Speeding up Answer Set Programming by Quantum Computing

Davide Della Giustina
dellagiustina.davide@spes.uniud.it
DMIF, University of Udine
Italy

Stefano Pessotto
pessotto.stefano001@spes.uniud.it
DMIF, University of Udine
Italy

Carla Piazza
carla.piazza@uniud.it
DMIF, University of Udine
Italy

Riccardo Romanello
riccardo.romanello@uniud.it
DMIF, University of Udine
Italy

ABSTRACT

Quantum Computing has become a more and more prominent research field in the last few decades. This growth in interest is mainly related to the so-called *quantum speed up* that some quantum procedure exhibits. The two main examples are Shor and Grover algorithms. The latter will be a key ingredient of this paper. In particular, we propose an attempt to speed up Answer Set Programming (ASP) exploiting Quantum Computing. We rely on two proposals in the literature that use quantum computation for: finding stable models of ASP programs; counting solutions of propositional formulae. For combining such proposals we embed in the quantum framework a third proposal from the literature, namely a purely classical approach for navigating the solution space of ASP models. We end up with a quantum method for counting stable models of ASP programs. After providing the details of our method, we briefly describe a Proof of Concept implementation of these techniques.

CCS CONCEPTS

• **Theory of computation** → *Constraint and logic programming*;
• **Computer systems organization** → **Quantum computing**;
• **Computing methodologies** → **Logic programming and answer set programming**.

KEYWORDS

Quantum Computing, Answer Set Programming, Facets Navigation

ACM Reference Format:
Davide Della Giustina, Stefano Pessotto, Carla Piazza, and Riccardo Romanello. 2024. Speeding up Answer Set Programming by Quantum Computing. In *The 33rd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '24), June 3–7, 2024, Pisa, Italy.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3660318.3660328

This work is partially supported by PRIN project NiRvAna—Noninterference and Reversibility Analysis in Private Blockchains" (20202FCJMH, CUP G23C22000400005) and by the GNCS INdAM project 2024 "Strutture di matrici e di funzioni per la sintesi di circuiti quantistici efficienti" (CUP E53C23001670001).

This work is licensed under a Creative Commons Attribution-ShareAlike International 4.0 License.
QUASAR '24, June 3, 2024, Pisa, Italy
© 2024 Copyright is held by the owner/author(s).
ACM ISBN 979-8-4007-0646-2/24/06.
https://doi.org/10.1145/3660318.3660328

INTRODUCTION

Quantum Computation has gained more and more relevance over the last few years. This great success is mainly due to the speed ups obtained by Shor and Grover in their two famous papers [9, 21]. With the goal of reproducing such speed ups in other algorithms, researchers have started their journey into Quantum Computation. They soon found out that such increment in speed are not easy to reproduce. Nevertheless, Quantum Computation has become a very fervent research area. The expressiveness of quantum computers has been investigated through the lens of Quantum Automata in [12, 16]. Problems related to graphs, combined with Quantum Computation, created a great interest in the researchers. On the one hand, a field of study is how graphs are encoded in the quantum circuit based model [5]. On the other hand, proposals have been made on how to solve problems related to graphs in the quantum annealing framework [10]. Moreover, string matching has been tackled with the tools of quantum computation in [3] and [6].

Despite all these results, classical computation still plays a key role when used alongside quantum algorithm. For example, the methods of formal verification have been adopted in proposals like [1, 2, 11] where the authors considered the problem of verifying the correctness of quantum programs/circuits exploiting purely classical computations. Other examples are [13, 17] where authors adopted SAT and ASP based techniques to solve the *purely* quantum problem of minimizing the number of CNOT gates in a circuit.

In this paper we propose a Quantum Algorithm for solving the problem of counting stable models of an ASP program, thus obtaining a quadratic speed up with respect to the classical counter part. Our proposal is based on the union of different techniques coming from [7, 14, 19].

In [7] the authors proposed a purely classical technique to navigate through the solution space of an ASP program—with the goal of finding stable models. Their algorithm is a classical visit of a graph built by using the notion of *facet*. They proved the existence of some particular quantities capable of speeding up the search, which unfortunately are computationally hard to obtain. The ultimate goal of such algorithm was to find a stable model for an ASP program.

The same problem was tackled in [14] too. In this case the authors defined a Quantum Algorithm to solve it. In particular, they devised a *Grover-based* technique that exploits an oracle O which is supposed to be able to identify stable models. Despite being almost

a *bruteforce* approach, it is actually the first attempt for solving ASP-related problems with quantum tools.

A different goal was pursued in [19], where given a propositional logic formula ϕ the aim is that of counting how many solutions it has. In particular, the problem is faced by introducing the notion of *Weighted Model Counting* (WMC) in which each of the assignments satisfying ϕ has a particular *weight*. WMC is the sum of the weights of all the assignments that satisfy ϕ.

What we propose here is a Quantum Algorithm to compute one of the quantities that can speed up the search process proposed in [7]. We are able to achieve such result with a quadratic speed up with respect to any classical approach by exploiting the algorithm proposed in [19]. Being able to compute such a quantity with a fast algorithm can open up the possibility of obtaining faster versions of the algorithm proposed in [7]:

- Start by applying algorithm [7];
- Every time the algorithm needs to be guided during the visit, call our quantum algorithm to speed up the computation of the guiding function.

The paper is structured as follows. Section 1 is devoted to the introduction of the basic concepts of both Quantum Computation and Answer Set Programming. For sake of readability and due to space limits, this section will not contain every single aspect of the two aforementioned topics. In Section 2 we will describe [7, 14, 19]. After that, we introduce the algorithm we devise to speed up the computation of the guiding function for [7]. Almost all the algorithms presented in this paper have been implemented and tested to show their effectiveness. The implementation is publicly available and it is briefly described in Section 3. Finally, in Section 4, we draw some conclusions, stressing the aim of this paper, while giving some ideas on how to improve this work.

1 PRELIMINARIES

In this section we introduce all the notions about quantum computation that will be useful throughout the paper. Since we will restrict ourselves just to what is necessary for this particular proposal, the reader may refer to [15] for further details.

1.1 Quantum Computing

1.1.1 Qubits and Unitaries. A *qubit* is a mathematical object defined as a normalized[1] 2-dimensional vector in the complex field. Usually, such vectors are represented with *Dirac's notation*:

- $|\psi\rangle$ is called a *ket* and represents a column vector;
- $\langle\psi| \coloneqq |\psi\rangle^\dagger$ is called a *bra* and represents a row vector, where the *dagger* indicates an element-wise application of the conjugate operator, followed by a transposition.

A combined use of a *bra* and a *ket* allows to compactly represent the scalar product of two vectors:

$$\langle\psi_1|\psi_2\rangle \coloneqq \langle\psi_1| \cdot |\psi_2\rangle = |\psi_1\rangle^\dagger \cdot |\psi_2\rangle \tag{1}$$

Being qubits 2-dimensional vectors, a basis of size 2 is required to represent them. One of the most commonly used bases is the *computational basis*:

$$|0\rangle \coloneqq \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle \coloneqq \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2}$$

However, other useful bases do exist, such as the one composed by the following 2 vectors:

$$|+\rangle \coloneqq \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad |-\rangle \coloneqq \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{3}$$

It is interesting to notice that any qubit state $|\psi\rangle$ given as a linear combination of the states of some basis, e.g. the computational basis:

$$|\psi\rangle \coloneqq \alpha\,|0\rangle + \beta\,|1\rangle \text{ with } \alpha, \beta \in \mathbb{C} : |\alpha|^2 + |\beta|^2 = 1 \tag{4}$$

can be rewritten in the following form:

$$|\psi\rangle \coloneqq e^{\imath\gamma} \left(\cos\frac{\theta}{2}\,|0\rangle + e^{\imath\phi} \sin\frac{\theta}{2}\,|1\rangle \right) \tag{5}$$

where the term $e^{\imath\phi}$ is called the *relative phase* of z, whereas $e^{\imath\gamma}$ represents its *global phase*. Due to their physical meaninglessness, global phase terms are often factored out and ignored in calculations.

The evolution of a quantum system—set of qubits joined by tensor product—is done through a unitary transformation.

Definition 1.1 (Unitary matrix). A matrix U implementing a quantum transformation is said to be *unitary* if the following condition holds:

$$U^\dagger U = U U^\dagger = I \tag{6}$$

Throughout this paper we will adopt the gate-based model of Quantum Computation. In such framework, every unitary corresponds to a *gate* that is applied to some specific qubits. Measurement operations are used to extract information from the quantum states.

1.1.2 Grover algorithm. Let χ be an n-ary boolean function:

$$\chi : \{0,1\}^n \mapsto \{0,1\} \tag{7}$$

and let $\overline{X}$ be the set of all χ's inputs that correspond to an output of 1:

$$\overline{X} \coloneqq \left\{ \bar{x} \in \{0,1\}^n : \chi(\bar{x}) = 1 \right\} \tag{8}$$

We call *Search Problem* the problem of finding (at least one) $\bar{x} \in \overline{X}$ for some χ given in input. We usually refer to $\overline{X}$ as the set of *solutions* to the problem. For convenience, define $N \coloneqq |\{0,1\}^n| = 2^n$ as the size of the *search space* and $M \coloneqq |\overline{X}|$ as the number of solutions.

Grover's Algorithm is a probabilistic quantum algorithm that allows to solve a Search Problem with $O\left(\sqrt{N}\right)$ expected queries to a quantum oracle for χ, i.e. it implements a procedure that exhibits a quadratic speed up over the best classical algorithm. First of all, the algorithm initializes all the qubits to a uniform superposition of all the elements of the search space, by applying an Hadamard gate $\mathcal{H}$ to every qubit:

$$\mathcal{H}^{\otimes n} |0\rangle^{\otimes n} = |+\rangle^{\otimes n} \tag{9}$$

The reader may notice that the generalization of the Hadamard gate $\mathcal{H}$ to n qubits—which is the gate we indicated with $\mathcal{H}^{\otimes n}$ in Equation 9—is often referred to as the *Walsh-Hadamard transform*, and represented by the letter $\mathcal{W}_n$.

After that, the aim of the procedure is to increase the amplitude of the element $\bar{x}$ in the superposition, while simultaneously

[1] In this context, a vector v is said to be *normalized* if $|v| = 1$.

decreasing all the other ones. Such task is carried out by repeatedly applying the *Grover Operator* $\mathcal{G}$. After some iterations, the amplitude of $\bar{x}$ is intuitively expected to be high enough so that the result of a measurement of all the qubits would result in $\bar{x}$ with high probability[2].

To be more specific, the operator $\mathcal{G}$ can be seen as the concatenation of two distinct steps:

(1) first, $\mathcal{G}$ queries the quantum oracle S_χ. In particular, the oracle is defined so that the effect on the quantum state it is applied to corresponds to a sign flip on the amplitude of $\bar{x}$, while all the other amplitudes are left unchanged:
$$S_\chi |x\rangle = (-1)^{\chi(x)} |x\rangle \tag{10}$$
Since S_χ can also be interpreted as a reflection about the $|\bar{x}\rangle$ vector, it can be represented with a *Hourseholder matrix*:
$$S_\chi = U_{\bar{x}} \coloneqq I_n - 2 |\bar{x}\rangle \langle \bar{x}| \tag{11}$$
where $I_n \in \mathbb{C}^{2^n \times 2^n}$ is the identity matrix. Clearly, S_χ is unitary and thus it is a proper quantum operator;
(2) after that, the operator $\mathcal{G}$ maps each of the amplitudes a_i to their reflection about the average of all the amplitudes of the state:
$$a_i \rightsquigarrow 2\left(\frac{\sum_{j=0}^{N-1} a_j}{N}\right) - a_i \tag{12}$$
Once again, the operation can be represented as a reflection[3], usually referred to as the *diffusion operator*:
$$-U_+ = -\mathcal{W}_n U_0 \mathcal{W}_n = \mathcal{W}_n \left(2 |0\rangle^{\otimes n} \langle 0|^{\otimes n} - I_n\right) \mathcal{W}_n \tag{13}$$

Combining all the steps, the Grover Operator can be written as follows:
$$\mathcal{G} \coloneqq -\mathcal{W}_n U_0 \mathcal{W}_n S_\chi \tag{14}$$

1.1.3 Quantum Counting. Exactly as in the Quantum Search problem setting, let χ be an n-ary boolean function and let $\overline{X}$ be the set of solutions to the equation $\chi(x) = 1$. The *Quantum Counting* problem is concerned with evaluation the size of the solution set—$M \coloneqq |\overline{X}|$. Let S_χ be a quantum oracle that implements χ using $n+1$ qubits.

Now consider the Grover Operator $\mathcal{G}$ that uses S_χ as an oracle: since it represents a rotation of an angle θ in the space spanned by $|\alpha\rangle$ and $|\beta\rangle$, it must have two eigenvectors $|a\rangle$, $|b\rangle$ that lie in the same subspace. Therefore, the vector $|+\rangle^{\otimes(n+1)}$ can be expressed as a linear combination of $|a\rangle$ and $|b\rangle$:
$$|+\rangle^{\otimes(n+1)} = c_a |a\rangle + c_b |b\rangle \tag{15}$$
where c_a and c_b are complex coefficients. Observe what follows:

- due to Equation 15, $|+\rangle^{\otimes(n+1)}$ can be regarded as a superposition of the states $|a\rangle$ and $|b\rangle$;
- in turn, this implies that if such a state is fed into the Phase Estimation algorithm, the circuit behaves linearly and produces thus in output digits that are either from the eigenvalue associated to $|a\rangle$ or from the one associated to $|b\rangle$, with probabilities related to c_a and c_b respectively;
- since $|a\rangle$ and $|b\rangle$ are the eigenvectors of a rotation, their associated eigenvalues can be respectively written as $e^{\iota\theta}$ and $e^{\iota(2\pi-\theta)}$. This means that estimating any of the two allows to easily retrieve the value of θ.

Eventually, the value of M can be computed as follows:
$$M \coloneqq 2N \sin^2 \frac{\theta}{2} \tag{16}$$

1.2 Answer Set Programming

In this section we introduce all the definitions related to ASP that are required throughout the paper. The reader may refer to [8] for further details.

Logic Programming is a declarative programming paradigm whose foundations stemmed from formal logic. In particular, logic programs are used in the fields of *knowledge representation* and *automated reasoning*. Logic programs are built on top of 5 main ingredients:

- a finite set C of *constants*;
- a finite set V of *variables*;
- a finite set F of *functions*;
- a finite set P of *predicates*;
- a function $\text{ar} : F \cup P \mapsto \mathbb{N}^+$, known as the *arity* of both functions and predicates.

By combining elements from the sets C, V, and F, *terms* can be built:

- each constant $c \in C$ and each variable $v \in V$ is a term;
- if $t_1, \ldots, t_n$ are terms and $f \in F$ is a function such that $\text{ar}(f) = n$, then $f(t_1, \ldots, t_n)$ is also a term.

A term with no variables is said to be *ground*.

An *atomic formula*—or, for short, an *atom*—is an object of the form:
$$p(t_1, \ldots, t_n) \tag{17}$$
where $t_1, \ldots, t_n$ are terms and $p \in P$ is a predicate such that $\text{ar}(p) = n$. An atom is said to be *ground* if it is built using only ground terms. Moreover, we call *literal* an atom or the negation of an atom.

A *rule* ρ is a syntactic object that can be expressed as follows:
$$\alpha \leftarrow \beta_1, \ldots, \beta_n, \neg\gamma_1, \ldots, \neg\gamma_m \tag{18}$$
where α, β_i, and γ_j are all atoms. We usually refer to α as the *head* of the rule, and to $\beta_1, \ldots, \beta_n, \neg\gamma_1, \ldots, \neg\gamma_m$ as the *body*. Some special kind of rules are allowed:

- a *fact* is a rule whose body is empty (i.e. $n = m = 0$);
- a *denial* is a rule without a head;
- a rule whose body does not contain any negated atom (i.e. $m = 0$) is called a *definite clause*[4].

A simultaneous substitution of every variable occurring in a rule ρ with a ground term produces a *ground instance* of ρ. Notice that each rule may have several ground instances. A *logical program* Π is a finite set of rules. In analogy to rules, a program is said to be *definite* if it contains only definite clauses. The *ground instance* of a program Π is the program defined as the union – for every rule $\rho \in \Pi$ – of all the possible ground instances of ρ.

The semantics of a logic program Π is defined with respect to a given set $\mathcal{U}$—called *universe*—of objects. An *interpretation* is an

[2] A probability greater than $1/2$ is usually required.

[3] Observe that since $|+\rangle^{\otimes n} = \mathcal{W}_n |0\rangle^{\otimes n}$, the operator $-U_+$ can also be regarded as a reflection about the initial state of the system, which is defined by Equation 9.

[4] Definite clauses are also known as *Horn clauses* in the literature.

assignment that binds some *values* to constants, functions, and relations. Let I be an interpretation for Π. I is referred to as a *model* if it satisfies the logical meaning of all the rules $\rho \in \Pi$. Thanks to Theorem 1.2, we can however limit ourselves to *Herbrand* universes and interpretations. In particular, fixed a program Π:

- its *Herbrand universe* $\mathcal{U}_\Pi$ is the set of all the ground instances of the terms that occur in Π;
- an *Herbrand interpretation* is an interpretation for Π in $\mathcal{U}_\Pi$. Herbrand interpretations that are also models of Π are called *Herbrand models*.

Theorem 1.2 (Herbrand Fundamental Theorem). *Let T be a conjunction of clauses. Then T has a model if and only if it has a Herbrand model.*

For convenience, we can also define the *Herbrand base* B_Π of a program Π:

$$B_\Pi \coloneqq \{\alpha : \alpha \text{ is a ground atom}\} \tag{19}$$

Observe that any $I \subseteq B_\Pi$ is a Herbrand interpretation for Π.

The whole point of Logical Programming is being able to compute logical consequences of programs that encode knowledge about some application domain. However, it is not always so easy to perform such task.

If we restrict ourselves to definite programs, the following lemma provides the foundations for being able to compute logical consequences:

Lemma 1.3. *Let Π be a definite program, and let I_1 and I_2 be two Herbrand models of Π. Then $I_1 \cap I_2$ is also a model of Π.*

Proof. Let $\alpha \leftarrow \beta_1, \ldots, \beta_n$ be a ground instance of a definite clause $\rho \in \Pi$, and assume that $\{\beta_1, \ldots, \beta_n\} \subseteq I_1 \cap I_2$. Then, $\{\beta_1, \ldots, \beta_n\} \in I_1$ and $\{\beta_1, \ldots, \beta_n\} \in I_2$. Since both I_1 and I_2 are models of Π, it holds that $\alpha \in I_1$ and $\alpha \in I_2$. Thus, $\alpha \in I_1 \cap I_2$. □

Corollary 1.4. *If Π is a definite program, then it has a minimum Herbrand model M_Π that is the intersection of all its models.*

Unfortunately, Lemma 1.3 does not hold in the case of *general* programs, i.e. programs that have rules that are not definite clauses. To address this problem, the *completion* of a program can be computed by replacing implications with double implications. Moreover, the set B_Π can be partitioned into three subsets I_Π^+, I_Π^-, R, where:

- I_Π^+ contains all the atoms that belong to every model of the completion;
- I_Π^- that is made of all the atoms that do not belong to any model of the completion;
- R defined as $B_\Pi \setminus \left(I_\Pi^+ \cup I_\Pi^-\right)$.

We call *well-founded model* the pair $\langle I_\Pi^+, I_\Pi^- \rangle$. Notice that it can be computed in polynomial time with respect to the size of the ground instance of the input program Π. Then, either one of the following two cases applies:

- if $I_\Pi^+ \cup I_\Pi^- = B_\Pi$, then the well-founded model identifies the unique, minimum model of Π;
- otherwise, the well-founded model represents some sort of "partial" model of Π.

In order to deal with the possibility of having multiple plausible interpretations, the notion of *stable model semantics* was introduced.

Definition 1.5 (Gelfond-Lifschitz reduct). The *Gelfond-Lifschitz reduct* Π^S of the (ground) program Π with respect to a *candidate model* $S \subseteq B_\Pi$ can be computed by applying the following transformations to Π:

(1) first, all the rules $\rho \in \Pi$ whose body contains an occurrence of a literal of type $\neg\alpha$ such that $\alpha \in S$ are removed;
(2) after that, any other negated literal is removed from the remaining rules' bodies.

Observe that for any Π and for any $S \subseteq B_\Pi$, Π^S is a definite program, and thus it has a minimum model M_{Π^S}. We say that S is a *stable model* – or, equivalently, an *answer set* – of Π if and only if the following condition holds:

$$S = M_{\Pi^S} \tag{20}$$

More in general, a candidate model $S \subseteq B_\Pi$ is a stable model of the program Π if it is a stable model of its ground instance. We denote by $\mathcal{AS}(\Pi)$ the set that contains all the stable models of Π.

Example 1.6. Consider the following (ground) program:

$$\Pi \coloneqq \{p \leftarrow \neg q, q \leftarrow \neg p, r \leftarrow p, r \leftarrow q\} \tag{21}$$

In addition, consider the candidate model $S \coloneqq \{p, r\}$. Let's now test whether $S \in \mathcal{AS}(\Pi)$ by applying the definition:

(1) the reduct of Π with respect to S is $\Pi^S \coloneqq \{p, r \leftarrow p, r \leftarrow q\}$;
(2) the unique minimum model of Π^S is $M_{\Pi^S} \coloneqq \{p, r\}$;
(3) since $S = M_{\Pi^S}$, we can conclude that $S \in \mathcal{AS}(\Pi)$.

Now consider another candidate model $S' \coloneqq \{p, q, r\}$, and apply the same procedure:

(1) the reduct with respect to S' is $\Pi^{S'} \coloneqq \{r \leftarrow p, r \leftarrow q\}$;
(2) the unique minimum model of the reduct is $M_{\Pi^{S'}} \coloneqq \emptyset$;
(3) since $S' \neq M_{\Pi^{S'}}$, we can conclude that $S' \notin \mathcal{AS}(\Pi)$.

□

Remark. The term *Answer Set Programming*—often shortened as *ASP*—refers to all those Logic Programming techniques whose goal is to compute the stable models of a program given in input. On the other hand, languages like Prolog, allow the use also of non definite programs, but negation is interpreted as negation as failure.

2 ANSWER SET PROGRAMMING VIA QUANTUM COMPUTATION

Before delving into the description of both the algorithms in [7, 14, 19] and our original proposal, we want to briefly recall their key aspects.

In [7] the authors proposed a graph-based classical algorithm to find the stable models of an ASP program. The same problem was solved in [14] but with a quantum algorithm. In this case, the authors proposed a Grover-based quantum technique to find stable models.

A different question was answered in [19]: given a propositional logic formula ϕ, we want to count in a *weighted* manner the number of interpretations that satisfy ϕ.

Hence, our proposal stands somehow in the middle between [7] and [19]. An issue in [7] is that one of the best quantity to guide the graph visit is computationally hard to determine. Therefore, we devise a Quantum Algorithm, based on the proposal of [19], to

compute such quantity with a quadratic speed up with respect to any classical approach.

2.1 Stable Models via Grover Operator [14]

In [14], the authors proposed a quantum algorithm to both find and count stable models of an ASP program. Their approach was based on a quantum phase-flip oracle O capable of identifying stable models. Such oracle O was then plugged into a Grover Operator and they let the search algorithm do the trick. Roughly speaking, the steps performed by the algorithm are as follows:

(1) It initializes an equiprobable superposition of all the interpretations $S \subseteq B_\Pi$
(2) The superposition is run through O. The result is that all the states that represent a desired solution—a stable model $S \in AS(\Pi)$—are marked with a phase-flip.
(3) The state is then fed into the diffusion operator, whose aim is to increase the amplitude of the solutions marked by the oracle. Simultaneously, amplitudes of the unmarked states are reduced
(4) The two previous steps are iterated until a the probability of the desired states is high enough—greater than 1/2.

This kind of approach almost mimics what in classical computation would be called *bruteforce*. Despite Grover algorithm quadratic speedup is undeniable, it also holds that a lot of real-world problems can be solved using vary effective heuristics [4].

Hence, the aim of this paper is to study variations of the Grover Algorithm in order to apply this modified version to some ASP-related problems. As stated in the introduction, our goal is not to provide some groundbreaking results about both Quantum Computation and ASP. On the other hand, what we want to show is that some *natural* overlaps are present in the literature. These kind of overlaps should encourage authors from different areas to work alongside.

2.2 Weighted Count of Propositional Models [19]

The Weighted Model Counting (QMC) problem has been introduced in [19], where a quantum algorithm for solving it has also been proposed.

Roughly speaking, the aim is to solve a tweaked version of a counting problem. Tweaked in the sense that some in the overall counting some elements are *heavier* than others.

Let ϕ be a propositional logic formula over a set L of propositional letters and let $n := |L|$. Let $\prec$ be some strict total order over the elements of L:

$$L = \{l_1, l_2, \cdots l_n\} \quad where$$
$$l_1 \prec l_2 \prec l_3 \prec \cdots \prec l_n$$

For each $S \subseteq L$, let $en(S) := s_1 s_2 \cdots s_n$ be the $n-$symbol binary string such that $s_i = 1$ if and only if $l_i \in S$. We refer to $en(S)$ as the *encoding* of S. With respect to such encoding, let $\chi : \{0,1\}^n \to \{0,1\}$ be the following function:

$$\chi(en(S)) := \begin{cases} 1 & \text{if } \phi(S) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

Moreover, let $w : \{1, 2, \cdots n\} \times 0, 1 \to \mathbb{R}^{\geq 0}$ be a *weight function* for χ. Informally, $w(i, b)$ is the *weight* of setting the i-th input of χ to the value b. The *cumulative weight* W_x of some *encoding* $x := \text{en}(S)$ is defined as follows:

$$W_x = \prod_{i=1}^{n} w(i, x_i)$$

The reader may notice that, considering the Search Problem associated to χ—finding those $en(S)$ for which the result is one—the function w can be regarded as a way to express some *preference* on the elements of the search space.

The *Weighted Model Counting* problem is the evaluation of the following quantity:

$$\text{WMC}(\chi, w) := \sum_{x:\chi(x)=1} W_x$$

As the name suggests, this problem is a weighted variation of the problem of counting all the solutions of a Search Problem.

In [19], authors proposed a variation of the Quantum Counting circuit to solve WMC. The core idea was to replace the the Hadamard gates of the Quantum Counting Circuits, with a generic *Rot* gate that mimic the behaviour of the weight function w.

Let S_χ be the quantum phase-flip oracle that implements the boolean function χ, as defined in 22. For sake of readability, we will consider and explain the case where the weights are normalized:

$$w(i, 0) + w(i, 1) = 1 \quad \forall i = 1, 2, \cdots n$$

The reader may refer to [19] for the discussion about the unnormalized case. Since each weight w_i encodes some kind of preference for the i-th bit being set to 1, we would like the gate *Rot* to relate each w_i to the probability of measuring a 1 on the corresponding qubit. Formally speaking, the gate *Rot* must initialize each qubit to the following state:

$$|\psi_i\rangle = \sqrt{1 - w_i}\,|0\rangle + \sqrt{w_i}\,|1\rangle$$

In [19], it has been shown that this can be achieved by performing a rotation R_y of an angle θ_i defined as follows:

$$\theta_i = 2 \arccos \sqrt{1 - w_i} = 2 \arcsin \sqrt{w_i}$$

Joining the effect of the rotation on each qubit, the operator *Rot* is defined as:

$$Rot := \left(\bigotimes_{i=1}^{n} R_y(\theta_i) \right) \otimes \mathcal{H}$$

where the rightmost Hadamard gate has been added to account for the number of solutions—we do not know if the number M of solutions to the problem is larger or smaller than half of the size $N = 2^n$ of the search space.

This operator is then used to define the so called *Weighted Grover Operator*:

$$G_w = -Rot\, U_0\, Rot\, S_\chi$$

To prove the overall behaviour of their algorithm, the authors in [19] started by noticing that it is indeed possible to write two orthonormal vectors $|\gamma\rangle$ and $|\delta\rangle$ such that the following hold:

- $|\psi\rangle := Rot|0\rangle^{\otimes(n+1)}$ can be expressed as a linear combination of these two vectors
- G_w is a rotation of an angle θ in the space spanned by $|\gamma\rangle$ and $|\delta\rangle$

What they obtained is that using Phase Estimation to estimate θ actually solves WMC, since:

$$\mathrm{WMC}(\chi, w) \coloneqq \sum_{x:\chi(x)=1} W_x = 2\sin^2\frac{\theta}{2}$$

2.3 Classically navigating Stable Models [7]

Fichte, Gaggl, and Rusovac presented in [7] a novel approach to the navigation of the solution space of an ASP program.

Given an ASP program Π such that $\mathcal{AS}(\Pi)$ denotes the set of all its stable models, we call:

- $\mathcal{BC}(\Pi) \coloneqq \bigcup \mathcal{AS}(\Pi)$ the set of Π's *brave consequences*;
- $\mathcal{CC}(\Pi) \coloneqq \bigcap \mathcal{AS}(\Pi)$ the set of its *cautious consequences*.

Roughly speaking, the brave consequences of Π are all the atoms that appear in some of its stable models, while the cautious consequences are those that occur in all of them. The set of *facets* of Π is then defined as follows:

$$\mathcal{F}(\Pi) \coloneqq \mathcal{F}^+(\Pi) \cup \mathcal{F}^-(\Pi) \tag{23}$$

where:

$$\mathcal{F}^+(\Pi) \coloneqq \mathcal{BC}(\Pi) \setminus \mathcal{CC}(\Pi)$$
$$\mathcal{F}^-(\Pi) \coloneqq \left\{\overline{\alpha} : \alpha \in \mathcal{F}^+(\Pi)\right\}$$

An interpretation $S \subseteq B_\Pi$ for Π is said to satisfy an *inclusive facet* $\alpha \in \mathcal{F}^+(\Pi)$ if $\alpha \in S$. Symmetrically, S is said to satisfy an *exclusive facet* $\overline{\alpha} \in \mathcal{F}^-(\Pi)$ if $\alpha \notin S$. We denote by $ic(f)$ the singleton ASP program that contains the integrity constraint[5] corresponding to the facet $f \in \mathcal{F}(\Pi)$:

$$ic(f) \coloneqq \begin{cases} \{\leftarrow \neg\alpha\} & \text{if } f \text{ is an atom } \alpha \\ \{\leftarrow \alpha\} & \text{otherwise} \end{cases} \tag{24}$$

Let Π be an ASP program. The result of the activation of a facet $f \in \mathcal{F}(\Pi)$ is the program Π' defined as:

$$\Pi' \coloneqq \Pi \cup ic(f) \tag{25}$$

We say that the activation of f denotes a *navigation step* from Π to Π'. Observe that recursively applying different navigation steps to a program Π produces a graph-like structure, which we may wish to navigate. In order to do so, a finite sequence $\delta \coloneqq \langle f_1, \ldots, f_k\rangle$ of facets defines a *route*, and denotes an ordered sequence of navigation steps from an initial program Π. Notice that faceted navigation is possible as long as $\mathcal{F}(\Pi) \neq \emptyset$. As a consequence, we are usually interested in identifying a set of *safe* routes, which we denote by:

$$\Delta_s^\Pi \coloneqq \left\{\delta \in \Delta^\Pi : \mathcal{AS}(\Pi^\delta) \neq \emptyset\right\} \tag{26}$$

where Δ^Π is the set of all possible routes on Π, and $\Pi^\delta \coloneqq \Pi \cup ic(f_1) \cup \cdots \cup ic(f_k)$, with $\delta \coloneqq \langle f_1, \ldots, f_k\rangle$. In [7], the authors define two *navigation modes*, i.e. functions that prune the solution space according to some strategy that involves routes and facets. Intuitively:

- *goal-oriented* navigation aims to narrow down the solution space until a unique solution is found;
- *free* navigation does not follow any particular strategy: it allows following unsafe routes, which can be *redirected* if $\mathcal{F}(\Pi)$ eventually becomes empty.

In addition, the authors also propose a *weighted* navigation variant: since the effect of activating a facet is basically unknown beforehand, they suggest to associate a weight to each facet in order to characterize the extent to which activating said facet affects the size of the solution space. Intuitively, weight functions can be used to guide the faceted navigation process in a meaningful manner.

2.4 Speeding up Stable Models navigation with Quantum WMC

Among the different weight functions that Fichte, Gaggl, and Rusovac considered and evaluated in [7], we take as example *Absolute Weight*, defined as follows[6]:

$$w_{\#\mathcal{AS}}(f, \Pi^\delta) \coloneqq |\mathcal{AS}(\Pi^\delta)| - |\mathcal{AS}(\Pi^{\langle\delta, f\rangle})| \tag{27}$$

The authors showed that $w_{\#\mathcal{AS}}$ is really effective when used to guide the navigation[7]. Unfortunately, computing absolute weights is #coNP-*complete* [7].

As original contribution of this paper, we now show how Quantum WMC can be used to compute absolute weights with a quadratic speed up over any classical method. Let Π be an ASP program such that $n \coloneqq |B_\Pi|$, and let $\delta \coloneqq \langle f_1, \ldots, f_k\rangle \in \Delta^\Pi$ be a route for Π. Our goal is to estimate the following quantity:

$$M^\delta \coloneqq |\mathcal{AS}(\Pi^\delta)| \tag{28}$$

Being able to solve such problem would result in a simple technique to compute the weight $w_{\#\mathcal{AS}}(f, \Pi^\delta)$ through Equation 27.

Let $\chi : \{0,1\}^n \mapsto \{0,1\}$ be a boolean function that outputs a 1 if and only if its input is a binary string that encodes a stable model of Π, and let S_χ be a quantum phase-flip oracle for χ. Now consider the weight function w defined as follows:

$$w(i,1) \coloneqq \begin{cases} 1 & \text{if } \exists\, j = 1, \ldots, k \; f_j = \alpha_i \\ 0 & \text{if } \exists\, j = 1, \ldots, k \; f_j = \overline{\alpha_i} \;\; \forall\, i = 1, \ldots, n \\ 1/2 & \text{otherwise} \end{cases} \tag{29}$$

and set each $w(i,0)$ so that w is normalized. Let $S \subseteq B_\Pi$ be an interpretation for Π and $en(S)$ be its binary encoding. The following observations can be done:

- if $S \notin \mathcal{AS}(\Pi^\delta)$, then either one of the following two cases applies:
  - δ contains an inclusive facet $f_j = \alpha_i$ such that $\alpha_i \notin S$. This means that $w(i,1) = 1$ and $w(i,0) = 0$. But since $\alpha_i \notin S$ by hypothesis, then the i-th bit of $en(S)$ is 0 and thus $W_{en(S)} = 0$;
  - otherwise, δ contains an exclusive facet $f_j = \overline{\alpha_i}$ such that $\alpha_i \in S$. In this case $w(i,1) = 0$, and since the i-th bit of $en(S)$ is 1 by hypothesis then again $W_{en(S)} = 0$.

[5] An *integrity constraint* is a rule with an empty head.

[6] The actual definition of $w_{\#\mathcal{AS}}$ that the authors show in their article is a bit more complex, as it needs to account for "fallback" routes in the case $\langle\delta, f\rangle$ is not a safe route. For simplicity, we ignore here the added complexity that derives from this fact.

[7] Fichte, Gaggl, and Rusovac proved in [7] that $w_{\#\mathcal{AS}}$ enjoys a list of properties that they define in order to compare several weight functions.

- if, instead, $S \in \mathcal{AS}(\Pi^\delta)$, then $W_{en(S)} = 1^k \cdot (1/2)^{n-k}$, where k is the number of facets of the route δ. Notice that k does not depend on S.

Therefore, applying the quantum circuit for Weighted Model Counting that we analyzed in Section 2.2 produces the following result:

$$\mathrm{WMC}(\chi, w) \coloneqq \sum_{S \in \mathcal{AS}(\Pi)} W_{en(S)} = |\mathcal{AS}(\Pi^\delta)| \cdot \left(1^k \cdot (1/2)^{n-k}\right) \tag{30}$$

Thus, M^δ can be retrieved using the following equality:

$$M^\delta \coloneqq |\mathcal{AS}(\Pi^\delta)| = \frac{\mathrm{WMC}(\chi, w)}{\left(1^k \cdot (1/2)^{n-k}\right)} = 2^{n-k} \cdot \mathrm{WMC}(\chi, w) \tag{31}$$

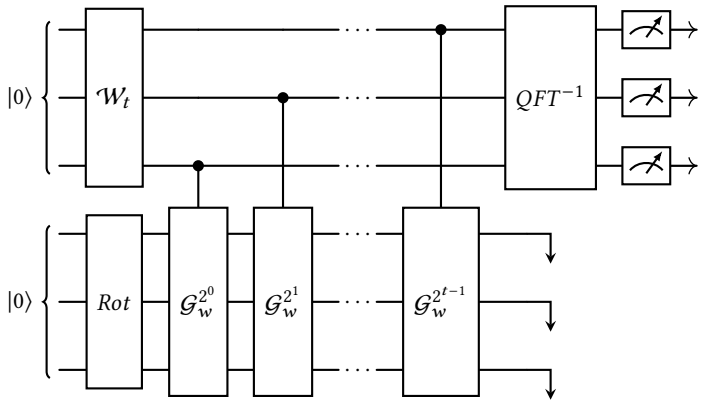

Figure 1: Circuit implementing the Quantum Weighted Model Counting algorithm.

In Figure 1 we depicted a quantum circuit solving the WMC problem. The result of such circuit is a key ingredient to compute $|\mathcal{AS}(\Pi^\delta)|$.

Example 2.1. To clarify the definition of w from Equation 29, reconsider the program Π from Example 1.6. In addition, let $\delta \coloneqq \langle p \rangle$, and observe that $\mathcal{AS}(\Pi^\delta) = \{\{p, r\}\}$. According to the definition and assuming $p \prec q \prec r$ – i.e. p corresponds to the first bit, q to the second one, and r to the third one – the weight function w is defined as follows:

$$\begin{aligned} w(1,0) &\coloneqq 0 \quad w(2,0) \coloneqq 1/2 \quad w(3,0) \coloneqq 1/2 \\ w(1,1) &\coloneqq 1 \quad w(2,1) \coloneqq 1/2 \quad w(3,1) \coloneqq 1/2 \end{aligned} \tag{32}$$

In Section 3, we show an implementation of the *Rot* gate that uses the weight function from Example 2.1, with the aim of counting the stable models of Π^δ.

3 IMPLEMENTATION

In this section we briefly describe the implementation we provide for the aforementioned algorithms. Our code is written in Python and it is publicly available on GitHub at the following address:

https://github.com/davidedellagiustina/qasp-solver

The code is extensively documented and the `README.md` file in the repository reports detailed build and execution instructions. Hence, in this section we focus on explaining which examples have been implemented.

3.1 Running the Examples

As previously mentioned, our GitHub repository also contains some examples, which can be found in the folder `./src/examples`. In order to run them, the command runner `just` can be used.

```
# Install `just' command runner (Ubuntu)
snap install --edge --classic just

# List of commands that `just' can run
just -l
```

Examples can be listed and run using the following commands:

```
# List all the available examples
just list

# Run the example <example_name>
just run <example_name>
```

The paragraphs that follow briefly explain the purpose of each example. Additionally, in each file, the head of the file is allocated to explain which ASP program has been taken into account for that particular example

3.1.1 `grover_search_known_m`. This example demonstrates how to use a plain implementation of the Grover Search algorithm to find one of the stable models of Π. In order to compute the optimal number of iterations needed to maximize the probability of measuring a correct solution, we assume here to know a priori the number M of answer sets of the considered program.

3.1.2 `quantum_counting`. Elaborating on the previous example, we show here how to use the Quantum Counting circuit in order to estimate the value of M. However, observe that using $m \coloneqq \lceil n/2 \rceil + 1 = 3$ in this case provides a very inaccurate estimate for the desired value.

Remark. As we already discussed, the raw output of the Quantum Counting algorithm is an estimate of an angle φ to m binary digits that is correct with probability $1 - \varepsilon$. Aside from the probability of success of the algorithm, observe that stating that $0.\varphi_0 \dots \varphi_{m-1}$ is an estimate for φ up to the m-th binary digit is equivalent to the following:

$$\begin{gathered} 0.\varphi_0 \dots \varphi_{m-1}000\dots \;\leq\; \varphi \;\leq\; 0.\varphi_0 \dots \varphi_{m-1}111\dots \\ \equiv \\ 0.\varphi_0 \dots \varphi_{m-1} \;\leq\; \varphi \;<\; 0.\varphi_0 \dots \varphi_{m-1} + 2^{-m} \end{gathered} \tag{33}$$

This is why our implementation outputs an interval for M, instead of a point estimate.

3.1.3 `quantum_counting_advanced`. In order to improve the accuracy of the estimate in the previous example, we show here the same algorithm but with an increased value of $m \coloneqq 5$. While the circuit takes significantly longer to simulate, the results are clearly better. In this case, the algorithm correctly identifies (with probability at least $1 - \varepsilon = 5/6$) that the considered ASP program has 2 stable models.

3.1.4 `oracle_construction`. This example shows an application of the Grover Search algorithm to the resolution of an ASP program

where the oracle is built by implementing the classical procedure that checks whether a given interpretation is a stable model or not.

3.1.5 `quantum_faceted_navigation`. This example implements an instance of the quantum circuit that solves Weighted Model Counting—as presented in [19]—to count the number of stable models of an ASP program during faceted navigation of its solution space. In particular, the gate *Rot* is built by instantiating the weight function shown in Example 2.1.

3.1.6 `grover_search_unknown_m`. As the name suggests, this is a variation of the example `grover_search_known_m` where we additionally assume that the number M of stable models of the program Π is not known in advance, but we still want to find one of its stable models.

3.1.7 `embedded_heuristic`. While `quantum_faceted_navigation` exploits an instance of the *Rot* gate to approximately count the stable models of an ASP program Π during faceted navigation of its solution space, this example shows how to use the same instance of *Rot* in order to also find (one of) those stable models.

In our code, we demonstrate the functioning of the techniques presented on some toy examples. Given the computational complexity of the problems addressed, scalability will remain a bottleneck nonetheless.

4 CONCLUSIONS AND FUTURE WORKS

In this paper we addressed the possible improvement that could be obtained by speeding up Answer Set Programming using Quantum Computation. We started by presenting all the required backgrounds in both Quantum Computation and Answer Set Programming. After that we delved into the proposals from [7, 14, 19] by explaining their internals and we addressed one of the problems described of [7], i.e., the computation of guiding functions. In order to speed up this particular process, we proposed a Quantum Algorithm built upon a tweaked version of the quantum algorithm proposed in [19] to solve WMC. We also presented an implementation of our result.

As stressed before, in this paper we showed how Quantum Computation could help Answer Set Programming. The other way round is possible too. In fact, in works like [17, 18] Answer Set Programming is used as a technique to tackle Quantum related problems. In that particular case, the issue was the synthesis of CNOT-minimal quantum circuits. Hence, a double link connection, as proposed in [20], is actually a path worth following.

In the future, we are interested in an in depth analysis of quantum algorithms for solving ASP related problems, to show how the two disciplines can co-exists.

REFERENCES

[1] Matthew Amy. 2019. Towards Large-scale Functional Verification of Universal Quantum Circuits. *Electronic Proceedings in Theoretical Computer Science* 287 (Jan. 2019), 1–21. https://doi.org/10.4204/eptcs.287.1

[2] Linda Anticoli, Carla Piazza, Leonardo Taglialegne, and Paolo Zuliani. 2016. Towards quantum programs verification: from quipper circuits to QPMC. In *Reversible Computation: 8th International Conference, RC 2016, Bologna, Italy, July 7-8, 2016, Proceedings 8*. Springer, 213–219.

[3] Domenico Cantone, Simone Faro, Arianna Pavone, and Caterina Viola. 2023. Longest Common Substring and Longest Palindromic Substring in $\tilde{O}(\sqrt{n})$ Time. arXiv:2309.01250 [cs.DS]

[4] A. Colorni, M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini, and M. Trubian. 1996. Heuristics from Nature for Hard Combinatorial Optimization Problems. *International Transactions in Operational Research* 3, 1 (1996), 1–21. https://doi.org/10.1111/j.1475-3995.1996.tb00032.x

[5] D. Della Giustina, C. Londero, C. Piazza, B. Riccardi, and R. Romanello. 2024. Quantum encoding of dynamic directed graphs. *Journal of Logical and Algebraic Methods in Programming* 136 (2024), 100925. https://doi.org/10.1016/j.jlamp.2023.100925

[6] Massimo Equi, Arianne Meijer van de Griend, and Veli Mäkinen. 2023. From Bit-Parallelism to Quantum String Matching for Labelled Graphs. arXiv:2302.02848 [quant-ph]

[7] Johannes K. Fichte, Sarah A. Gaggl, and Dominik Rusovac. 2021. Rushing and Strolling among Answer Sets – Navigation Made Easy. *CoRR* (12 2021). https://doi.org/10.48550/arXiv.2112.07596

[8] Michael Gelfond and Yulia Kahl. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press. https://doi.org/10.1017/CBO9781139342124

[9] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, 212–219. https://doi.org/10.1145/237814.237866

[10] Massimiliano Incudini, Fabio Tarocco, Riccardo Mengoni, Alessandra Di Pierro, and Antonio Mandarino. 2022. Computing graph edit distance on quantum devices. *Quantum Machine Intelligence* 4, 2 (2022), 24. https://doi.org/10.1007/s42484-022-00077-x arXiv:2111.10183 [quant-ph]

[11] Marco Lewis, Sadegh Soudjani, and Paolo Zuliani. 2023. Formal Verification of Quantum Programs: Theory, Tools, and Challenges. *ACM Transactions on Quantum Computing* 5, 1 (2023), 1–35.

[12] Carlo Mereghetti, Beatrice Palano, and Priscilla Raucci. 2024. Unary Quantum Finite State Automata with Control Language. *Applied Sciences* 14, 4 (2024). https://doi.org/10.3390/app14041490

[13] Giulia Meuli, Mathias Soeken, and Giovanni De Micheli. 2018. SAT-based {CNOT, T} quantum circuit synthesis. In *Reversible Computation: 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings 10*. Springer, 175–188.

[14] David A. Meyer, James Pommersheim, and Jeffrey B. Remmel. 2004. Finding stable models via quantum computation. In *International Workshop on Non-Monotonic Reasoning*.

[15] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum computation and Quantum Information*. Cambridge University Press.

[16] Carla Piazza and Riccardo Romanello. 2022. Mirrors and Memory in Quantum Automata. In *Quantitative Evaluation of Systems: 19th International Conference, QEST 2022, Warsaw, Poland, September 12–16, 2022, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 359–380. https://doi.org/10.1007/978-3-031-16336-4_18

[17] Carla Piazza and Riccardo Romanello. 2023. Synthesis of CNOT minimal quantum circuits with topological constraints through ASP. In *CEUR WORKSHOP PROCEEDINGS*, Vol. 3586. CEUR-WS, 37–49.

[18] Carla Piazza, Riccardo Romanello, Robert Wille, et al. 2023. An ASP Approach for the Synthesis of CNOT Minimal Quantum Circuits. In *CEUR WORKSHOP PROCEEDINGS*, Vol. 3428. CEUR-WS.

[19] Fabrizio Riguzzi. 2019. Quantum Weighted Model Counting. In *European Conference on Artificial Intelligence*.

[20] Alex Della Schiava, Carla Piazza, and Riccardo Romanello. 2023. Graph-Theoretical Arguments in Support of a Quantum Declarative Manifesto. *CEUR Workshop Proceedings* 3428 (2023).

[21] Peter Shor. 2003. Lecture Notes in Quantum Computation.