere University, Tampere, Finland
{masood.feyzbakhshrankooh, tomi.janhunen}@tuni.fi

Abstract

We establish a novel relation between delete-free planning, an important task for the AI Planning community also known as relaxed planning, and logic programming. We show that given a planning problem, all subsets of actions that could be ordered to produce relaxed plans for the problem can be bijectively captured with stable models of a logic program describing the corresponding relaxed planning problem. We also consider the supported model semantics of logic programs, and introduce one causal and one diagnostic encoding of the relaxed planning problem as logic programs, both capturing relaxed plans with their supported models. Our experimental results show that these new encodings can provide major performance gain when computing optimal relaxed plans.

1 Introduction

AI Planning is the task of finding a sequence of actions, called a plan, that when applied to a given initial state transforms it to a state that satisfies all members of a given set of goal conditions. According to the STRIPS formulation of AI Planning, states and goal conditions are represented by sets of atomic propositions, and each action can have separate sets of atomic propositions as its preconditions, positive effects, and negative effects. Delete-free planning problems are those for which actions have no negative effects. A given planning problem can be relaxed into a delete-free problem, optimal solving of which provides lower bound of the optimal cost of the original problem. However, computing this lower bound, denoted by h^+, is NP-equivalent [Bylander, 1994]. Also, h^+ is hard to approximate [Betz and Helmert, 2009].

Computing h^+ in an efficient way is important for multiple reasons. There have been many admissible heuristic functions that approximate h^+ in polynomial time by computing lower bounds. Examples are the 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]. The informativeness of these heuristic functions cannot be evaluated unless we can compute the exact value of h^+. Also, efficient solving of relaxed planning problems is in itself of importance, as there exist planning tasks of interest for the AI community whose actions are all delete-free [Gefen and Brafman, 2011; Robinson *et al.*, 2014]. Another reason for the importance of efficient optimal relaxed planning is that optimal plans for non-relaxed planning problems can always be produced by iterative solving and reformulating relaxed planning tasks [Haslum, 2012].

Several approaches to solving relaxed planning problems have previously been introduced. The approaches include Boolean satisfiability (SAT) based encodings [Rankooh and Rintanen, 2022b], integer programming based models [Imai and Fukunaga, 2015; Rankooh and Rintanen, 2022a]; and a minimum-cost hitting set based method introduced by Haslum et al. [2012]. Here, we take an approach based on the stable and supported models of logic programs [Gelfond and Lifschitz, 1988; Marek and Subrahmanian, 1992]. Such models provide the semantical basis for answer set programming (ASP); see, e.g., [Brewka *et al.*, 2011] for an overview. The ASP paradigm offers rule-based general-purpose modeling languages for knowledge representation and reasoning.

Both stable and supported models implement a form of minimality, i.e., atomic propositions are *false by default*. This is highly useful in the context of AI planning since state predicates are falsified in this sense and the encodings of planning problems can concentrate on specifying which state predicates become true or remain true inertially. This tends to lead to more compact encodings compared to those based on pure SAT and, furthermore, enable memory savings if native answer set solvers are used for actual computations.

In this work, we give an encoding of relaxed planning problems in ASP. We show that all relaxed plans can be bijectively captured with stable models of the produced logic program. While the supported model semantics of logic programs cannot be directly employed for this purpose, we show how by guaranteeing acyclicity in an underlying graph of the logic program, one may deploy supported models to harvest (optimal) relaxed plans of the planning problem. The logic program produced in this way inherits the causal nature of our stable model based encoding, in the sense that the direction of explanations provided by the rules is from causes/preconditions to effects. By reversing this direction, we

*This is an extended abstract of [Rankooh and Janhunen, 2023] that received the Best Paper Award of ICLP 2023.

establish a diagnostic encoding, which while still using the supported model semantics of logic programs, is shown to be more efficient than our causal encoding by our empirical study. Our experimental results show that our diagnostic supported model based encoding enables CLASP [Gebser *et al.*, 2015] to solve more problems compared to the integer programming solver based state-of-the-art method.

2 Preliminaries

A STRIPS-style planning problem is a 5-tuple $\Pi = \langle X, I, A, G, cost\rangle$ where X is a finite set of Boolean state variables, also called *atomic propositions*. The initial state I and the set of goal conditions G are subsets of X. The finite set A is the set of actions. Each member $\vec{a}$ of A is a triple $\langle pre(\vec{a}), add(\vec{a}), del(\vec{a})\rangle$, where $pre(\vec{a})$, $add(\vec{a})$ and $del(\vec{a})$ are sets of atomic propositions denoting the set of preconditions, positive effects, and negative effects of $\vec{a}$, which are the atomic propositions that $\vec{a}$ requires, adds, and deletes, respectively. The cost function $cost$ maps members of A to non-negative integers. We use the vector sign to distinguish actions from the corresponding atoms that represent them in logic programs.

States are represented as subsets of X. The successor $s' = exec_{\vec{a}}(s)$ of a state s with respect to action $\vec{a} \in A$ is defined if $pre(\vec{a}) \subseteq s$, where the definition of s' is $(s \setminus del(\vec{a})) \cup add(\vec{a})$. An action sequence $\vec{a_1}, ..., \vec{a_n}$ is executable (in a state s) if $exec_{\vec{a_1},...,\vec{a_n}}(s) = exec_{\vec{a_n}}(...exec_{\vec{a_2}}(exec_{\vec{a_1}}(s))...)$ 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 $\pi = \vec{a_1}, ..., \vec{a_n}$ for Π, is defined by $\Sigma_{i=1,...,n} cost(\vec{a_i})$. An optimal plan for Π is a plan with minimal cost.

For a given STRIPS planning problem $\Pi = \langle X, I, A, G, cost\rangle$, the delete relaxation [Bonet and Geffner, 2001] is defined as $\Pi^+ = \langle X, I, A^+, G, cost\rangle$, where A^+ is defined from A by replacing the set of negative effects of each member of A with the empty set. Without loss of generality, we can define $\Pi^+ = \langle X, \emptyset, A^+, G, cost\rangle$, with an additional requirement that all members of I have been removed from G, and also from the preconditions and effects of members of A^+.

A plan for Π^+ is called a relaxed plan for the original problem Π. The minimal cost of plans of Π^+ is denoted by $h^+(\Pi)$. If there is no relaxed plan for Π, we set $h^+(\Pi)$ to ∞.

In order to translate planning problems to logic programs, we consider logic programs that consist of *rules* of the forms:

$$a \leftarrow b_1, \ldots, b_n, \texttt{not } c_1, \ldots, \texttt{not } c_m. \quad (1)$$

$$\{a\} \leftarrow b_1, \ldots, b_n, \texttt{not } c_1, \ldots, \texttt{not } c_m. \quad (2)$$

The symbols $a, b_1, \ldots, b_n$ with $n \geq 0$, and $c_1, \ldots, c_m$ with $m \geq 0$ occurring in the rules are (propositional) *atoms* and "not" denotes *negation by default*. Rules of the forms (1) and (2) are known as *normal* and *choice rules*, respectively [Simons *et al.*, 2002]. Intuitively, each rule r gives a reason to derive its *head* $\mathrm{head}(r) = a$ if the conditions in its *body* $\mathrm{body}(r)$ are met, i.e., atoms involved can be either derived or not by other rules. For a choice rule r of form (2), the derivation of $\mathrm{head}(r)$ is optional, enabling an exception to $\mathrm{head}(r)$ being false by default. We write $\mathrm{body}^+(r)$ and $\mathrm{body}^-(r)$ for the sets of atoms $b_1, \ldots, b_n$ (resp. $c_1, \ldots, c_m$) occurring positively (resp. negatively) in $\mathrm{body}(r)$. We say that r is a positive rule if $\mathrm{body}^-(r)$ is empty.

The *signature* of a logic program P, denoted by $\mathrm{At}(P)$, is the set of atoms that occur in P. The *positive dependency graph* of P is $\mathrm{DG}^+(P) = \langle \mathrm{At}(P), \succeq\rangle$ where $a \succeq b$ holds for $a, b \in \mathrm{At}(P)$ if $\mathrm{head}(r) = a$ and $b \in \mathrm{body}^+(r)$ for some rule $r \in P$. If $a \succeq b$, we say that a depends on b, and also denote this by $\langle a, b\rangle \in \mathrm{DG}^+(P)$.

An *interpretation* $I \subseteq \mathrm{At}(P)$ determines which atoms $a \in \mathrm{At}(P)$ are *true* ($a \in I$) and which are *false* ($a \notin I$). Then I satisfies a rule $r \in P$ of form (1), denoted $I \models r$, if the satisfaction of the body, denoted $I \models \mathrm{body}(r)$, implies that $\mathrm{head}(r) \in I$, i.e., $I \models \mathrm{head}(r)$. For a choice rule r of form (2), $I \models r$ unconditionally. Moreover, the interpretation I is a (classical) *model* of P if $I \models r$ holds for every $r \in P$. Each positive normal program P has a unique *least model* $\mathrm{LM}(P)$ obtained as the intersection $\bigcap\{I \subseteq \mathrm{At}(P) \mid I \models P\}$.

Given an interpretation I, the *reduct* r^I of r with respect to I is obtained by partially evaluating the negative conditions of r. For a normal rule (1), $r^I = \emptyset$ if $c_i \in I$ for some $1 \leq i \leq m$ and $r^I = \{a \leftarrow b_1, \ldots, b_n\}$ otherwise. For a choice rule (2), the latter case additionally requires that $a \in I$. Finally, for an entire logic program P, the reduct $P^I = \bigcup_{r \in P} r^I$ and I is a *stable model* of P iff $I = \mathrm{LM}(P^I)$. We distinguish the *supporting rules* of P with respect to I, denoted by $\mathrm{SR}_P(I)$, which are the normal rules whose bodies are satisfied, and the choice rules whose bodies and heads are satisfied. A model $I \models P$ is *supported* (by P) when $I = \{\mathrm{head}(r) \mid r \in \mathrm{SR}_P(I)\}$. Each stable model of P is supported, but supported models are not necessarily stable, such as $I = \{a\}$ for $P = \{a \leftarrow a.\}$.

3 Relaxed Plans Captured with Stable Models of Logic Programs

Let $\Pi = \langle X, I, A, G, cost\rangle$ be a STRIPS planning problem, $\Pi^+ = \langle X, \emptyset, A^+, G, cost\rangle$ be the delete relaxation of Π, and P be a logic program consisting of rules of the form (1) $g \leftarrow \texttt{not } g$ for every $g \in G$; (2) $\{a\} \leftarrow q_1, \ldots, q_n$ for every $\vec{a} \in A$ with $pre(\vec{a}) = \{q_1, \ldots, q_n\}$; (3) $p \leftarrow a$ for every $\vec{a} \in A$ and $p \in add(\vec{a})$. Intuitively, the first rule guarantees all goal atoms to be true in a model. The second rule explains the necessary conditions for the execution of an action $\vec{a}$. The third rule enforces the positive effects in case $\vec{a}$ has been chosen to be in the model. We formally show that P captures the relaxed plans of Π as its stable models.

Theorem 1. *There is a bijection $f(A') = \bigcup_{\vec{a} \in A'}(add(\vec{a}) \cup \{a\})$ between all subsets A' of A^+ which can be ordered to produce a relaxed plan for Π, and all stable models of P.*

Theorem 1 shows that if P is augmented with an optimization constraint requiring minimization over the summation of the costs of actions in the answer sets, the cost of an optimal stable model of P is equal to $h^+(\Pi)$. The program P can be seen as a *causal* encoding of relaxed plans of P. That is because the direction of relaxed plan computation in P is from preconditions to actions, and from actions to effects. In other words, the direction is from causes to effects.

4 Relaxed Plans Captured with Supported Models of Logic Pograms

In this section, we recall the instrumentation of logic programs with acyclicity constraint, which allows capturing the stable models of a given logic program P with the supported models of another program $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ [Bomanson *et al.*, 2016]. We provide an adaptation of this method based on the structure of program P explained above. We then review the vertex elimination method, used previously for cycle prevention in the produced models of SAT formulas [Rankooh and Rintanen, 2022c; Rankooh and Janhunen, 2022]. We next show how this method could also be used to translate $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ to new programs whose supported models represent stable models of P and relaxed plans of Π.

4.1 Instrumentation of Logic Programs with Acyclicity Constraint

We adopt the *acyclicity translation* $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ of a logic program P [Bomanson *et al.*, 2016] that deploys special *dependency atoms* $\mathtt{dep}(x, y)$ to express the activation of the respective arc $\langle x, y\rangle \in \mathrm{DG}^+(P)$ in the acyclicity constraint. For the sake of the compactness of the output program, we circumvent the introduction of dependency atoms for actions, by establishing dependencies only between atoms of the original planning problem. This way, the underlying graphs for which acyclicity must be guaranteed become considerably smaller than $\mathrm{DG}^+(P)$.

The idea is to *instrument* P explained in the previous section with additional rules that capture *well-support* for atoms $p \in X$. For each pair $\langle p, q\rangle$, if there exists $\vec{a} \in A$ such that $p \in add(a)$ and $q \in pre(a)$, the potential dependency of p on q is expressed using a choice rule $\{\mathtt{dep}(p, q)\} \leftarrow q$. Also, atoms $\mathtt{ws}(a_1, p), \ldots, \mathtt{ws}(a_k, p)$, for actions $\{\vec{a}_1, \ldots, \vec{a}_k\}$ that add p enforce the well-support for p in terms of k rules $p \leftarrow \mathtt{ws}(a_i, p)$ for $i = 1, ..., k$. For an atom $p \in X$, the rule (3) below captures the option that the well-support for p is provided by some action $\vec{a}$ such that $pre(\vec{a}) = \{q_1, \ldots, q_n\}$ and $p \in add(\vec{a})$.

$$\{\mathtt{ws}(a, p)\} \leftarrow \mathtt{dep}(p, q_1), \ldots, \mathtt{dep}(p, q_n). \tag{3}$$

Also, the rule $a \leftarrow \mathtt{ws}(a, p)$ captures the atom a in the supported models, in the case that it has been used to provide well-support for p. As in program P, we need a rule $g \leftarrow \mathtt{not}\ g$ for every $g \in G$ to guarantee that every goal atom has been produced by some action.

For $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ obtained in this way, the distinction between stable and supported models disappears if we insist on *acyclic models* I for which the digraph induced by the set of arcs $\{\langle a, b\rangle \mid \mathtt{dep}(a, b) \in I\}$ is acyclic. We deploy the following result:

Proposition 1 ([Bomanson *et al.*, 2016])**.** *If M is a stable model of P, then $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ has an acyclic supported model N such that $M = N \cap \mathrm{At}(P)$. If N is an acyclic supported model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$, then $M = N \cap \mathrm{At}(P)$ is a stable model of P.*

Similarly to the stable model based encoding, $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ is a causal encoding, expressing the inference in the direction from preconditions to actions, and from actions to effects.

4.2 Vertex Elimination Graphs

The concept of vertex elimination for digraphs was originally introduced by Rose and Tarjan [1975]. Given a digraph $\mathcal{G} = \langle V, E\rangle$, an ordering of V is a bijection $\alpha : \{1, \ldots, n\} \to V$. For a vertex v, the *fill-in* of v, denoted by $F(v)$, is the set of arcs from the in-neighbors of v to the out-neighbors of v, formally defined by

$$F(v) = \{\langle x, y\rangle \mid \langle x, v\rangle \in E, \langle v, y\rangle \in E, x \neq y\}. \tag{4}$$

The *v-elimination* graph of $\mathcal{G}$ is produced by removing v from $\mathcal{G}$, and adding the fill-in of v to the resulting graph. Formally, $\mathcal{G}(v) = \langle V \setminus \{v\}, E(v) \cup F(v)\rangle$, where $E(v) = \{\langle x, y\rangle \mid \langle x, y\rangle \in E, x \neq v, y \neq v\}$.

Given a digraph $\mathcal{G}$ and an ordering α of its vertices, the *elimination process* of $\mathcal{G}$ according to α is the sequence $\mathcal{G} = \mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_{n-1}$, where $\mathcal{G}_i$ is the $\alpha(i)$-elimination graph of $\mathcal{G}_{i-1}$ for $i = 1, \ldots, n-1$.

The fill-in of the digraph $\mathcal{G}$ according to α, denoted by $F_\alpha(\mathcal{G})$, is the set of all arcs added to $\mathcal{G}$ in the vertex elimination process. Formally, $F_\alpha(\mathcal{G})$ is defined by (5), where $F_{i-1}(\alpha(i))$ is the fill-in of $\alpha(i)$ in $\mathcal{G}_{i-1}$, with $\mathcal{G}_0 = \mathcal{G}$:

$$F_\alpha(\mathcal{G}) = \bigcup_{i=1}^{|V|-1} F_{i-1}(\alpha(i)). \tag{5}$$

The vertex elimination graph of $\mathcal{G}$ according to α, denoted by $\mathcal{G}^*_\alpha$, is the union of all graphs produced in the elimination process of $\mathcal{G}$ according to α:

$$\mathcal{G}^*_\alpha = \langle V, E \cup F_\alpha(\mathcal{G})\rangle. \tag{6}$$

One important property of vertex elimination is that if the original graph $\mathcal{G}$ has a directed cycle, then $\mathcal{G}^*_\alpha$ will have a cycle of length 2, regardless of the ordering α.

4.3 The Causal Encoding

Consider $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ explained above. Let $\mathcal{G}$ be the graph of all dependencies of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$. Formally, $\mathcal{G} = \langle X, E\rangle$, where $E = \{\langle p, q\rangle \mid \mathtt{dep}(p, q) \in \mathrm{At}(\mathrm{Tr}_{\mathrm{ACYC}}(P))\}$. Also, for each supported model M of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$, let $\mathcal{G}_M$ be the graph of all dependencies in M, i.e., $\mathcal{G}_M = \langle X, E_M\rangle$, where $E_M = \{\langle p, q\rangle \mid \mathtt{dep}(p, q) \in M\}$. Assume that α is an ordering of the members of X, $\mathcal{G} = \mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_{n-1}$ is the elimination process of $\mathcal{G}$ according to α, and for $i = 1, \ldots, n$, $F_{i-1}(\alpha(i))$ is the fill-in of $\alpha(i)$ in $\mathcal{G}_{i-1}$. Let $\mathcal{G}^*_\alpha = \langle X, E^*\rangle$ and $\mathcal{G}^*_{M,\alpha} = \langle X, E^*_M\rangle$ be the vertex elimination graphs of $\mathcal{G}$ and $\mathcal{G}_M$ according to α, respectively.

We produce the causal supported model semantics based encoding of Π as logic program P_c by adding the following rules to $\mathrm{Tr}_{\mathrm{ACYC}}(P)$. For every $\langle p, q\rangle \in F_{i-1}(\alpha(i))$, add

$$\mathtt{dep}(p, q) \leftarrow \mathtt{dep}(p, \alpha(i)), \mathtt{dep}(\alpha(i), q). \tag{7}$$

Also, for every p and q such that $\langle p, q\rangle \in \mathcal{G}^*_\alpha$ and $\langle q, p\rangle \in \mathcal{G}^*_\alpha$, we add

$$f \leftarrow \mathtt{dep}(p, q), \mathtt{dep}(q, p), \mathtt{not}\ f. \tag{8}$$

Intuitively, for any vertex ordering α, and any supported model M of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$, the rule (7) extends M by atoms representing the arcs in $\mathcal{G}^*_{M,\alpha}$, the vertex elimination graph of $\mathcal{G}_M$ according to α, while the rule (8) guarantees that $\mathcal{G}^*_{M,\alpha}$ has no cycle of length 2. We have the following result:

Theorem 2. *There exists a permutation π of members of $A' \subseteq A^+$ such that π is a relaxed plan for Π iff P_c has a supported model M such that $A' = \{\vec{a} \in A^+ \mid a \in M\}$.*

4.4 The Diagnostic Encoding

Inferring causes from effects can be understood as diagnostic inference [Russell and Norvig, 2020]. In our causal encoding, we express the inference direction from preconditions to dependencies, from dependencies to well-supports, and from well-supports to effects. We can alternatively reverse all these directions to produce a diagnostic encoding.

In our diagnostic encoding P_d, we assume that all atoms could possibly be in the model by using the rule $\{p\}$ for every $p \in X$. However, if p is in the model, then it must have well-support by at least one action. We establish this by adding $\{\texttt{ws}(a,p)\} \leftarrow p$ for every $\vec{a} \in A$ such that $p \in add(\vec{a})$, and also $f \leftarrow p, \texttt{not ws}(a_1,p), \ldots, \texttt{not ws}(a_m,p), \texttt{not } f$ for $p \in X$ and all actions $\vec{a}_1, ..., \vec{a}_m$ that could add p. To represent the inference from well-supports to dependencies, we add $\texttt{dep}(p,q) \leftarrow \texttt{ws}(a,p)$ for $\vec{a} \in A$, $q \in pre(\vec{a})$, and $p \in add(\vec{a})$. Finally, to establish the inference direction from dependencies to preconditions, we add $q \leftarrow \texttt{dep}(p,q)$. As in P_c, all rules in the forms of (7) and (8) must be included to enforce acyclicity in the supported model. Moreover, we add $a \leftarrow \texttt{ws}(a,p)$ for $\vec{a} \in A$ and $p \in add(\vec{a})$, to enable an action atom a to represent its cost in the minimization constraint, and also $g \leftarrow \texttt{not } g$ for every $g \in G$ to guarantee that goal atoms are included in the model.

It is quite easy to check that if P_d has a supported model M, then M is also a supported model of P_c. On the other hand, it can be shown in a straightforward manner that if N is a supported model of P_c, then $N \setminus L$ is a supported model of P_d, where L is the set of atoms $\texttt{dep}(p,q)$ for which there is no action $\vec{a}$ such that $\texttt{ws}(a,p) \in N$ and $q \in pre(\vec{a})$. Thus, we have the following result:

Theorem 3. *There exists a permutation π of members of $A' \subseteq A^+$ such that π is a relaxed plan for Π iff P_d has a supported model M such that $A' = \{\vec{a} \in A^+ \mid a \in M\}$.*

5 Empirical Results

We have implemented our encoding methods inside the HSP* planner [Haslum, 2015]. The implementation is available under the ASPTOOLS collection[1]. All experiments have been run on a cluster of Linux machines with Intel Xeon 2.40 GHz CPUs, using a timeout of 1800 seconds per problem, and a memory limit of 8 GB. For our supported model based encodings, where vertex elimination is used, for determining the order of vertex elimination, we have implemented the *minimum degree* heuristic, i.e., eliminating a vertex with a minimal total number of incoming and outgoing arcs in the graph produced after the elimination of previously eliminated vertices.

Our three implemented encodings are (1) our stable model based encoding P; (2) our causal supported model based encoding P_c; and (3) our diagnostic supported model based encoding P_d. As the solver we use CLASP 3.3.5, which is capable of optimizing over both stable and supported models. The CLASP solver searches for stable models by default. We enable the search for supported models only for our P_c and P_d encodings. As the optimization strategy we use the unsatisfiable core (USC) method. Henceforth, we refer to the solver obtained by combing CLASP with our P, P_c, and P_d encodings simply by the name of the corresponding encoding.

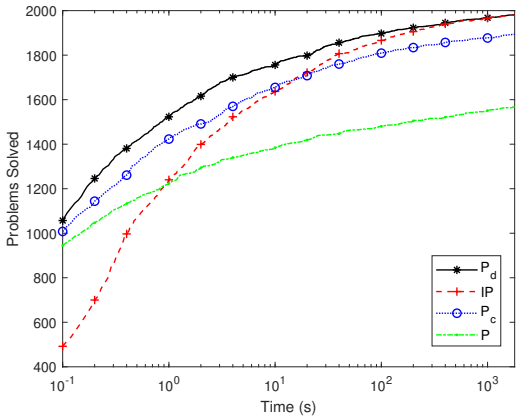

Figure 1: Cumulative numbers of problems solved by the competing methods

To evaluate the efficiency of our methods, we have compared them based on the total time of encoding and solving with IP, the integer programming based encoding by Rankooh and Rintanen [2022a], which uses IBM ILOG CPLEX Optimization Studio 20.1[2] as the optimizer.

As benchmark problem sets, we use the STRIPS planning problem sets of the *planning repository*[3]. In total, 2212 problem instances from 84 problem sets are used for comparison.

The cumulative number of problems solved by all methods are presented in Figure 1. Our supported model based encodings significantly outperform the stable model based one, with the diagnostic encoding performing visibly faster than the causal one. Also, even though the number of problems solved within 1800 seconds by our diagnostic encoding is not much higher than that of IP, P_d solves problems considerably faster than IP. In fact, regardless of the time limit, P_d solves more problems compared to any other solver.

6 Conclusions

In this work, we study the previously uninvestigated application of ASP solvers to optimal relaxed planning. Three different encodings of relaxed planning problems into logic programs are provided, one based on the stable model semantics, and two based on the supported model semantics of logic programs. According to our empirical results, all our encodings enable CLASP to outperform the state-of-the-art method if the time limit is small. Moreover, our diagnostic supported model based method outperforms the state-of-the-art solver on the studied benchmark problems regardless of the used time limit.

[1] https://github.com/asptools/software

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

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

Acknowledgements

Financial support from the Research Council of Finland (Project XAILOG, #345633) is gratefully acknowledged.

References

[Betz and Helmert, 2009] Christoph Betz and Malte Helmert. Planning with h^+ in theory and practice. In Bärbel Mertsching, Marcus Hund, and Muhammad Zaheer Aziz, editors, *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*, pages 9–16. Springer, 2009.

[Bomanson *et al.*, 2016] Jori Bomanson, Martin Gebser, Tomi Janhunen, Benjamin Kaufmann, and Torsten Schaub. Answer set programming modulo acyclicity. *Fundam. Informaticae*, 147(1):63–91, 2016.

[Bonet and Geffner, 2001] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[Bylander, 1994] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[Gebser *et al.*, 2015] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Javier Romero, and Torsten Schaub. Progress in clasp series 3. In *LPNMR 2015*, pages 368–383, 2015.

[Gefen and Brafman, 2011] Avitan Gefen and Ronen I. Brafman. The minimal seed set problem. In Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp, and Malte Helmert, editors, *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*. AAAI, 2011.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of ICLP'88*, pages 1070–1080, 1988.

[Haslum *et al.*, 2012] Patrik Haslum, John K. Slaney, and Sylvie Thiébaux. Minimal landmarks for optimal delete-free planning. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*, pages 353–357. AAAI Press, 2012.

[Haslum, 2012] Patrik Haslum. Incremental lower bounds for additive cost planning problems. In Lee McCluskey, Brian Charles Williams, José Reinaldo Silva, and Blai Bonet, editors, *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI, 2012.

[Haslum, 2015] Patrik Haslum. Hsp* code and documentation http://users.cecs.anu.edu.au/patrik/un-hsps.html., 2015.

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI, 2009.

[Imai and Fukunaga, 2015] Tatsuya Imai and Alex Fukunaga. 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, 2015.

[Keyder and Geffner, 2008] Emil Keyder and Hector Geffner. Heuristics for planning with action costs revisited. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors, *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 588–592. IOS Press, 2008.

[Marek and Subrahmanian, 1992] Victor Witold Marek and Venkatramanan Siva Subrahmanian. The relationship between stable, supported, default and autoepistemic semantics for general logic programs. *Theor. Comput. Sci.*, 103(2):365–386, 1992.

[Mirkis and Domshlak, 2007] Vitaly Mirkis and Carmel Domshlak. Cost-sharing approximations for h+. In Mark S. Boddy, Maria Fox, and Sylvie Thiébaux, editors, *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, pages 240–247. AAAI, 2007.

[Rankooh and Janhunen, 2022] Masood Feyzbakhsh Rankooh and Tomi Janhunen. Efficient computation of answer sets via SAT modulo acyclicity and vertex elimination. In Georg Gottlob, Daniela Inclezan, and Marco Maratea, editors, *Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings*, volume 13416 of *Lecture Notes in Computer Science*, pages 203–216. Springer, 2022.

[Rankooh and Janhunen, 2023] Masood Feyzbakhsh Rankooh and Tomi Janhunen. Capturing (optimal) relaxed plans with stable and supported models of logic programs. *Theory Pract. Log. Program.*, 23(4):1–15, 2023.

[Rankooh and Rintanen, 2022a] Masood Feyzbakhsh Rankooh and Jussi Rintanen. Efficient computation and informative estimation of h+ by integer and linear programming. In Akshat Kumar, Sylvie Thiébaux, Pradeep Varakantham, and William Yeoh, editors, *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24, 2022*, pages 71–79. AAAI Press, 2022.

[Rankooh and Rintanen, 2022b] Masood Feyzbakhsh Rankooh and Jussi Rintanen. Efficient encoding of cost

optimal delete-free planning as SAT. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Virtual Event, February 22 - March 1, 2022*, pages 9910–9917. AAAI Press, 2022.

[Rankooh and Rintanen, 2022c] Masood Feyzbakhsh Rankooh and Jussi Rintanen. Propositional encodings of acyclicity and reachability by using vertex elimination. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, 2022 Virtual Event, February 22 - March 1, 2022*, pages 5861–5868. AAAI Press, 2022.

[Robinson *et al.*, 2014] Nathan Robinson, Sheila A. McIlraith, and David Toman. Cost-based query optimization via AI planning. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 2344–2351. AAAI Press, 2014.

[Rose and Tarjan, 1975] Donald J. Rose and Robert Endre Tarjan. Algorithmic aspects of vertex elimination. In *Proceedings of the 7th Annual ACM Symposium on Theory of Computing*, pages 245–254, 1975.

[Russell and Norvig, 2020] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

[Simons *et al.*, 2002] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.