

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

MASOOD FEYZBAKSH RANKOOH and TOMI JANHUNEN

Tampere University, Tampere, Finland

(e-mails: masood.feyzbakshrankooh@tuni.fi, tomi.janhunen@tuni.fi)

submitted 1 June 2023; revised 22 June 2023; accepted 22 June 2023

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, with our diagnostic encoding outperforming state-of-the-art approaches to relaxed planning regardless of the given time limit when measured on a wide collection of STRIPS planning benchmarks.

KEYWORDS: planning heuristics, answer set programming, optimal delete-free planning, supported models, acyclicity constraint, dependency graph, vertex elimination

1 Introduction

AI planning, an active research area of artificial intelligence, 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 (also called add effects), and negative effects (also called delete 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. This lower bound, denoted by h^+ , could be used as a heuristic in an A*-like search scheme to find an optimal solution for the original problem. Computing h^+ is, however, NP-equivalent (Bylander 1994). Also, h^+ is hard to approximate (Betz and Helmert 2009).

Optimally solving relaxed planning problems 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 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} (Mirakis and Domshlak 2007). The informativeness of these heuristic functions cannot be evaluated unless we can compute the exact value of h^+ . Using such a measure for informativeness could lead to devising more informative heuristic functions. Moreover, efficient solving of relaxed planning problems is in itself of importance, because there exist planning tasks of interest for the AI community whose actions are all delete-free. Examples of such tasks are the minimal seed-set problem (Gefen and Brafman 2011) and the problem of determining join orders in relational database query plan generation (Robinson *et al.* 2014). Another reason for the importance of efficient optimal relaxed planning is the fact that optimal plans for non-relaxed planning problems can always be produced by iterative solving and reformulating relaxed planning tasks (Haslum 2012). By repeatedly finding optimal plans for newly produced relaxed problems, while reformulating the non-relaxed problem in each iteration, one can reach a point where the found optimal plan for the last relaxed problem is actually an optimal plan for the original problem.

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 (Haslum *et al.* 2012). In this work we take a new 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, for example, the overview by Brewka *et al.* (2011). The ASP paradigm offers general-purpose modeling languages for knowledge representation and reasoning.

A typical encoding of a search problem in ASP aims at a one-to-one correspondence between answer sets and the solutions of the problem. This is in perfect harmony with AI planning where sequences of actions (plans) form solutions to problems at hand. Indeed, many AI planning problems have been encoded as logic programs (Son and Balduccini 2018) and AI planning also played a role in the early development of the ASP paradigm (Lifschitz 1999) in the first place. Both stable and supported models implement a form of minimality, that is, 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. The difference between stable and supported models is also interesting in this setting, since ASP solvers may compute answer sets based on either semantics. Stable models are also supported models but not vice versa in general. The gap between the two semantics vanishes if a logic program is suitably instrumented, for example, in terms of acyclicity constraints (Bomanson *et al.* 2016). These observations open up new avenues when it comes to encoding planning problems as logic programs as well as choosing an approach for computing plans as answer sets.

In this work, we establish a new relation between relaxed planning and logic programs. We give an encoding of relaxed planning problems in ASP. We show that all subsets of actions that could be ordered to produce relaxed plans can be bijectively captured with stable models of the produced logic program. This enables the previously uninvestigated

usage of off-the-shelf answer set solvers for computing the value of h^+ . 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 provide 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 when given small time limits these new encodings can significantly outperform the previous approaches to relaxed planning when measured on STRIPS planning benchmarks. Moreover, regardless of the used time limit, 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.

Logic programming has recently been employed for computing heuristics for lifted planning tasks. Corrêa *et al.* 2021; 2022 employed Datalog programs to calculate h^{add} (Bonet and Geffner 2001) and h^{FF} (Hoffmann and Nebel 2001), respectively, for lifted planning tasks. However, the objective of our work differs from theirs. While both h^{add} and h^{FF} are non-admissible estimations of h^+ and can be computed in polynomial time for ground instances, we aim to compute h^+ itself. Furthermore, this work focuses on ground planning tasks. Although the generalization of our current approach to lifted planning is relatively simple, we leave it for future research.

The rest of this article is organized as follows. In Section 2, we recall basic concepts and definitions of planning problems, relaxed planning, logic programs, and their stable and supported model semantics. Then, in Section 3, we show how relaxed plans can be captured with stable models of an encoding of relaxed planning problems into logic programs. In Section 3, we first show how a logic program can be augmented with a dynamically varying digraph whose acyclicity guarantees a shift in the semantics from stable models to supported models. We then recall how vertex elimination can be used to check whether a given digraph is acyclic. Based on the supported model semantics and the vertex elimination method, we explain our causal and diagnostic encodings of relaxed planning problems. We present practical evidence in Section 5 based on an experimental evaluation of the resulting encoding for answer set and supported model optimization. This analysis is based on 2212 problem instances from 84 STRIPS planning problem sets. Finally, we conclude the paper in Section 6.

2 Preliminaries

Since we intend to establish a connection between AI planning and ASP, we provide necessary formal definitions with respect to both of these paradigms.

2.1 AI planning and relaxed plans

A STRIPS 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 a non-negative integer. 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 is $s' = (s \setminus del(\vec{a})) \cup add(\vec{a})$. An action sequence $\vec{a}_1, \dots, \vec{a}_n$ is executable (in state s) if $exec_{\vec{a}_1, \dots, \vec{a}_n}(s) = exec_{\vec{a}_n}(\dots exec_{\vec{a}_2}(exec_{\vec{a}_1}(s))\dots)$ 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 $\pi = \vec{a}_1, \dots, \vec{a}_n$ for Π , is defined by $\sum_{i=1, \dots, n} cost(\vec{a}_i)$. 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^+ . We use this latter definition of relaxation in the rest of the paper.

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 ∞ .

2.2 Answer set programming

In this work, we consider logic programs that consist of *rules* of the forms:

$$a \leftarrow b_1, \dots, b_n, \text{not } c_1, \dots, \text{not } c_m, \quad (1)$$

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

The symbols a, b_1, \dots, b_n with $n \geq 0$, and c_1, \dots, 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* $head(r) = a$ if the conditions in its *body* $body(r)$ are met, that is, atoms involved can be either derived or not by other rules. For a choice rule r of form (2), the derivation of $head(r)$ is optional, enabling an exception to $head(r)$ being false by default. We write $body^+(r)$ and $body^-(r)$ for the sets of atoms b_1, \dots, b_n (resp. c_1, \dots, c_m) occurring positively (resp. negatively) in $body(r)$. We say that r is a positive rule if $body^-(r)$ is empty.

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

An *interpretation* $I \subseteq At(P)$ determines which atoms $a \in At(P)$ 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 $I \models body(r)$, implies that $head(r) \in I$, that is, $I \models head(r)$. 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* $\text{LM}(P)$ obtained as the intersection $\bigcap \{I \subseteq \text{At}(P) \mid I \models P\}$.

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 $r^I = \{a \leftarrow b_1, \dots, b_n\}$ 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 $I = \text{LM}(P^I)$. For the purposes of this work, it is also useful to distinguish the *supporting rules* of P with respect to I , denoted by $\text{SR}_P(I)$, which are the normal rules whose bodies are satisfied, and the choice rules whose bodies and heads are satisfied. Then, a model $I \models P$ is *supported* (by P) when $I = \{\text{head}(r) \mid r \in \text{SR}_P(I)\}$. 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

Typically, modeling planning problems as answer set programs is done by assuming a number of time steps for the output plan, which is also mirrored in the structure of the produced logic program (Son *et al.* 2006). Here, however, we show that, as long as finding relaxed plans are concerned, one can encode the planning problem in such a way that there will be no need for a multi-step structure.

Let $\Pi = \langle X, I, A, G, \text{cost} \rangle$ be a relaxed STRIPS planning problem, $\Pi^+ = \langle X, \emptyset, A^+, G, \text{cost} \rangle$ be the delete relaxation of Π , and P be a logic program consisting of rules of the form (1) $g \leftarrow \text{not } g$ for every $g \in G$; (2) $\{a\} \leftarrow q_1, \dots, q_n$ for every $\vec{a} \in A$ with $\text{pre}(\vec{a}) = \{q_1, \dots, q_n\}$; (3) $p \leftarrow a$ for every $\vec{a} \in A$ and $p \in \text{add}(\vec{a})$. 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 show that more relaxed semantics of models could not play the same role. Example 1 shows that neither the classical models nor the supported models of P are generally suitable for capturing the relaxed plans of Π correctly.

Example 1

Consider a planning problem $\Pi = \langle X, I, A, G, \text{cost} \rangle$, where $X = \{p, q\}$, $I = \emptyset$, $G = \{p\}$, $A = \{\vec{a}, \vec{b}\}$, $\text{pre}(\vec{a}) = \text{add}(\vec{b}) = \{p\}$, $\text{add}(\vec{a}) = \text{pre}(\vec{b}) = \{q\}$, and the cost function cost is arbitrary. This problem has no relaxed plan, as \vec{a} and \vec{b} are codependent. The logic program P explained above consists of the following rules:

$$\begin{array}{ll} \{a\} \leftarrow p. & \{b\} \leftarrow q. \\ q \leftarrow a. & p \leftarrow b. \\ p \leftarrow \text{not } p. & \end{array}$$

It is easy to check that $M = \{a, b, p, q\}$ is both a classical and a supported model for P . However, P has no stable model, due to circularities involved in the encoding. ■

We now 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 .

Proof

We first show that f is well-defined, that is, if $\pi = \vec{a}_1, \dots, \vec{a}_m$ is a permutation of members of A' such that π is a relaxed plan for Π , then $M = f(A')$ is a stable model of P . For every $g \in G$, g must be added by some action in π . Thus, the reduct P^M consists of rules of the form (1) $a \leftarrow q_1, \dots, q_n$ for every $\vec{a} \in \pi$ and $pre(\vec{a}) = \{q_1, \dots, q_n\}$, and (2) $p \leftarrow a$ for every $\vec{a} \in \pi$ and $p \in add(\vec{a})$. Clearly, M is model for P^M . By bounded induction on the lengths of prefixes of π , we show that M is a subset of any model for P^M . As we explained above, the initial state of the relaxed problem is (safely) assumed to be an empty set. Therefore, \vec{a}_1 cannot have any precondition. Thus, P^M includes a rule of the form (a_1) , and $add(\vec{a}_1) \cup \{a_1\}$ is a subset of any model for P^M . Assume that for $1 \leq j < m$, $\bigcup_{i=1, \dots, j} add(\vec{a}_i) \cup \{a_1, \dots, a_j\}$ is a subset of any model for P^M . Since \vec{a}_{j+1} is executable in $exec_{\vec{a}_1, \dots, \vec{a}_j}(\emptyset)$, $pre(\vec{a}_{j+1})$ is a subset of $\bigcup_{i=1, \dots, j} add(\vec{a}_i)$. Because P^M includes the two types of rules explained above for \vec{a}_{j+1} , we conclude that $\bigcup_{i=1, \dots, j+1} (add(\vec{a}_i) \cup \{a_i\})$ is a subset of any model for P^M .

Clearly, f is injective. We now show that f is also surjective, that is, if M is a stable model of P , then there exists $A' \subseteq A^+$ such that $M = f(A')$, and A' can be permuted to produce a relaxed plan for Π . Let $A' = \{\vec{a} \mid a \in M\}$. We have $G \subseteq M$ because for every $g \in G$, P includes the rule $g \leftarrow \text{not } g$. The reduct P^M consists of rules of the form (1) $a \leftarrow q_1, \dots, q_n$ for every $\vec{a} \in A'$ and $pre(\vec{a}) = \{q_1, \dots, q_n\}$ and (2) $p \leftarrow a$ for every $\vec{a} \in A'$ and $p \in add(\vec{a})$. If p is added by some action $\vec{a} \in A'$, then clearly we must have $p \in M$. On the other hand, for every $p \in X$ if $p \in M$, then p is added by some action $\vec{a} \in A'$, otherwise $M \setminus \{p\}$ would also be a model for P^M , contradicting that M is the least model for P^M . We conclude that $M = f(A')$ and if A' can be ordered to produce a sequence of actions executable in I , then that sequence is also a relaxed plan for Π .

For the sake of contradiction, assume that A' cannot be ordered to produce a sequence of actions executable in I . Let A'' be a (possibly empty) proper subset of A' such that its members (if any) can be ordered to produce a sequence of actions executable in I , and furthermore, let A'' be maximal in the sense that there is no subset of A' with such a property that is also a proper superset of A'' . Let $M' = \bigcup_{\vec{a} \in A''} add(\vec{a}) \cup \{a \mid \vec{a} \in A''\}$. Clearly, M' is a proper subset of M . Let $\vec{a} \in A'$ and $pre(\vec{a}) = \{q_1, \dots, q_n\}$. If $\vec{a} \in A''$, M' trivially satisfies $a \leftarrow q_1, \dots, q_n$. On the other hand, for every $\vec{a} \in A' \setminus A''$, the maximality of A'' implies that at least one precondition of \vec{a} is not in M' , and therefore, $a \leftarrow q_1, \dots, q_n$ is vacuously satisfied. We conclude that M' is a model for P^M , contradicting that M is the least model for P^M . □

Theorem 1 shows that if P is augmented with an optimization constraint requiring minimization over the summation of the costs of the 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 explaining the logic 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. Alternatively, a *diagnostic* encoding would explain the logic of relaxed

plan computation from effects to actions, and from actions to preconditions. In the next section, we show how this latter paradigm could be used for computing relaxed plans.

4 Relaxed plans captured with supported models of logic programs

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 $\text{Tr}_{\text{ACYC}}(P)$ which are acyclic with respect to an underlying graph (Bomanson *et al.* 2016). We provide an adaptation of this method based on the structure of program P explained above. We then review the so-called 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 vertex elimination could also be used to translate $\text{Tr}_{\text{ACYC}}(P)$ to a new program P_c such that the supported models of P_c represent acyclic supported models of $\text{Tr}_{\text{ACYC}}(P)$, and thus, stable models of P and relaxed plans of Π . Based on the structure of P_c , we introduce another logic program P_d which describes the relaxed plans diagnostically. We prove that the supported models of P_d represent those of P_c , thereby capturing the stable models of P and relaxed plans of Π .

4.1 Instrumentation of logic programs with acyclicity constraint

We adopt the *acyclicity translation* $\text{Tr}_{\text{ACYC}}(P)$ of a logic program P (Bomanson *et al.* 2016) that deploys special *dependency atoms* $\text{dep}(x, y)$ to express the activation of the respective arc $\langle x, y \rangle \in \text{DG}^+(P)$ in the acyclicity constraint. For the sake of the compactness of the output program, instead of using the exact method, we customize the translation method considering the structure of the program P explained above. In particular, 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 $\text{DG}^+(P)$.

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 $p \in \text{add}(\vec{a})$ and $q \in \text{pre}(\vec{a})$, the potential dependency of p on q is expressed using a choice rule $\{\text{dep}(p, q)\} \leftarrow q$. Also, atoms $\text{ws}(a_1, p), \dots, \text{ws}(a_k, p)$, for actions $\{\vec{a}_1, \dots, \vec{a}_k\}$ that add p enforce the well-support for p in terms of k rules $p \leftarrow \text{ws}(a_i, p)$ for $i = 1, \dots, k$. 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 $\text{pre}(\vec{a}) = \{q_1, \dots, q_n\}$ and $p \in \text{add}(\vec{a})$.

$$\{\text{ws}(a, p)\} \leftarrow \text{dep}(p, q_1), \dots, \text{dep}(p, q_n). \quad (3)$$

Also, the rule $a \leftarrow \text{ws}(a, p)$ 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 $g \leftarrow \text{not } g$ for every $g \in G$ to guarantee that every goal atom has been produced.

For $\text{Tr}_{\text{ACYC}}(P)$ 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 $\{\langle a, b \rangle \mid \text{dep}(a, b) \in I\}$ is acyclic. We deploy the following result:

Proposition 1 (Bomanson et al. (2016))

If M is a stable model of P , then $\text{Tr}_{\text{ACYC}}(P)$ has an acyclic supported model N such that $M = N \cap \text{At}(P)$. If N is an acyclic supported model of $\text{Tr}_{\text{ACYC}}(P)$, then $M = N \cap \text{At}(P)$ is a stable model of P .

Example 2

Consider Π to be the planning problem of Example 1. The program $\text{Tr}_{\text{ACYC}}(P)$ consists of the following rules:

$$\begin{array}{ll} \{\text{dep}(p, q)\} \leftarrow q. & \{\text{dep}(q, p)\} \leftarrow p. \\ \{\text{ws}(a, q)\} \leftarrow \text{dep}(q, p). & \{\text{ws}(b, p)\} \leftarrow \text{dep}(p, q). \\ q \leftarrow \text{ws}(a, q). & p \leftarrow \text{ws}(b, p). \\ a \leftarrow \text{ws}(a, q). & b \leftarrow \text{ws}(b, p). \\ p \leftarrow \text{not } p. & \end{array}$$

It can easily be checked that $M = \{a, b, p, q, \text{ws}(a, q), \text{ws}(b, p), \text{dep}(p, q), \text{dep}(q, p)\}$ is the only supported model for $\text{Tr}_{\text{ACYC}}(P)$. However, this model is not acyclic, as it contains both $\text{dep}(p, q)$ and $\text{dep}(q, p)$. ■

Similarly to the stable model-based encoding, $\text{Tr}_{\text{ACYC}}(P)$ is a causal encoding, expressing the inference in the direction from preconditions to actions, and from actions to effects. However, there are additional concepts in this encoding, namely dependencies and well-support. In fact, in $\text{Tr}_{\text{ACYC}}(P)$, preconditions are assumed to cause dependencies, which in turn cause well-support and effects. Here, well-support atoms $\text{ws}(a, p)$ take the causal role that action atoms a have in P . The action atoms are only included in $\text{Tr}_{\text{ACYC}}(P)$ to represent their cost in the minimization constraint. The rules in Example 2 establish the inference direction from preconditions to dependencies (the first row), from dependencies to well-support (the second row), and from well-support to effects (the third and the fourth rows). The final rule captures the goal condition (as before).

4.2 Vertex elimination graphs

The concept of vertex elimination graphs has been recently shown effective for guaranteeing acyclicity in constraint programs with underlying 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 $\alpha : \{1, \dots, n\} \rightarrow V$. 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

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

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 $\mathcal{G} = \mathcal{G}_0, \mathcal{G}_1, \dots, \mathcal{G}_{n-1}$, where \mathcal{G}_i is the $\alpha(i)$ -elimination graph of \mathcal{G}_{i-1} for $i = 1, \dots, n - 1$.

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} :

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

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

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

For any digraph \mathcal{G} , the number of arcs of the vertex elimination graph depends on the ordering function α . It has been shown that the problem of finding the optimal ordering function, the one resulting in the smallest number of arcs in the vertex elimination graph, is NP-complete (Rose and Tarjan 1975). Nevertheless, there are effective heuristics for finding empirically useful orderings. Examples are the *minimum fill-in* and *minimum degree* that accordingly choose a vertex for removal at each step during the elimination process. One important property of vertex elimination is that if the original graph \mathcal{G} has a directed cycle, then \mathcal{G}_α^* will have a cycle of length 2, regardless of the ordering α .

4.3 The causal encoding based on supported models

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

$$\text{dep}(p, q) \leftarrow \text{dep}(p, \alpha(i)), \text{dep}(\alpha(i), q). \quad (7)$$

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

$$f \leftarrow \text{dep}(p, q), \text{dep}(q, p), \text{not } f. \quad (8)$$

Intuitively, for any vertex ordering α , and any supported model M of $\text{Tr}_{\text{ACYC}}(P)$, 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.

Theorem 2

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

Proof

(\implies) If Π has a relaxed plan $\pi = \vec{a}_1, \dots, \vec{a}_m$, then according to Theorem 1, $\bigcup_{i=1, \dots, m} \text{add}(\vec{a}_i) \cup \{a_1, \dots, a_m\}$ is a stable model of P . By Proposition 1, $\text{Tr}_{\text{ACYC}}(P)$ has an acyclic supported model N such that $\{a \in N \mid \vec{a} \in A^+\} = \{a_1, \dots, a_m\}$. Let $\mathcal{G}_N = \langle X, E_N \rangle$, where $E_N = \{\langle p, q \rangle \mid \text{dep}(p, q) \in N\}$, and let $\mathcal{G}_{N, \alpha}^* = \langle X, E_{N, \alpha}^* \rangle$ be the vertex elimination graph of \mathcal{G}_N according to α . Since \mathcal{G}_N is acyclic, X can be ordered by topological sorting according to \mathcal{G}_N . Now, if the vertex elimination process adds the arc $\langle p, q \rangle$, then p must be ordered before q by the topological sorting. Therefore, $\mathcal{G}_{N, \alpha}^*$ is also acyclic. It should now be easy to check that $N \cup \{\text{dep}(p, q) \mid \langle p, q \rangle \in E_{N, \alpha}^*\}$ is a supported model of P_c .

(\impliedby) Let M be a supported model for P_c . We first show that M is acyclic. Let $\mathcal{G}_M = \langle X, E_M \rangle$, where $E_M = \{\langle p, q \rangle \mid \text{dep}(p, q) \in M\}$. Assume that $k > 1$ is the smallest number for which there exist a cycle of length k in \mathcal{G}_M . Then there are atoms $\text{dep}(p_1, p_2), \dots, \text{dep}(p_{k-1}, p_k), \text{dep}(p_k, p_1)$ in M . According to the rule (8), k cannot be equal to 2. Let $i = \text{argmin}_{1 \leq j \leq k} \alpha^{-1}(p_j)$. Then p_i is the vertex in the mentioned cycle that is eliminated before all other vertices in the cycle according to α . According to the rule (7), $\text{dep}(p_{i-1}, p_{i+1}) \in M$ (with indices considered modulo k), and therefore \mathcal{G}_M has a cycle of length $k - 1$, a contradiction. Let $N = M \cap \text{At}(\text{Tr}_{\text{ACYC}}(P))$. A straightforward investigation shows that N is a supported model of $\text{Tr}_{\text{ACYC}}(P)$. By Proposition 1, $N' = N \cap \text{At}(P)$ is a stable model of P . Since $A' = \{\vec{a} \in A^+ \mid a \in N'\}$, by Theorem 1, there exists a permutation π of members of A' such that π is a relaxed plan for Π . \square

4.4 The diagnostic encoding based on supported models

One major approach to solving problems in the AI planning field is to perform backward search, also known as regression, in the search space (Ghallab *et al.* 2004). In this approach, actions are assumed to act in reverse, that is, producing their preconditions given they have some effects relevant to the current search node. The main drawback of this approach is that it can easily produce dead-end states, which are not reachable from the initial state. The notion of reversibility of actions has been shown to be quite effective for detecting dead-end states. However, determining the reversibility of actions is itself challenging, and might even need a logic program (Faber *et al.* 2022) of its own. Nevertheless, the problem of detecting the dead-ends is an easy one in the case of relaxed planning, and can be done in polynomial time as a preprocessing method (Hoffmann and Nebel 2001). Therefore, this backward approach has promise to be efficient for relaxed planning.

Inferring causes from effects can be understood as diagnostic inference (Russell and Norvig 2020). In our causal encoding, we expressed 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 $\{\text{ws}(a, p)\} \leftarrow p$ for every $\vec{a} \in A$ such that $p \in \text{add}(\vec{a})$, and also $f \leftarrow p, \text{not ws}(a_1, p), \dots, \text{not ws}(a_m, p), \text{not } f$ for $p \in X$ and all actions $\vec{a}_1, \dots, \vec{a}_m$ that could add p . The first rule provides the possibility of well-support atoms being in a supported model, while the second rule requires at least one of

the well-support atoms to be in the model. To represent the inference from well-supports to dependencies, we add $\text{dep}(p, q) \leftarrow \text{ws}(a, p)$ for $\vec{a} \in A$, $q \in \text{pre}(\vec{a})$, and $p \in \text{add}(\vec{a})$. Finally, to establish the inference direction from dependencies to preconditions, we add $q \leftarrow \text{dep}(p, q)$. 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 $a \leftarrow \text{ws}(a, p)$ for $\vec{a} \in A$ and $p \in \text{add}(\vec{a})$, to enable an action atom a to represent its cost in the minimization constraint, and also $g \leftarrow \text{not } g$ 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 $\text{dep}(p, q)$ for which there is no action \vec{a} such that $\text{ws}(a, p) \in N$ and $q \in \text{pre}(\vec{a})$. Thus, we have the following result:

Theorem 3

Let A' be any subset of A^+ . The program P_d has a supported model M such that $A' = \{\vec{a} \mid a \in M\}$ iff P_c has a supported model N such that $A' = \{\vec{a} \mid a \in N\}$.

Theorem 2 and Theorem 3 can be used to establish Corollary 1.

Corollary 1

Let A' be any subset of A^+ . There exists a permutation π of members of 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 collection¹. All experiments have been run on a cluster of Linux machines with Intel Xeon 2.40 GHz CPUs, using a timeout of 1800 s 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, that is, eliminating a vertex with 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)-based search, which our preliminary experiments showed to significantly outperform the branch-and-bound strategy for the mentioned encodings. Although CLASP offers a variety of search strategies, we only use the default one. Therefore, the solver parameters have not been tuned to produce the best performance for our new methods. Henceforth, we refer to the method obtained by combing CLASP with our P , P_c , and P_d encodings simply by the name of the corresponding encoding.

¹ <https://github.com/asptools/software>

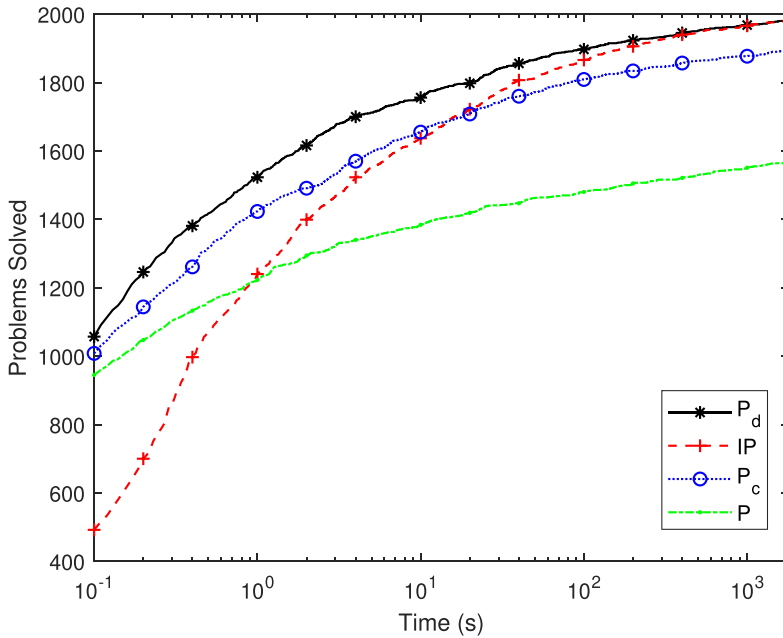


Fig. 1. Cumulative numbers of problems solved by the competing methods.

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 20.1² as the optimizer. Regardless of the given time limit, IP has shown to outperform previously introduced methods for optimal relaxed planning including the Boolean satisfiability-based encoding used by Rankooh and Rintanen (2022b), the integer programming-based model introduced by Imai and Fukunaga (2015), and the minimum-cost hitting set-based method introduced by Haslum *et al.* (2012). Since IP has also been implemented inside the HSP* planner (Haslum 2015), all competing methods share the same code for reading the input problem, grounding, and preprocessing.

As benchmark problem sets, we use the STRIPS planning problem sets found in the *planning repository*³. From IPC domains, domains from both optimal and so-called *satisficing* tracks have been considered. In total, 2212 problem instances from 84 problem sets are used for comparison. Note that this is exactly the benchmark set used by Rankooh and Rintanen (2022a) for comparing IP with previously introduced methods.

The cumulative number of problems solved by all methods are presented in Figure 1. Out of the 2212 problems under evaluation, the cost of an optimal relaxed plan was computed in 1800 s for 1980, 1982, 1894, and 1567 problems by IP, P_d , P_c , and P , respectively. As it can be seen 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 s by our diagnostic encoding is not much higher than that of IP, P_d solves problems

² <https://www.ibm.com/products/ilog-cplex-optimization-studio>

³ <https://github.com/AI-Planning/classical-domains>

considerably faster than IP. In fact, regardless of the time limit, P_d solves more problems compared to any other solver. Particularly, P_d solves 1091 problems in less than 0.1 s, more than double the 516 problems solved by IP within the same time limit.

6 Conclusions and future research

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.

One direction to extend the current work is to study the impact of our new encodings and ASP solvers when employed for computing heuristic values inside state-of-the-art planners. Since our best encoding enables CLASP to solve almost half of the studied benchmark problems in less than one tenth of second, a direct usage of h^+ computed by CLASP seems to be promising. Also, the usage of USC as the optimization strategy allows for computing lower bounds for h^+ within any given time limit. It seems interesting to study the informativeness of such lower bounds in comparison to other commonly used heuristics such as LM-cut, another lower bound of h^+ , when given the same amount of time for computation.

Acknowledgements

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

Conflict of interest

The authors declare no competing interests.

References

- 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*. Lecture Notes in Computer Science, vol. 5803. Springer, 9–16.
- BOMANSON, J., GEBSER, M., JANHUNEN, T., KAUFMANN, B. AND SCHAUB, T. 2016. Answer set programming modulo acyclicity. *Fundamenta Informaticae* 147, 1, 63–91.
- BONET, B. AND GEFFNER, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129, 1-2, 5–33.
- BREWKA, G., EITER, T. AND TRUSZCZYNSKI, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54, 12, 92–103.
- BYLANDER, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69, 1-2, 165–204.

- CORRÊA, A. B., FRANCÈS, G., POMMERENING, F. AND HELMERT, M. 2021. Delete-relaxation heuristics for lifted classical planning. In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021*. AAAI Press, 94–102.
- CORRÊA, A. B., POMMERENING, F., HELMERT, M. AND FRANCÈS, G. 2022. The FF heuristic for lifted classical planning. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Virtual Event, February 22 - March 1, 2022*. AAAI Press, 9716–9723.
- FABER, W., MORAK, M. AND CHRPA, L. 2022. Determining action reversibility in STRIPS using answer set programming with quantifiers. In *Practical Aspects of Declarative Languages - 24th International Symposium, PADL 2022, Philadelphia, PA, USA, January 17-18, 2022, Proceedings*. Lecture Notes in Computer Science, vol. 13165. Springer, 42–56.
- GEBSER, M., KAMINSKI, R., KAUFMANN, B., ROMERO, J. AND SCHAUB, T. 2015. Progress in clasp series 3. In *LPNMR 2015*. 368–383.
- 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, Freiburg, Germany June 11-16, 2011*. AAAI.
- GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Proceedings of ICLP'88*. 1070–1080.
- GHALLAB, M., NAU, D. S. AND TRAVERSO, P. 2004. *Automated planning - theory and practice*. Elsevier.
- 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, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI.
- HASLUM, P. 2015. Hsp* code and documentation. <http://users.cecs.anu.edu.au/patrik/un-hsps.html>.
- 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, 353–357.
- 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, Thessaloniki, Greece, September 19–23, 2009*. AAAI.
- HOFFMANN, J. AND NEBEL, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14, 253–302.
- 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.
- KEYDER, E. AND GEFFNER, H. 2008. Heuristics for planning with action costs revisited. In *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21–25, 2008, Proceedings*. Frontiers in Artificial Intelligence and Applications, vol. 178. IOS Press, 588–592.
- LIFSCHITZ, V. 1999. Answer set planning. In *Proceedings of ICLP 1999*, 23–37.
- MAREK, V. W. AND SUBRAHMANIAN, V. S. 1992. The relationship between stable, supported, default and autoepistemic semantics for general logic programs. *Theoretical Computer Science* 103, 2, 365–386.
- 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, Providence, Rhode Island, USA, September 22-26, 2007*. AAAI, 240–247.
- RANKOOH, M. F. AND JANHUNEN, T. 2022. Efficient computation of answer sets via SAT modulo acyclicity and vertex elimination. In *Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings*,

- G. Gottlob, D. Incezan, and M. Maratea, Eds. Lecture Notes in Computer Science, vol. 13416. Springer, 203–216.
- RANKOOH, M. F. AND RINTANEN, J. 2022a. Efficient computation and informative estimation of h^+ by integer and linear programming. In *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13–24, 2022*. AAAI Press, 71–79.
- RANKOOH, M. F. AND RINTANEN, J. 2022b. 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*. AAAI Press, 9910–9917.
- RANKOOH, M. F. AND RINTANEN, J. 2022c. 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*. AAAI Press, 5861–5868.
- 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, July 27–31, 2014, Québec City, Québec, Canada*. AAAI Press, 2344–2351.
- 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*, 245–254.
- RUSSELL, S. AND NORVIG, P. 2020. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson.
- SIMONS, P., NIEMELÄ, I. AND SOININEN, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138, 1–2, 181–234.
- SON, T. C. AND BALDUCCINI, M. 2018. Answer set planning in single- and multi-agent environments. *Künstliche Intelligenz* 32, 2–3, 133–141.
- SON, T. C., BARAL, C., NAM, T. H. AND MCILRAITH, S. A. 2006. Domain-dependent knowledge in answer set planning. *ACM Transactions on Computational Logic* 7, 4, 613–657.