EPiC Series in Computing

Volume 94, 2023, Pages 270–287

Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning

A Fast and Accurate ASP Counting Based Network Reliability Estimator

Mohimenul Kabir and Kuldeep S Meel

National University of Singapore, Singapore

Abstract

The quantification of system reliability is fundamental to the assessment of a system's safety and resilience, and has been of interest to decision-makers. Since quantifying the system reliability is shown to be computationally intractable, researchers aim to find approximations. Existing approaches to approximate reliability either suffer from poor scalability or lack of correctness guarantees. Answer Set Programming (ASP) is a powerful tool for knowledge representation that can specify complex combinatorial problems. In recent years, the new applications of ASP have propelled the emergence of well-engineered ASP systems. This paper proposes a new ASP counting based framework, RelNet-ASP, to approximate or estimate the reliability of a system or network. The framework reduces the problem of reliability estimation to an approximate model counting problem on ASP programs, offering formal guarantees of the estimated reliability. The experimental evaluation demonstrates that RelNet-ASP outperforms state-of-the-art techniques in terms of both runtime performance and accuracy.

1 Introduction

Modern societies rely on complex inter-connected networks to ensure the supply of essential services such as power, telecommunication, food, water, and transportation. Consequently, the analysis of the reliability of these networks is of critical importance. Broadly, the problem of network reliability is to determine the probability that a network would behave as per intended specifications in the presence of unreliable components [38].

In this paper, we focus on the (s,t)-reliability problem: Given a graph $G = (V, E)$ wherein each edge has a *probability function* W of being active, for two nodes s and t, what is the probability that s and t are connected [1] (denoted as $r(G, s, t, \mathsf{W})$). The problem of network reliability is a fundamental problem in the field of engineering. From the perspective of computer science, the seminal work of Valiant showed the #P-completeness of network reliability problem even under the case when every edge fails with an identical probability of $\frac{1}{2}$ [44]. Given the computational intractability of #P, the exact methods are limited to networks of small size or with certain bounded properties, such as treewidth and diameter [23, 6], making it necessary to

[1] The techniques proposed in this paper naturally extends to the case wherein nodes also behave stochastically. Furthermore, techniques also extend to K-terminal reliability problem wherein instead of a single source and target node, we have a set of source and target nodes.

develop approximate techniques for network reliability computation. Approximate techniques have received much attention in recent years. In particular, there has been a growing interest in *probably approximately correct* (PAC) methods for network reliability estimation [27] wherein the estimate computed by the underlying techniques is within $(1+\varepsilon)$-factor of the ground truth with a confidence of at least $1-\delta$ (defined formally in Section 2), wherein the tolerance parameter (ε) and confidence parameter (δ) are specified by the user.

Monte Carlo methods have been proposed to attain PAC guarantees [18]; the high-level idea is to construct a random variable whose expectation is equal to $r(G,s,t,\mathsf{W})$. One such indicator variable Y that achieves the desired objective is randomly sampling a subgraph and then checking if s and t are connected. The standard probability arguments show that the number of samples depends on the relative variance of Y ($\frac{\sigma_Y^2}{\mu_Y^2}$), where σ_Y^2 and μ_Y are the variance and mean of the random variable Y, respectively. Since we have $\sigma_Y^2 \neq \mu_Y$, the number of samples required is often high when μ_Y is close to 0, also known as *rare event situations*. Several efforts have been made to make Monte Carlo techniques scalable in rare event situations [43, 7, 20]. Another related line of work has focused on the all-terminal variant of the problem [33], where we are interested in determining the probability if a pair of nodes are connected; techniques developed in this context do not translate to (s,t)-reliability (or the case of $K-$terminal reliability when K is not the order of magnitude of $|E|$).

In another line of work, Paredes et al. [12] proposed a counting-based framework, RelNet, which aims to estimate network unreliability, which seeks to compute an estimate $1-r(G,s,t,\mathsf{W})$. Unfortunately, an approximation of network unreliability $1-r(G,s,t,\mathsf{W})$ does not translate to an approximation of network reliability $r(G,s,t,\mathsf{W})$, for (ε,δ)-guarantee. To summarize, the design of scalable techniques for network reliability estimation remains a major challenge.

The primary contribution of this paper is a framework, called RelNet-ASP, that reduces the problem of network reliability to that of Answer Set Programming (ASP)-counting; ASP [37] is a declarative programming paradigm that has its root in logic programming and non-monotonic reasoning. An ASP program consists of *logical rules* defined over propositional atoms and ASP counting seeks to count the number of solutions of an ASP program. Our investigations are motivated by the recent surge of interest in the development of ASP counters [31]. Our empirical evaluation demonstrates that RelNet-ASP significantly outperforms prior state-of-the-art approaches when accounting for accuracy and runtime performance. In particular, RelNet achieved a TAP score[2] of 2262, while the nearest scalable estimator achieved a TAP score of 2853.

The remainder of the paper is organized as follows: Section 2 presents background knowledge and notations, and Section 3 outlines prior works in network reliability estimation. Section 4 presents our approach to reduce network reliability estimation to the ASP counting problem. Section 5 presents the proofs of theorems supporting our framework. Section 6 presents our experiment evaluation. Finally, section 7 concludes the paper.

2 Background

Before going to the technical description, we introduce some background about propositional satisfiability, graph theory, ASP, and weighted to unweighted model counting.

[2]The TAP score [2] is a performance metric assessing both accuracy and efficiency; the lower the better.

Propositional Satisfiability. A propositional *atom* v takes either value 0 (false, resp.) or 1 (true resp.). A *literal* ℓ is either an atom (positive literal) or its negation (negated literal). A *clause* $C = \bigvee_i \ell_i$ is a *disjunction* of literals. A *conjunctive normal form (CNF)* formula $\phi = \bigvee_j C_j$ is a *conjuction* of clauses. We denote the set of propositional atoms in ϕ using notation $\mathsf{atoms}(\phi)$.

An assignment is a mapping $\tau : X \to \{0,1\}$, where $X \subseteq \mathsf{atoms}(\phi)$. For an atom $v \in X$, we define $\tau(\neg v) = 1 - \tau(v)$. An assignment τ satisfies ϕ if τ evaluates ϕ to be true.

Two clauses C_i and $C_{i'}$ are *logically equivalent*, denoted as $C_i \leftrightarrow C_{i'}$, if C_i and $C_{i'}$ have the same truth value for all assignments over $\mathsf{atoms}(C_i \wedge C_{i'})$. The logical relation between clauses C_i and $C_{i'}$ is known as *equivalence* and for notational convenience, we denote an equivalence $C_i \leftrightarrow C_{i'}$ as a tuple of C_i and $C_{i'}$. Given an equivalence $C_i \leftrightarrow C_{i'}$, if C_i ($C_{i'}$ resp.) consists of only one literal, then the atoms of $C_{i'}$ (C_i resp.) *define* the truth value of C_i ($C_{i'}$ resp.).

Graph Theory. Let $G = (V, E)$ be a graph, where $V = \mathsf{Node}(G)$ is the set of the nodes, and $E = \mathsf{Edge}(G)$ is the set of edges. Each edge in $e \in E$ is represented as a tuple $e = (a, b)$, where nodes $a, b \in \mathsf{Node}(G)$ are two endpoints of e. If there is an edge $(a, b) \in E$, then node a (b resp.) is *adjacent* to node b (a resp.). A graph G' is a *subgraph* of G, denoted as $G' = (V', E')$, if $V' \subseteq V$ and $E' \subseteq E$. Given two arbitrary nodes $s, t \in \mathsf{Node}(G')$, if there exists a set of edges in G' that connects nodes s and t or there is a path in G' with nodes s and t, then G' is referred to as (s, t)-connected subgraph where nodes s and t are the source and target nodes, respectively. The nodes s and t are also called *terminal* nodes. We use the notation $\mathsf{Subgraph}(G, s, t)$ to denote all (s, t)-connected subgraphs of G.

In this work, our graphs are probabilistic and the probabilities are assigned to the edges. The probability of edge e is represented by $\mathsf{W}(\mathsf{e})$, which determines the likelihood that edge e is active and the likelihood of edge e failing is represented by $1 - \mathsf{W}(\mathsf{e})$. A graph is *unweighted* if $\forall e \in \mathsf{Edge}(G), \mathsf{W}(\mathsf{e}) = 1/2$; otherwise, the graph is *weighted*. Given a subgraph G', $\mathsf{Pr}(G')$ is defined as $\prod_{e_i \in \mathsf{Edge}(G')} \mathsf{W}(\mathsf{e_i}) \times \prod_{e_i \in \mathsf{Edge}(G) \setminus \mathsf{Edge}(G')} (1 - \mathsf{W}(\mathsf{e_i}))$, i.e., the probability of a subgraph G' is calculated as the product of the probabilities of its edges that are active and the complement of the probabilities of its edges that are inactive. The reliability of graph G w.r.t. source node s, target node t, and probability over edges W, is defined as $r(G, u, v, \mathsf{W}) = \sum_{G' \in \mathsf{Subgraph}(G,s,t)} \mathsf{Pr}(G')$, i.e., the reliability of a graph G, with respect to source node s, target node t and edge probability W, is defined as the sum of the probabilities of all (s, t)-connected subgraphs of G.

We introduce two well known operations on graphs. The *removal* of edge e on a graph $G = (V, E)$, denoted as $G \setminus e$, which is defined as $(V, E \setminus \{e\})$, i.e., deleting the edge e from graph G. The *contraction* of edge $e = (a, b)$ on a graph $G = (V, E)$, denoted as G/e, which is defined as (V', E'), where the node set $V' = V \setminus \{a, b\} \cup \{c\}$, the new node c is not present in V and the edge set $E' = E \cup \{(d, c) | d \notin \{a, b\}$ and node d is either adjacent to a or $b\} \setminus \{e' \in E |$ one of the endpoints of e' is either a or $b\}$, i.e., contraction of edge e merges two endpoints of e into a newly introduced node.

Answer Set Programming. An answer set program P expresses a set of logical relationships between a set of propositional atoms. The set of atoms of program P is denoted as $\mathsf{atoms}(P)$. A *normal (logic) program* P is a set of rules that are expressions of the following form:

$$\text{Rule } r\text{: } \; a \leftarrow b_1, \ldots, b_m, \mathsf{not}\ c_1, \ldots, \mathsf{not}\ c_n \tag{1}$$

In rule r, not denotes *default negation* or *Clark's negation*, indicating *failure to prove* [10]. In rule r (Equation (1)), atom 'a' is called the *head* of r, denoted $\mathsf{Head}(r)$ and atom set

$\{b_1, \ldots, b_m, c_1, \ldots, c_n\}$ is called the *body* of r, denoted $\mathsf{Body}(r)$. More specifically, the atom sets $\{b_1, \ldots, b_m\}$ is *positive body atoms* of r, denoted $\mathsf{Body}(r)^+$ and $\{c_1, \ldots, c_n\}$ is *negative body atoms* of r, denoted $\mathsf{Body}(r)^-$. While the body of a rule is a set of atoms, we use $\mathsf{Body}(r)$ to denote the conjunction $b_1 \wedge \ldots b_m \wedge \neg c_1 \wedge \ldots \neg c_n$ in the following discussion, and we use the notation $\mathsf{Rule}(.)$ to express a rule.

In ASP, an interpretation $M \subseteq \mathsf{atoms}(P)$ lists the true atoms, i.e., an atom a is true (resp. false) if and only if $a \in M$ (resp. $a \notin M$). An interpretation M satisfies $\mathsf{Body}(r)$, denoted as $M \models \mathsf{Body}(r)$, if and only if $\mathsf{Body}(r)^+ \subseteq M$ and $\mathsf{Body}(r)^- \cap M = \emptyset$, where notation not is interpreted classically, i.e., $I \models \mathsf{not}\ c_i$ if $I \not\models c_i$. Thus, a rule (like Equation (1)) interprets that if atoms in $\mathsf{Body}(r)^+$ *hold* by some rules in P but not atoms in $\mathsf{Body}(r)^-$, then $\mathsf{Head}(r)$ also holds. So, an interpretation M satisfies a rule r, denoted as $M \models r$, if $(M \models \mathsf{Body}(r)) \rightarrow (M \models \mathsf{Head}(r))$. An interpretation M is a *model* of P, denoted as $M \models P$, if $\forall_{r \in \mathsf{Rules}(P)} M \models r$. Given an interpretation M, the *Gelfond-Lifschitz (GL) reduct* of a program P is defined as $P^M = \{\mathsf{Head}(r) \leftarrow \mathsf{Body}(r)^+ | r \in \mathsf{Rules}(P), \mathsf{Body}(r)^- \cap M = \emptyset\}$. An interpretation M is an answer set of P if M is the minimal model of P^M. The answer sets of program P is denoted as $\mathsf{AS}(P)$.

The ASP counting problem is to find the number of answer sets $|\mathsf{AS}(P)|$ of a given program P. Given a program P, the PAC-style ASP counting involves estimating a count c such that $\mathsf{Pr}(\frac{|\mathsf{AS}(P)|}{1+\varepsilon} \leq c \leq (1+\varepsilon) \times |\mathsf{AS}(P)|) \geq 1 - \delta$, where $\varepsilon, \delta \in [0, 1]$. Similarly, the PAC-style network reliability estimation involves estimating $\hat{r}$ such that $\mathsf{Pr}(\frac{r}{1+\varepsilon} \leq \hat{r} \leq (1+\varepsilon) \times r) \geq 1 - \delta$, where r is the ground truth.

We use some notations from faceted answer set navigation [3]. The faceted answer set navigation shows that for a given atom $a \in \mathsf{atoms}(P)$ and $f \in \{a, \mathsf{not}\ a\}$ $\mathsf{AS}(P \cup \mathsf{Rule}(\leftarrow f)) = \{\tau \in \mathsf{AS}(P) | \tau \models \mathsf{Rule}(\leftarrow f)\}$, i.e., adding an *integrity constraint* to a program P filters out answer sets of P that do not satisfy the integrity constraint. More specifically, $\mathsf{AS}(P \cup \mathsf{Rule}(\leftarrow a)) = \{\tau \in \mathsf{AS}(P) | a \notin \tau\}$ and $\mathsf{AS}(P \cup \mathsf{Rule}(\leftarrow \mathsf{not}\ a)) = \{\tau \in \mathsf{AS}(P) | a \in \tau\}$.

Clark's completion [10] or program completion procedure provides a preliminary translation of a normal program P into a propositional formula $\mathsf{Comp}(P)$. For each atom $a \in \mathsf{atoms}(P)$, we compute $\mathsf{Comp}(P)$ as follows:

1. If there exist rules $r_1, \ldots, r_k \in \mathsf{Rules}(P)$ such that $\mathsf{Head}(r_1) = \ldots = \mathsf{Head}(r_k) = a$, then include a propositional formula $(a \leftrightarrow (\mathsf{Body}(r_1) \vee \ldots \vee \mathsf{Body}(r_k)))$ to $\mathsf{Comp}(P)$.

2. Otherwise, add $\neg a$ to $\mathsf{Comp}(P)$.

Finally, $\mathsf{Comp}(P)$ is the conjunction of all propositional formulas considered above. Note that $\mathsf{Comp}(P)$ is not syntactically equivalent to P. It is well established that an answer set of P satisfies $\mathsf{Comp}(P)$ but not vice versa [36].

However, $\mathsf{Comp}(P)$ is syntactically equivalent to P for a restricted class of normal programs. To characterize the restricted class of program, we define the *positive dependency graph*, denoted $\mathsf{DG}(P)$, of a program P as follows: the vertices of $\mathsf{DG}(P)$ are $\mathsf{atoms}(P)$ and there exists an edge from b to a if there exists a rule $r \in \mathsf{Rules}(P)$ such that $a = \mathsf{Head}(r)$ and $b \in \mathsf{Body}(r)^+$ [32]. A program P is *tight* if there is no cycle in $\mathsf{DG}(P)$. Otherwise, program P is *non-tight*. The tight program is the restricted class of normal program having a one-to-one correspondence between answer sets of P and the models of $\mathsf{Comp}(P)$ [36].

A set of atoms $L \subseteq \mathsf{atoms}(P)$ forms a *loop* in P if for two arbitrary atoms $x, y \in L$, there is a path from x to y in $\mathsf{DG}(P)$ and all atoms (nodes) on the path are in L. Specifically, a rule r is an *external supporting rule* of a loop L in P if $\mathsf{Head}(r) \in L$ and $\mathsf{Body}(r)^+ \cap L = \emptyset$ and $\mathsf{ExtRule}(L)$ denotes the set of external supporting rules of loop L. Lin and Zhao [36] showed

that the atoms on a loop must be *supported by* atoms external to the loop. Thus, the loop formula $\mathsf{LF}(L, P)$ [35] of a loop L is defined as follows:

$$\mathsf{LF}(L, P) = (\bigwedge_{a \in L} a) \rightarrow \bigvee_{r \in \mathsf{ExtRule}(L)} \mathsf{Body}(r)$$

Finally, the loop formula $\mathsf{LF}(P)$ of program P is defined as the conjunction of loop formulas for all loops L in P.

Weighted to Unweighted Model Counting. The chain formula [9] is a restricted class of propositional formulas and has been found to be useful for reducing *weighted model counting* to *unweighted model counting*. Let we are interested in computing chain formula corresponding to the weight of $\frac{m}{2^k}$ (obtained after possible reduction), where $m > 0$ is a natural number, and $k < 2^m$ is a positive odd number. Let $c_1, \ldots, c_m$ be the binary representation of k, where c_m be the least significant bit. Then we can formulate the chain formula $\phi_{k,m}$ over m propositional atoms $b_1, \ldots, b_m$ using the following notation:

$$\phi_{k,m}(b_1, \ldots, b_m) = (b_1 C_1 (b_2 C_2 \ldots (b_{m-1} C_{m-1} b_m) \ldots))$$

where $C_i = \vee$, if $c_i = 1$, otherwise $C_i = \wedge$. Chakraborty et al. [9] showed that the size of $\phi_{k,m}$ is linear with m and chain formula $\phi_{k,m}$ has k satisfying assignments.

Although the chain formula is not expressed in CNF, it can be transformed into a CNF formula by introducing a new set of Boolean atoms and equivalences. This transformation is known as the *Tseitin transformation* and can be expressed in the CNF formula without exponential increase [42].

The transformation to a chain formula $\phi_{k,m}(b_1, \ldots, b_m)$ works as follows: the encoding first introduces an equivalence and a new atom for the innermost simple Boolean expression of $\phi_{k,m}$; the simple Boolean expression is $(b_{m-1} C_{m-1} b_m)$, and the equivalence is $t_{m-1} \leftrightarrow (b_{m-1} C_{m-1} b_m)$, where t_{m-1} is a new propositional atom. Then the transformation introduces another equivalence and a new atom for the second innermost simple Boolean expression of $\phi_{k,m}$ (if any), namely, the equivalence is $t_{m-2} \leftrightarrow (b_{m-2} C_{m-2} (b_{m-1} C_{m-1} b_m))$. However, a truth value of the Boolean expression $(b_{m-1} C_{m-1} b_m)$ defines the truth value of t_{m-1}. Thus, the new equivalence can be written as $t_{m-2} \leftrightarrow (b_{m-2} C_{m-2} t_{m-1})$. The transformation continues in this way until the transformation encounters the simple Boolean expression $b_1 C_1 t_2$. Thus, the transformation generates a total of $m-2$ propositional atoms $(t_2, \ldots, t_{m-1})$ and derives a total of $m-1$ equivalences. For simplification, we introduce one additional equivalence, $t_1 \leftrightarrow (b_1 C_1 t_2)$, to the set of equivalences. Given a chain formula $\phi_{k,m}$, let denote the transformation using the notation $\mathsf{T}(\phi_{k,m})$.

The transformation introduces a new set of propositional atoms, which are logically defined by the original set of atoms $\{b_1, \ldots, b_m\}$. As a result, an assignment over the atom set of $\{b_1, \ldots, b_m\}$ uniquely defines the truth value of $\{t_1, \ldots, t_{m-1}\}$. For arbitrary assignment over atom set of $\{b_1, \ldots, b_m\}$, the truth values of t_{i-1} and $b_{i-1} C_{i-1} O_i$ are same, for $i \in [1, m-1]$, where O_i is the other operand of C_i except b_{i-1}. It follows that if an assignment τ over $\{b_1, \ldots, b_m\}$ satisfies $\phi_{k,m}$, then τ evaluates t_1 to be true. Thus, $\phi_{k,m}$ and $\mathsf{T}(\phi_{k,m}) \wedge \{t_1 \leftrightarrow 1\}$ have the same number of satisfying assignments. As a result, $\mathsf{T}(\phi_{k,m}) \wedge \{t_1 \leftrightarrow 1\}$ preserves the number of satisfying assignments of the original chain formula $\phi_{k,m}$.

Example 1. *Construct the chain formula for* $k = 5$ *and* $m = 3$.
The binary representation of 5 *using* 3 *bits is* 101. *Therefore, we have* $\phi_{k,m}(b_1, b_2, b_3) = (b_1 \vee (b_2 \wedge b_3))$, $\mathsf{T}(\phi_{k,m}) = \{t_1 \leftrightarrow (b_1 \vee t_2), t_2 \leftrightarrow (b_2 \wedge b_3)\}$. *Finally,* $\mathsf{T}(\phi_{k,m}) \wedge \{t_1 \leftrightarrow 1\}$ *has* 5 *satisfying assignments.*

3 Related Work

Valiant [44] initiated the complexity study of the network reliability problem and showed that the problem is #P-complete. The exact techniques are often based on the enumeration of *cut sets/ path sets* to account for disjoint events [1] or the usage of *recursive* or *online decompositions* of disjoint events [40, 43].

Monte Carlo (MC) techniques are often employed to design approximate techniques; a straightforward design of MC techniques struggles to scale due to *rare event situations*. To address the aforementioned limitation, researchers have proposed variations of MC, including multilevel splitting [21], generalized splitting [5], permutation MC-based Lomonosov's Turnip [20], splitting sequential MC [43], subset simulation [48], among others. The *stopping rule algorithm* [11] is an approximation algorithm offering PAC guarantees based on *Sequential analysis*. Dagum et al. [11] proposed a new algorithm based on the stopping rule algorithm by improving its sample size. The *gamma Bernoulli approximation scheme* [25] can calculate network reliability with PAC guarantees and improve the running time of [11] by utilizing the *central limit theorem* and achieving a lower order on the running time. Subsequently, Huber [26] proposed a two-stage algorithm that combines the gamma Bernoulli approximation scheme and the algorithm of [11] to improve the sample size further.

Advanced sampling techniques, e.g., line sampling and variance reduction method [8], employing graphical models, offer guarantee-less approaches to quantify network reliability. Another line of research involves statistical learning theories with numerical simulation, which offers techniques for network reliability estimation [28]. Furthermore, specialized techniques borrowed from data mining, such as hierarchical and unsupervised spectral clustering, combined with sampling, provide valuable insights into reliability and risk assessment [47, 22, 14].

Another related line of work has focused on the problem of network unreliability, i.e., to estimate $1 - r(G, s, t, \mathsf{W})$. Paredes et al. [12] proposed a CNF model counting-based PAC-style framework for network unreliability estimation. It is worth remarking that PAC-approximation of $1 - r(G, s, t, \mathsf{W})$ can not be used to derive PAC-approximation of $r(G, s, t, \mathsf{W})$. Akin to CNF-based techniques, the network reliability can be computed via *plausibility reasoning* [16] on ASP; however, this technique performs well particularly on lower treewidth encodings and most encodings do not offer a lower treewidth [24].

ASP Counters The complexity study shows that ASP counting is #·co-NP-complete [16], while the complexity drops to #P for *normal* programs. Various techniques exist for ASP counting, including *unfounded set detection* in propositional model counters [4], dynamic programming on *tree decomposition* [15], and *cycle breaking* from positive dependency graphs [13]. Another line of work combines hashing-based techniques with ASP solving to yield PAC-style ASP counting [31].

4 Approach

We now discuss our primary contribution: RelNet-ASP, an ASP counting based approach to network reliability estimation. RelNet-ASP takes in a graph $G = (V, E)$, a probability function over edges W, terminal nodes s and t, and computes an estimate of $r(G, s, t, \mathsf{W})$.

Figure 1 provides a high-level overview of RelNet-ASP. The RelNet-ASP consists of three phases, which we discuss below:

- (Phase 1) Given a graph G and terminal nodes(s and t), RelNet-ASP generates an ASP

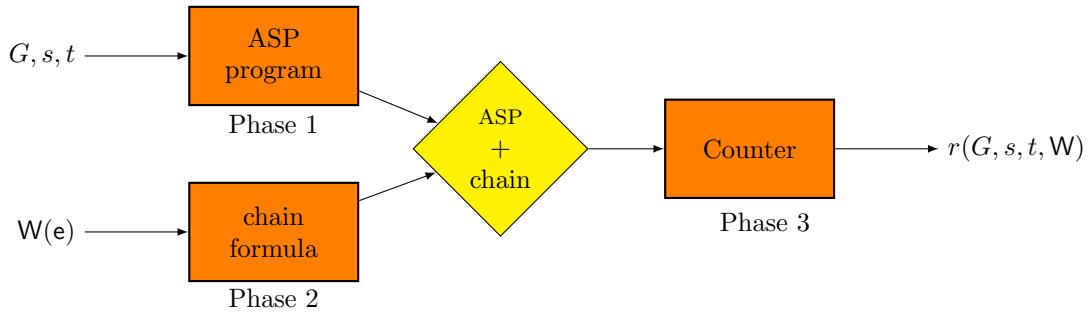


Figure 1: The architecture of RelNet-ASP.

program $\mathcal{P}_{G,s,t}$ such that the answer sets of $\mathcal{P}_{G,s,t}$ correspond one-to-one to the (s,t)-connected subgraphs of G.

- (Phase 2) We encode the arbitrary edge probabilities W into ASP rules by relying on the notion of chain formulas; the notation Ch^e denotes the resulting ASP rules for arbitrary edge $e \in E$.
- (Phase 3) RelNet-ASP relies on a state-of-the-art ASP counter to compute (exact/approximate) estimate of $|\mathsf{AS}(\mathcal{P}_{G,s,t} \wedge \bigwedge_{e \in E} \mathsf{Ch}^e)|$, which is normalized to return the desired estimate of $r(G, s, t, \mathsf{W})$.

In the remainder of the section, we discuss the above phases in detail:

4.1 Generate ASP Program (Phase 1)

We assume that the input graph G is unweighted, i.e., each edge has the identical probability of $\frac{1}{2}$. RelNet-ASP generates a program $\mathcal{P}_{G,s,t}$ such that $|\mathsf{AS}(\mathcal{P}_{G,s,t})| = |\mathsf{Subgraph}(G, s, t)|$. In program $\mathcal{P}_{G,s,t}$, capital letters (e.g., X, Y) denote arbitrary objects and small letters denote specific objects (e.g., s, t). The program $\mathcal{P}_{G,s,t}$ uses a set of atoms to represent the nodes and edges of the graph, the activity of an edge, as well as the reachability of a node from the source node s. These atoms may include:

- node(X) and edge(X,Y) represent that X is a node and (X,Y) is an edge in the input graph, respectively
- source(X) and target(Y) represent X and Y are the source and target nodes, respectively
- in(X,Y) represents that edge(X,Y) is active
- reached(X) represents that node X is reachable from the source node s

Using these atoms described above, the program $\mathcal{P}_{G,s,t}$ uses a set of rules to represent the structure of the graph, the activity of edges, and the reachability of nodes from node s.

The encoding of Listing 1 presents $\mathcal{P}_{G,s,t}$. In the encoding, the atoms in(X,Y) act as *choice variables*[3] of ASP [19]. The node s is the source node and the program starts graph traversal starting from node s. The atom reached(s) is true initially, i.e., the source node s is

[3]In unweighted case, the atom in(X,Y) is assigned to true (i.e., edge (X,Y) is active) with probability of 0.5.

```
1 % transitive definition of reachability
2 reached(X) ← source(X).
3 reached(Y) ← in(X,Y), reached(X).
4 reached(X) ← in(X,Y), reached(Y).
5 % target node must be reachable
6 ← target(X), not reached(X).
```

Listing 1: Program $\mathcal{P}_{G,s,t}$

reachable from the s. Then the program transitively determines the reachable nodes from node s (Listing 1, lines 3 and 4). These rules state that if one of the endpoints of an active edge is reachable from s, then the other endpoint of the active edge is reachable from s. Finally, the program adds a constraint that target node t must be reachable from node s (Listing 1, line 6). The constraint ensures that only the connected subgraphs that include a path from the source node s to the target node t are considered as a part of the answer sets.

K-terminal Reliability The ASP program Listing 1 can be used if there are k terminal nodes, where $k > 2$. In this case, we consider one of the terminal nodes as the source node and the remaining $k-1$ nodes as target nodes. The constraint in line 6 of program 1 would validate that all $k-1$ target nodes will be reachable from the source node. The constraint would be represented by a rule that states that $\forall t, t$ is a target, and the atom $\mathsf{reached}(\mathrm{t})$ must be true.

4.2 Chain formula in ASP (Phase 2)

In this subsection, RelNet-ASP encodes arbitrary edge probabilities relying on the concept of chain formula. Finally, RelNet-ASP generates an ASP program for a given chain formula such that the number of answer sets of the program is proportional to the corresponding weight of the chain formula.

From Chain Formula to ASP Recall that the chain formula is discussed in Section 2 and a chain formula $\phi_{k,m}$ can be viewed as a set of equivalences $\mathsf{T}(\phi_{k,m})$. Given a chain formula, Algorithm 1 of RelNet-ASP derives an ASP program Ch. For each equivalence of $\mathsf{T}(\phi_{k,m})$, Algorithm 1 introduces at most two ASP rules. Recall from the chain formula definition, for each equivalence $(a, b) \in \mathsf{T}(\phi_{k,m})$, the part b is either a conjunction or disjunction of two atoms. For each equivalence of form: $(a, p \vee q) \in \mathsf{T}(\phi_{k,m})$, Algorithm 1 introduces two rules: $\mathsf{Rule}(a \leftarrow p)$ and $\mathsf{Rule}(a \leftarrow q)$ to Ch. For each equivalence of form: $(a, p \wedge q) \in \mathsf{T}(\phi_{k,m})$, Algorithm 1 introduces one rule: $\mathsf{Rule}(a \leftarrow p, q)$ to Ch.

There are two interesting characteristics of the Ch program. First, program Ch is a tight program. More specifically, the positive dependency graph of Ch has directed edges from b_i to t_i and from t_{i+1} to t_i, where $i \in [1, m-2]$. It follows that the positive dependency graph of Ch is acyclic; thus, the rules of Ch form a tight logic program. Second, as per the construction of Algorithm 1, the Clark completion of Ch corresponds to $\mathsf{T}(\phi_{k,m})$. As $\phi_{k,m}$ has k satisfying assignments, $\mathsf{T}(\phi_{k,m}) \wedge \{t_1 \leftrightarrow 1\}$ has exactly k solutions. Conversely, $\mathsf{Rewrite}(\phi_{k,m}, m, e) \wedge \{t_1 \leftrightarrow 0\}$ has exactly $2^m - k$ solutions. As Ch is a tight program, the answer sets of Ch correspond to the satisfying assignments of $\mathsf{T}(\phi_{k,m})$. It follows that $\mathsf{Ch} \wedge \mathsf{Rule}(\leftarrow \mathsf{not}\ t_1)$ has exactly k answer sets, whereas $\mathsf{Ch} \wedge \mathsf{Rule}(\leftarrow t_1)$ has $2^m - k$ answer sets.

Algorithm 1 ChainASP($\phi_{k,m}$)

Input: $\phi_{k,m}$
Output: ASP program Ch

```
1: for each (a, b) ∈ T(φ_{k,m}) do
2:     if b is of form p ∨ q then
3:         Ch.add(Rule(a ← p))
4:         Ch.add(Rule(a ← q))
5:     else if b is of form p ∧ q then
6:         Ch.add(Rule(a ← p, q))
7: return Ch
```

Encoding edge probabilities in RelNet-ASP RelNet-ASP employs the chain formula concept in ASP (as outlined in Algorithm 1) to encode arbitrary edge probabilities. More specifically, RelNet-ASP constructs an ASP program Ch^e, for an edge $e \in \mathsf{Edge}(G)$. If an edge $e = (a, b)$ has a probability $\mathsf{W(e)} = \frac{k}{2^m}$, RelNet-ASP uses Algorithm 1 to generate an ASP program Ch^e, which deals with the edge probability $\frac{k}{2^m}$ by introducing k new answer sets when in(a,b) is true, i.e., edge(a,b) is active and $2^m - k$ new answer sets when in(a,b) is false, i.e., edge(a,b) is not active.

Example 2. *Consider an edge e such that* $\mathsf{W(e)} = \frac{5}{2^3}$. *Recall from Example 1:* $\phi_{k,m}(b_1, b_2, b_3) = (b_1 \vee (b_2 \wedge b_3))$. $\mathsf{T}(\phi_{k,m})$ *is the conjunction of the equivalences:* $\{t_2 \leftrightarrow (b_2 \wedge b_3),\ t_1 \leftrightarrow (b_1 \vee t_2)\}$. *From* $\mathsf{T}(\phi_{k,m})$, *the program* Ch^e *consists of the following rules:* $\{\mathsf{Rule}(t_2 \leftarrow b_2, b_3), \mathsf{Rule}(t_1 \leftarrow b_1), \mathsf{Rule}(t_1 \leftarrow t_2)\}$. *Finally,* $\mathsf{Ch}^e \wedge \mathsf{Rule}(\leftarrow \mathsf{not}\ t_1)$ *has* 5 *answer sets and* $\mathsf{Ch}^e \wedge \mathsf{Rule}(\leftarrow t_1)$ *has* 3 *answer sets.*

4.3 From Reliability Estimation to ASP counting (Phase 3)

Algorithm 2 of RelNet-ASP reduces the network reliability estimation problem to an ASP counting problem. The algorithm takes in a weighted graph G and terminal nodes s and t, and generates a chain formula augmented ASP program $\mathcal{Q}$. Initially, $\mathcal{Q}$ equals to $\mathcal{P}_{G,s,t}$ and num is 0. For each edge $e = (X, Y) \in \mathsf{Edge}(G)$ where $\mathsf{W(e)} = \frac{k}{2^m}$, the algorithm invokes Algorithm 1, which outputs an ASP program Ch^e (Algorithm 2, Line 5). Then the algorithm adds a new rule: $\mathsf{Rule}(\mathsf{in(X,Y)} \leftarrow t_1^e)$ to Ch^e, augments $\mathcal{Q}$ with Ch^e, and increments num by m (Algorithm 2, Line 7 and 8). The program $\mathcal{Q}$ considers the probability of each of the edges; more specifically, $\mathcal{Q}$ is the conjunction of $\mathcal{P}_{G,s,t}$ and $\bigwedge_{e \in \mathsf{Edge}(G)} \mathsf{Ch}_e$. Finally, the algorithm returns the tuple ($\mathcal{Q}$, num). If Algorithm 2 returns ($\mathcal{Q}$, num), then network reliability $r(G, s, t, \mathsf{W})$ is $|\mathsf{AS}(\mathcal{Q})|$ divided by 2^{num}. As the network reliability $r(G, s, t, \mathsf{W})$ is a constant factor of $|\mathsf{AS}(\mathcal{Q})|$, an ASP counter with (ε, δ) guarantees offers (ε, δ) guarantees on network reliability estimation.

5 Theoretical Analysis

In this section, we establish the correctness of our scheme. To this end, Lemma 1 establishes the correctness of Phase 1. Finally, we combine Lemma 1 and Lemma 2 to prove the main theorem of the paper (Theorem 1).

Lemma 1. *Given a unweighted graph G, source and target nodes s and t, respectively,* $|\mathsf{Subgraph}(G, s, t)| = |\mathsf{AS}(\mathcal{P}_{G,s,t})|$.

Algorithm 2 ProcessProb(G, s, t, W)

Input: G, s, t, W
Output: Chain formula augmented ASP program

1: $\mathcal{Q} \leftarrow \mathcal{P}_{G,s,t}$
2: $\mathsf{num} \leftarrow 0$
3: **for each** $e = (X, Y) \in \mathsf{Edge}(G)$ **do**
4: compute k, m such that $\mathsf{W(e)} = \frac{k}{2^m}$
5: $\mathsf{Ch}^e \leftarrow \mathsf{ChainASP}(\phi_{k,m})$
6: $\mathsf{Ch}^e.\mathsf{add}(\mathsf{Rule}(\mathsf{in(X,Y)} \leftarrow t_1^e))$
7: $\mathcal{Q}.\mathsf{add}(\mathsf{Ch}^e)$
8: $\mathsf{num} \leftarrow \mathsf{num} + m$
9: **return** $(\mathcal{Q}, \mathsf{num})$

Proof. As the probability of a graph is associated with its edges, we represent a subgraph using the set of active edges it contains. We use the notation $\tau_{\downarrow \mathsf{in}}$ to refer the set of $\mathsf{in}(X, Y)$ atoms such that $\mathsf{in}(X, Y) \in \tau$ or the set of active edges under answer set τ. We use the shortforms $\mathsf{in}/2$ and $\mathsf{reached}/1$ to refer to the set of active edges and reachable nodes from the source node s, respectively. More specifically, an arbitrary set of $\mathsf{in}/2$ atoms $\{\mathsf{in}(X, Y) | (X, Y) \in \mathsf{Edge}(G)\}$ refers to a subgraph of G where the corresponding edges $\mathsf{edge}(X, Y)$ are active. We consider the following statements to proof the lemma.

S1 If $\tau \in \mathsf{AS}(\mathcal{P}_{G,s,t})$, then τ corresponds to a (s, t)-connected subgraph of G

S2 If $\tau_1, \tau_2 \in \mathsf{AS}(\mathcal{P}_{G,s,t})$ and $\tau_{1\downarrow \mathsf{in}} = \tau_{2\downarrow \mathsf{in}}$, then $\tau_1 = \tau_2$

S3 Each subgraph has a unique set of active edges.

S4 Each (s, t)-connected subgraph of G corresponds to a unique answer set of $\mathcal{P}_{G,s,t}$

Proof of **S1**. For an answer set $\tau \in \mathsf{AS}(\mathcal{P}_{G,s,t})$, we construct a unique (s, t)-connected subgraph G' of the input graph G. Let G' be the subgraph of G such that $\mathsf{Node}(G') = \mathsf{Node}(G)$ and $(X, Y) \in \mathsf{Edge}(G')$ if and only if $\mathsf{in}(X, Y) \in \tau$. More specifically, according to Listing 1, $\mathsf{reached}(s) \in \tau$ and $\mathsf{reached}(t) \in \tau$. For a node $X \in \mathsf{Node}(G)$, if $\mathsf{reached}(X) \in \tau$, then node X is reachable in subgraph G' from source node s because Listing 1 computes the reachable nodes from source node s transitively and the active edges of G' are same as the corresponding active edges of answer set τ. Thus, the subgraph G' is (s, t)-connected.

Proof of **S2**. Proof by contradiction. Assume that there are two answer sets τ_1 and τ_2 such that $\tau_{1\downarrow \mathsf{in}} = \tau_{2\downarrow \mathsf{in}}$ and $\tau_1 \neq \tau_2$. Without loss of generality, assume that $\mathsf{reached}(i_1) \in \tau_1 \setminus \tau_2$. Let's say $x_k = \mathsf{reached}(i_1)$ (initially $k = 1$). There is a rule r_k such that $\mathsf{Head}(r_k) = \mathsf{reached}(i_1)$, $\tau_1 \models \mathsf{Body}(r_k) = \mathsf{in}(i_1, i_2), \mathsf{reached}(i_2)$ and $x_{k+1} = \mathsf{reached}(i_2) \notin \tau_2$ because if $\mathsf{reached}(i_2) \in \tau_2$ then $\mathsf{reached}(i_1) \in \tau_2$. Note that the same conclusion $\mathsf{reached}(i_2) \in \tau_2$ holds if $\mathsf{Body}(r_k) = \mathsf{in}(i_2, i_1), \mathsf{reached}(i_2)$. Similarly, for x_{k+1}, there is another rule r_{k+1}. Thus, the atom set of $\{x_i\}$ is an infinite sequence of $\mathsf{reached}/1$ atoms. If each of x_i is distinct, then it contradicts that $\tau_1 \setminus \tau_2$ has at most $|\mathsf{atoms}(P)|$ atoms. Otherwise, $x_i = x_j$ for some $i < j$, which follows that there is an *unfounded set* between the atom set of $\{x_i, \ldots, x_j\}$, which is a contradiction because no answer set contains an unfounded set. Thus, there is no such x_k atom.

Proof of **S3**. The statement follows from the definition of subgraph.

Proof of **S4**. We can prove this using construction. Recall that each subgraph is a set of $\mathsf{in}(X, Y)$ atoms. For each $(s,\ t)$-connected subgraph G', we construct a unique assignment τ over $\mathsf{atoms}(\mathcal{P}_{G,s,t})$ and show that τ maps to a unique answer set of $\mathcal{P}_{G,s,t}$.

Given an $(s,\ t)$-connected subgraph G', initially we have $\tau = \{\mathsf{in}(X,Y)|(X,Y) \in \mathsf{Edge}(G')\} \cup \{\mathsf{source}(s), \mathsf{target}(t)\}$. We start graph traversal on G' from source node s and $\tau = \tau \cup \{\mathsf{reached}(s)\}$. We determine the reachable nodes from node s transitively, i.e., if one of the endpoints of an edge e of G' is reachable from node s, then the other endpoint of e is also reachable from node s. If a node $X \in \mathsf{Node}(G')$ is reachable from source node s, then we append $\mathsf{reached}(X)$ to τ. Upon finishing the graph traversal on G', we get all reachable nodes from source node s under subgraph G'. Note that the subgraph G' is $(s,\ t)$-connected, so $\mathsf{reached}(t) \in \tau$.

The assignment τ satisfies all rules of $\mathcal{P}_{G,s,t}$ because we traverse G' transitively. The assignment τ satisfies the Clark completion of $\mathcal{P}_{G,s,t}$. Now there are two cases to consider: either τ satisfies $\mathsf{LF}(\mathcal{P}_{G,s,t})$ or τ does not satisfy the loop formula $\mathsf{LF}(\mathcal{P}_{G,s,t}, L)$, where L is one of the loops of $\mathsf{DG}(\mathcal{P}_{G,s,t})$. We show that τ is an answer set of $\mathcal{P}_{G,s,t}$ by proving that τ satisfies $\mathsf{LF}(\mathcal{P}_{G,s,t})$. For the purpose of contradiction, assume that there is a loop L in $\mathsf{DG}(\mathcal{P}_{G,s,t})$ such that $\tau \not\models \mathsf{LF}(\mathcal{P}_{G,s,t}, L)$. It is easy to verify that loop L consists of $\mathsf{reached}/1$ atoms. Let $L = \{\mathsf{reached}(i_1), \ldots \mathsf{reached}(i_k)\}$. Therefore, $\tau \models \bigwedge_{a \in L} a$ and $\tau \not\models \bigvee_{r \in \mathsf{ExtRule}(L)} \mathsf{Body}(r)$. But the condition contradicts to our graph traversal technique because we traverse the subgraph G' transitively. So, there is no such loop L, which follows that τ is an answer set of $\mathcal{P}_{G,s,t}$. □

To summarize, each (s, t) corresponds to a unique answer set of $\mathcal{P}_{G,s,t}$ and each answer set of $\mathcal{P}_{G,s,t}$ corresponds to a unique $(s,\ t)$-connected subgraph. Thus, Lemma 1 is proved.
The following lemma is useful to prove Theorem 1.

Lemma 2. *Given a graph G, terminal nodes s, t, edge probabilities W, if Algorithm 2 returns $(\mathcal{Q}_{G,s,t,\mathsf{W}}, \mathsf{num})$ and $e = (a, b) \in \mathsf{Edge}(G)$ is a weighted edge with $\mathsf{W}(\mathsf{e}) = \frac{k}{2^m}$, then*

$$|\mathsf{AS}(\mathcal{Q}_{G,s,t,\mathsf{W}})| = k \times |\mathsf{AS}(\mathcal{Q}_{G/e,s,t,\mathsf{W}} \setminus \mathsf{Ch}^e)| + (2^m - k) \times |\mathsf{AS}(\mathcal{Q}_{G\setminus e,s,t,\mathsf{W}} \setminus \mathsf{Ch}^e)|$$

Proof. There is *exactly one* rule $r = \mathsf{Rule}(\mathsf{in}(a, b) \leftarrow t_1^e) \in \mathcal{Q}$ such that $\mathsf{Head}(r) = \mathsf{in}(a, b)$. If M is an answer set of $\mathcal{Q}_{G,s,t,\mathsf{W}}$, from Clark completion semantics, we have that $t_1^e \in M$ implies in(a,b) $\in M$. Similarly, if t_1^e does not belong to an answer set M, then in(a,b) does not belong to M.

$\mathsf{atoms}(\mathsf{Ch}_e) \cap \mathsf{atoms}(\mathcal{Q}_{G,s,t,\mathsf{W}} \setminus \mathsf{Ch}_e) = \{\mathsf{in}(a, b)\}$. Following the above discussion on atoms $\mathsf{in}(a, b)$ and t_1^e, we can write the following equation on the number of answer sets of $\mathcal{Q}_{G,s,t,\mathsf{W}}$:

$$\begin{aligned}|\mathsf{AS}(\mathcal{Q}_{G,s,t,\mathsf{W}})| = {} & |\mathsf{AS}(\mathsf{Ch}_e \wedge \mathsf{Rule}(\leftarrow t_1^e))| \times |\mathsf{AS}(\mathcal{Q}_{G,s,t,\mathsf{W}} \setminus \mathsf{Ch}^e \wedge \mathsf{Rule}(\leftarrow \mathsf{in}(a, b)))| \\ & + |\mathsf{AS}(\mathsf{Ch}_e \wedge \mathsf{Rule}(\leftarrow \mathsf{not}\ t_1^e))| \times |\mathsf{AS}(\mathcal{Q}_{G,s,t,\mathsf{W}} \setminus \mathsf{Ch}^e \wedge \mathsf{Rule}(\leftarrow \mathsf{not}\ \mathsf{in}(a, b)))|\end{aligned}$$

Note that the *set minus* operation considers a program as a set of rules and removes the corresponding rules from a program. $\mathcal{Q}_{G,s,t,\mathsf{W}} \wedge \mathsf{Rule}(\leftarrow \mathsf{in}(a, b))$ implies that $\mathsf{in}(a, b)$ is false, i.e., edge (a, b) is removed from the input graph G. On the other hand, $\mathcal{Q}_{G,s,t,\mathsf{W}} \wedge \mathsf{Rule}(\leftarrow \mathsf{not}\ \mathsf{in}(a, b))$ implies that $\mathsf{in}(a, b)$ is true, i.e., edge (a, b) is active in the input graph G, which is known as edge contraction in graph theory. Thus, we have the following equation:

$$|\mathsf{AS}(\mathcal{Q}_{G,s,t,\mathsf{W}})| = (2^m - k) \times |\mathsf{AS}(\mathcal{Q}_{G\setminus e,s,t,\mathsf{W}} \setminus \mathsf{Ch}^e)| + k \times |\mathsf{AS}(\mathcal{Q}_{G/e,s,t,\mathsf{W}} \setminus \mathsf{Ch}^e)|$$

□

We are now ready to state the main theorem of the paper.

Theorem 1. *Given a graph G, terminal nodes s, t, edge probabilities W, if Algorithm 2 returns $(\mathcal{Q}_{G,s,t,\mathsf{W}}, \mathsf{num})$, then $r(G,s,t,\mathsf{W}) = \frac{|\mathsf{AS}(\mathcal{Q}_{G,s,t,\mathsf{W}})|}{2^{\mathsf{num}}}$.*

Proof. We use induction on the number of edges $|\mathsf{Edge}(G)|$.
Base case: The theorem holds for $|\mathsf{Edge}(G)| = 0$ because if there is no edge then there is no (s,t)-connected subgraph. If there is no edge, then $|\mathsf{AS}(\mathcal{Q}_{G,s,t,\mathsf{W}})| = |\mathsf{AS}(\mathcal{P}_{G,s,t})| = 0$. So, $r(G,s,t,\mathsf{W})$ is 0. For $|\mathsf{Edge}(G)| = 1$, assume that $e_1 = (a,b) \in \mathsf{Edge}(G)$ and $\mathsf{W}(\mathsf{e_1}) = \frac{k}{2^m}$. There are two cases to consider: (i) nodes s and t are the two endpoints of e_1 and (ii) otherwise. The case (ii) is trivial because the reliability is zero. In case (i), the two endpoints a and b are reachable when edge e_1 is active, i.e., in(a,b) is true; and according to the Clark completion, we have that $t_1^{e_1}$ is true. It follows that $|\mathsf{AS}(\mathcal{Q}_{G,s,t,\mathsf{W}})| = k$ and the network reliability is calculated as $r(G,s,t,\mathsf{W}) = \frac{|\mathsf{AS}(\mathcal{Q}_{G,s,t,\mathsf{W}})|}{2^m} = \frac{k}{2^m}$.
Induction step: Assume that the theorem is true for $|\mathsf{Edge}(G)| \leq i$ (induction hypothesis). We proof that the theorem holds for $|\mathsf{Edge}(G)| = i{+}1$. Assume that $e_{i+1} = (a,b) \in \mathsf{Edge}(G)$ and $\mathsf{W}(\mathsf{e_{i+1}}) = \frac{k}{2^m}$. We use the following equation to compute the reliability of graph $r(G,s,t,\mathsf{W})$:

$$\begin{aligned} r(G,s,t,\mathsf{W}) &= \mathsf{W}(\text{edge } \mathsf{e_{k+1}} \text{ fails}) \times r(G \setminus e_{k+1}, s, t, \mathsf{W}) \\ &\quad + \mathsf{W}(\text{edge } \mathsf{e_{k+1}} \text{ active}) \times r(G/e_{k+1}, s, t, \mathsf{W}) \\ &= \big(1 - \frac{k}{2^m}\big) \times r(G \setminus e_{k+1}, s, t, \mathsf{W}) + \frac{k}{2^m} \times r(G/e_{k+1}, s, t, \mathsf{W}) \\ &= \frac{1}{2^m} \times \big(k \times r(G/e_{k+1}, s, t, \mathsf{W}) + (2^m - k) \times r(G \setminus e_{k+1}, s, t, \mathsf{W})\big) \end{aligned}$$

Both of the graphs $G \setminus e_{k+1}$ and G/e_{k+1} have at most i edges. So, we can apply induction hypothesis on both of them.

$$\begin{aligned} r(G,s,t,\mathsf{W}) &= \frac{1}{2^m} \times \big(k \times \frac{|\mathsf{AS}(\mathcal{Q}_{G/e_{k+1},s,t,\mathsf{W}})|}{2^{\mathsf{num}-m}} + (2^m - k) \times \frac{|\mathsf{AS}(\mathcal{Q}_{G\setminus e_{k+1},s,t,\mathsf{W}})|}{2^{\mathsf{num}-m}}\big) \\ &= \frac{1}{2^{\mathsf{num}}} \times \big(k \times |\mathsf{AS}(\mathcal{Q}_{G/e_{k+1},s,t,\mathsf{W}})| + (2^m - k) \times |\mathsf{AS}(\mathcal{Q}_{G\setminus e_{k+1},s,t,\mathsf{W}})|\big) \end{aligned}$$

Now we can apply Lemma 2.

$$r(G,s,t,\mathsf{W}) = \frac{1}{2^{\mathsf{num}}} \times (|\mathsf{AS}(\mathcal{Q}_{G,s,t,\mathsf{W}})|)$$

□

6 Experimental Results

We implemented a prototype of RelNet-ASP that is configured to use different ASP counters: lp2sat+#CNF, aspmc+#CNF, and ApproxASP ApproxASP provides (ε, δ)-guarantees. Another approach for ASP counting is based on translation to CNF. To this end, we have incorporated two standard translations from ASP to CNF: (i) lp2sat[4] and (ii) aspmc [13], and the resulting CNF formulas are fed to exact (SharpSAT-TD [34]) and approximate (ApproxMC [41]) CNF counters.

[4]We refer to the standard translations [29, 30] for counting answer sets by CNF counters.

Baseline. We compare the performance of RelNet-ASP with the prior state-of-the-art tools for network reliability estimation. As exact methods for reliability estimation, we considered state space partition (SSP) [38], provan and ball (Cut) [39], a decomposition method based on the *binary decision diagram* (BDD) [23], and ProbLog [17]. ProbLog computes the marginal probability of a query given a *probabilistic logic* program. In experiment, we ran ProbLog with Sentinal Decision Diagram as the underlying knowledge compilation tool. These methods were chosen as they are widely used in the literature and have been proven to be effective in estimating network reliability. In addition to these exact methods, we also compared RelNet-ASP with several approximate methods. In particular, we compared DKLR [11] and GBAS [25], and 2-stage DKLR [26] (Huber22). In line with prior studies, we set $\varepsilon = 0.8$ and $\delta = 0.2$ for all techniques that provide (ε, δ)-guarantees.

Instances. We conducted experiments on a set of real-world graph networks. We extracted these graph networks from power grids [46] and online social networks [45]. The dataset comprised 710 undirected graph networks, with a maximum of 1000 nodes and 1500 edges. In line with the previous research on network reliability estimation [12], we assumed that each edge had a probability of $\frac{1}{8}$. For each graph network, we randomly chose two nodes as terminal nodes to estimate network reliability.

Objective. The objective of the experimental evaluation was to assess the performance of RelNet-ASP in terms of runtime and accuracy. We compare the performance of RelNet-ASP against several state-of-the-art techniques to compute network reliability: these techniques vary in the accuracy of their estimates. It is crucial to use a metric that captures both accuracy and runtime performance. Therefore, we used the Time Accuracy Penalty (TAP) score [2] as the metric for comparison.

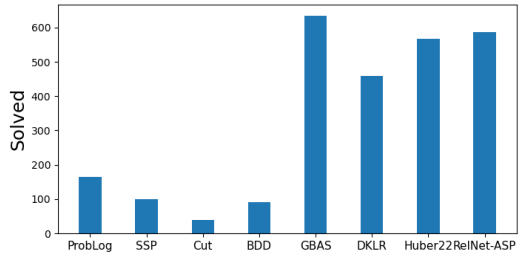
The Time Accuracy Penalty (TAP) score utilizes a *reliable dataset*, which is a set of instances with known ground truth[5]. To compute the ground truth, we applied exact network reliability methods with a memory capacity of 16 GB and a time limit of 10000 seconds to each instance. Finally, we could construct a reliable dataset consisting of 171 instances. Our adapted κ-TAP score follows the definition of TAP score; for a given tool and an instance, the κ-TAP score is defined as follows:

$$\kappa\text{-TAP}(t,i) = \begin{cases} 2 \times \mathcal{T}, & \text{for timeout or memout or error} \\ t + \mathcal{T} \times \frac{\mathcal{R}}{\kappa}, & \text{for } \mathcal{R} < \kappa \\ 2 \times \mathcal{T} - (\mathcal{T} - t) \times \exp(\kappa - \mathcal{R}) & \text{for } \mathcal{R} \geq \kappa \end{cases}$$

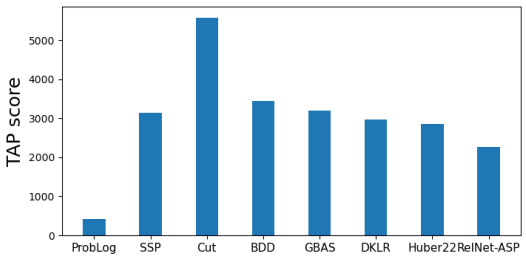
where $\kappa = 1 + \varepsilon$, t is the time to estimate reliability, $\mathcal{T} = 3600$ is the timeout (in seconds), $\mathcal{R} = \mathsf{max}(\frac{\hat{r}}{r}, \frac{r}{\hat{r}})$ is the relative error, and r and $\hat{r}$ are ground truth and estimated reliability, respectively. We use the term TAP to refer κ-TAP score in the following discussion.

The experimental results demonstrate that RelNet-ASP, with ApproxASP as the underlying ASP counter, outperforms existing reliability estimators in terms of both runtime and accuracy metrics. Current estimators typically prioritize either higher accuracy or runtime efficiency, but RelNet-ASP strikes a balance between the two, rendering it a promising technique for network reliability estimation.

[5]The TAP score of [2] relies on *conjectured* value of Z. However, our experimental setup has no *reliable algorithm*. Thus, we rely on instances having known ground truth.



(a) Number of solved instances; the higher the better.



(b) TAP score; the lower the better.

Figure 2: The performance comparison of RelNet-ASP with other network reliability estimators.

Environmental Settings. All experiments were carried out on a high-performance computer cluster, where each node consists of an 2xE5-2690v3 CPU running with 2x12 real cores and 96GB of RAM. The runtime was limited to 3600 seconds and the memory limit was to limited to 4GB.

6.1 Performance Analysis of RelNet-ASP

Table 1 compares the performance of RelNet-ASP with baseline techniques. We compare the performance of the techniques in terms of three metrics: the number of solved instances, observed tolerance, and TAP score. The observed tolerance is defined as $\max(\frac{r}{\hat{r}}-1, \frac{\hat{r}}{r}-1)$, where r and $\hat{r}$ are the ground truth and estimated reliability, respectively. For better visualization, the performance comparison is shown in bar plots in Figure 2.

From Table 1, it is clear that while GBAS is able to count for more instances but the runtime performance comes at the cost of accuracy – this observation is consistent in the previous analysis of Monte Carlo methods. On the other hand, ProbLog achieves the lowest TAP score, but its scalability is limited due to the complexity of exact counting. However, among approximate methods, RelNet-ASP achieves the lowest TAP score. Therefore, RelNet-ASP achieves a good balance between the two when considering both runtime performance and accuracy.

Table 2 illustrates the performance of RelNet-ASP w.r.t. different underlying ASP counters. The table demonstrates that ApproxASP outperforms other underlying approximate answer set counters in terms of performance. Furthermore, RelNet-ASP, in conjunction with exact answer set counters (e.g., lp2sat+SharpSAT-TD and aspmc+SharpSAT-TD), is able to output the ground truth of network reliability, indicating the validity of RelNet-ASP as an estimator

	Exact methods				Approximate methods			
	ProbLog	SSP	Cut	BDD	GBAS	DKLR	Huber22	RelNet-ASP
Solved (710)	164	100	38	92	635	460	567	587
Tolerance (avg)	0	0	0	0	0.423	0.078	0.132	0.036
TAP (avg)	**428**	3133	5583	3442	3198	2972	2853	2262

Table 1: The performance of RelNet-ASP compared to different reliability estimators.

for network reliability.

	RelNet-ASP				
	aspmc		lp2sat		
	SharpSAT-TD	ApproxMC	SharpSAT-TD	ApproxMC	ApproxASP
Solved (710)	91	214	78	474	587
Tolerance (avg)	0	0.065	0	0.076	0.036
TAP (avg)	3630	2568	3982	2452	**2262**

Table 2: The performance of RelNet-ASP w.r.t. different underlying ASP counters.

7 Conclusion

The study has introduced a novel approach for estimating network reliability by combining ASP counting and theories derived from weighted model counting. The proposed tool, RelNet-ASP, leverages the expressive modeling capabilities of ASP and takes advantage of the latest advancements in ASP counting techniques. RelNet-ASP offers a fresh approach to weighted answer set counting. The experimental evaluation demonstrates the scalability of RelNet-ASP in terms of accuracy and efficiency, outperforming existing tools for network reliability estimation. These findings underscore the potential of ASP as a powerful formalism for reliability analysis.

Acknowledgement This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019-0004], Ministry of Education Singapore Tier 2 grant MOE-T2EP20121-0011, and Ministry of Education Singapore Tier 1 Grant [R-252-000-B59-114]. The computational work for this article was performed on resources of the National Supercomputing Centre, Singapore (https://www.nscc.sg).

References

[1] Jacob A Abraham. An improved algorithm for network reliability. In *IEEE Transactions on Reliability*, volume 28, pages 58–61. IEEE, 1979.

[2] Durgesh Agrawal, Yash Pote, and Kuldeep S Meel. Partition function estimation: A quantitative study. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 8 2021.

[3] Christian Alrabbaa, Sebastian Rudolph, and Lukas Schweizer. Faceted answer set navigation. In *Rules and Reasoning: Second International Joint Conference, RuleML+ RR 2018, Luxembourg, Luxembourg, September 18–21, 2018, Proceedings*, pages 211–225. Springer, 2018.

[4] Rehan Aziz, Geoffrey Chu, Christian Muise, and Peter J Stuckey. Stable model counting and its application in probabilistic logic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[5] Zdravko I Botev, Pierre L'Ecuyer, Gerardo Rubino, Richard Simard, and Bruno Tuffin. Static network reliability estimation via generalized splitting. In *INFORMS Journal on Computing*, volume 25, pages 56–71. INFORMS, 2013.

[6] Eduardo Canale, Pablo Romero, and Gerardo Rubino. Factorization theory in diameter constrained reliability. In *2016 8th international workshop on Resilient Networks Design and Modeling (RNDM)*, pages 66–71. IEEE, 2016.

[7] Hector Cancela and Mohamed El Khadiri. A recursive variance reduction algorithm for estimating communication network reliability. In *IEEE Transactions on Reliability*, volume 44, pages 595–602. IEEE, 1995.

[8] Héctor Cancela, Mohamed El Khadiri, Gerardo Rubino, and Bruno Tuffin. Balanced and approximate zero-variance recursive estimators for the network reliability problem. In *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, volume 25, pages 1–19. ACM New York, NY, USA, 2014.

[9] Supratik Chakraborty, Dror Fried, Kuldeep S Meel, and Moshe Y Vardi. From weighted to unweighted model counting. In *IJCAI*, pages 689–695, 2015.

[10] Keith L Clark. Negation as failure. *Logic and data bases*, pages 293–322, 1978.

[11] Paul Dagum, Richard Karp, Michael Luby, and Sheldon Ross. An optimal algorithm for Monte Carlo estimation. *SIAM Journal on computing*, 29(5):1484–1496, 2000.

[12] Leonardo Duenas-Osorio, Kuldeep S Meel, Roger Paredes, and Moshe Y Vardi. Counting-based reliability estimation for power-transmission grids. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[13] Thomas Eiter, Markus Hecher, and Rafael Kiesel. Treewidth-aware cycle breaking for algebraic answer set counting. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 18, pages 269–279, 2021.

[14] Yi-Ping Fang and Enrico Zio. Unsupervised spectral clustering for hierarchical modelling and criticality analysis of complex networks. *Reliability Engineering & System Safety*, 116:64–74, 2013.

[15] Johannes K Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Answer set solving with bounded treewidth revisited. In *Logic Programming and Nonmonotonic Reasoning: 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings 14*, pages 132–145. Springer, 2017.

[16] Johannes K Fichte, Markus Hecher, Mohamed A Nadeem, and TU Dresden. Plausibility reasoning via projected answer set counting–a hybrid approach. In *IJCAI*, volume 22, pages 2620–2626, 2022.

[17] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015.

[18] George S Fishman. A comparison of four Monte Carlo methods for estimating the probability of s-t connectedness. In *IEEE Transactions on reliability*, volume 35, pages 145–155. IEEE, 1986.

[19] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Answer set solving in practice. *Synthesis lectures on artificial intelligence and machine learning*, 6(3):1–238, 2012.

[20] Ilya B Gertsbakh and Yoseph Shpungin. *Models of network reliability: analysis, combinatorics, and Monte Carlo*. CRC press, 2016.

[21] Paul Glasserman, Philip Heidelberger, Perwez Shahabuddin, and Tim Zajic. Multilevel splitting for estimating rare event probabilities. In *Operations Research*, volume 47, pages 585–600. Informs, 1999.

[22] C Gomez, M Sanchez-Silva, and Leonardo Duenas-Osorio. Clustering methods for risk assessment of infrastructure network systems. *Applications of statistics and probability in civil engineering*,

pages 1389–1397, 2011.

[23] Gary Hardy, Corinne Lucet, and Nikolaos Limnios. K-terminal network reliability measures with binary decision diagrams. In *IEEE Transactions on Reliability*, volume 56, pages 506–515. IEEE, 2007.

[24] Markus Hecher. Treewidth-aware reductions of normal ASP to SAT–is normal ASP harder than SAT after all? *Artificial Intelligence*, 304:103651, 2022.

[25] Mark Huber. A Bernoulli mean estimate with known relative error distribution. In *Random Structures & Algorithms*, volume 50, pages 173–182. Wiley Online Library, 2017.

[26] Mark Huber. Tight relative estimation in the mean of Bernoulli random variables. *arXiv preprint arXiv:2210.12861*, 2022.

[27] Mark Huber and Bo Jones. Faster estimates of the mean of bounded random variables. *Mathematics and Computers in Simulation*, 161:93–101, 2019.

[28] Jorge Eduardo Hurtado. *Structural reliability: statistical learning perspectives*, volume 17. Springer Science & Business Media, 2004.

[29] Tomi Janhunen. Representing normal programs with clauses. In *ECAI*, volume 16, page 358, 2004.

[30] Tomi Janhunen and Ilkka Niemelä. Compact translations of non-disjunctive answer set programs to propositional clauses. *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, pages 111–130, 2011.

[31] Mohimenul Kabir, Flavio O Everardo, Ankit K Shukla, Markus Hecher, Johannes K Fichte, and Kuldeep S Meel. ApproxASP–a scalable approximate answer set counter. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 5755–5764, 2022.

[32] Kanchana Kanchanasut and Peter J Stuckey. Transforming normal logic programs to constraint logic programs. *Theoretical Computer Science*, 105(1):27–56, 1992.

[33] David R Karger. A fast and simple unbiased estimator for network (un)reliability. In *2016 IEEE 57th annual symposium on foundations of computer science (FOCS)*, pages 635–644. IEEE, 2016.

[34] Tuukka Korhonen and Matti Järvisalo. Integrating tree decompositions into decision heuristics of propositional model counters. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[35] Joohyung Lee and Vladimir Lifschitz. Loop formulas for disjunctive logic programs. In *ICLP*, volume 2916, pages 451–465. Springer, 2003.

[36] Fangzhen Lin and Yuting Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. In *Artificial Intelligence*, volume 157, pages 115–137. Elsevier, 2004.

[37] Victor W Marek and Miroslaw Truszczyński. Stable models and an alternative logic programming paradigm. *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398, 1999.

[38] Roger Paredes, Leonardo Duenas-Osorio, and Isaac Hernandez-Fajardo. Decomposition algorithms for system reliability estimation with applications to interdependent lifeline networks. volume 47, pages 2581–2600. Wiley Online Library, 2018.

[39] J Scott Provan and Michael O Ball. Computing network reliability in time polynomial in the number of cuts. *Operations Research*, 32(3):516–526, 1984.

[40] Suresh Rai and Arun Kumar. Recursive technique for computing system reliability. In *IEEE transactions on reliability*, volume 36, pages 38–44. IEEE, 1987.

[41] Mate Soos, Stephan Gocht, and Kuldeep S Meel. Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling. In *International Conference on Computer Aided Verification*, pages 463–484. Springer, 2020.

[42] Grigori S Tseitin. On the complexity of derivation in propositional calculus. *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pages 466–483, 1983.

[43] Radislav Vaisman, Dirk P Kroese, and Ilya B Gertsbakh. Splitting sequential Monte Carlo for efficient unreliability estimation of highly reliable networks. In *Structural Safety*, volume 63, pages

1–10. Elsevier, 2016.

[44] Leslie G Valiant. The complexity of enumeration and reliability problems. In *SIAM Journal on Computing*, volume 8, pages 410–421. SIAM, 1979.

[45] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 37–42, 2009.

[46] Bart Wiegmans. Gridkit: European and north-american extracts. 47317, 2016.

[47] Chao Yin and Ahsan Kareem. Computation of failure probability via hierarchical clustering. *Structural Safety*, 61:67–77, 2016.

[48] Konstantin M Zuev, Stephen Wu, and James L Beck. General network reliability problem and its efficient solution by subset simulation. *Probabilistic Engineering Mechanics*, 40:25–35, 2015.