Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
KR in the Wild

SAT-based ASP Solving and Optimization via a General Transitive Closure Framework

Masood Feyzbakhsh Rankooh and **Matti Järvisalo**
HIIT, Department of Computer Science, University of Helsinki, Finland
{masood.feyzbakhshrankooh,matti.jarvisalo}@helsinki.fi

Abstract

Answer set programming (ASP) in the NP fragment can be solved by translating into propositional satisfiability (SAT). However, for non-tight programs, this requires additional encodings to enforce acyclicity of the underlying dependency graph, making acyclicity handling a key challenge in translating ASP encodings of decision and optimization problems into SAT and maximum satisfiability (MaxSAT). Various SAT encodings of acyclicity exploiting structural graph properties have recently been proposed for various settings. Focusing on ASP, we show that such encodings are captured by a generalized transitive closure framework. The framework can be instantiated for obtaining various types of refined transitive closure encodings. We consider four concrete instantiations framework, analyzing their correctness and size. Putting the framework into practice, we show through extensive empirical evaluation that current state-of-the-art SAT and MaxSAT solvers are competitive with and often even outperform state-of-the-art native ASP solvers on both decision and optimization problems.

1 Introduction

Answer set programming (ASP) (Lifschitz 2019; Niemelä 1999a) is a successful rule-based declarative paradigm that allows for modelling and efficiently solving NP-hard decision and optimization problems. In terms of modelling, the ASP language is particularly convenient as a high-level representation language for modelling problems as logic programs.

In terms of solving, state-of-the-art ASP solvers allow for efficiently finding intended solutions corresponding to stable models of an ASP program at hand. The core solving algorithms are based on Boolean satisfiability (SAT) solving (Biere et al. 2021), in particular, the dominant conflict-driven clause learning (CDCL) algorithm (Marques Silva and Sakallah 1999; Marques-Silva, Lynce, and Malik 2021). However, ASP solvers integrate specialized algorithmic techniques as extensions of CDCL in order to exactly capture the stable model semantics. In particular, stable models are more restrictive than the classical propositional semantics, and require—specifically in the general case of for non-tight ASP programs—enforcing acyclicity of the so-called dependency graph of underlying an ASP program.

The key challenge in translating normal ASP programs is in developing encodings for enforcing the acyclicity of the dependency graph of a given program. Classical approaches to this task rely on transitive closure constructions that capture reachability between pairs of vertices, but these encodings can become prohibitively large. More recent approaches exploit structural properties of the dependency graph, for instance via techniques such as vertex elimination (Rankooh and Janhunen 2022) and related approaches, to obtain more compact encodings by restricting the set of path compositions that need to be considered (see Section 2 for a more thorough review of related literature).

In this work, we formalize a general transitive closure framework for obtaining effective monolithic SAT encodings of ASP, and put it into practice through several instantiations for solving ASP encodings of decision and optimization using state-of-the-art SAT and MaxSAT solvers. Our acyclicity encoding framework, targeting the dependency graph underlying ASP programs, is parameterized by a set of transitivity triples that determine where path composition is permitted. We formally establish sufficient conditions under which such restricted transitivity encodings are correct. As we show, both classical transitive closure encodings and more recent approaches based on vertex elimination and cycle elimination (Rankooh and Rintanen 2022; Rankooh, Niskanen, and Järvisalo 2025) are captured as specific instantiations of the framework. Indeed, the general framework provides a unified perspective to relationships between existing encodings, and provides a systematic basis for deriving new encodings and analyzing their size in terms of structural graph parameters such as elimination width and ordered cycle rank. As concrete instantiations of the framework, we introduce an ordered transitive closure encoding and a feedback vertex set based encoding. In addition, we establish new complexity results for the cycle elimination based encoding—first time employed for SAT encodings of ASP in this work—by relating its size to the well-known graph parameter of cycle rank. Putting the encodings to use for solving ASP encodings of decision and optimization problems "in the wild", we present results from an extensive evaluation comparing the efficiency of state-of-the-art SAT and MaxSAT solvers and state-of-the-art native ASP solvers on both decision and optimization problems. To the best of our knowledge, the use of modern MaxSAT solvers for ASP optimization via translations to MaxSAT has not been thoroughly evaluated to-date. The results show that, through

the structure-based encodings, current SAT and MaxSAT solvers are complementary to and even surpass (in particular in the case of optimization) native ASP solvers in their efficiency, with further promise ahead in making use of future improvements in SAT and MaxSAT solving for improving the state of the art in ASP solving.

2 Related Work

Despite the attractiveness of ASP as a high-level modelling language, the number of state-of-the-art native ASP solvers available today remains limited. Among these, Clasp (Gebser, Kaufmann, and Schaub 2012) is arguably the most widely used solver, alongside Wasp (Alviano et al. 2013), both building on earlier systems such as Smodels (Syrjänen and Niemelä 2001) and DLV (Leone et al. 2006). This situation contrasts with SAT and maximum satisfiability (MaxSAT) (Bacchus, Järvisalo, and Martins 2021), where a wide range of solvers are actively developed.

Motivated by this contrast, several approaches have aimed at leveraging advances in SAT and MaxSAT solving for ASP. Early SAT-based ASP solvers, including ASSAT (Lin and Zhao 2002) and Cmodels (Giunchiglia, Lierler, and Maratea 2004), rely on computing classical models of Clark's completion (Clark 1977) and iteratively eliminating non-stable models by adding loop formulas (Lin and Zhao 2002). However, since acyclicity of the dependency graph is not enforced during search, this approach may require introducing exponentially many loop formulas in the worst case, which limits its competitiveness compared to modern native ASP solvers.

To overcome this limitation, polynomial translations of normal ASP programs into SAT have been extensively studied (Lin and Zhao 2003; Janhunen 2004), as well as extensions to richer frameworks such as SAT modulo theories, including bit-vectors (Nguyen, Janhunen, and Niemelä 2011), difference logic (Janhunen, Niemelä, and Sevalnev 2009), and integer programming (Liu, Janhunen, and Niemelä 2012). Another line of work avoids fully monolithic encodings by extending SAT solvers with specialized propagators to handle acyclicity as a global constraint (Gebser, Janhunen, and Rintanen 2014a). More recently, progress has been made in developing effective monolithic SAT encodings for ASP. In particular, acyclicity constraints can be compiled into SAT using the vertex elimination technique (Rose, Tarjan, and Lueker 1976; Rankooh and Rintanen 2022), which exploits structural properties of the dependency graph. Empirical results show that this approach narrows the performance gap between translation-based methods and native ASP solvers.

Classical encodings of acyclicity are based on transitive closure constructions that capture reachability between all pairs of vertices, but these encodings are often prohibitively large. To address this, several structure-based refinements have been proposed, including approaches based on vertex elimination (Rankooh and Rintanen 2022; Rankooh and Janhunen 2022), as well as more recent techniques such as cycle elimination and refined ordering encodings (Rankooh, Niskanen, and Järvisalo 2025). These approaches share the common principle of enforcing acyclicity only over a carefully selected subset of edge compositions, guided by structural properties of the graph, leading to more compact encodings. In the context of ASP, prior work has primarily focused on the vertex elimination encoding as a way of exploiting structural properties. Furthermore, empirical evaluations of such encodings have so far been limited to decision problems when compared against native ASP solvers.

3 Preliminaries

3.1 Answer Set Programming

We consider *weight constraint programs* (WCPs) consisting of rules of the following forms:

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

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

$$a \leftarrow k \leq [b_1 = w_1\,,\ldots,\,b_n = w_n,\ \texttt{not}\ c_1 = w_{n+1}\,,\ldots,\,\texttt{not}\ c_m = w_{n+m}]. \tag{3}$$

The symbols a, $b_1\,,\ldots,b_n$, and $c_1\,,\ldots,c_m$ are propositional atoms, and not denotes negation by default. The bound k and the weights $w_1\,,\ldots,w_{n+m}$ are non-negative integers. Rules of the forms (1)–(3) are called *normal*, *choice*, and *weight* rules, respectively.

Each rule r derives its *head* $\mathrm{head}(r)$ when the conditions in its *body* $\mathrm{body}(r)$ are satisfied. For a choice rule, the derivation of the head is optional. For a weight rule, the sum of weights of satisfied body literals must reach the bound k. We write $\mathrm{body}^+(r)$ and $\mathrm{body}^-(r)$ for the sets of atoms occurring positively and negatively in the body of r, respectively.

A *normal logic program* consists of normal rules only. A program is *positive* if it contains no negation by default. For a program P, the *definition* of an atom a in P is $\mathrm{def}_P(a) = \{r \in P \mid \mathrm{head}(r) = a\}$. The *signature* of P is the set of atoms $\mathrm{At}(P) = \bigcup_{r\in P}(\{\mathrm{head}(r)\}\cup\mathrm{body}^+(r)\cup \mathrm{body}^-(r))$.

The *positive dependency graph* of P is the directed graph $\mathrm{DG}^+(P) = \langle \mathrm{At}(P), \succeq\rangle$, where $a \succeq b$ holds if there exists a rule $r \in P$ with $\mathrm{head}(r) = a$ and $b \in \mathrm{body}^+(r)$. A *strongly connected component* (SCC) of $\mathrm{DG}^+(P)$ is a maximal set of atoms that are mutually reachable via directed paths. A program is considered non-tight if its positive dependency graph contains at least one non-trivial SCC, i.e., an SCC consisting of more than one vertex.

An *interpretation* is a total function $I\colon \mathrm{At}(P) \to \{\top, \bot\}$ that assigns a truth value to each atom. For an atom $a \in \mathrm{At}(P)$, we write $I \models a$ if $I(a) = \top$. An interpretation I satisfies a normal or weight rule r if whenever the body of r is satisfied under I, the head $\mathrm{head}(r)$ is true under I. Choice rules are satisfied unconditionally. An interpretation M is a *classical model* of a program P, written $M \models P$, if it satisfies all rules in P.

Every positive program has a unique *least model*: the pointwise minimum truth assignment among all models of P. This corresponds to the intersection of all sets of atoms that are true in some model of P. Given an interpretation

I, the *reduct* P^I is obtained by removing all negative body literals that are false under I and adjusting bounds accordingly. An interpretation I is an *stable model* of P if it coincides with the least model of its reduct, i.e., if $I = \mathrm{LM}(P^I)$. A classical model M is *supported* if every atom assigned true by M occurs as the head of some rule whose body is satisfied under M. Every stable model is supported, but the converse does not hold in general.

Example 1. *Consider the program P consisting of the following rules:*

$$a \leftarrow b,\ c. \qquad \{b\}. \qquad c \leftarrow 3 \leq [a = 1,\ b = 2,\ \texttt{not}\ b = 3].$$

The signature of P is $\mathrm{At}(P) = \{a, b, c\}$. The positive dependency graph $\mathrm{DG}^+(P)$ contains edges (a, b), (a, c), (c, a), and (c, b), and has two SCCs: $\{a, c\}$ and $\{b\}$. The program has two stable models, $\{b\}$ and $\{c\}$. The interpretation $\{a, b, c\}$ is supported but not stable.

3.2 Graph Properties and Acyclicity

Let $G = (V, E)$ be a directed graph, where V is a finite set of vertices and $E \subseteq V \times V$ is a set of directed edges. A directed graph is acyclic if and only if all of its SCCs are singletons. A set $F \subseteq V$ is a *feedback vertex set* (FVS) of G if removing F and all incident edges yields an acyclic graph. Equivalently, $G[V \setminus F]$ contains no directed cycle. Feedback vertex sets provide a vertex-based characterization of acyclicity.

The *transitive closure* of G is the directed graph $G^+ = (V, E^+)$ with $(u, v) \in E^+$ if and only if there exists a directed path from u to v in G. While the transitive closure provides a complete representation of reachability in a graph, it represents strictly more information than is needed to guarantee acyclicity: to detect directed cycles, it is not necessary to know all reachable pairs of vertices; it suffices to know whether *any* vertex can reach itself. As a result, representing the full transitive closure introduces a large number of reachability relations that are irrelevant for cycle detection and can significantly increase the size of the representation. This observation motivates the use of alternative graph constructions that preserve just enough reachability information to detect cycles, while avoiding the overhead of representing all pairs reachability. In the following, we review vertex and cycle elimination graphs, which achieve this by maintaining only selected reachability relations and are therefore better suited for compact acyclicity encodings.

Vertex elimination (VE) graphs (Rose, Tarjan, and Lueker 1976) are closely related to acyclicity in directed graphs. Let $G = (V, E)$ be a directed graph, and let $\alpha : \{1, \ldots, |V|\} \to V$ be a bijection that fixes an ordering of the vertices. For a vertex $v \in V$, the *fill-in* of v is the set $F(v) = \{(x, y) \mid (x, v) \in E,\ (v, y) \in E,\ x \neq y\}$. Intuitively, the fill-in contains all edges that represent paths of length two passing through v. Eliminating a vertex v means removing v and all edges incident to it, and then adding all edges in $F(v)$. The resulting graph is $G(v) = (V \setminus \{v\},\ E(v) \cup F(v))$, where $E(v) = \{(x, y) \in E \mid x \neq v \text{ and } y \neq v\}$.

Given an ordering α, the vertex elimination process removes the vertices one by one following α. The sequence of elimination steps produces a sequence of graphs $G = G_0^{ve}, G_1^{ve}, \ldots, G_{|V|-1}^{ve}$, where for each $i = 1, \ldots, |V| - 1$, the graph G_i^{ve} is obtained from G_{i-1}^{ve} by eliminating the vertex $\alpha(i)$. At each elimination step, new fill-in edges may be added. The union of all fill-in edges introduced during the process is called the *cumulative fill-in* and is defined as $F_\alpha^{ve}(G) = \bigcup_{i=1}^{|V|-1} F_{i-1}(\alpha(i))$, where $F_{i-1}(\alpha(i))$ is the fill-in of $\alpha(i)$ computed in G_{i-1}^{ve}. The *vertex elimination graph* corresponding to the ordering α is then $G_\alpha^* = (V,\ E \cup F_\alpha^{ve}(G))$. A key property of vertex elimination graphs is that if the original graph G contains a directed cycle, then for any ordering α, the graph G_α^* contains a directed cycle of length two. This property makes vertex elimination useful for enforcing acyclicity, since such short cycles are easy to detect and forbid.

The *ordered elimination width* $\delta_\alpha(G)$ of G under the ordering α (Hunter and Kreutzer 2007) is the maximum number of outgoing edges of the vertex $\alpha(i)$ in the graph G_{i-1}^{ve} over all $i = 1, \ldots, |V|$. By this measure, the number of edges in the vertex elimination graph G_α^* is bounded by $\mathcal{O}(|E| + \delta_\alpha(G) \cdot |V|)$. Finding an ordering that minimizes the elimination width is NP-complete (Rose, Tarjan, and Lueker 1976). In practice, however, simple heuristics often work well. Common examples include the *minimum fill-in* and *minimum degree* heuristics, which at each step choose a vertex whose elimination introduces few new edges or has a low degree in the current graph.

Example 2 (Vertex Elimination)**.** *Figure 2(a) shows the directed graph. The remaining figures illustrate the vertex elimination process. Vertices with low degree are eliminated first in order to keep the amount of fill-in small. When v_6 is eliminated, only the edge (v_4, v_5) is added, as shown in Figure 1(b). Next, eliminating v_2 introduces the edge (v_3, v_4), as shown in Figure 1(c). Eliminating v_3 then results in the graph shown in Figure 1(d). No additional fill-in edges are created, and the final vertex elimination graph is shown in Figure 1(e).*

Cycle elimination graphs (Rose, Tarjan, and Lueker 1976) allow for describing how directed cycles can be removed step by step by deleting vertices and simplifying the remaining graph structure. Unlike vertex elimination, which adds edges and therefore increases connectivity, cycle elimination works in the opposite direction. Repeatedly collapsing strongly connected components gradually removes connectivity from the graph. This makes cycle elimination fundamentally different from vertex-elimination-based approaches to acyclicity.

Let $G = (V, E)$ be a directed graph. For a vertex $v \in V$, the *cycle elimination fill-in* of v is the set of all outgoing edges of v in the transitive closure G^+ of G. In other words, the fill-in consists of all pairs (v, u) such that u is reachable from v in G. Given a vertex ordering α, the cycle elimination process produces a sequence of directed graphs $G = G_0^{ce}, G_1^{ce}, \ldots, G_{|V|-1}^{ce}$. For each $i = 1, \ldots, |V| - 1$, the graph G_i^{ce} is obtained from G_{i-1}^{ce} by first removing the vertex $\alpha(i)$ together with all edges incident to it, and then iden-

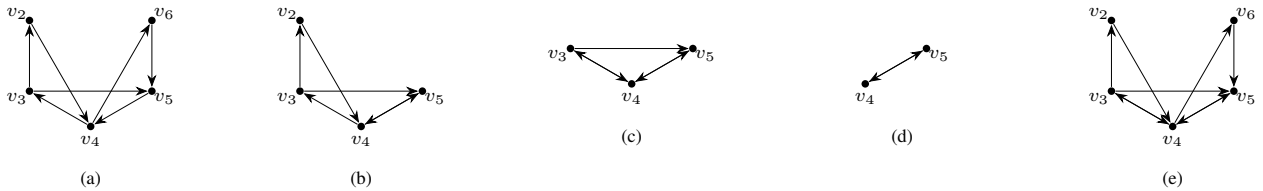

Figure 1: An example of the vertex elimination process.

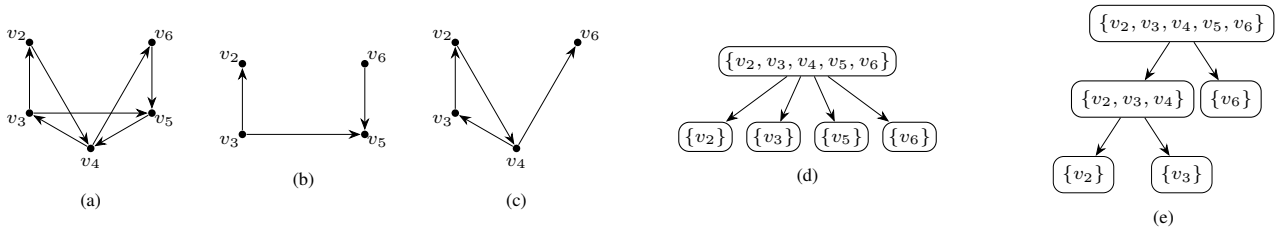

Figure 2: Cycle elimination examples and the corresponding cycle elimination decomposition trees.

tifying the strongly connected components of the remaining graph and removing all edges that leave these components. In this way, edges that could contribute to cycles are progressively eliminated.

The union of all fill-in edges introduced during this process is the *cumulative cycle elimination fill-in* $F^{ce}_{\alpha}(G) = \bigcup_{i=1}^{|V|-1} F^{ce}_{i-1}(\alpha(i))$, where $F^{ce}_{i-1}(\alpha(i))$ denotes the cycle elimination fill-in of $\alpha(i)$ computed in G^{ce}_{i-1}. The *cycle elimination graph* corresponding to the ordering α is then defined as $G^{+}_{\alpha} = (V, F^{ce}_{\alpha}(G))$. By construction, the graph G^{+}_{α} is always a subgraph of the transitive closure G^{+} of G. In general, it can be a strict subgraph, since the fill-in $F^{ce}_{i-1}(\alpha(i))$ may include only some of the outgoing edges of $\alpha(i)$ in G^{+}. Despite this restriction, G^{+}_{α} retains a crucial property of the full transitive closure: if the original graph G contains a directed cycle, then for any ordering α, the graph G^{+}_{α} contains a directed cycle of length two (Rankooh, Niskanen, and Järvisalo 2025). This makes cycle elimination graphs suitable for enforcing acyclicity.

Cycle rank (Eggan 1963) is a classical measure of how far a directed graph is from being acyclic and is closely related to the cycle elimination process. The cycle rank $r(G)$ of a directed graph G is defined recursively as follows. If G is acyclic, then $r(G) = 0$. If G is strongly connected and nonempty, then $r(G) = 1 + \min_{v \in V} r(G - v)$. Otherwise, $r(G)$ is the maximum cycle rank among the strongly connected components of G. Computing the cycle rank is NP-hard even for sparse directed graphs (Gruber 2012). However, if the order in which vertices are removed is fixed, as in the cycle elimination process, one obtains the notion of *ordered cycle rank*. Given an ordering α, the ordered cycle rank $r_{\alpha}(G)$ is defined by applying the same recursion as above, but restricting vertex removals to follow α, so that at each step only the next vertex in the order may be removed. This yields a tractable measure that directly reflects the structure revealed by cycle elimination.

The cycle elimination process can be represented using a tree-based structure called the *cycle elimination decomposition tree*. This structure makes explicit how vertices are removed and how the graph breaks into smaller strongly connected components during the process. Let $G = (V, E)$ be a directed graph and let α be a fixed ordering of the vertices. We run the cycle elimination process on G according to α and construct a rooted tree $T = (\mathcal{N}, \mathcal{A})$, where each node is labeled by a subset of vertices. The root of the tree is labeled by the full vertex set V. For a node $X \in \mathcal{N}$, let $G[X]$ denote the subgraph of G induced by the vertices in X. If $|X| = 1$, then X is a leaf of the tree. Otherwise, let v be the vertex in X that appears first in the ordering α. The vertex v is eliminated at node X.

After eliminating v from $G[X]$, we consider the strongly connected components of the remaining graph $G[X \setminus \{v\}]$. For each such strongly connected component C, we add a child node of X to the tree, labeled by the vertex set $V(C)$. This process is then applied recursively to each child node. For every vertex $v \in V$, there is a unique node of the tree at which v is eliminated. We denote this node by X_v, which is the node whose label contains v and in which v is the first vertex under the ordering α.

This construction directly mirrors the cycle elimination process. Each node of the tree corresponds to the elimination of one vertex, followed by a decomposition of the remaining graph into strongly connected components. As a result, the depth of the cycle elimination decomposition tree is exactly the ordered cycle rank $r_{\alpha}(G)$.

Example 3 (Cycle Elimination)**.** *Figures 2(d–e) show two cycle elimination decomposition trees constructed for the graph in Fig 2(a) using different vertex orderings. These trees illustrate how the chosen ordering directly influences both the structure of the decomposition and the resulting ordered cycle rank. In Fig. 2(d), the elimination process starts with vertex v_4. After removing v_4, the remaining graph breaks into strongly connected components that each contain a single vertex. As a result, the decomposition tree has depth one, which equals the ordered cycle rank for any or-*

dering that begins with v_4. Figure 2(e) shows the decomposition obtained when the process starts by eliminating v_5. In this case, the remaining graph contains the nontrivial strongly connected component $\{v_2, v_3, v_4\}$, which requires an additional elimination step. Removing v_4 from this component splits it into the two single-vertex components $\{v_2\}$ and $\{v_3\}$; the resulting decomposition tree has depth two.

4 ASP as SAT Modulo Acyclicity

We recall the standard translation of logic programs under stable model semantics into propositional satisfiability modulo acyclicity. The translation follows earlier work on ASP-to-SAT encodings (Gebser, Janhunen, and Rintanen 2014b; Bomanson et al. 2016) and consists of three phases: instrumentation with an explicit acyclicity constraint, normalization, and propositional completion.

To make the minimality condition of stable models explicit, the program is instrumented with additional rules that encode the acyclicity of the positive dependency graph. For atoms occurring in non-trivial strongly connected components, *dependency atoms* $\mathtt{dep}(a, b)$ are introduced to represent the activation of dependency edges (a, b). For each such dependency, a choice rule $\{\mathtt{dep}(a, b)\} \leftarrow b$ is added. To enforce well-supportedness, for each defining rule $r \in \mathrm{def}_P(a)$ a fresh atom $\mathtt{ws}(r)$ is introduced. Assuming that the positive body of r is ordered so that $b_1, \ldots, b_l$ belong to the same SCC as a, while $b_{l+1}, \ldots, b_n$ do not, well-support is captured by the rule

$$\begin{aligned}\mathtt{ws}(r) \leftarrow\ & \mathtt{dep}(a, b_1)\,, \ \ldots\,, \ \mathtt{dep}(a, b_l),\\ & b_{l+1}\,, \ \ldots\,, \ b_n, \ \mathtt{not}\ c_1\,, \ \ldots\,, \ \mathtt{not}\ c_m.\end{aligned} \tag{4}$$

for normal and choice rules, and by

$$\begin{aligned}\mathtt{ws}(r) \leftarrow\ & k \leq [\mathtt{dep}(a, b_1) = w_1\,, \ \ldots\,, \ \mathtt{dep}(a, b_l) = w_l,\\ & b_{l+1} = w_{l+1}\,, \ \ldots\,, \ b_n = w_n,\\ & \mathtt{not}\ c_1 = w_{n+1}\,, \ \ldots\,, \ \mathtt{not}\ c_m = w_{n+m}].\end{aligned} \tag{5}$$

for weight rules. A constraint is added to ensure that every true atom is well-supported. The resulting program, denoted $\mathrm{Tr}_{\mathrm{ACYC}}(P)$, is interpreted under *ASP modulo acyclicity* semantics, where supported models must induce an acyclic dependency graph.

Weight constraint programs (WCPs) are then rewritten into equivalent normal logic programs. Extended rule types, such as choice rules and weight rules, can be expressed using normal rules by introducing auxiliary atoms (Bomanson, Gebser, and Janhunen 2014). We denote by $\mathrm{Tr}_{\mathrm{NORM}}(P)$ the result of applying a fixed normalization scheme to a WCP P. Normalization preserves stable models up to the original program signature and enables the subsequent translation steps to operate on normal rules only.

The final phase translates the instrumented normal logic program into a propositional formula using Clark's completion. For each rule r, a fresh atom $\mathtt{bt}(r)$ is introduced to represent the satisfaction of the rule body:

$$\mathtt{bt}(r) \ \leftrightarrow \bigwedge_{b \in \mathrm{body}^+(r)} b \ \wedge \bigwedge_{c \in \mathrm{body}^-(r)} \neg c. \tag{6}$$

Each ordinary atom a is then defined by

$$a \ \leftrightarrow \bigvee_{r \in \mathrm{def}_P(a)} \mathtt{bt}(r). \tag{7}$$

The equivalences (6) and (7) are clausified using a Tseitin-style transformation to keep the translation linear in size. Dependency and well-support atoms introduced during instrumentation are treated as ordinary atoms and preserved by the completion. We denote the resulting propositional theory by $\mathrm{Tr}_{\mathrm{COMP}}(P)$.

Proposition 1. *(Bomanson et al. 2016) Let P be a weight constraint program. An interpretation $M\colon \mathrm{At}(P) \to \{\top, \bot\}$ is a stable model of P if and only if there exists a propositional interpretation $N\colon \mathrm{At}(\mathrm{Tr}_{\mathrm{COMP}}(\mathrm{Tr}_{\mathrm{NORM}}(\mathrm{Tr}_{\mathrm{ACYC}}(P)))) \to \{\top, \bot\}$ such that $N \models \mathrm{Tr}_{\mathrm{COMP}}(\mathrm{Tr}_{\mathrm{NORM}}(\mathrm{Tr}_{\mathrm{ACYC}}(P)))$, $M(a) = N(a)$ for all $a \in \mathrm{At}(P)$, and the dependency graph induced by $\{(a, b) \mid N(\mathtt{dep}(a, b)) = \top\}$ is acyclic.*

5 A General Transitive Closure Framework

In this section, we present a general transitive-closure framework. It is based on the classical transitive-closure encoding that we review first, but enforces transitivity only on a selected set of vertex triples while still guaranteeing acyclicity.

5.1 Classical Transitive Closure Encoding

Let $G = (V, E)$ be a directed graph and let $\leq$ be a fixed total order on V. For symmetry reduction, we introduce propositional variables $e_{u,v}$ only for pairs of distinct vertices with $u < v$. For convenience, we use the notation $e^*_{u,v}$ to denote $e_{u,v}$ if $u < v$ and $\neg e_{v,u}$ otherwise. For every dependency atom $\mathtt{dep}(u, v)$ occurring in the input, we add the clause

$$\neg\, \mathtt{dep}(u, v) \ \vee \ e^*_{u,v}. \tag{8}$$

This ensures that whenever a directed edge (u, v) is selected, the corresponding order literal is set consistently with the symmetry-reduced representation. As a consequence of this encoding, directed 2-cycles are inherently forbidden, since $e^*_{u,v}$ and $e^*_{v,u}$ cannot be simultaneously satisfied.

The classical transitive-closure encoding enforces transitivity using the transitive closure $G^+ = (V, E^+)$ of G, where $(u, v) \in E^+$ holds iff there exists a directed path from u to v in G. For every $(u, w, v) \in V \times V \times V$, it adds

$$\neg e^*_{u,w} \ \vee \ \neg e^*_{w,v} \ \vee \ e^*_{u,v}. \tag{9}$$

5.2 Generalized Transitive Closure Encoding

We generalize the classical transitive-closure encoding by enforcing transitivity only for a selected set $T \subseteq V \times V \times V$ of triples. We write $\Phi^{T,+}(G)$ for the formula consisting of all clauses (8) and all clauses (9) for $(u, w, v) \in T$. To reason about correctness, we use the concepts of the underlying graph of a model and T-transitive closure.

Definition 5.1 (Underlying graph of a model)**.** *Let Φ be a propositional formula whose atoms include dependency atoms $\mathtt{dep}(u, v)$, and let M be a model of Φ. The* underlying graph *of M is the directed graph $G_M = (V, E_M)$, where $E_M = \{(u, v) \mid M \models \mathtt{dep}(u, v)\}$.*

Definition 5.2 (T-transitive closure)**.** *Let $G = (V, E)$ be a directed graph and $T \subseteq V \times V \times V$. The* T-transitive closure *of G is the directed graph $G^{T,+} = (V, E^{T,+})$, where $E^{T,+} \subseteq V \times V$ is the least set of edges satisfying the following conditions: $(u, v) \in E$ implies $(u, v) \in E^{T,+}$, and whenever $(u, w, v) \in T$, $(u, w) \in E^{T,+}$, and $(w, v) \in E^{T,+}$, we have $(u, v) \in E^{T,+}$.*

Intuitively, the graph $G^{T,+}$ captures reachability in G when transitivity is permitted only along vertex triples explicitly listed in T. In contrast to the classical transitive closure G^+, the graph $G^{T,+}$ may contain strictly fewer edges, reflecting the fact that only selected compositions of paths are enforced.

Definition 5.3 (Cycle coverage)**.** *Let $G = (V, E)$ be a directed graph, $T \subseteq V \times V \times V$, and C be a directed cycle in G. The set T* covers the cycle *C if there exist distinct vertices $u, v \in C$ such that $(u, v) \in E^{T,+}$ and $(v, u) \in E^{T,+}$.*

This notion says that the restricted closure $G^{T,+}$ is still strong enough to witness the cyclicity of C via mutual reachability of some vertex pair on the cycle. When T covers C, any satisfying assignment of the corresponding transitivity constraints must encode a directed 2-cycle for that pair, which conflicts with acyclicity in the symmetry-reduced representation.

We now establish the soundness and completeness of our generalized transitive-closure constraints when they are combined with an arbitrary formula Φ that selects a directed graph via the dependency atoms. Intuitively, $\Phi^{T,+}(G)$ acts as a *cycle detector* over the underlying graph induced by a model: if T covers all directed cycles of G, then any cyclic choice of dependency edges would force a pair of opposite reachability literals to become true in the restricted closure, which is impossible under the symmetry-reduced representation. Consequently, adding $\Phi^{T,+}(G)$ does not rule out any acyclic solution allowed by Φ (completeness), but it eliminates every cyclic solution of Φ (soundness).

Lemma 5.1. *For a directed graph $G = (V, E)$ and a set $T \subseteq V \times V \times V$, consider a model M of $\Phi \wedge \Phi^{T,+}(G)$ with underlying graph G_M. Denote by $G_M^{T,+}$ the T-transitive closure of G_M. If $(u, v) \in E_M^{T,+}$, then $M \models e^*_{u,v}$.*

Proof. We prove the claim by induction on the construction of $G_M^{T,+}$ as defined in Definition 5.2. For the base case, suppose that $(u, v) \in E_M^{T,+}$ holds due to $(u, v) \in E_M$. Then trivially $M \models e^*_{u,v}$. For the inductive step, suppose $(u, v) \in E_M^{T,+}$ holds because there exists a vertex $w \in V$ and a triple $(u, w, v) \in T$ such that $(u, w) \in E_M^{T,+}$ and $(w, v) \in E_M^{T,+}$. By the induction hypothesis, we have $M \models e^*_{u,w}$ and $M \models e^*_{w,v}$. Since $(u, w, v) \in T$, the corresponding transitivity clause $\neg e^*_{u,w} \vee \neg e^*_{w,v} \vee e^*_{u,v}$ appears in $\Phi^{T,+}(G)$. Because M satisfies this clause and both $e^*_{u,w}$ and $e^*_{w,v}$ are true under M, we have $M \models e^*_{u,v}$. Thus, in all cases, $(u, v) \in E_M^{T,+}$ implies $M \models e^*_{u,v}$. □

Theorem 5.1 (Correctness of generalized transitive closure encoding)**.** *Let $G = (V, E)$ be a directed graph, Φ be a propositional formula whose atoms include dependency atoms $\mathtt{dep}(u, v)$, and let $T \subseteq V \times V \times V$. Assume that T covers every directed cycle of G. Then the following hold:*

(i) If Φ has a model M whose underlying graph G_M is acyclic, then $\Phi \wedge \Phi^{T,+}(G)$ is satisfiable by an extension of M.

(ii) If $\Phi \wedge \Phi^{T,+}(G)$ is satisfiable by a model M, and N is the restriction of M to the atoms occurring in Φ, then the underlying graph G_N is acyclic.

Proof. (i) Assume that Φ has a model M such that its underlying graph G_M is acyclic. Let $\prec$ be a topological order of G_M. We extend M to a model M' of $\Phi^{T,+}(G)$ by setting, for all distinct $p, q \in V$, $M' \models e^*_{p,q}$ iff $p \prec q$. By construction, M' agrees with M on all atoms of Φ, and hence $M' \models \Phi$. Consider any transitivity clause of $\Phi^{T,+}(G)$ corresponding to a triple $(p, q, r) \in T$, namely $\neg e^*_{p,q} \vee \neg e^*_{q,r} \vee e^*_{p,r}$. If $M' \models e^*_{p,q}$ and $M' \models e^*_{q,r}$, then $p \prec q$ and $q \prec r$, and by transitivity of $\prec$ we obtain $p \prec r$, hence $M' \models e^*_{p,r}$. Thus all clauses of $\Phi^{T,+}(G)$ are satisfied, and $M' \models \Phi \wedge \Phi^{T,+}(G)$.

(ii) Assume that $\Phi \wedge \Phi^{T,+}(G)$ is satisfiable and let M be a model of it. Let N be the restriction of M to the atoms of Φ. Suppose, for a contradiction, that the underlying graph G_N contains a directed cycle C. Since M extends N, the same cycle exists in G_M. Because T covers every directed cycle of G, it also covers C. By Definition 5.3, there exist distinct vertices $u, v \in C$ such that $(u, v) \in E_M^{T,+}$ and $(v, u) \in E_M^{T,+}$, where $G_M^{T,+}$ denotes the T-transitive closure of G_M. By Lemma 5.1, this implies $M \models e^*_{u,v}$ and $M \models e^*_{v,u}$. Since $u \neq v$, the symmetry-reduced definition of e^* makes these literals negations of each other, contradicting that $M \models \Phi^{T,+}(G)$. Hence, G_N must be acyclic. □

Encoding Size. The formula $\Phi^{T,+}(G)$ contains exactly $|T|$ transitivity clauses from (9), for a total of $|E| + |T|$ clauses. The variables are the dependency variables $\mathtt{dep}(a, b)$ for unordered pairs $\{a, b\}$ that occur in E or in some triple of T. This is $\mathcal{O}(|V|^2)$ variables. In practice, however, the number of introduced variables can be considerably smaller as some instantiations of our framework restrict T to a small subset of $V \times V \times V$. In the next section, we show how the size of the resulting encodings can be bounded in terms of structural measures of the graph G.

6 Instantiating the Framework

We now describe several concrete instantiations of the generalized transitive closure framework obtained by different choices of the transitivity triple set T. In each case, we briefly argue correctness by verifying cycle coverage with respect to the T-transitive closure $G^{T,+}$, and analyze the size of the resulting encoding.

6.1 Full Transitive Closure

For the naive instantiation corresponding to the full transitive closure, we set $T = V \times V \times V$. This enforces transitivity for all vertex triples and thus corresponds to explicitly

encoding the full transitive closure. Cycle coverage is immediate as every possible triple is included and hence $G^{T,+}$ coincides with the classical transitive closure G^+. The encoding introduced $\mathcal{O}(|V|^2)$ variables and $\mathcal{O}(|V|^3)$ clauses, which can be prohibitively large.

6.2 Alternative Transitive Closure Encoding

A more compact instantiation (Cussens et al. 2017) restricts transitivity to triples $T = \{(v, w, u) \mid v, w \in V,\ (w, u) \in E\}$. Here, transitivity is enforced only when the first edge is an arbitrary reachability edge and the second is an original edge. Cycle coverage holds because the resulting T-transitive closure $G^{T,+}$ is identical to the classical transitive closure G^+. The number of clauses is $\mathcal{O}(|V||E|)$ and variable $\mathcal{O}(|V|^2)$. This encoding is asymptotically smaller than the full transitive closure but still potentially large.

6.3 Ordered Transitive Closure Encoding

This instantiation further reduces the number of enforced transitivity constraints by fixing a vertex ordering α and restricting transitivity to triples $T = \{(v, w, u) \mid v, w \in V,\ (w, u) \in E,\ \alpha(v) < \alpha(w),\ \alpha(v) < \alpha(u)\}$. Thus, transitivity is enforced only when path composition respects the given ordering. Cycle coverage is achieved by letting x be the smallest vertex on a directed cycle C under α, for every other vertex y on C we have $(x, y) \in E^{T,+}$. In particular, if y is the in-neighbor of x on C, both (x, y) and (y, x) belong to $E^{T,+}$. Compared to the alternative transitive closure encoding, this instantiation reduces the number of transitivity clauses by roughly a factor of two, still using $\mathcal{O}(|V|^2)$ variables.

6.4 Feedback Vertex Set–Based Encoding

A further refined instantiation employs a given feedback vertex set $F \subseteq V$ together with a vertex ordering α. In this instantiation, transitivity is restricted to triples $T = \{(v, w, u) \mid v \in F,\ (w, u) \in E,\ \alpha(v) < \alpha(w) \text{ if } w \in F\}$. Thus, transitivity is enforced only for compositions rooted at vertices in the feedback vertex set, with ordering constraints applied among vertices of F. Cycle coverage holds because every directed cycle intersects F. Letting x be the vertex of minimum α on a cycle within F, for any other vertex y on the same cycle we obtain $(x, y) \in E^{T,+}$, and in particular for the predecessor y of x on the cycle both (y, x) and (x, y) belong to $E^{T,+}$. As a result, the T-transitive closure graph $G^{T,+}$ contains the necessary mutual reachability to detect cycles. The size of the encoding depends on $|F|$: the number of transitivity clauses is $\mathcal{O}(|F||E|)$, and the number of variables is $\mathcal{O}(|F||V|)$, yielding a substantially more compact encoding when F is small.

6.5 Vertex Elimination Encoding

The next instantiation is based on the vertex elimination process induced by a fixed ordering α (Rankooh and Janhunen 2022). Let G^*_α be the vertex elimination graph obtained by cumulatively adding fill-in edges during elimination, and for each vertex w let $F(w)$ denote the set of fill-in edges introduced when w is eliminated. We restrict transitivity to the set $T = \{(v, w, u) \mid (v, u) \in F(w)\}$, that is, transitivity is enforced only along reachability shortcuts explicitly created by the elimination process. Intuitively, each fill-in edge represents a valid shortcut created by vertex elimination, and restricting transitivity in this way makes the T-transitive closure graph $G^{T,+}$ reflect exactly these shortcuts. To see why cycle coverage holds, let C be a directed cycle in G, and let x and y be the last two vertices of C under the ordering α. During vertex elimination, all other vertices of C are eliminated before x and y, and the connectivity of C is successively bypassed through fill-in edges. As a result, the current elimination graph contains both edges (x, y) and (y, x). Because each such shortcut arises from a fill-in step and is encoded by a triple in T, both edges belong to the T-transitive closure graph $G^{T,+}$, forming a directed 2-cycle. Hence, $G^{T,+}$ captures mutual reachability for every directed cycle of G. The number of transitivity clauses equals the number of fill-in edges, which is $\mathcal{O}(\delta^2_\alpha(G)|V|)$ (Rankooh and Rintanen 2022), where $\delta_\alpha(G)$ is the elimination width, and the total number of variables is $\mathcal{O}(|E| + \delta_\alpha(G)|V|)$. This yields the standard vertex-elimination-based encoding of acyclicity, modified by symmetry-reducing variables.

6.6 Cycle Elimination Encoding

Finally, we detail an instantiation based on the cycle elimination process induced by a fixed ordering α. Let G^+_α be the cycle elimination graph obtained by cumulatively collecting the cycle elimination fill-in during the process, and let X_v denote the node of the cycle elimination decomposition tree at which vertex v is eliminated. We restrict transitivity to the set $T = \{(v, w, u) \mid w, u \in X_v,\ w \neq u\}$, that is, transitivity is enforced only among vertices that coexist in the same cycle elimination component at the moment v is eliminated. Intuitively, each such component represents a strongly connected subgraph whose internal reachability is relevant for detecting cycles, and restricting transitivity to these sets ensures that $G^{T,+}$ reflects exactly the reachability information exposed by cycle elimination. To see why cycle coverage holds, let C be a directed cycle in G, and let x be the first vertex of C under the ordering α. By the definition of the cycle elimination decomposition tree, all other vertices of C remain in the same strongly connected component at the moment x is eliminated, and hence all vertices of $C \setminus \{x\}$ are members of X_x. In particular, if y is the predecessor of x on the cycle C, then both (x, y) and (y, x) belong to the T-transitive closure graph $G^{T,+}$, yielding a directed 2-cycle. Thus $G^{T,+}$ captures mutual reachability for every directed cycle of G. The size of the encoding can be analyzed level by level using the cycle elimination decomposition tree. At each level, the total number of transitivity triples added to T is at most $|E|$, since each level corresponds to a decomposition into strongly connected components of a subgraph of G. Likewise, the number of dependency variables introduced per level is at most $|V|$. Summing over all levels yields a total of $\mathcal{O}(r_\alpha(G)|E|)$ transitivity clauses and $\mathcal{O}(r_\alpha(G)|V|)$ variables, where $r_\alpha(G)$ is the ordered cycle rank. This yields a cycle-elimination-based acyclicity encoding whose size is parameterized by the ordered cycle rank of G.

7 Empirical Evaluation

We report on an extensive empirical evaluation of the performance of SAT and MaxSAT solvers on the (Max)SAT encodings of typical ASP benchmarks using the various acyclicity encodings against the current state-of-the-art native ASP solvers. We emphasize that modern MaxSAT solvers for ASP optimization have not been thoroughly evaluated to-date to the best of our knowledge.

As the tool chain, we first use the tools lp2acyc, lp2normal2, and lp2sat from the ASPTOOLS library (Janhunen 2018) for instrumentation, normalization, and translation to SAT modulo graphs, respectively. Our implementations of the acyclicity encoding then translate the resulting SAT modulo graph instances to SAT (in case of decision problems) or MaxSAT (for optimization problems). The implemented translators are available at https://bitbucket.org/coreo-group/aspbymaxsat.

In our implementation of the encodings, for the vertex elimination based encoding (*VE*) we compute the elimination ordering using the minimum-degree heuristic, i.e., iteratively eliminating a vertex with the smallest degree at each step. For the cycle elimination based encoding (*CE*), we use a simple elimination ordering based on maximum degree, with the aim of efficiently breaking strongly connected components. For the feedback vertex set based encoding (*FVS*), vertices with maximum degree are iteratively added to the set until a feedback vertex set is obtained. For the ordered transitive closure encoding (*TC-ord*), we use the vertex ordering directly from the atom numbering provided by lp2sat. These heuristic methods are intentionally simple, and it is likely that employing more sophisticated strategies could further improve the compactness of the encodings and thereby solver performance. The full transitive closure encoding (*TC*) and the alternative transitive closure encoding (*TC-2*) are used as defined earlier.

For decision problems, we use Kissat (Biere et al. 2020) 4.0.2 as the SAT solver on the SAT encodings. For optimization, we mainly focus on the results obtained using the solution-improving search style MaxSAT solver Pacose (Paxian, Reimer, and Becker 2018) (version MSE2024). We also provide a separate comparison of the runtime performance resulting from using different MaxSAT solvers, representing different algorithmic approaches to MaxSAT: the implicit hitting set based MaxSAT solver MaxHS (Davies and Bacchus 2011) 5.0.0, the core-guided OLL-based MaxSAT solver UWrMaxSAT (Piotrów 2020) 1.7.0, and the clause-learning branch-and-bound MaxSAT solver WMaxCDCL (Coll et al. 2025) (version MSE2024). All solvers were run in their default settings without parameter tuning.

We compare to the state-of-the-art native ASP solvers Clingo (Gebser et al. 2019) 5.7.1 and WASP (Alviano et al. 2015) 2.0 both on decision and optimization problems. For Clingo, on optimization problems, we consider both its default branch-and-bound (BB) algorithm and the alternative core-guided OLL optimization algorithm (USC) (Andres et al. 2012) Clingo offers. We note that both the earlier native solvers Smodels and DLV, and the earlier-proposed SAT-based approaches a la ASSAT and Cmodels using loop formulas (as the approach can introduce a prohibitive number of loop formulas on non-tight programs (Giunchiglia, Lierler, and Maratea 2006; Janhunen 2006)) are known not to be competitive with Clingo and WASP.

We focus on non-tight ASP benchmarks, since tight programs can be translated through Clark's completion directly to SAT/MaxSAT without need for the acyclicity encodings and are as such uninteresting for the evaluation. The optimization benchmark sets (with their corresponding ASP Competition year in parentheses) are *Bayesian Network Learning (Bayes)* (2017), *Connected Maximum-Density Still Life (Connect)* (2015), *Markov Network Learning (Markov)* (2017), *Traveling Salesman Problem (TSP)* (2017), and *Valves Location Optimization (Valves)* (2015). The decision benchmark sets are *Combined Configuration (Comb)* (2015), *Knight Tour With Holes (Knight)* (2015), *Labyrinth (Lab)* (2015), *Maze Generation (Maze)* (2011), and *Random NonTight (Rand)* (2007), We additionally include a set of 300 Hamiltonian cycle (Ham) benchmarks using encoding from (Niemelä 1999b), derived from 30 randomly generated planar graphs with 60, 70, . . . , 150 vertices.

The experiments were run under Linux on computing nodes with 2.40-GHz Intel Xeon Gold 6148 CPUs and 381-GB memory, using a per-instance time limit of 1200 seconds (following the ASP Competitions) and a memory limit of 16 GB. All solvers were executed using a single thread.

Overviews of the results are shown in Tables 1–2 and Figure 3. Tables 1–2 show the number of solved instances on decision and optimization problems, respectively, for the various (Max)SAT encodings, with a comparison to Clingo and Wasp. Overall, we observe that especially the CE encoding yields very competitive results compared to the state-of-the-art native ASP solvers. CE results in the greatest number of instances solved for both decision and optimization problems. All encodings apart from TC-2 result in the (Max)SAT solver outperforming Clingo and Wasp on the entire benchmark set. The relative performance of the (Max)SAT solver compared to Clingo and Wasp depends on the problem domain, with somewhat complementary performance. The SAT solver outperforms the ASP solvers most noticeably on Ham, and the MaxSAT solver very significantly on Connect. Figure 3 provides an alternative view to the results, showing the number of solved instances over per-instance time, over 30 randomly sampled instances per benchmark domain to balance the impact of individual domains[1]. Figure 3(b) in particular shows that the MaxSAT solver outshines the ASP solvers overall.

Finally, we consider the relative performance of different state-of-the-art MaxSAT solvers on the optimization problems (Table 3) using the well-performing CE encoding. As Clasp can also be used directly as a MaxSAT solver, we also include it (column "Clasp (MaxSAT)") in its default settings in this comparison. The results highlight how the variety of MaxSAT solvers available today, implementing different types of SAT-based algorithms, have the potential of signif-

[1]We confirmed by repeating the sampling 10 times that the so obtained median runtimes exhibit similar trends as the data presented in Figure 3)

Benchmark	Acyclicity Encoding + Kissat						Clingo	Wasp
	VE	CE	FVS	TC-ord	TC-2	TC		
Comb	68	62	56	56	48	34	67	19
Ham	294	300	299	296	300	265	217	286
Knight	27	30	28	29	29	7	40	40
Lab	216	217	212	211	214	113	223	195
Maz	50	50	46	50	39	0	50	50
Rand	14	14	14	14	14	14	14	14
Total	669	673	655	656	644	433	611	604

Table 1: Decision problems: Number of instances solved within 1200 s.

Benchmark	Acyclicity Encoding + Pacose						Clingo		Wasp
	VE	CE	FVS	TC-ord	TC-2	TC	BB	USC	
Bayes	50	50	50	45	46	16	44	32	33
Connect	109	111	110	106	106	101	18	73	79
Markov	33	30	7	20	8	0	34	0	0
TSP	0	0	0	0	0	0	0	0	0
Valve	12	13	12	12	11	12	58	23	47
Total	204	204	179	183	171	129	154	128	159

Table 2: Optimization problems: Number of instances solved within 1200 s.

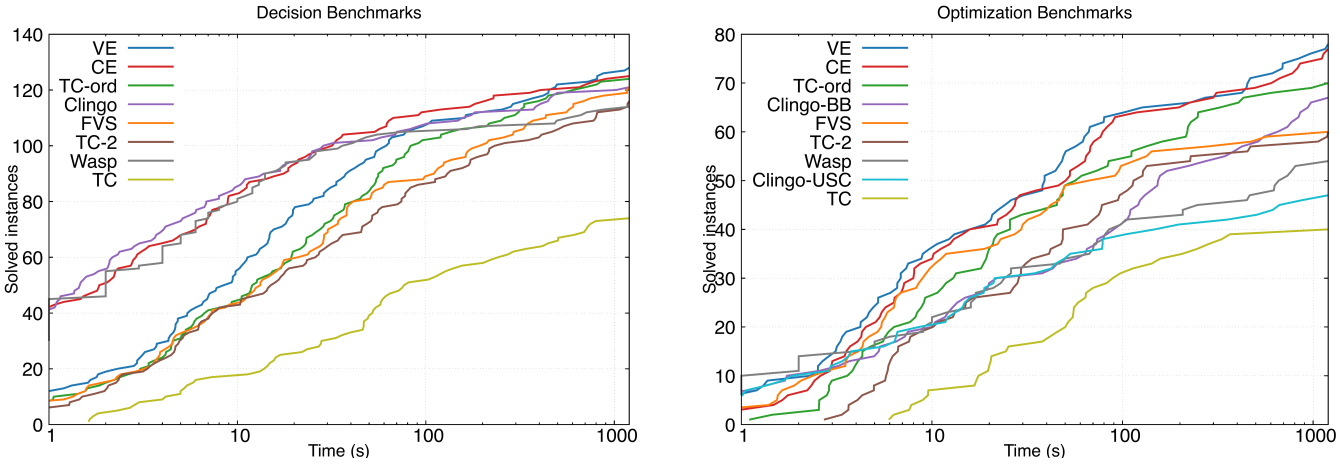

Figure 3: Number of instances solved over per-instance time. Left: decision benchmarks, right: optimization benchmarks.

Domain	Pacose	UWrMaxSat	WMaxCDCL	MaxHS	Clasp (MaxSAT)	VBS
Bayes	50	55	58	49	48	58
Connect	111	82	48	78	16	112
Markov	30	16	47	16	57	57
TSP	0	23	9	4	0	23
Valve	13	16	9	12	15	16
Total	204	192	171	159	136	266

Table 3: Number of solved instances by different MaxSAT solvers using the CE encoding.

icantly scaling up ASP optimization on a per-domain basis. While we focused on Pacose in reporting the earlier comparison, we observe that, e.g., on TSP benchmarks especially UWrMaxSat excels: it solves 23 TSP benchmarks, whereas the native ASP solvers cannot solve any (see Table 2). Interestingly, Clasp as a MaxSAT solver excels on Markov, solving 57 benchmarks against Clingo solving 34 in its default mode, while neither Clingo using USC nor Wasp can solve any of the Markov benchmarks. All in all, the diversity of MaxSAT solvers offers significant advances to the state of the art in ASP optimization.

8 Conclusions

A key challenge in making use of recent and on-going advances in SAT and MaxSAT solving for ASP is developing effective ways of enforcing acyclicity of the underlying dependence graphs of ASP programs in order to capture the stable model semantics. We formalized a general framework for such encodings, capturing various recently-proposed refined structure-based transitive closure encodings of acyclicity as well as novel ones in a unifying way. Instantiating several encodings that fit the framework, we showed through an extensive empirical evaluation that by employing the encodings—such as on based on cycle elimination—state-of-the-art SAT and MaxSAT solvers in particular turn out to be competitive and even outperform state-of-the-art native ASP solvers. Developing further encodings as instantiation of the framework; making use of future improvements in SAT and MaxSAT and tuning them specifically for ASP; employing the various alternative encoding either within a parallel and/or portfolio solving approach; and foundations research towards developing an in-depth understanding on which types of MaxSAT algorithms work best when are examples of interesting directions for future work.

Acknowledgements

This work was financially supported by Research Council of Finland (grant 356046) and Helsinki Institute for Information Technology HIIT. The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

AI Declaration

The authors have not employed any generative AI tools.

References

Alviano, M.; Dodaro, C.; Faber, W.; Leone, N.; and Ricca, F. 2013. WASP: A native ASP solver based on constraint learning. In Cabalar, P., and Son, T. C., eds., *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, volume 8148 of *Lecture Notes in Computer Science*, 54–66. Springer.

Alviano, M.; Dodaro, C.; Leone, N.; and Ricca, F. 2015. Advances in WASP. In Calimeri, F.; Ianni, G.; and Truszczynski, M., eds., *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings*, volume 9345 of *Lecture Notes in Computer Science*, 40–54. Springer.

Andres, B.; Kaufmann, B.; Matheis, O.; and Schaub, T. 2012. Unsatisfiability-based optimization in clasp. In Dovier, A., and Costa, V. S., eds., *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, Budapest, Hungary, September 4-8, 2012*, volume 17 of *LIPIcs*, 211–221. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Bacchus, F.; Järvisalo, M.; and Martins, R. 2021. Maximum satisfiability. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. 929–991.

Biere, A.; Fazekas, K.; Fleury, M.; and Heisinger, M. 2020. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Balyo, T.; Froleyks, N.; Heule, M.; Iser, M.; Järvisalo, M.; and Suda, M., eds., *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, 51–53. University of Helsinki.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2021. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Bomanson, J.; Gebser, M.; Janhunen, T.; Kaufmann, B.; and Schaub, T. 2016. Answer set programming modulo acyclicity. *Fundam. Informaticae* 147(1):63–91.

Bomanson, J.; Gebser, M.; and Janhunen, T. 2014. Improving the normalization of weight rules in answer set programs. In *JELIA 2014*, 166–180.

Clark, K. L. 1977. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*, Advances in Data Base Theory, 293–322. New York: Plemum Press.

Coll, J.; Li, C. M.; Li, S.; Habet, D.; and Manyà, F. 2025. Solving weighted maximum satisfiability with branch and bound and clause learning. *Comput. Oper. Res.* 183:107195.

Cussens, J.; Järvisalo, M.; Korhonen, J. H.; and Bartlett, M. 2017. Bayesian network structure learning with integer programming: Polytopes, facets and complexity. *J. Artif. Intell. Res.* 58:185–229.

Davies, J., and Bacchus, F. 2011. Solving MAXSAT by solving a sequence of simpler SAT instances. In Lee, J. H., ed., *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, 225–239. Springer.

Eggan, L. C. 1963. Transition graphs and the star-height of regular events. *Michigan Mathematical Journal* 10:385–397.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.* 19(1):27–82.

Gebser, M.; Janhunen, T.; and Rintanen, J. 2014a. Answer set programming as SAT modulo acyclicity. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, 351–356. IOS Press.

Gebser, M.; Janhunen, T.; and Rintanen, J. 2014b. Answer

set programming as SAT modulo acyclicity. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, 351–356. IOS Press.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.* 187:52–89.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2004. Sat-based answer set programming. In McGuinness, D. L., and Ferguson, G., eds., *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, 61–66. AAAI Press / The MIT Press.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2006. Experiments with sat-based answer set programming. In *Proceedings of the Workshop on Logic and Search (LaSh)*.

Gruber, H. 2012. Digraph complexity measures and applications in formal language theory. *Discret. Math. Theor. Comput. Sci.* 14(2):189–204.

Hunter, P., and Kreutzer, S. 2007. Digraph measures: Kelly decompositions, games, and orderings. In *Proceedings of SODA 2007*, 637–644. SIAM.

Janhunen, T.; Niemelä, I.; and Sevalnev, M. 2009. Computing stable models via reductions to difference logic. In Erdem, E.; Lin, F.; and Schaub, T., eds., *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*, volume 5753 of *Lecture Notes in Computer Science*, 142–154. Springer.

Janhunen, T. 2004. Representing normal programs with clauses. In de Mántaras, R. L., and Saitta, L., eds., *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, 358–362. IOS Press.

Janhunen, T. 2006. Some (in)translatability results for normal logic programs and propositional theories. *J. Appl. Non Class. Logics* 16(1-2):35–86.

Janhunen, T. 2018. Cross-translating answer set programs using the ASPTOOLS collection. *Künstliche Intell.* 32(2-3):183–184.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.* 7(3):499–562.

Lifschitz, V. 2019. *Answer Set Programming*. Springer.

Lin, F., and Zhao, Y. 2002. ASSAT: computing answer sets of a logic program by SAT solvers. In Dechter, R.; Kearns, M. J.; and Sutton, R. S., eds., *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, 112–118. AAAI Press / The MIT Press.

Lin, F., and Zhao, J. 2003. On tight logic programs and yet another translation from normal logic programs to propositional logic. In Gottlob, G., and Walsh, T., eds., *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, 853–858. Morgan Kaufmann.

Liu, G.; Janhunen, T.; and Niemelä, I. 2012. Answer set programming via mixed integer programming. In Brewka, G.; Eiter, T.; and McIlraith, S. A., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*. AAAI Press.

Marques Silva, J. P., and Sakallah, K. A. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers* 48(5):506–521.

Marques-Silva, J.; Lynce, I.; and Malik, S. 2021. Conflict-driven clause learning SAT solvers. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. 133–182.

Nguyen, M.; Janhunen, T.; and Niemelä, I. 2011. Translating answer-set programs into bit-vector logic. In Tompits, H.; Abreu, S.; Oetsch, J.; Pührer, J.; Seipel, D.; Umeda, M.; and Wolf, A., eds., *Applications of Declarative Programming and Knowledge Management - 19th International Conference, INAP 2011, and 25th Workshop on Logic Programming, WLP 2011, Vienna, Austria, September 28-30, 2011, Revised Selected Papers*, volume 7773 of *Lecture Notes in Computer Science*, 95–113. Springer.

Niemelä, I. 1999a. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.* 25(3-4):241–273.

Niemelä, I. 1999b. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.* 25(3-4):241–273.

Paxian, T.; Reimer, S.; and Becker, B. 2018. Dynamic polynomial watchdog encoding for solving weighted maxsat. In Beyersdorff, O., and Wintersteiger, C. M., eds., *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, 37–53. Springer.

Piotrów, M. 2020. Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems. In *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*, 132–136. IEEE.

Rankooh, M. F., and Janhunen, T. 2022. Efficient computation of answer sets via SAT modulo acyclicity and vertex elimination. In Gottlob, G.; Inclezan, D.; and Maratea, M., eds., *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*, 203–216. Springer.

Rankooh, M. F., and Rintanen, J. 2022. Propositional encodings of acyclicity and reachability by using vertex elimination. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, 5861–5868. AAAI Press.

Rankooh, M. F.; Niskanen, A.; and Järvisalo, M. 2025. Cost-optimal delete-free classical planning via maximum satisfiability. In *Proceedings of the 22st International Conference on Principles of Knowledge Representation and Reasoning, KR 2025, Melbourne, Australia. November 11-17, 2025*.

Rose, D.; Tarjan, R.; and Lueker, G. 1976. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal of Computing* 5(2):266–283.

Syrjänen, T., and Niemelä, I. 2001. The smodels system. In Eiter, T.; Faber, W.; and Truszczynski, M., eds., *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings*, volume 2173 of *Lecture Notes in Computer Science*, 434–438. Springer.