Computing Secure Sets in Graphs using Answer Set Programming

Michael Abseher, Bernhard Bliem, Günther Charwat⋆, Frederico Dusberger and Stefan Woltran

Institute of Information Systems 184/2
Vienna University of Technology
Favoritenstrasse 9–11, 1040 Vienna, Austria
`[abseher,bliem,gcharwat,dusberg,woltran]@dbai.tuwien.ac.at`

Abstract. The notion of secure sets is a rather new concept in the area of graph theory. Applied to social network analysis, the goal is to identify groups of entities that can repel any attack or influence from the outside. In this paper we tackle this problem by utilizing Answer Set Programming (ASP). It is known that verifying whether a set is secure in a graph is already co-NP-hard. Therefore, the problem of enumerating all secure sets is challenging for ASP and its systems. In particular, encodings for this problem seem to require disjunction and also recursive aggregates. Here, we provide such encodings and analyze their performance using the Clingo system. Furthermore, we study several problem variants, including multiple secure or insecure sets, and weighted graphs.

Keywords: secure sets, security in graphs, alliances in graphs

1 Introduction

Studies of group behavior, social interactions and relationships are at the core of social sciences. On a graph-theoretic level, these interactions can be modeled as a social network, which is a graph where vertices represent entities (e.g., persons, actors or agents) and edges denote some relation among the entities (e.g., acquaintanceship, influence or friendship). Social network analysis then deals with the study of metrics and properties of the network. This includes identification of "important" entities such as bridges or central vertices as well as structural characteristics such as cliques, clusters or cohesive groups (see, e.g., [22, 27]).

Here, we consider *secure sets*, a relatively new concept that was introduced by Brigham et al. [6]. Intuitively, a secure set is a group of entities that can withstand any attack from the outside and, in particular, even all subsets of the group can defend themselves against attacks. More precisely, the Secure Set problem asks for a non-empty set of vertices S in a graph G such that for each subset $X \subseteq S$, $|N[X] \cap S| \geq |N[X] \setminus S|$ holds. Here, $N[X]$ denotes the closed

⋆ Corresponding author. E-mail: `gcharwat@dbai.tuwien.ac.at` neighborhood of X in G, i.e., the set X together with all vertices adjacent to some vertex in X. The concept of secure sets can be applied, for instance, in the area of opinion research to analyse group behaviour and to identify insusceptible peer groups, or in strategic settings where entities occupy spots on a map such that they can defend themselves against any attack from neighbors. Identifying secure sets is intractable; in particular it is known that already the verification task of checking if a given set is secure in a given graph is co-NP-complete [21].

In this work we study secure sets as well as several problem variants, including minimal secure sets and a partitioning of the graph into several secure or insecure sets. Additionally, we consider weighted graphs. To allow for an efficient and extensible programming model, we employ the declarative paradigm of Answer Set Programming (ASP) [5], which has been shown to be very effective for tackling computationally complex problems [1, 24, 25]. Its *Guess & Check* approach allows for easy specification and is often very efficient in solving NP-complete problems in practice. By allowing also disjunctive logic programs, ASP can even capture problems up to the second level of the polynomial hierarchy [10] like, for instance, Strategic Companies, Minimal Diagnosis or Complex Optimization of Answer Sets [1].

Given the definition of SECURE SET and the related complexity-theoretic results, enumerating secure sets is a challenging task, even when applying ASP. We identify the following obstacles toward a suitable ASP encoding:

- Since the verification problem is already co-NP-hard, encodings require the full expressive power of ASP, in particular disjunction (unless NP = co-NP).
- Such encodings typically rely on a recursive schema (the so-called saturation technique) for the verification part.
- From the definition of secure sets we can conclude that recursive aggregates have to be employed [13].

Aggregates in ASP have been thoroughly investigated [13, 14, 20, 26]. Due to various underlying semantics, the implementation of aggregates differs between individual ASP systems. Moreover, for the sake of simplicity, some systems impose restrictions on aggregates such that not everything that is syntactically and semantically expressible can also be used in practice. Therefore, we give a detailed account of how our desired encodings can be realized using Clingo 4, one of the most prominent ASP systems.

Besides providing ASP-based implementations for SECURE SET, we believe that the developed encodings can also serve as fruitful benchmarks for ASP systems. In particular, there exists a vast amount of benchmarks for problems in NP, but problems at the second level of the polynomial hierarchy have not been investigated to this extent (with some exceptions, such as [1], [4] or [18]).

To summarize, our main contributions are the following:

- We provide encodings for SECURE SET and related problem variants, which to the best of our knowledge have not been tackled with ASP yet. Our encodings require the "full power" of ASP, i.e., disjunction (in combination with the saturation technique) plus aggregates.

– We state theoretical observations in the form of new characterizations for secure sets and provide new complexity results for some of the problem variants.
– We present several optimizations for our encodings and provide an experimental analysis, thereby comparing the run-time performance of the encoded variants.

This paper is structured as follows. Section 2 covers the required background on secure sets and ASP. In Section 3 we present our ASP encodings for Secure Set. Problem variants are discussed in Section 4. Experimental results for the encodings are reported in Section 5. Section 6 concludes the paper with a discussion of our results.

2 Preliminaries

In this section, we define the notions of secure sets and then give a brief introduction to ASP. Beside the basic ASP syntax and semantics we further describe the concept of the saturation technique and the commonly used language extension for aggregates.

2.1 Secure Sets

Let $G = (V, E)$ be a simple graph with vertex set V and edge set E. The set of vertices adjacent to a vertex $v \in V$, the *open neighborhood* of v, is denoted by $N(v)$. The *closed neighborhood* $N[v]$ of a vertex $v \in V$ is the open neighborhood of v together with the vertex v itself, formally $N[v] = N(v) \cup \{v\}$. For a set $S \subseteq V$, $N[S] = \bigcup_{v \in S} N[v]$ defines the closed neighborhood of S, and $N(S) = N[S] \setminus S$ is the open neighborhood of S.

A *defensive alliance* of $G = (V, E)$ is a subset $S \subseteq V$ such that for every $x \in S$ the inequality $|N[x] \cap S| \geq |N[x] \setminus S|$ is satisfied. For a better understanding, one can think of the vertices of $N[x] \cap S$ as the defenders of x and those of $N[x] \setminus S$ as the attackers of x. This means that for any vertex contained in a defensive alliance there are at least as many defenders as there are attackers and so any attack on a single vertex can be repelled.

A stronger notion is that of *secure sets* [6] where simultaneous attacks on more than a single vertex have to be repelled. Formally, this is captured as follows.

Definition 1. *Given a graph $G = (V, E)$, a non-empty set $S \subseteq V$ is* secure *in G if for* all *subsets $X \subseteq S$ the following inequality holds.*

$$|N[X] \cap S| \geq |N[X] \setminus S| \qquad (1)$$

On the contrary, we say that a non-empty set $S \subseteq V$ is *insecure* in G in case Inequality (1) is not satisfied for some $X \subseteq S$. Identifying secure sets is at the core of all secure set problem variants that we will consider in this paper.

2.2 Answer Set Programming

ASP [5] is a declarative language where a *program* Π is a finite set of *rules*

$$a_1 | \dots | a_k \leftarrow b_1, \dots, b_m, \mathit{not}\, b_{m+1}, \dots, \mathit{not}\, b_n.$$

where $a_1, \dots, a_k, b_1, \dots, b_n$ are *atoms*. The constituents of a rule $r \in \Pi$ are its *head* $h(r) = \{a_1, \dots, a_k\}$, and its *body* consisting of $b^+(r) = \{b_1, \dots, b_m\}$ and $b^-(r) = \{b_{m+1}, \dots, b_n\}$. Intuitively, r states that if an answer set contains all of $b^+(r)$ and none of $b^-(r)$, then it contains some element of $h(r)$. An *interpretation* I is a subset of atoms over the domain. I satisfies a rule r iff $I \cap h(r) \neq \emptyset$ or $b^-(r) \cap I \neq \emptyset$ or $b^+(r) \setminus I \neq \emptyset$. I is a *model* of a set of rules iff it satisfies each rule. I is an *answer set* of a program Π iff it is a subset-minimal model of the program $\Pi^I = \{h(r) \leftarrow b^+(r) \mid r \in \Pi, b^-(r) \cap I = \emptyset\}$, called the *Gelfond-Lifschitz reduct* of Π with respect to I [19]. For a program Π we denote the set of its answer sets by $\mathcal{AS}(\Pi)$.

ASP systems allow first-order atoms of the form $p(t_1, \dots, t_l)$ where p is a *predicate* of arity $l \geq 0$ and each t_i is either a variable or a constant. An atom is *ground* if it is free of variables. For any program Π, let U_Π be the set of all constants appearing in Π. For non-ground programs, the above semantics can be utilized by first applying, to each rule $r \in \Pi$, all possible substitutions from the variables in r to elements of U_Π.

The Saturation Technique. The *saturation technique* allows us to represent problems on the second level of the polynomial hierarchy by encoding the co-NP-check in ASP [11]. It relies on the fact that rule heads may contain disjunctions. This way, all atoms that are subject to a guess can be jointly contained in an answer set.

The idea for employing this step to compute solutions for problems on the second level of the polynomial hierarchy is the following. First of all, we guess a solution candidate for which we want to know if all possibilities of it being in fact no valid solution fail. Therefore, at the same time we also guess (using disjunction) a potential witness for the solution candidate being invalid. If this second guess indeed does not yield such a witness, we derive a designated atom that causes all atoms in the disjunction for guessing witnesses to be set to true. By doing so, all models not amounting to a witness are "saturated" with all the atoms in this disjunction. So, if a valid solution has been guessed, all guesses of potential witnesses thus collapse to a unique maximal answer set. On the other hand, if we have managed to guess a witness, we kill the solution candidate by means of a constraint. Invalid solution candidates are then discarded by the minimal model semantics because each model of the program that does not encode a witness is saturated and is thus not a *minimal* model of the reduct. For more details, we refer the reader to [11].

A concept that is often applied in combination with saturation is that of a "loop" (see, e.g., [12]). A loop allows to avoid (unstratified) default negation on the saturated parts of the program by "iterating" over the witnesses to be checked.

Aggregates. In the following we give an overview on the semantics of aggregates. In order to cope with simple arithmetic operations the basic ASP language is commonly extended by aggregate functions (cf. the ASP-Core-2 input language [7] and, e.g., [13]).

Syntax. An *aggregate element* e is of the form

$$t_1, \ldots, t_m : l_1, \ldots, l_n$$

where $t_1, \ldots, t_m$ are terms (denoted by $\mathrm{Term}(e)$) and $l_1, \ldots, l_n$ are atoms, (denoted by $\mathrm{Conj}(e)$). An *aggregate atom* has the form

$$\texttt{\#aggr}\{e_1; \ldots ; e_n\} \prec t$$

where $\texttt{\#aggr}$, called an *aggregate function*, is one of $\{\texttt{\#count}, \texttt{\#sum}, \texttt{\#max}, \texttt{\#min}\}$, $e_1, \ldots, e_n$ are aggregate elements, $\prec \in \{<, \leq, >, \geq, =, \neq\}$ is an *aggregate relation* and t is a term. ASP programs under this extension allow aggregate atoms in rule bodies.

In a given program Π an aggregate atom $\texttt{\#aggr}\{e_1; \ldots ; e_n\} \prec t$ is called *recursive* if it involves a cyclic dependency, i.e., if it contains an aggregate element e with an atom $a \in \mathrm{Conj}(e)$ and there is a rule r with an atom $b \in h(r)$ such that a and b cannot be stratified, i.e., there is no unique ordering for the rules involving a and b w.r.t. their evaluation. Otherwise it is called *non-recursive.*

Example 1. The program

$$\Pi : \{p(1) \leftarrow \texttt{\#sum}\{X : p(X)\} > 0\}.$$

contains a recursive aggregate since the atom $p(1)$ in the head of the rule depends on an aggregate atom involving $p(1)$. $\triangle$

Semantics. Aggregate functions are evaluated with respect to an interpretation I. A straightforward semantics for aggregates was proposed by Dell'Armi et al. [8] and Faber et al. [13]. In the following we describe the semantics for the ground case. The valuation of an aggregate element $e = \langle t_1, \ldots, t_m : l_1, \ldots, l_n \rangle$ is defined by

$$I(e) = \begin{cases} t_1 & \text{if } I \models \mathrm{Conj}(e) \\ 0 & \text{otherwise,} \end{cases}$$

i.e., the projection of $\mathrm{Term}(e)$ on the first term, if $\mathrm{Conj}(e)$ evaluates to true under I, and 0 otherwise. An aggregate function $\texttt{\#aggr}\{e_1; \ldots ; e_n\}$ is now simply evaluated as the application of the function denoted by $\texttt{\#aggr}$ on $\bigcup_{i=1}^{n} I(e_i)$. If $\bigcup_{i=1}^{n} I(e_i)$ is not in the domain of $\texttt{\#aggr}$, then $I(\texttt{\#aggr}\{e_1; \ldots ; e_n\}) = \bot$. Based on these definitions, $\texttt{\#aggr}\{e_1; \ldots ; e_n\} \prec t \in I$ holds iff

(i) $I(\texttt{\#aggr}\{e_1; \ldots ; e_n\}) \neq \bot$ and
(ii) $I(\texttt{\#aggr}\{e_1; \ldots ; e_n\}) \prec t$ holds.

Note that when aggregates are involved, the reduct of a program Π under an interpretation I is obtained by deleting the rules in which a body literal is false. For programs restricted to the basic ASP language, the semantics when using this notion of a reduct is equivalent to the semantics using the Gelfond-Lifschitz reduct [13].

Example 2. Consider the program

$$\Pi = \{q \leftarrow \texttt{\#sum}\{1 : q\} = 1.\},$$

and interpretations $\{\}$, $\{q\}$, for which we obtain:

$$\Pi^{\{\}} = \emptyset, \qquad \Pi^{\{q\}} = \Pi$$

Under the subset-minimal model semantics, $\{\}$ is the only answer set of Π, since $\{q\}$ is not minimal and thus not a valid answer set. $\triangle$

In contrast, a different way of defining the semantics of aggregates, which is implemented in, e.g., Clingo, is by translating them to another formula which is evaluated instead. Ferraris and Lifschitz proposed such a translation to evaluate so-called weight constraints [16], which can also be used to define the semantics of aggregates [14, 20]. In a nutshell, the aggregate is translated to a conjunction of implications reflecting the intuitive meaning of the aggregate.

It has been shown that both semantics are equivalent when positive programs are considered [15]. This result suggests that we can provide an encoding that can equivalently be run with DLV and Clingo. However, the current version of DLV (Dec 16, 2012) [3] does not support negative numbers and the WASP system (Jun 12, 2013) [2], which utilizes a modified version of DLV, does not support recursive aggregates. We therefore restrict ourselves to a realization using Clingo (4.4.0) [17].

Finally, as this method will be used afterwards in our encodings, we also want to highlight the fact that aggregates, in case they occur in the head of a rule, allow to model choices in a more natural way than by guessing all combinations and restricting via additional constraints. In fact, this kind of rule is commonly referred to as a choice rule. For instance, the rule

```
1 {q(X) : p(X)}.
```

models the fact that at least one `X` that is in the extension of the predicate `p/1` must also occur in the extension of the predicate `q/1`.

3 Solving the Secure Set Problem with ASP

In this section we show how the Secure Set problem can be solved by means of ASP. In particular, for a graph G, our goal is to enumerate all secure sets of G. The presented encodings are *fixed*, which means that they are independent of the concrete input instance. For an input graph $G = (V, E)$ we define its corresponding representation in ASP as a program Π_G where

$$\Pi_G = \{\texttt{vertex}(v) \mid v \in V\} \cup \{\texttt{edge}(u, v), \texttt{edge}(v, u) \mid \{u, v\} \in E\}.$$

As the definition of secure sets suggests, recursive aggregates are required to cope with the co-NP-hard task of verifying whether a guessed set of vertices is secure. In what follows, we give three proposals for such encodings. The main obstacle here is to get the recursive aggregates to work. In Clingo, it is required that the value the outcome of the aggregate is compared to is fixed, and moreover that predicates used in negative terms of aggregates have to stem from "outside" the saturation.

3.1 Loop Encoding

In order to fulfill the requirements above, we give a slightly different characterization of secure sets:

Lemma 1. *Let $G = (V, E)$ be a graph. A non-empty set $S \subseteq V$ is secure in G iff for all $X \subseteq S$*

$$|N[X] \cap S| + |(N[S] \setminus N[X]) \setminus S| - |N[S] \setminus S| \geq 0.$$

Proof. Recall that a non-empty set S is secure iff for all $X \subseteq S$

$$|N[X] \cap S| \geq |N[X] \setminus S|.$$

We now add $|(N[S] \setminus N[X]) \setminus S|$ to both sides. Note that $J = (N[S] \setminus N[X]) \setminus S$ and $J' = (N[X] \setminus S)$ are obviously disjoint and that $J \cup J' = N[S] \setminus S$. It follows that $|J| + |J'| = |N[S] \setminus S|$ and thus we obtain

$$|N[X] \cap S| + |(N[S] \setminus N[X]) \setminus S| \geq |N[S] \setminus S|$$

yielding the desired result. $\square$

Note that this characterization has a subtraction involved ($|N[S] \setminus S|$) but this number is independent from X, the set which has to be dealt with in the saturation part of the encoding.

In the following we present the ASP encoding for enumerating all secure sets of a given graph G using the concept of *loops*. To this extent, we require a total order over the vertices in G. We will use this to "loop" over all vertices, and to check whether certain conditions are fulfilled. We assume that the order is given by a positive logic program $\Pi_<$ where predicates `inf/1`, `succ/2` and `sup/1` specify the infimum, successor relation and supremum of the vertices, respectively. Note that (variants of) $\Pi_<$ occasionally reappear in literature, see, for example, [9].

Encoding 1: Secure Set - Loop Encoding (Π_{loop})

$$1\{\texttt{inS}(V) : \texttt{vertex}(V)\}. \quad (1)$$

$$\texttt{outS}(V) \leftarrow \texttt{vertex}(V), \mathit{not}\, \texttt{inS}(V). \quad (2)$$

$$\texttt{attackSet}(V) \leftarrow \texttt{inS}(U), \texttt{edge}(U,V), \texttt{outS}(V). \quad (3)$$

$$\texttt{inX}(V) \mid \texttt{outX}(V) \leftarrow \texttt{inS}(V). \quad (4)$$

$$\texttt{defendSet}(V) \leftarrow \texttt{inX}(V). \quad (5)$$

$$\texttt{defendSet}(V) \leftarrow \texttt{inX}(U), \texttt{edge}(U,V), \texttt{inS}(V). \quad (6)$$

$$\texttt{okupto}(U,V) \leftarrow \texttt{vertex}(U), \texttt{inf}(V), \texttt{outX}(V). \quad (7)$$

$$\texttt{okupto}(U,V) \leftarrow \texttt{vertex}(U), \texttt{inf}(V), \texttt{outS}(V). \quad (8)$$

$$\texttt{okupto}(U,V) \leftarrow \texttt{vertex}(U), \texttt{inf}(V), \texttt{inX}(V), \mathit{not}\, \texttt{edge}(U,V). \quad (9)$$

$$\texttt{okupto}(U,W) \leftarrow \texttt{okupto}(U,V), \texttt{succ}(V,W), \texttt{outX}(W). \quad (10)$$

$$\texttt{okupto}(U,W) \leftarrow \texttt{okupto}(U,V), \texttt{succ}(V,W), \texttt{outS}(W). \quad (11)$$

$$\texttt{okupto}(U,W) \leftarrow \texttt{okupto}(U,V), \texttt{succ}(V,W), \texttt{inX}(W), \mathit{not}\, \texttt{edge}(U,W). \quad (12)$$

$$\texttt{ok}(U) \leftarrow \texttt{okupto}(U,V), \texttt{sup}(V). \quad (13)$$

$$\texttt{inactAtt}(U) \leftarrow \texttt{ok}(U), \texttt{outS}(U), \texttt{edge}(U,V), \texttt{inS}(V). \quad (14)$$

$$\begin{aligned}\texttt{defended} \leftarrow\ & \#\mathrm{sum}\{1, V, \mathit{pos} : \texttt{defendSet}(V); \\ & 1, V, \mathit{pos} : \texttt{inactAtt}(V); \\ & -1, V, \mathit{neg} : \texttt{attackSet}(V)\} \geq 0.\end{aligned} \quad (15)$$

$$\texttt{inX}(V) \leftarrow \texttt{defended}, \texttt{inS}(V). \quad (16)$$

$$\texttt{outX}(V) \leftarrow \texttt{defended}, \texttt{inS}(V). \quad (17)$$

$$\leftarrow \mathit{not}\, \texttt{defended}. \quad (18)$$

Our loop encoding, Π_{loop}, is depicted in Encoding 1. In rule 1, a non-empty set $S \subseteq V$, denoted by `inS/1`, is guessed. Furthermore, predicate `outS/1` represents the vertices in $V \backslash S$ (rule 2). Vertices denoted by `attackSet/1` are in $N[S] \backslash S$ (rule 3). We then guess all sets $X \subseteq S$, marked with `inX/1`, and the other vertices (in $S \backslash X$) are given by `outX/1` (rule 4). Predicate `defendSet/1` specifies the set $N[X] \cap S$ (rules 5–6).

Rules 7–14 define the set $(N[S] \backslash N[X]) \backslash S$, that is, the set of *inactive attackers*. Intuitively, an inactive attacker is an attacker (i.e., a vertex in $N[S] \backslash S$) that is not adjacent to any vertex in X. In order to obtain the set of inactive attackers (without using default negation on saturation-dependent atoms), we have to loop. To be more precise, to determine for a vertex $u \in V$ whether $u \in (N[S] \setminus N[X])$ holds, we "loop" over all vertices $v \in V$ and check if one of the following conditions holds: (a) $v \in S \backslash X$ (rules 7,10); (b) $v \in V \backslash S$ (rules 8,11); or (c) $v \in X, \{u, v\} \notin E$ (rules 9,12). Finally, the predicate `inactAtt/1` denotes the set of inactive attackers (rule 14).

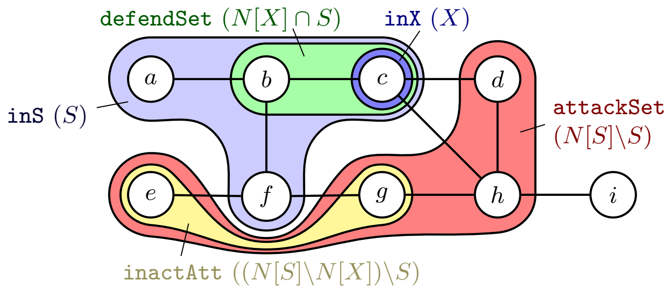
The set $X \subseteq S$ is defended if X satisfies the inequality given in Lemma 1 (rule 15). In case `defended/0` is obtained, we *saturate* by setting all vertices in S

to both `inX/1` and `outX/1` (rules 16–17); On the other hand, if X is not defended against the attackers, the answer set is removed (rule 18). In that case, due to the minimal model semantics, no answer set containing S is returned. Specifically, in case any $X \subseteq S$ cannot be defended, S is not secure and is therefore not returned as a solution.

The relation between secure sets of a graph G and the answer sets of the loop encoding is captured in the following proposition. The proof of Proposition 1 is given in Appendix A.

Proposition 1. *Let G be a graph and $\mathcal{S}$ be the collection of secure sets in G. Furthermore, let $\mathcal{A} = \mathcal{AS}(\Pi_G \cup \Pi_< \cup \Pi_{loop})$. Then, for every $S \in \mathcal{S}$, there exists an $A \in \mathcal{A}$ such that $\{v \mid \texttt{inS}(v) \in A\} = S$. Furthermore, for every $A \in \mathcal{A}$ it holds that $\{v \mid \texttt{inS}(v) \in A\} \in \mathcal{S}$. Finally, $|\mathcal{S}| = |\mathcal{A}|$.*

Example 3. Figure 1 shows an example graph G_{ex} with currently selected sets $S = \{a, b, c, f\}$ and $X = \{c\}$. The figure illustrates the corresponding defend set $N[X] \cap S = \{b, c\}$, attack set $N[S] \backslash S = \{d, e, g, h\}$ and the set of inactive attackers $(N[S] \backslash N[X]) \backslash S = \{e, g\}$. Since $|\{b, c\}| + |\{e, g\}| - |\{d, e, g, h\}| = 2 + 2 - 4 = 0 \geq 0$ holds, X is defended. For S to be secure, all $X \subseteq S$ have to be defended. For $X = \{c, f\}$, we have $N[X] \cap S = \{b, c, f\}$, $N[S] \backslash S = \{d, e, g, h\}$ and $(N[S] \backslash N[X]) \backslash S = \{\}$. Here, $|\{b, c, f\}| + |\{\}| - |\{d, e, g, h\}| = 3 + 0 - 4 = -1 \ngeq 0$, and therefore S is not secure. △



defendSet ($N[X] \cap S$)
inX (X)
inS (S)
attackSet ($N[S] \backslash S$)
inactAtt ($(N[S] \backslash N[X]) \backslash S$)
a
b
c
d
e
f
g
h
i

Fig. 1: Example graph G_{ex} with $S = \{a, b, c, f\}$ and $X = \{c\}$.

3.2 Loop Encoding with Restriction to Border Vertices

Lemma 1 can be strengthened by defining secure sets based on their *border*:

Definition 2. *Let $G = (V, E)$ and $S \subseteq V$. The* border *$b(S)$ of S is $b(S) = \{x \mid \{x, y\} \in E, x \in S, y \notin S\}$, i.e., the set of all vertices in S which are adjacent to at least one vertex $y \notin S$.*

Lemma 2. *Let $G = (V, E)$ be a graph. A non-empty set $S \subseteq V$ with border $b(S)$ is secure in G iff for all $X \subseteq b(S)$*

$$|N[X] \cap S| + |(N[S] \setminus N[X]) \setminus S| - |N[S] \setminus S| \geq 0.$$

Proof. Given a secure set S, since $b(S) \subseteq S$, the "only if" direction follows directly from Lemma 1. As for the other direction: Since the only negative part of the inequality ($|N[S] \setminus S|$) is independent of all $x \in S \setminus b(S)$, checking the inequality for all $X \subseteq b(S)$ suffices to verify that S is secure. □

Encoding 2: Secure Set - Border Loop Encoding ($\Pi_{border-loop}$)

$$1\{\texttt{inS}(V) : \texttt{vertex}(V)\}. \tag{1}$$
$$\texttt{outS}(V) \leftarrow \texttt{vertex}(V), \textit{not}\, \texttt{inS}(V). \tag{2}$$
$$\texttt{attackSet}(V) \leftarrow \texttt{inS}(U), \texttt{edge}(U,V), \texttt{outS}(V). \tag{3}$$
$$\texttt{border}(U) \leftarrow \texttt{inS}(U), \texttt{outS}(V), \texttt{edge}(U,V). \tag{4}$$
$$\texttt{inX}(V) \mid \texttt{outX}(V) \leftarrow \texttt{border}(V). \tag{5}$$
$$\texttt{outX}(V) \leftarrow \texttt{inS}(V), \textit{not}\, \texttt{border}(V). \tag{6}$$
$$\texttt{defendSet}(V) \leftarrow \texttt{inX}(V). \tag{7}$$
$$\texttt{defendSet}(V) \leftarrow \texttt{inX}(U), \texttt{edge}(U,V), \texttt{inS}(V). \tag{8}$$
$$\texttt{okupto}(U,V) \leftarrow \texttt{vertex}(U), \texttt{inf}(V), \texttt{outX}(V). \tag{9}$$
$$\texttt{okupto}(U,V) \leftarrow \texttt{vertex}(U), \texttt{inf}(V), \texttt{outS}(V). \tag{10}$$
$$\texttt{okupto}(U,V) \leftarrow \texttt{vertex}(U), \texttt{inf}(V), \texttt{inX}(V), \textit{not}\, \texttt{edge}(U,V). \tag{11}$$
$$\texttt{okupto}(U,W) \leftarrow \texttt{okupto}(U,V), \texttt{succ}(V,W), \texttt{outX}(W). \tag{12}$$
$$\texttt{okupto}(U,W) \leftarrow \texttt{okupto}(U,V), \texttt{succ}(V,W), \texttt{outS}(W). \tag{13}$$
$$\texttt{okupto}(U,W) \leftarrow \texttt{okupto}(U,V), \texttt{succ}(V,W), \texttt{inX}(W), \textit{not}\, \texttt{edge}(U,W). \tag{14}$$
$$\texttt{ok}(U) \leftarrow \texttt{okupto}(U,V), \texttt{sup}(V). \tag{15}$$
$$\texttt{inactAtt}(U) \leftarrow \texttt{ok}(U), \texttt{outS}(U), \texttt{edge}(U,V), \texttt{inS}(V). \tag{16}$$
$$\begin{aligned}\texttt{defended} \leftarrow\ & \#\text{sum}\{1, V, \textit{pos} : \texttt{defendSet}(V); \\ & 1, V, \textit{pos} : \texttt{inactAtt}(V); \\ & -1, V, \textit{neg} : \texttt{attackSet}(V)\} \geq 0.\end{aligned} \tag{17}$$
$$\texttt{inX}(V) \leftarrow \texttt{defended}, \texttt{inS}(V). \tag{18}$$
$$\texttt{outX}(V) \leftarrow \texttt{defended}, \texttt{inS}(V). \tag{19}$$
$$\leftarrow \textit{not}\, \texttt{defended}. \tag{20}$$

Regarding an appropriate ASP encoding, Lemma 2 allows us to restrict the guess on $X \subseteq b(S)$ where $b(S) \subseteq S$ holds. We adapt Π_{loop} to additionally take into account the *border* of S. In particular, it is sufficient to add a rule that gives the border, and to modify the guess of X accordingly. In Encoding 2 ($\Pi_{border-loop}$) we again guess non-empty sets $S \subseteq V$ and obtain `attackSet/1` (rules 1–3). Predicate `border/1` represents the set $b(S)$ (rule 4). Following Lemma 2, it suffices to guess sets $X \subseteq b(S)$ (rule 5). Furthermore, vertices in $S \backslash b(S)$ are marked with `outX/1` (rule 6). The remaining parts of $\Pi_{border-loop}$ correspond to the rules in Π_{loop}.

We capture the relation between secure sets and answer sets in the following proposition. Note that the correctness of Proposition 2 can be shown similarly to the proof of Proposition 1 given in Appendix A.

Proposition 2. *Let G be a graph and $\mathcal{S}$ be the collection of secure sets in G. Furthermore, let $\mathcal{A} = \mathcal{AS}(\Pi_G \cup \Pi_< \cup \Pi_{border-loop})$. Then, for every $S \in \mathcal{S}$, there exists an $A \in \mathcal{A}$ such that $\{v \mid \texttt{inS}(v) \in A\} = S$. Furthermore, for every $A \in \mathcal{A}$ it holds that $\{v \mid \texttt{inS}(v) \in A\} \in \mathcal{S}$. Finally, $|\mathcal{S}| = |\mathcal{A}|$.*

3.3 Alternative Secure Set Characterization

In order to avoid the loop from above, we provide an alternative characterization for secure sets, which to the best of our knowledge has not appeared in the literature yet. The intuition is to guess instead of the set $X \subseteq S$ a partition of the attack set $N[S] \backslash S$ into active and inactive attackers, and then check whether for such a partition a suitable set X exists.

Definition 3. *Let $G = (V, E)$ and $S \subseteq V$. We call any subset $B \subseteq N[S] \setminus S$ a* partial boundary *of S and identify by $\chi(B) = \{X \subseteq S \mid (N[X] \setminus S) = B\}$ the subsets of S that cover B. B is called a* valid partial boundary *if $\chi(B) \neq \emptyset$.*

Lemma 3. *Let $G = (V, E)$ and $S \subseteq V$ with $S \neq \emptyset$. Then S is secure in G iff for all valid partial boundaries B of S and all $X \in \chi(B)$, $|N[X] \cap S| \geq |B|$.*

Proof. "If" direction: Suppose S is not secure. Then there exists $X \subseteq S$, such that $|N[X] \cap S| < |N[X] \setminus S|$. Let $B = N[X] \setminus S$. We show that B is indeed a valid partial boundary. Obviously, $B \subseteq (N[S] \setminus S)$. Furthermore, $X \in \chi(B)$ and by assumption $|N[X] \cap S| < |B|$. "Only if" direction: Suppose there is a valid partial boundary B and an $X \in \chi(B)$ such that $|N[X] \cap S| < |B|$. By definition $X \subseteq S$ and $B = N[X] \setminus S$. Then S is not secure. □

We can strengthen the result as follows.

Definition 4. *Let $G = (V, E)$ and $S \subseteq V$, and $B \subseteq N[S] \setminus S$. Moreover, let $b(S)$ be the border of S. Then $\chi'(B) = \{X \subseteq b(S) \mid (N[X] \setminus S) = B\}$.*

Lemma 4. *Let $G = (V, E)$ and $S \subseteq V$ with $S \neq \emptyset$. Then S is secure in G iff for all valid partial boundaries B and all $X \in \chi'(B)$, $|N[X] \cap S| \geq |B|$ holds.*

Proof. The "only if" direction follows directly from Lemma 3 and the fact that $\chi'(B) \subseteq \chi(B)$ for all partial boundaries B of S. For the other direction note that $|N[X] \cap S| \geq |B|$ holds for $X \in \chi(B) \setminus \chi'(B)$ since for each $Y \in \chi(B)$ there is an $Y' \in \chi'(B)$ with $Y' \subseteq Y$ and that in this case $|N[Y] \cap S| \geq |N[Y'] \cap S|$. □

The main advantage of this definition is that we have for each relevant $X \subseteq S$, i.e., for each $X \subseteq b(S)$, the set $(N[S] \setminus N[X]) \setminus S$ directly given via $N[S] \setminus B$ where B is some partial boundary of S with $X \in \chi(B)$.

In Encoding 3 we again first guess S (rules 1–2), and obtain the (directed) edges from S to $N[X]\backslash S$, denoted by `borderEdge/2`, as well as the `attackSet/1` (rules 3–4). Next, the partial boundaries $B \subseteq N[S]\backslash S$, or active attackers (denoted by `actAtt/1`), are guessed (rule 5). In order to obtain the sets X where $\chi'(B) = \{X \subseteq b(S) \mid (N[X] \setminus S) = B\}$, $\chi' \neq \emptyset$ holds, each vertex in B must be adjacent to at least one vertex in $X \subseteq b(s)$, denoted by `inX/1` (rule 6). Furthermore, if there is a vertex $v \in N[X]\backslash S$ with $v \notin B$, then we can set X to be defended, since $X \notin \chi'(B)$ but $X \in \chi'(B')$ with $B \subset B'$ (rule 7). Note that this is required to be able to saturate. Rules 8–10 of Π_{alt} correspond to rules 7, 8 and 17 of $\Pi_{border-loop}$, i.e., they define the defend set and check whether the inequality given in Lemma 2 holds. Analogous to Π_{loop} and $\Pi_{border-loop}$, we saturate on the guessed `inX/1`, `actAtt/1` and `inactAtt/1` predicates.

Due to Lemma 4, we thus obtain the secure sets.

Proposition 3. *Let G be a graph and $\mathcal{S}$ be the collection of secure sets in G. Furthermore, let $\mathcal{A} = \mathcal{AS}(\Pi_G \cup \Pi_{alt})$. Then, for every $S \in \mathcal{S}$, there exists an $A \in \mathcal{A}$ such that $\{v \mid \texttt{inS}(v) \in A\} = S$. Furthermore, for every $A \in \mathcal{A}$ it holds that $\{v \mid \texttt{inS}(v) \in A\} \in \mathcal{S}$. Finally, $|\mathcal{S}| = |\mathcal{A}|$.*

Encoding 3: Secure Set - Alternative Encoding (Π_{alt})

$$1\{\texttt{inS}(V) : \texttt{vertex}(V)\}. \tag{1}$$
$$\texttt{outS}(V) \leftarrow \texttt{vertex}(V), \textit{not}\ \texttt{inS}(V). \tag{2}$$
$$\texttt{borderEdge}(U,V) \leftarrow \texttt{inS}(U), \texttt{outS}(V), \texttt{edge}(U,V). \tag{3}$$
$$\texttt{attackSet}(V) \leftarrow \texttt{borderEdge}(U,V). \tag{4}$$
$$\texttt{actAtt}(V) \mid \texttt{inactAtt}(V) \leftarrow \texttt{attackSet}(V). \tag{5}$$
$$\texttt{inX}(U) : \texttt{borderEdge}(U,V) \leftarrow \texttt{actAtt}(V), \texttt{outS}(V). \tag{6}$$
$$\texttt{defended} \leftarrow \texttt{inactAtt}(U), \texttt{inX}(V), \texttt{edge}(U,V). \tag{7}$$
$$\texttt{defendSet}(V) \leftarrow \texttt{inX}(V). \tag{8}$$
$$\texttt{defendSet}(V) \leftarrow \texttt{inX}(U), \texttt{edge}(U,V), \texttt{inS}(V). \tag{9}$$
$$\begin{aligned}\texttt{defended} \leftarrow\ \#\text{sum}\{&1, V, \textit{pos} : \texttt{defendSet}(V);\\ &1, V, \textit{pos} : \texttt{inactAtt}(V);\\ &-1, V, \textit{neg} : \texttt{attackSet}(V)\} \geq 0.\end{aligned} \tag{10}$$
$$\texttt{inX}(V) \leftarrow \texttt{defended}, \texttt{inS}(V). \tag{11}$$
$$\texttt{actAtt}(V) \leftarrow \texttt{defended}, \texttt{attackSet}(V). \tag{12}$$
$$\texttt{inactAtt}(V) \leftarrow \texttt{defended}, \texttt{attackSet}(V). \tag{13}$$
$$\leftarrow \textit{not}\ \texttt{defended}. \tag{14}$$

3.4 Using a #Count-Aggregate as an Alternative to #Sum

The encodings presented so far rely on the #sum-aggregate for checking whether a set X is defended (see Π_{loop} rule 15, $\Pi_{border-loop}$ rule 17, and Π_{alt} rule 10). The #sum-aggregate is required because the size of the attack set is subtracted from the combined size of the defend and inactive attacker sets. By adding $|V|$ to both sides of the inequality in Lemma 2, and since $|V| - |N[S]\backslash S| = |V\backslash(N[S]\backslash S)|$, we get rid of the negative part in the inequality. Thus, it is possible to use the #count-aggregate by replacing the respective rule with the following:

$$\begin{aligned}
\texttt{size}(N) &\leftarrow N = \#\text{count}\{V : \texttt{vertex}(V)\}.\\
\texttt{defended} &\leftarrow\ \#\text{count}\{V, \mathit{pos} : \texttt{defendSet}(V);\\
&\qquad\qquad\quad V, \mathit{pos} : \texttt{inactAtt}(V);\\
&\qquad\qquad\quad V, \mathit{neg} : \texttt{vertex}(V), \mathit{not}\, \texttt{attackSet}(V)\} \geq N, \texttt{size}(N).
\end{aligned}$$

Note that default negation is allowed here, since `attackSet/1` is independent of the saturation. Furthermore, instead of adding $|V|$, $|N[S]\backslash S|$ could also simply be shifted to the right side of the inequality. However, this drastically increases the size of the grounding.

3.5 Search Space Pruning

In order to improve the performance of our encodings, it is desirable to define "strengthening" constraints. Here, we define constraints that kill insecure sets $S \subseteq V$, independent of the guess for $X \subseteq S$. A vertex $v \in S$ cannot be defended if $|N(v) \cap S| < \lfloor |N(v)|/2 \rfloor$, or, in other words, less than half of its neighbors are in S. In that case S cannot be secure. In ASP, this is represented as follows:

$$\begin{aligned}
\texttt{deg}(V, D) &\leftarrow \texttt{vertex}(V), D = \#\text{count}\{U : \texttt{edge}(U, V)\}.\\
&\leftarrow \texttt{inS}(V), \texttt{deg}(V, D), \#\text{count}\{U : \texttt{edge}(U, V), \texttt{inS}(U)\} < D/2.
\end{aligned}$$

Note that degree `deg/2` can be computed during grounding (i.e., it is independent of any guess). Furthermore, ASP implements integer arithmetic, therefore $D/2$ always corresponds to $\lfloor |N(v)|/2 \rfloor$.

4 Secure Set Problem Variants

In this section we consider several extensions of the SECURE SET problem. At its core, they are based on the notion of a set being *secure* as given in Definition 1. First, we briefly illustrate how ASP can be applied in order to obtain minimal secure sets (MINIMUM SECURE SET problem). Then, we consider variants where the graph is partitioned into several secure sets (SECURE SET EQUILIBRIUM problem) or secure and insecure sets (SECURE-INSECURE SET PARTITIONING problem). Finally, in the WEIGHTED SECURE SET problem vertices additionally get assigned weights.

4.1 Minimum Secure Set Problem

In certain settings it might be desirable to obtain only minimal sets of entities that are secure, i.e., that can repel any attack from the outside. Consider, for example, a social network, modeled as a simple graph, where entities (e.g., persons, agents, ...) are represented as vertices and possible influence of entities on other entities is modeled in form of edges. With the notion of *minimal security* we can identify small groups of entities that are resistant to any influence from the outside. We capture this formally as follows.

Definition 5. *Given a graph $G = (V, E)$, a non-empty set $S \subseteq V$ is* minimal-secure *in G iff it is secure in G and there exists no (non-empty) secure set $S' \subseteq V$ in G with $|S'| < |S|$.*

In order to solve the corresponding Minimum Secure Set problem by means of ASP, it is sufficient to add the program

$$\Pi_{min} = \{\#\text{minimize}\{1, X, inS : \texttt{inS}(X), \texttt{vertex}(X)\}\}.$$

to the previously presented encodings. With this, only cardinality-minimal secure sets are obtained, i.e., the answer sets with a minimal number of `inS/1` occurrences are returned. We make this explicit in the following proposition.

Proposition 4. *Let G be a graph and $\mathcal{S}$ be the collection of minimal-secure sets in G. Furthermore, let $\mathcal{A} = \{\Pi_G \cup \Pi_{min} \cup \Pi\}$ where $\Pi \in \{\Pi_< \cup \Pi_{loop}, \Pi_< \cup \Pi_{border-loop}, \Pi_{alt}\}$. Then, for every $S \in \mathcal{S}$, there exists an $A \in \mathcal{A}$ such that $\{v \mid \texttt{inS}(v) \in A\} = S$. Furthermore, for every $A \in \mathcal{A}$ it holds that $\{v \mid \texttt{inS}(v) \in A\} \in \mathcal{S}$. Finally, $|\mathcal{S}| = |\mathcal{A}|$.*

4.2 Secure Set Equilibrium Problem

In the Secure Set Equilibrium problem we are looking for a partition of the graph's vertices into several subsets, such that each subset is secure. Consider, for example, a strategic game where each player can occupy spots on a map. Players aim at selecting spots in such a way that they are capable of defending themselves against any attack from their neighbors. In case every player can repel any attacks from other players, we have a secure set equilibrium. Formally, we define the problem as follows.

Definition 6. *Given a graph $G = (V, E)$ and an integer s with $s \geq 2$, a* secure set equilibrium *is a partition of G into s non-empty sets $\mathcal{E} = \{S_1, \ldots, S_s\}$ such that every $S \in \mathcal{E}$ is secure in G.*

First, we show that the corresponding verification problem is co-NP-complete. This result (together with the definition of secure sets) suggests that we again require the full power of ASP (disjunction and aggregates) to solve Secure Set Equilibrium, i.e., to enumerate the secure set equilibria of a graph.

The verification problem for Secure Set Equilibrium asks whether for a given graph $G = (V, E)$ and a partition $\mathcal{E}$ over V into s sets, is $\mathcal{E}$ a secure set equilibrium in G? We will now show that the following proposition holds.

Proposition 5. Secure Set Equilibrium Verification *is co-NP-complete.*

Proof. In order to show co-NP membership, we consider the complement problem of Secure Set Equilibrium Verification. Given a graph $G = (V, E)$ and a partition $\mathcal{E}$ over V into s sets, is there a set $S \in \mathcal{E}$ that is insecure? It is easy to see that this problem is in NP. It suffices to guess a set $S \in \mathcal{E}$ and a set $X \subseteq S$ and to check whether $|N[X] \cap S| < |N[X] \setminus S|$ holds in G. This check is feasible in polynomial time.

For proving hardness, we assume that $s = 2$, but the result can be generalized to any (fixed) $s \geq 2$. We provide a reduction from Secure Set Verification, which was shown to be co-NP-complete [21][1]. This problem is defined as follows: Given a graph $G = (V, E)$ and a non-empty set S with $S \subseteq V$, is S secure in G?

Let $G = (V, E)$ be a graph and S be a non-empty set $S \subseteq V$. We construct a graph $G'_S = (V'_S, E'_S)$ as follows. For each vertex $x \in V \backslash S$ we introduce a set V'_x of fresh vertices with $|V'_x| = |N[x] \cap S|$. Furthermore, let $E'_x = \{\{x, y\} \mid y \in V'_x\}$ be the set of edges that connect x with the vertices in V'_x. Finally, let a be a fresh vertex. Then, $V'_S = V \cup \bigcup_{x \in V \backslash S} V'_x \cup \{a\}$ and $E'_S = E \cup \bigcup_{x \in V \backslash S} E'_x$. This construction of G'_S is feasible in polynomial time with at most $|S| \cdot (|V| - |S|) + 1$ additional vertices. We now show that given a graph $G = (V, E)$ and a non-empty set $S \subseteq V$, S is secure in G iff $\mathcal{E} = \{S, T\}$ with $T = V'_S \backslash S$ is a secure set equilibrium in $G'_S = (V'_S, E'_S)$.

$\Leftarrow$ Suppose that $\mathcal{E} = \{S, T\}$ is a secure set equilibrium in G'_S. Let $N[X]$ ($N'_S[X]$) be the closed neighborhood of a set X in G (G'_S). By assumption, for all $X \subseteq S$ we have $|N'_S[X] \cap S| \geq |N'_S[X] \backslash S|$ in G'_S. Furthermore, by construction $N'_S[X] = N[X]$ holds. Hence, S is also secure in G.

$\Rightarrow$ Now suppose that S is secure in G. By the same argument as above, S is secure in G'_S. It remains to show that T is secure in G'_S. Since $a \in T$, T is non-empty. Furthermore, to be secure, for each $X \subseteq T$ the inequality

$$|N'_S[X] \cap T| \geq |N'_S[X] \setminus T|$$

must hold. By construction of G'_S, for the left side we have $|N'_S[X] \cap T| \geq |X \cup \bigcup_{x \in X} V'_x|$. Since X and all V'_x are disjoint, $|X \cup \bigcup_{x \in X} V'_x| = |X| + \sum_{x \in X} |V'_x| = |X| + \sum_{x \in X} |N[x] \cap S|$. For the right side of the inequality we have $|N'_S[X] \setminus T| = |N'_S[X] \cap S)| = |N[X] \cap S|$. Overall, we have $|X| + \sum_{x \in X} |N[x] \cap S| \geq |N[X] \cap S|$ which obviously holds for all $X \subseteq T$. Hence, T is secure and $\mathcal{E}$ is a secure set equilibrium in G'_S. $\square$

The encoding for enumerating the secure set equilibria is depicted in Π_{equ} (Encoding 4). Here, the sets $\mathcal{E}$ are identified as input facts $\texttt{sec}(X)$ where $1 \leq X \leq s$, contained in a program Π_s. Observe that this encoding closely resembles Π_{alt}. In a nutshell, the arity of predicates is adapted in order to distinguish between different sets in $\mathcal{E}$. Rules 1 to 3 construct a disjoint partition of all vertices into s sets. Here, $\texttt{inS}(S, V)$ denotes that vertex V is contained in set $S \in \mathcal{E}$. Note that rule 2, which ensures non-emptiness of sets in $\mathcal{E}$, could be equivalently

[1] Note that in [21] the problem is called Is Secure.

expressed as a constraint with #count-aggregate. Vertices not contained in S are represented by the binary predicate `outS`/2. Most importantly, observe that predicate `defended`/1 in rules 8 and 11 is parameterized by the set the currently considered vertex is assigned to. Then, the constraint in rule 15 (together with the saturation applied in rules 12 to 14) guarantees that an answer set is only returned iff all sets in $\mathcal{E}$ are secure. This is captured in the following proposition.

Proposition 6. *Let G be a graph and s be an integer with $s \geq 2$. Furthermore, let $\mathcal{S}$ be the collection of secure set equilibria in G containing s secure sets, and let $\mathcal{A} = \mathcal{AS}(\Pi_G \cup \Pi_s \cup \Pi_{equ})$. Then, for every $\mathcal{E} \in \mathcal{S}$, there exists an $A \in \mathcal{A}$ such that $\{\{v \mid \texttt{inS}(X, v) \in A\} \mid 1 \leq X \leq s\} = \mathcal{E}$. Furthermore, for every $A \in \mathcal{A}$ it holds that $\{\{v \mid \texttt{inS}(X, v) \in A\} \mid 1 \leq X \leq s\} \in \mathcal{S}$.*

Encoding 4: Secure Set Equilibrium (Π_{equ})

$$
\begin{aligned}
1\{\texttt{inS}(S,V) : \texttt{sec}(S)\}1 &\leftarrow \texttt{vertex}(V). && (1)\\
&\leftarrow \texttt{sec}(S), \mathit{not}\, \texttt{inS}(S,V) : \texttt{vertex}(V). && (2)\\
\texttt{outS}(S,V) &\leftarrow \texttt{vertex}(V), \texttt{sec}(S), \mathit{not}\, \texttt{inS}(S,V). && (3)\\
\texttt{borderEdge}(S,U,V) &\leftarrow \texttt{inS}(S,U), \texttt{outS}(S,V), \texttt{edge}(U,V). && (4)\\
\texttt{attackSet}(S,V) &\leftarrow \texttt{borderEdge}(S,U,V). && (5)\\
\texttt{actAtt}(S,V) \mid \texttt{inactAtt}(S,V) &\leftarrow \texttt{attackSet}(S,V). && (6)\\
\texttt{inX}(S,U) : \texttt{borderEdge}(S,U,V) &\leftarrow \texttt{actAtt}(S,V), \texttt{outS}(S,V). && (7)\\
\texttt{defended}(S) &\leftarrow \texttt{inactAtt}(S,U), \texttt{inX}(S,V), \texttt{edge}(U,V). && (8)\\
\texttt{defendSet}(S,V) &\leftarrow \texttt{inX}(S,V). && (9)\\
\texttt{defendSet}(S,V) &\leftarrow \texttt{inX}(S,U), \texttt{edge}(U,V), \texttt{inS}(S,V). && (10)\\
\texttt{defended}(S) &\leftarrow \texttt{sec}(S), && (11)\\
&\qquad \#\mathrm{sum}\{1, V, \mathit{pos} : \texttt{defendSet}(S,V);\\
&\qquad\qquad 1, V, \mathit{pos} : \texttt{inactAtt}(S,V);\\
&\qquad\qquad -1, V, \mathit{neg} : \texttt{attackSet(S,V)}\} \geq 0.\\
\texttt{inX}(S,V) &\leftarrow \texttt{defended}(S), \texttt{inS}(S,V). && (12)\\
\texttt{actAtt}(S,V) &\leftarrow \texttt{defended}(S), \texttt{attackSet}(S,V). && (13)\\
\texttt{inactAtt}(S,V) &\leftarrow \texttt{defended}(S), \texttt{attackSet}(S,V). && (14)\\
&\leftarrow \texttt{sec}(S), \mathit{not}\, \texttt{defended}(S). && (15)
\end{aligned}
$$

Symmetry breaking. Observe that in Proposition 6, $|\mathcal{S}| = |\mathcal{A}|$ does *not* hold. Since secure sets in Π_{equ} are identified by some ID X, $1 \leq X \leq s$, results contain symmetric solutions that differ only in permutations of secure set IDs. To overcome this problem, the encoding is refined as follows.

$$\begin{aligned}\texttt{setAllowed}(1,V) &\leftarrow \texttt{inf}(V).\\ \texttt{setAllowed}(S,V) &\leftarrow \texttt{setAllowed}(S,U),\texttt{succ}(U,V).\\ \texttt{setAllowed}(S+1,V) &\leftarrow \texttt{inS}(S,U),\texttt{succ}(U,V),\texttt{sec}(S+1).\end{aligned}$$

As in program Π_{loop}, we assume that a lexicographical order over all vertices is given by `inf`/1 and `succ`/2 by the program $\Pi_{<}$. The smallest vertex must be contained in set 1, denoted by `setAllowed`/2 (rule 1). Following the successor relationship, subsequent vertices can be contained in sets to which some preceding vertex has already been assigned (rule 2), or they may be added to the next unoccupied set (rule 3). Finally, the assignment of vertices to some set in $\mathcal{E}$, given in Π_{equ}, rule 1, is changed to $1\{\texttt{inS}(S,V):\texttt{setAllowed}(S,V)\}1 \leftarrow \texttt{vertex}(V)$.

4.3 Secure-Insecure Set Partitioning Problem

A natural extension to the problems discussed in the previous sections is to distinguish between sets of vertices that are secure, and such that are insecure. This can be captured as follows.

Definition 7. *Given a graph $G = (V, E)$ and integers s and i with $s, i \geq 1$, a* secure-insecure set partitioning *is a partitioning of G into $s + i$ non-empty sets $\mathcal{P} = \{\mathcal{S}, \mathcal{I}\}$ with $\mathcal{S} = \{S_1, \ldots, S_s\}$ and $\mathcal{I} = \{I_1, \ldots, I_i\}$. Furthermore, every $S \in \mathcal{S}$ is secure and every $I \in \mathcal{I}$ is insecure in G.*

It is easy to see that Secure-Insecure Set Partitioning Verification (Given a graph $G = (V, E)$ and a partition $\mathcal{P} = \{S, I\}$ of V, is $\mathcal{P}$ an secure-insecure set partition?) is co-NP-hard.

Proposition 7. Secure-Insecure Set Partitioning Verification *is co-NP-hard.*

Proof (sketch). We again show this by reduction from Secure Set Verification. Let $G = (V, E)$ be a graph and $S \subseteq V$. We construct a graph $G'_S = (V'_S, E'_S)$ and a set $S' \subseteq V'_S$ as follows. Let a, b, c be fresh vertices. Then, $V'_S = V \cup \{a, b, c\}$, $E'_S = E \cup \{\{a, b\}, \{a, c\}\}$ and $S' = S \cup \{b, c\}$. Now assume a partition $\mathcal{P} = \{S', I'\}$ of G'_S with $I' = V'_S \backslash S'$. It is easy to see that S is secure in G iff S' is secure in G'_S. Furthermore, $a \in I'$ and a is not defended. Hence, I' is always insecure in G'_S. □

In ASP, we identify secure sets by input facts $\texttt{sec}(X)$ with $1 \leq X \leq s$, and denote insecure sets by $\texttt{insec}(Y)$ with $s + 1 \leq Y \leq s + i$. Furthermore, with predicate `set`/1 we give all sets in $\mathcal{P}$. Let $\Pi_{s,i}$ be the program that contains the respective input facts.

To solve the Secure-Insecure Set Partitioning problem, it is sufficient to adapt Encoding 4 as follows. In Encoding 5, rules 1–14 consider both secure and insecure sets of the partition. Rule 15 (together with saturation rules 12–14) guarantees that sets in $\mathcal{S}$ are secure. For sets in $\mathcal{I}$, rule 16 removes all answer sets where $X \subseteq I$ with $I \in \mathcal{I}$ is defended.

Proposition 8. *Let G be a graph and s and i be integers with $s, i \geq 1$. Furthermore, let $\mathcal{S}$ be the collection of secure-insecure set partitionings in G, where for each $\mathcal{P} \in \mathcal{S}$ we have $\mathcal{P} = \{\mathcal{S}^{\mathcal{P}}, \mathcal{I}^{\mathcal{P}}\}$ where $\mathcal{S}^{\mathcal{P}}$ with $|\mathcal{S}^{\mathcal{P}}| = s$ denotes the secure sets and $\mathcal{I}^{\mathcal{P}}$ with $|\mathcal{I}^{\mathcal{P}}| = i$ denotes the insecure sets. Moreover, let $\mathcal{A} = \mathcal{AS}(\Pi_G \cup \Pi_{s,i} \cup \Pi_{sec-insec})$.*

Then, for every $\mathcal{P} = \{\mathcal{S}^{\mathcal{P}}, \mathcal{I}^{\mathcal{P}}\} \in \mathcal{S}$, there exists an $A \in \mathcal{A}$ such that $\{\{v \mid \texttt{inS}(X, v) \in A\} \mid 1 \leq X \leq s\} = \mathcal{S}^{\mathcal{P}}$ and $\{\{v \mid \texttt{inS}(Y, v) \in A\} \mid s < Y \leq s + i\} = \mathcal{I}^{\mathcal{P}}$. Furthermore, for every $A \in \mathcal{A}$ there exists a $\mathcal{P} \in \mathcal{S}$ such that $\{\{v \mid \texttt{inS}(X, v) \in A\} \mid 1 \leq X \leq s\} = \mathcal{S}^{\mathcal{P}}$ and $\{\{v \mid \texttt{inS}(Y, v) \in A\} \mid s < Y \leq s + i\} = \mathcal{I}^{\mathcal{P}}$.

Encoding 5: Secure-Insecure Set Partitioning ($\Pi_{sec-insec}$)

$$1\{\texttt{inS}(P,V) : \texttt{set}(P)\}1 \leftarrow \texttt{vertex}(V). \tag{1}$$
$$\leftarrow \texttt{set}(P), \textit{not}\, \texttt{inS}(P,V) : \texttt{vertex}(V). \tag{2}$$
$$\texttt{outS}(P,V) \leftarrow \texttt{vertex}(V), \texttt{set}(P), \textit{not}\, \texttt{inS}(P,V). \tag{3}$$
$$\texttt{borderEdge}(P,U,V) \leftarrow \texttt{inS}(P,U), \texttt{outS}(P,V), \texttt{edge}(U,V). \tag{4}$$
$$\texttt{attackSet}(P,V) \leftarrow \texttt{borderEdge}(P,U,V). \tag{5}$$
$$\texttt{actAtt}(P,V) \mid \texttt{inactAtt}(P,V) \leftarrow \texttt{attackSet}(P,V). \tag{6}$$
$$\texttt{inX}(P,U) : \texttt{borderEdge}(P,U,V) \leftarrow \texttt{actAtt}(P,V), \texttt{outS}(P,V). \tag{7}$$
$$\texttt{defended}(P) \leftarrow \texttt{inactAtt}(P,U), \texttt{inX}(P,V), \texttt{edge}(U,V). \tag{8}$$
$$\texttt{defendSet}(P,V) \leftarrow \texttt{inX}(P,V). \tag{9}$$
$$\texttt{defendSet}(P,V) \leftarrow \texttt{inX}(P,U), \texttt{edge}(U,V), \texttt{inS}(P,V). \tag{10}$$
$$\begin{aligned}\texttt{defended}(P) \leftarrow\ & \texttt{set}(P),\\ & \#\text{sum}\{1, V, \textit{pos} : \texttt{defendSet}(P,V);\\ & \qquad 1, V, \textit{pos} : \texttt{inactAtt}(P,V);\\ & \quad -1, V, \textit{neg} : \texttt{attackSet}(\texttt{P},\texttt{V})\} \geq 0.\end{aligned} \tag{11}$$
$$\texttt{inX}(P,V) \leftarrow \texttt{defended}(P), \texttt{inS}(P,V). \tag{12}$$
$$\texttt{actAtt}(P,V) \leftarrow \texttt{defended}(P), \texttt{attackSet}(P,V). \tag{13}$$
$$\texttt{inactAtt}(P,V) \leftarrow \texttt{defended}(P), \texttt{attackSet}(P,V). \tag{14}$$
$$\leftarrow \texttt{sec}(S), \textit{not}\, \texttt{defended}(S). \tag{15}$$
$$\leftarrow \texttt{insec}(I), \texttt{defended}(I). \tag{16}$$

Note that this ASP program in general returns several answer sets for the same assignment of vertices to sets in $\mathcal{P}$. This is due to the fact that there may be some $X', X'' \subseteq I$ for $I \in \mathcal{I}$ with $X' \neq X''$, where both X' and X'' are not defended. In order to obtain a single answer set for each secure-insecure set partition, the ASP solver Clingo supports projection to atoms `inS`/2 (via command-line parameter `-project`). Symmetry breaking can be applied in a similar way as for Secure Set Equilibrium, but one has to distinguish between secure and insecure sets appropriately.

4.4 Weighted Secure Set Problem

In the Weighted Secure Set problem each vertex is associated with a positive weight. This weight can be regarded as the "influence" or "power" of a vertex in the graph. For instance, the weighted graph could be a model of a social network with both powerful and easily persuadable entities, or a map where spots are of different strategic importance. We adapt Definition 1 to cover weighted graphs.

Definition 8. *Given a graph $G = (V, E)$ and a function $w : V \to \mathbb{N}^+$ that assigns to each vertex a weight, a non-empty set $S \subseteq V$ is* weighted-secure *in G iff for* each *subset $X \subseteq S$ the following inequality holds.*

$$\sum_{u \in N[X] \cap S} w(u) \geq \sum_{v \in N[X] \setminus S} w(v)$$

Again, we first study the complexity of the corresponding verification problem, i.e., given a weighted graph $G = (V, E)$ with weights $w(v)$ for all $v \in V$ and a set $S \subseteq V$, is S weighted-secure in G?

Proposition 9. Weighted Secure Set Verification *is co-NP-complete.*

Proof (sketch). Membership can be shown by an algorithm that checks whether the inequality in Definition 8 holds for $X \subseteq S$ where the check is feasible in polynomial time. Hardness is obtained by reduction from Secure Set Verification where we assign $w(v) = 1$ to all $v \in V$. □

To solve this problem with ASP, we specify the additional input by facts $\mathtt{w}(V, W)$, where each vertex gets assigned a weight. Then, it is sufficient to adapt the #sum-aggregate to also consider weights.

$$\begin{aligned}\mathtt{defended} \leftarrow\ \#\mathrm{sum}\{&W, V, pos : \mathtt{defendSet}(V), \mathtt{w}(V, W);\\ &W, V, pos : \mathtt{inactAtt}(V), \mathtt{w}(V, W);\\ -\ &W, V, neg : \mathtt{attackSet}(V), \mathtt{w}(V, W)\} \geq 0.\end{aligned}$$

Note that the #count-aggregate as advocated in Section 3.4 can no longer be used here. However, the previously presented problem variants can be adapted in a straight-forward way to additionally support weights.

5 Experimental Results

In this section we present the results of our performance analysis. In detail, we compare the efficiency of the Loop Encoding (Encoding 1), the Border Loop Encoding (Encoding 2) and our Alternative Encoding (Encoding 3) for both enumerating all secure sets and for enumerating all minimum secure sets.

Furthermore we investigate the runtime behavior of our encodings for the two novel problem statements Secure Set Equilibrium and Secure-Insecure Set Partitioning.

For all the encodings under investigation we use the search-space pruning approach described above to increase performance. Since our experiments did not show a significant difference in the runtime between using the #count-aggregate instead of the #sum-aggregate we omit a separate discussion of the counting alternative.

5.1 Benchmark Setup

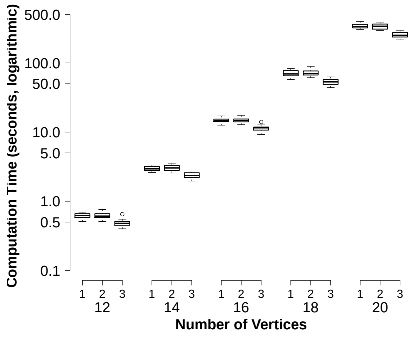
We evaluate our encodings based on a set of graphs (generated using the Erdös-Rényi random graph model) of different sizes n between 12 and 20 vertices. The graphs were generated using a fixed edge probability p (i.e., the probability of adding an edge between a pair of vertices). In our tests, we investigated the influence of the edge probability for a range between 20 and 100 percent. This allows us to analyze the impact of the size of the input graph and the impact of the graph density on the overall performance of our encodings separately. For each edge probability and for each graph size, 25 instances were generated and tested. The benchmark results were obtained using a machine with two Intel Xeon E5345 @ 2.33GHz processors and 48 GB RAM running Debian 7.7 (wheezy). Each test run, using Clingo 4.4.0 [17], was limited to a single core and 128 MB RAM with a time limit of 15 minutes. Due to the fact that preliminary tests showed no significant variance between the results of repeatedly executed experiments, we only used one test run for each instance and encoding to obtain the computation times needed to solve the various problem variants.

5.2 Benchmark Discussion for Secure Set

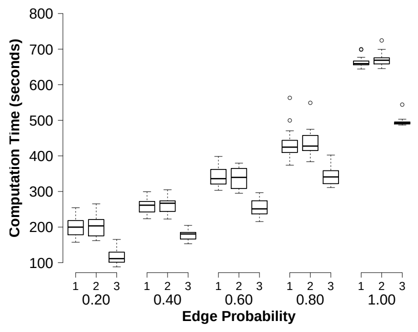
In this part of the paper we will discuss the performance characteristics of the Encodings 1 - 3 and analyze the similarities and differences of the encoding variants both in terms of runtime and grounding effort.

Performance comparison of the different encodings. When interpreting Figure 2, one can see that the original loop encoding and the loop encoding with restriction to the border vertices are almost equally efficient while the alternative encoding outperforms them in every case. We assume that the reason why the restriction to border vertices does not show a significant performance improvement is the fact that rules 4-6 of Encoding 2 introduce a kind of indirection that mitigates the desired performance gain. In contrast to this, the strengths of the alternative encoding (where, like in Encoding 1, only the border vertices are considered within the subset check) are visible over the whole range of instances under investigation and we assume that this also holds for larger graphs.

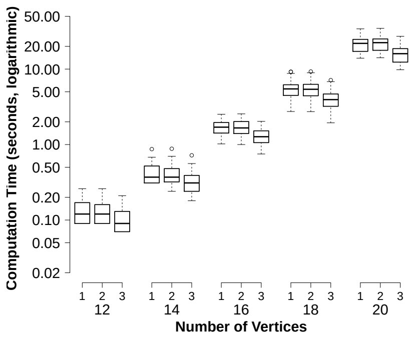
Influence of the graph size on computation time. Figures 2a and 2c show the program's runtime in the presence of different graph sizes. In particular note the exponential explosion of the computation time in relation to the graph size for both problem variants. Already for 22 vertices the time limit of 15 minutes was exceeded by almost every instance. For the optimization problem, the variation



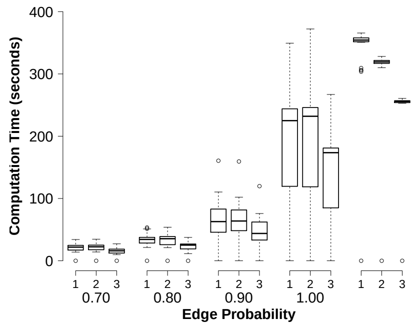
(a) Enumeration problem, $p = 0.50$ (grouped by graph size)



(b) Enumeration problem, $n = 20$ (grouped by edge probability)



(c) Minimization problem, $p = 0.50$ (grouped by graph size)

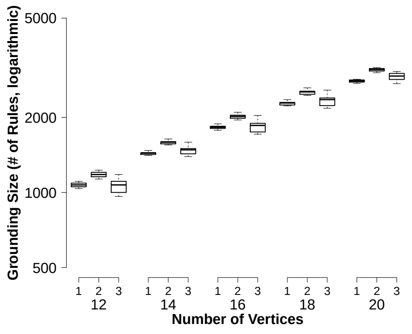


(d) Minimization problem, $n = 20$ (grouped by edge probability)

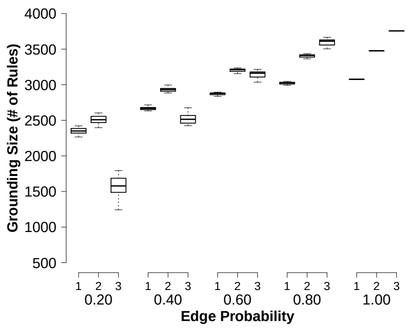
Fig. 2: Performance results for Encodings 1 - 3.

in runtime is much higher as the structure of the graph instance plays an important role (see also the discussion of the influence of the edge probability on the program runtime right below). Figures 2a and 2c only cover the experiments for instances of edge probability 0.5, but these observations also apply to other edge probabilities which are not depicted here.

Influence of the edge probability on computation time. Figures 2b and 2d illustrate the strong dependency of the computation time on the edge probability for a fixed graph size of 20 vertices. We assume that this can be explained by the fact that "small" solutions (i.e., secure sets which contain a small number of vertices) become less likely when the degree of connectedness increases. Interestingly, the optimization encodings are much more sensitive to changes in the edge probability than the enumeration variant. This is because the solver does not have to check any larger candidates for secure sets in the encodings of the optimization problem after a solution of smaller size has been found, while it still has to check the whole search space when solving the enumeration variant.



(a) Enumeration problem, $p = 0.50$ (grouped by graph size)



(b) Enumeration problem, $n = 20$ (grouped by edge probability)

Fig. 3: Grounding results for Encodings 1 - 3.

Finally, note that for $p = 1$ all generated instances represent the same graph (i.e. a clique), which explains the very small difference in runtime.

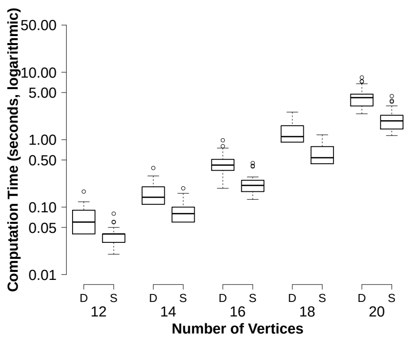
Influence of the graph size on grounding size. Figures 3a and 3b illustrate the grounding effort for our encodings in relation to the graph size and the edge probability. The actual values shown in the figures correspond to the number of lines in DIMACS format which Clingo outputs using the flag `-mode=gringo` for each of the input instances.

It is no surprise that the size of the grounding grows exponentially with the size of the input, but it is interesting to see that the grounding size of the alternative encoding grows much faster than for the other encodings when the edge probability increases although there is no loop involved. This circumstance is most likely caused by the fact that the alternative encoding offers for each vertex three possibilities to choose from within the saturation (namely `actAtt`, `inactAtt` and `inX`), while for the encodings incorporating a loop, there are just two choices (`inX` and `outX`) incorporated in the saturation. Note that the time spent for grounding never exceeded half a second per instance, implying that the grounding step is no bottleneck in the cases we investigated.

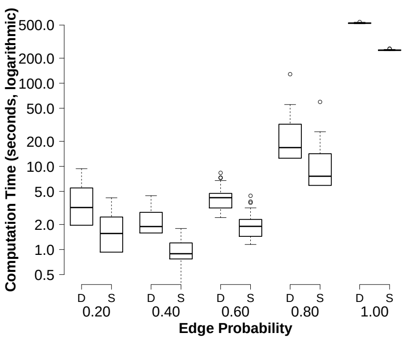
5.3 Benchmark Discussion for Secure Set Equilibrium

In Figures 4a and 4b we present the results of our experimental evaluation for the problem Secure Set Equilibrium (Encoding 4) with two secure sets in the default variant (D) and in the variant with symmetry breaking optimization (S). Both figures underline the fact that our approach for symmetry breaking indeed pays off.

Even more interesting is the effect of the edge probability on the runtime of our encodings: Between 20% and 60% we observe a slight decline in the diagram with its valley at the box-plot covering the results for 40% edge probability which seems strange at the first glance.



(a) Enumeration problem, $p = 0.50$ (grouped by graph size)



(b) Enumeration problem, $n = 20$ (grouped by edge probability)

Fig. 4: Performance results for Secure Set Equilibrium. Variants: D - "Default Encoding", S - "Encoding with Symmetry Breaking Optimization"

An explanation for this behavior is that for sparse and dense graphs, there are often many solutions. For instance, a clique or a graph consisting only of isolated vertices has lots of trivial solutions. Partitioning the graph into two sets of size $n/2$ suffices in both cases.[2] However, other graphs, in which the edges are distributed more randomly, might have much less solutions or even have no solutions at all and thus can be solved faster.

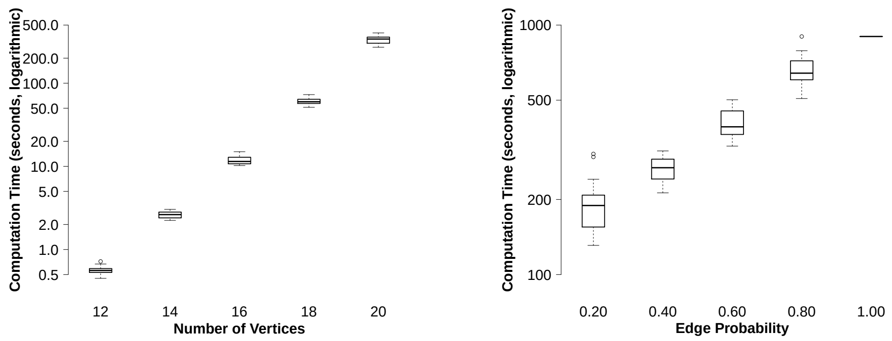
5.4 Benchmark Discussion for Secure-Insecure Set Partitioning

Figures 5a and 5b illustrate the influence of graph size and edge probability on the runtime needed to enumerate all solutions for the Secure-Insecure Set Partitioning problem with one secure and one insecure set.

At the first glance, the runtime behavior seems to be almost identical to those dependencies we observed when analyzing the performance characteristics of our encodings for the Secure Set problem. But on closer inspection one can see that the impact of the edge probability on the runtime is much higher for the Secure-Insecure Set Partitioning. This becomes even more apparent as the instance representing a clique containing 20 vertices could not be solved within the time limit of 15 minutes for this problem while there was no timeout even for cliques when considering our encodings for Secure Set.

We assume that the increased solving effort is caused by the fact that the same secure-insecure set partition could appear in multiple answer sets and that also the use of projection cannot mitigate the increased effort for computing the additional answer sets.

[2] Indeed, for cliques, n has to be even to ensure that there are any solutions for the Secure Set Equilibrium with two secure sets, because in any other case one of the sets would have more attackers than defenders.



(a) Enumeration problem, $p = 0.50$ (grouped by graph size)

(b) Enumeration problem, $n = 20$ (grouped by edge probability)

Fig. 5: Performance results for Secure-Insecure Set Partitioning.

6 Conclusion

In this paper we have studied the problem of finding secure sets in graphs. We have presented some alternative characterizations of secure sets that we put to work in an ASP-based implementation.

From an ASP modeling point of view, finding secure sets in graphs is an interesting problem as it requires disjunction as well as aggregates. It is therefore an attractive candidate for testing the performance of ASP systems that allow solving of problems harder than NP. In this work we have presented different encodings that we believe to be useful for benchmarking ASP systems in the future. This is of particular interest because so far problems harder than NP are underrepresented among common collections of benchmark problems.

Recursive aggregates have posed a challenge in writing encodings for the Secure Set problem. Our work witnesses that even some natural problems seem to require relatively involved tricks in order to properly encode the required arithmetics by means of aggregates in ASP. Moreover, current restrictions in the DLV solver prevented us from gathering experimental data for this system.

Our experiments show a strong dependency of the execution time on the graph size and the edge probability for all of our encodings, and the problem of enumerating all secure sets in a graph is shown to be very challenging for today's ASP solvers even for relatively small instances. Our encoding that refrains from looping over vertices (Encoding 3) outperforms our other encodings in all cases and is therefore assumed to be a good starting point for further investigations.

We also showed the extensibility of this encoding by providing several new problem variants based on the original definition of Secure Set and the experiments we conducted for these variants underline the need for efficient solvers with enough expressive power to capture recursive aggregates as needed for all the problems we discussed in our work.

We hope that in future versions of ASP systems restrictions with respect to (recursive) aggregates are alleviated allowing further research on the secure set problem as well as on similar problems requiring the full power of ASP. In future work, we want to perform further analyses for such problems, which should include other ASP systems (such as DLV) in order to obtain a more comprehensive picture of how ASP can cope with them.

Acknowledgments

The authors would like to thank Mario Alviano, Wolfgang Faber, Roland Kaminski and Francesco Ricca for their helpful comments about recursive aggregates and their implementation in various ASP systems. Furthermore, the authors would also like to thank the reviewers of the ASPOCP workshop 2014 and the JLC special issue reviewers for their helpful and constructive feedback.

This work was supported by the Austrian Science Fund (FWF): P25607 and Y698, and by the Vienna University of Technology special fund "Innovative Projekte" (9006.09/008).

References

1. M. Alviano, F. Calimeri, G. Charwat, M. Dao-Tran, C. Dodaro, G. Ianni, T. Krennwallner, M. Kronegger, J. Oetsch, A. Pfandler, J. Pührer, C. Redl, F. Ricca, P. Schneider, M. Schwengerer, L. K. Spendier, J. P. Wallner, and G. Xiao. The fourth answer set programming competition: Preliminary report. In *Proc. of LPNMR'13*, volume 8148 of *LNCS*, pages 42–53. Springer, 2013.
2. M. Alviano, C. Dodaro, W. Faber, N. Leone, and F. Ricca. WASP: A native ASP solver based on constraint learning. In *Proc. of LPNMR'13*, volume 8148 of *LNCS*, pages 54–66. Springer, 2013.
3. M. Alviano, W. Faber, N. Leone, S. Perri, G. Pfeifer, and G. Terracina. The disjunctive Datalog system DLV. In *Proc. of Datalog'10*, volume 6702 of *LNCS*, pages 282–301. Springer, 2011.
4. Asparagus – A web-based benchmarking environment for answer set programming. `http://asparagus.cs.uni-potsdam.de`. Accessed May 19, 2014.
5. G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.
6. R. C. Brigham, R. D. Dutton, and S. T. Hedetniemi. Security in graphs. *Discrete Applied Mathematics*, 155(13):1708–1714, 2007.
7. F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub. ASP-Core-2: 4th ASP competition official input language format. `https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf`, 2013.
8. T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer. Aggregate functions in DLV. In *Proc. of ASP'03*, volume 78, pages 274–288. CEUR-WS, 2003.
9. U. Egly, S. A. Gaggl, and S. Woltran. Answer-set programming encodings for argumentation frameworks. *Argument & Computation*, 1(2):147–177, 2010.
10. T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3–4):289–323, 1995.

11. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.

12. T. Eiter, G. Ianni, and T. Krennwallner. Answer set programming: A primer. In *Proc. of Summer School on Reasoning Web 2009*, volume 5689 of *LNCS*, pages 40–110. Springer, 2009.

13. W. Faber, G. Pfeifer, and N. Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011.

14. P. Ferraris. Answer sets for propositional theories. In *Proc. of LPNMR'05*, volume 3662 of *LNCS*, pages 119–131. Springer, 2005.

15. P. Ferraris. Logic programs with propositional connectives and aggregates. *ACM Trans. Comput. Log.*, 12(4):25, 2011.

16. P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5(1–2):45–74, 2005.

17. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):105–124, 2011.

18. M. Gebser, T. Schaub, S. Thiele, and P. Veber. Detecting inconsistencies in large biological networks with answer set programming. *Theory and Practice of Logic Programming*, 11(2–3):323–360, 2011.

19. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4):365–386, 1991.

20. A. Harrison, V. Lifschitz, and F. Yang. On the semantics of Gringo. In *Proc. of ASPOCP'13*, pages 129–142, 2013.

21. Y. Y. Ho. *Global Secure Sets of Trees and Grid-like Graphs*. PhD thesis, University of Central Florida, Orlando, USA, 2011.

22. D. Knoke and S. Yang. *Social Network Analysis*, volume 154 of *Quantitative Applications in the Social Sciences*. Sage, 2008.

23. I. Niemelä, P. Simons, and T. Soininen. Stable model semantics of weight constraint rules. In *Proc. of LPNMR'99*, volume 1730 of *LNCS*, pages 317–331. Springer, 1999.

24. M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An A-Prolog decision support system for the space shuttle. In *Proc. of PADL'01*, pages 169–183. Springer, 2001.

25. F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iiritano, and N. Leone. Team-building with answer set programming in the Gioia-Tauro seaport. *Theory and Practice of Logic Programming*, 12(3):361–381, 2012.

26. T. C. Son and E. Pontelli. A constructive semantic characterization of aggregates in answer set programming. *Theory and Practice of Logic Programming*, 7(3):355–375, 2007.

27. S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

A Proof of Proposition 1

Let $G = (V, E)$ be a graph and $\Pi = \Pi_G \cup \Pi_< \cup \Pi_{loop}$. W.l.o.g., let the vertices of V be denoted by integers $1, \ldots, n$ according to the order specified by $\Pi_<$. In the following, when we say that an interpretation satisfies a non-ground rule r, we mean it satisfies all ground instantiations of r over $1, \ldots, n$. Furthermore, for any interpretation I and i-ary predicate p, we write I_p to denote the set of atoms $I \cap \{p(v_1, \ldots, v_i) \mid v_1, \ldots, v_i \in V\}$.

By definition of answer sets, we can restrict our attention to interpretations being a subset of the Herbrand-base of Π, which consists of all ground predicates obtained from the predicate symbols in Π and domain elements (i.e., the vertices V in G).

We prove the assertion of the theorem as follows:

1. We show that for every non-empty secure set S in G, there is an answer set of Π.
2. We show that for every answer set of Π there is a non-empty secure set in G.
3. We show that there are as many secure sets in G as there are answer sets in Π.

(1) Let S be a non-empty secure set in G. We construct the following interpretation:

$$\begin{aligned}
A = {} & \{\texttt{vertex}(v) \mid v \in V\} \cup \{\texttt{edge}(u, v), \texttt{edge}(v, u) \mid \{u, v\} \in E\} \\
& \cup \{\texttt{inf}(1),\ \texttt{sup}(n)\} \cup \{\texttt{succ}(i,\ i + 1) \mid 0 < i < n\} \\
& \cup \{\texttt{inS}(v),\ \texttt{inX}(v),\ \texttt{outX}(v),\ \texttt{defendSet}(v) \mid v \in S\} \\
& \cup \{\texttt{outS}(v) \mid v \in V \setminus S\} \\
& \cup \{\texttt{attackSet}(v),\ \texttt{inactAtt}(v) \mid v \in N(S)\} \\
& \cup \{\texttt{okupto}(u, v) \mid (u, v) \in V^2\} \\
& \cup \{\texttt{ok}(u) \mid u \in V\} \\
& \cup \{\texttt{defended}\}
\end{aligned}$$

We first show that A is a model of Π. The rules originating from Π_G and $\Pi_<$ are all satisfied by construction of A; as S is non-empty, rule 1 of Π_{loop} is satisfied; rule 2 is satisfied because for any $v \in V$ either $\texttt{inS}(v) \in A$ or $\texttt{outS}(v) \in A$. For rule 3, let $u, v \in V$ and consider the ground rule $\texttt{attackSet}(v) \leftarrow \texttt{inS}(u), \texttt{edge}(u, v), \texttt{outS}(v)$. If $u \in S$, $v \notin S$ and $(u, v) \in E$, the body is satisfied by A by definition, but so is the head, since then $v \in N(S)$ and thus $\texttt{attackSet}(v) \in A$. Otherwise, the body is not satisfied by A, and thus the rule is satisfied by A. Rule 4 is satisfied by A since for each $v \in S$, $\texttt{inX(v)} \in A$ and $\texttt{outX(v)} \in A$. Rules 5 and 6 are satisfied by A, since $A_{\texttt{defendSet}} = A_{\texttt{inX}}$. Rules 7–12: A contains any instantiation of $\texttt{okupto/2}$, forming the heads of the rules. Thus all ground instances of these rules are trivially satisfied by A. Similarly for rule 13, A contains any instantiation of $\texttt{ok/1}$. Rule 14 is treated analogously to

Rule 3. Rule 15 is trivially satisfied by A since $\texttt{defended} \in A$. For rules 16 and 17, we oberve that $A_{\texttt{inS}} = A_{\texttt{inX}} = A_{\texttt{outX}}$. Finally, A falsifies the body of rule 18, because $\texttt{defended} \in A$.

In order to show that A is answer set of Π, it remains to prove that for each $B \subset A$ it holds that $B \not\models gr(\Pi)^A$. Suppose to the contrary that there is some $B \subset A$ with $B \models gr(\Pi)^A$. As $B \models gr(\Pi_G \cup \Pi_<)^A$, $A_p = B_p$ holds for $p \in \{\texttt{vertex}, \texttt{edge}, \texttt{inf}, \texttt{sup}, \texttt{succ}\}$. Due to the semantics of choice rules [23], rule 1 ensures that $A_{\texttt{inS}} = B_{\texttt{inS}}$ holds. Moreover, $A_{\texttt{outS}} = B_{\texttt{outS}}$ follows from rule 2. As the extensions of the predicates in the body of rule 3 are the same under A and B, rule 3 enforces $A_{\texttt{attackSet}} = B_{\texttt{attackSet}}$. If $\texttt{defended} \in B$, then rules 16 and 17 cause $A_{\texttt{inX}} = B_{\texttt{inX}}$ and $A_{\texttt{outX}} = B_{\texttt{outX}}$. This allows us to conclude $A_p = B_p$ for $p \in \{\texttt{defendSet}, \texttt{okupto}, \texttt{ok}, \texttt{inactAtt}\}$ due to rules 5–14. But then $B = A$, contradicting our assumption that $B \subset A$. We conclude that $\texttt{defended} \notin B$. As B satisfies rule 15, it hence falsifies the body, so $|B_{\texttt{defendSet}}| + |B_{\texttt{inactAtt}}| < |B_{\texttt{attackSet}}|$.

Let $X = \{v \in V \mid \texttt{inX}(v) \in B\}$, which is a subset of S due to rule 4. Due to rules 5 and 6, $N[X] \cap S \subseteq \{v \in V \mid \texttt{defendSet}(v) \in B\}$. It can be shown by induction that $(N[S] \setminus N[X]) \setminus S \subseteq \{u \in V \mid \texttt{inactAtt}(u) \in B\}$ due to rules 7–14: We first show that $V \setminus N[X] \subseteq \{u \in V \mid \texttt{ok}(u) \in B\}$. Let $u \in X$ be some arbitrary vertex. For $v = 1$, due to rules 7–9, we get $\texttt{okupto}(u, 1)$ if $1 \notin X$, or if $1 \in X$ and $v \notin N(u)$. For $v \in \{2, \ldots, n\}$ we have: Assume that $\texttt{okupto}(u, v-1) \in B$. Then, due to rules 10-12, we have $\texttt{okupto}(u, v) \in B$ if $v \notin X$ or jointly $v \in X$ and $v \notin N(u)$. Due to rule 13, $\texttt{ok}(u) \in B$ if for *all* $v \in V$ we have that $v \notin X$, or $v \in X$ and $v \notin N(u)$ holds. Hence, $V \setminus N[X] \subseteq \{u \in V \mid \texttt{ok}(u) \in B\}$. Finally, by rule 14, we have $\texttt{inactAtt}(u) \in B$ if $\texttt{ok}(u) \in B$ and $u \in N[S] \setminus S$. Restricting vertices to $V \setminus N[X]$, for the left side, we have $(V \setminus N[X]) \cap (N[S] \setminus S) = (N[S] \setminus N[X]) \setminus S$, the right side is $\{u \in V \mid \texttt{ok}(u) \in B\} \cap (N[S] \setminus S) \subseteq \{u \in V \mid \texttt{inactAtt}(u) \in B\}$. Hence, $(N[S] \setminus N[X]) \setminus S \subseteq \{u \in V \mid \texttt{inactAtt}(u) \in B\}$. Therefore $|N[X] \cap S| + |(N[S] \setminus N[X]) \setminus S| \leq |B_{\texttt{defendSet}}| + |B_{\texttt{inactAtt}}|$. But we have seen that $|B_{\texttt{defendSet}}| + |B_{\texttt{inactAtt}}| < |B_{\texttt{attackSet}}|$ and, due to rule 3, $N[S] \setminus S = \{v \in V \mid \texttt{attackSet}(v) \in B\}$ holds. So we conclude that $|N[X] \cap S| + |(N[S] \setminus N[X]) \setminus S| < |N[S] \setminus S|$, which contradicts our assumption that S is secure cf. Lemma 1. Therefore A is an answer set of Π.

(2) Let A be an answer set of Π and let $S = \{v \mid \texttt{inS}(v) \in A\}$. First we observe that A must have the following form in order to be an answer set of Π: As A satisfies Π_G and $\Pi_<$, it must hold that $A_{\texttt{vertex}} = \{\texttt{vertex}(v) \mid v \in V\}$, $A_{\texttt{edge}} = \{\texttt{edge}(u, v),\ \texttt{edge}(v, u) \mid \{u, v\} \in E\}$, $A_{\texttt{inf}} = \{\texttt{inf}(1)\}$, $A_{\texttt{sup}} = \{\texttt{sup}(n)\}$, and $A_{\texttt{succ}} = \{\texttt{succ}(i, i+1) \mid 0 < i < n\}$. In Π_{loop}, rule 1 ensures that $S \neq \emptyset$, and rule 2 causes that $A_{\texttt{outS}} = \{\texttt{outS}(v) \mid v \in V \setminus S\}$. Rule 3 then yields $\texttt{attackSet}(u) \in A$ for each $u \in N(S)$. It must hold that $\texttt{defended} \in A$ due to rule 18. Then rules 16–17 cause that, for any $v \in V$, $\texttt{inX}(v)$ and $\texttt{outX}(v)$ must be in A whenever $v \in S$, and neither of these atoms is in A if $v \notin S$ due to rules 3 and 16–17. This in turn entails that $\texttt{okupto}(u, v) \in A$ for any $u, v \in V$ due to rules 7–12, which enforces $\texttt{ok}(u) \in A$ for all $u \in V$. Rules 14

then yields $\texttt{inactAtt}(u) \in A$ for each $u \in N(S)$. For any $v \in V$, rules 5–6 entail $\texttt{defendSet}(v) \in A$ whenever $v \in S$.

We now show that the non-empty set S is secure in G by contradiction: Suppose it is not; by Lemma 1 there is a set $X \subseteq S$ with $|N[X] \cap S| + |(N[S] \setminus N[X]) \setminus S| < |N[S] \setminus S|$. We construct an interpretation B as follows:

$$
\begin{aligned}
B = {} & \{\texttt{vertex}(v) \mid v \in V\} \cup \{\texttt{edge}(u,v), \texttt{edge}(v,u) \mid \{u,v\} \in E\} \\
& \cup \{\texttt{inf}(1),\ \texttt{sup}(n)\} \cup \{\texttt{succ}(i,\ i+1) \mid 0 < i < n\} \\
& \cup \{\texttt{inS}(v) \mid v \in S\} \cup \{\texttt{outS}(v) \mid v \in V \setminus S\} \\
& \cup \{\texttt{inX}(v) \mid v \in X\} \cup \{\texttt{outX}(v) \mid v \in S \setminus X\} \\
& \cup \{\texttt{attackSet}(v) \mid v \in N(S)\} \\
& \cup \{\texttt{defendSet}(v) \mid v \in N[X] \cap S\} \\
& \cup \{\texttt{okupto}(u,v) \mid (u,v) \in V^2,\ \{1,\dots,v\} \subseteq (V \setminus X) \cup (X \setminus N(u))\} \\
& \cup \{\texttt{ok}(u) \mid u \in V,\ X \cap N(u) = \emptyset\} \\
& \cup \{\texttt{inactAtt}(u) \mid u \in N(S) \setminus N(X)\}
\end{aligned}
$$

By this construction of B and our observations on the form of A, it is easy to see that $B \subset A$. It remains to be shown that B is a model of $gr(\Pi)^A$. As $A_p = B_p$ for $p \in \{\texttt{vertex}, \texttt{edge}, \texttt{inf}, \texttt{sup}, \texttt{succ}\}$, B is a model of $gr(\Pi_G \cup \Pi_<)^A$. Due to $A_{\texttt{inS}} = B_{\texttt{inS}}$ and $A_{\texttt{outS}} = B_{\texttt{outS}}$, rules 1 and 2 in the reduct are satisfied. By rule 3 and construction of S, $\{v \in V \mid \texttt{attackSet}(v) \in A\} = N(S)$, so B satisfies rule 3 in the reduct. Similarly, by rules 5–6 and construction of S and X, $\{v \in V \mid \texttt{defendSet}(v) \in A\} = N[X] \cap S$, so B satisfies rules 5–6 in the reduct. Rule 4 is obviously satisfied by B in the reduct as either $\texttt{inX}(v)$ or $\texttt{outX}(v)$ is in B for any $v \in S$. To show that B satisfies rules 7–9 in the reduct, consider an arbitrary $u \in V$. For any $v \in \{2,\dots,n\}$, $\texttt{inf}(v) \notin B$, so these rules are all satisfied; so consider $v = 1$. If $v \notin X$, then $\texttt{okupto}(u,v) \in B$ and rules 7–8 in the reduct are satisfied; otherwise $\texttt{okupto}(u,v)$ is in B by construction iff $v \notin N(u)$, which entails that B satisfies either the head of rule 9 in the reduct or it falsifies the negative body. In this way, it can be shown using induction (similarly as before) that B satisfies also rules 10–12 in the reduct. Rule 13 is also satisfied by B in the reduct: Whenever $\texttt{okupto}(u,n) \in B$, by construction $\texttt{ok}(u)$ is also in B because $\{1,\dots,n\} \subseteq (V \setminus X) \cup (X \setminus N(u))$ holds iff $V = (V \setminus X) \cup (X \setminus N(u))$, which is equivalent to $X \cap N(u) = \emptyset$. As for rule 14, consider arbitrary $u, v \in V$ and suppose $\texttt{inS}(v)$, $\texttt{outS}(u)$, $\texttt{edge}(u,v)$ and $\texttt{ok}(u)$ are all in B. Then we know $v \in S$, $u \notin S$ and $u \in N(v)$ (which together entail $u \in N(S)$), and finally $X \cap N(u) = \emptyset$ (which entails $u \notin N(X)$). By construction of B, $\texttt{inactAtt}(u) \in B$ follows from $u \in N(S)$ and $u \notin N(X)$, so B satisfies rule 14 in the reduct. We constructed B such that $|B_{\texttt{defendSet}}| + |B_{\texttt{inactAtt}}| = |N[X] \cap S| + |(N[S] \setminus N[X]) \setminus S|$, which is by assumption less than $|N[S] \setminus S| = |B_{\texttt{attackSet}}|$, so B satisfies rule 15 in the reduct. Finally, B satisfies rules 16–17 in the reduct because $\texttt{defended} \notin B$, and rule 18 has no counterpart in the reduct as $\texttt{defended} \in S$. We have thus shown that A is not the least model of $gr(\Pi_G \cup \Pi_< \cup \Pi_{loop})^A$, which contradicts A being an answer set of Π. So S is secure in G.

(3) To show the one-to-one correspondence between non-empty secure sets in G and answer sets of Π, observe that in (1) the constructed answer sets are different for distinct secure sets. On the other hand, suppose that there are two different answer sets of Π. They cannot differ in the extensions of `vertex`/1, `edge`/2, `inf`/1, `sup`/1 and `succ`/2, as then they would not satisfy Π_G or $\Pi_<$. They must both contain `defended`, as otherwise they would not satisfy rule 18. If the two answer sets agree on the extension of `inS`/1, then, as we have seen in (2), they must both agree on the extensions of all other predicates, contradicting the assumption that the answer sets are distinct. But if they differ in `inS`/1, our construction in (2) yields different secure sets. So there are as many non-empty secure sets in G as answer sets of Π.