Finding Stable Models via Quantum Computation

D. A. Meyer and **J. B. Remmel**
Department of Mathematics
University of California/San Diego
La Jolla, CA 92903-0112

J. Pommersheim
Department of Mathematics
University of California/San Diego
La Jolla, CA 92903-0112
and
Department of Mathematics and Computer Science
Pomona College
Claremont, CA 91711

Abstract

Quantum computers have the potential to out-perform classical computers—certain quantum algorithms run much faster than any known alternative classical algorithm. For example, Grover showed that a quantum computer can search an unordered list of N items in time $O(\sqrt{N})$, providing a quadratic speed-up over the classical algorithm. In this paper, we show that we can modify Grover's search algorithm to give an algorithm that finds stable models of an Answer Set Program with a similar quadratic improvement over the classical algorithm. Marek and Remmel showed that Answer Set Programming (ASP) programs can uniformly solve all NP-search problems, so our quantum algorithm to find stable models of ASP programs also solves all NP-search problems. It follows that Answer Set Programming could provide a programming language for quantum computation.

Introduction

The history of electronic computing has been marked by the fact that the computers of each new generation are dramatically smaller and faster than their predecessors. Nevertheless, from a computational point of view, each generation of computers is essentially the same: the machines are built out of simple logic gates. The Church-Turing thesis asserts that this is inevitable since any computer can be simulated with at most a polynomial factor slowdown by a probabilistic Turing machine. The promise of quantum computation is that we may be able to use the laws of quantum mechanics to build a quantum computer that can out-perform classical computers, violating the classical Church-Turing thesis. Quantum computation originated with a suggestion by Feynman that although there appears to be no efficient way of simulating a multiparticle quantum system on a classical computer, there would be a way to run a simulation efficiently on a computer that took advantage of the properties of the quantum world (Feynman 1982). Two formal models for quantum computers—the quantum Turing machine (Deutsch 1985) and quantum computational networks, *i.e.*, quantum gate arrays (Deutsch 1989)—were defined subsequently by Deutsch.

Since these initial results, evidence that quantum computers can out-perform classical computers has been accumulating. In the work that stimulated the greatest surge of interest in the subject, Shor showed that there exist polynomial time quantum algorithms for two of the most famous problems in computer science: factoring and discrete log (Shor 1994). Bernstein and Vazirani gave the first formal evidence that quantum computers violate the modern form of the Church-Turing thesis. They proved that the recursive Fourier sampling problem can be solved in polynomial time on a quantum Turing machine, but relative to an oracle, requires superpolynomial time on a classical probabilistic Turing machine (Bernstein and Vazirani 1997).

Nevertheless, there are limits to the power of quantum computers. For example, Bennett, Bernstein, Brassard and Vazirani proved that, relative to a random oracle, with probability 1, the class NP cannot be solved on a quantum Turing machine in time $o(2^{n/2})$ (Bennett et. al. 1997). This bound is tight, since Grover's search algorithm (Grover 1996) shows that one can accept any language in NP in time $O(2^{n/2})$ on a quantum Turing machine.

These results on the power of quantum computation leave a wide range of possible applications of quantum algorithms to be explored. Our explorations in this paper are guided by recent developments in Knowledge Representation, especially the appearance of a new generation of systems (Cholewiński et. al. 1996; Niemelä and Simons 1996; Eiter et. al. 1998) based on the so-called Answer Set Programming (ASP) paradigm (Niemelä 1998; Cadoli and Palipoli 1988; Marek and Truszczyński 1999; Lifschitz 1999). We shall focus on one particular ASP formalism, namely, the Stable Semantics for Logic Programs (SLP) (Gelfond and Lifschitz 1988).

The underlying methods of ASP are similar to those used in Logic Programming (Apt 1990) and Constraint Programming (Jaffar and Maher 1994; Marriott and Stuckey 1998). That is, like Logic Programming, ASP is a declarative formalism and the semantics of all ASP systems are based on logic. Like Constraint Programming, certain clauses of an ASP program act as *constraints*. There is, however, a fundamental difference between ASP programs and Constraint Logic programs: in Constraint Programming, the constraints act on individual elements of the Herbrand base of the program, while the constraint clauses in ASP programs act more globally, placing restrictions on which subsets of the Herbrand base can be acceptable answers for the program.

For example, suppose that we have a problem Π whose solutions are *subsets* of some Herbrand base H. In order to

solve the problem, an ASP programmer essentially writes a logic program P that describes the constraints on the subsets of H that can be answers to Π. The basic idea is that the program P should have the property that there is an easy decoding of solutions of Π from stable models of P, and that all solutions of Π can be obtained from stable models of P through this decoding. The program P is then submitted to an ASP engine such as *smodels* (Niemelä and Simons 1996), `dlv` (Eiter et. al. 1998) or DeReS (Cholewiński et. al. 1996), which computes the stable models of the program P. Thus the ASP engine finds the stable models of the program (if any exists) and one reads off the solutions to Π from these stable models. Notice that all solutions are equally good in the sense that any solution found in the process described above is acceptable.

In the Aswer Set Programming paradigm, the semantics of logic program P can be defined in two stages. First, we assume, as in standard Logic Programming, that we interpret P over the Herbrand universe of P determined by the predicates and constants that occur in P. Since the set of constants occurring in the program is finite, we can ground the program in these constants to obtain a finite propositional logic program P_g. The stable models of P are by definition the stable models of P_g. The process of grounding is performed by a separate grounding engine such as *lparse* (Niemelä and Simons 1996). Second, the grounded program P_g is passed to the engine computing the stable models.

The basic complexity result for SLP propositional programs is due to Marek and Truszczyński, who showed that the problem of deciding whether a finite propositional logic program has a stable model is NP-complete (Marek and Truszczyński 1991). For DATALOG (with negation), an analogous result has been obtained by Schlipf (Schlipf 1995). Marek and Remmel strengthened both of these results by showing that ASP based on SLP is complete for NP-search problems (Marek and Remmel 2003). This means that the search engines developed so far are appropriate to solve a vast array of practical problems. Currently, systems based on the ASP paradigm are being tested on problems related to planning, product configuration, combinatorial optimization problems and other domains. For example, ASP systems have been applied to circuit verification problems (Heljanko and Niemelä 2001), product configuration problems (Soininen et. al. 2001), information extraction engines for the web (**?**), and updating database specifications (Eiter et. al. 2001).

The main goal of this paper is show that if we had a quantum computer, then we could develop a *quantum* search engine for Answer Set Programming using a suitable extension of Grover's basic quantum search algorithm. Our extension of Grover's algorithm gives a quadratic speed-up over the naïve search algorithm for stable models. The importance of this marriage of Answer Set Programming with quantum computation is that ASP provides an effective language for computation and thus could the basis of a practical "quantum programming language" for quantum computation.

Quantum algorithms

As we noted in the Introduction, in practice any classical computation is implemented by an array of logic gates, each acting on a finite set of bits. A somewhat surprising result, discovered in the investigation of thermodynamic limits to computation, is that in principle these gates can each be *reversible* (Bennett 1973). For example, Toffoli (Toffoli 1981) showed that a finite set of reversible gates is *universal* for classical computation, since any reversible boolean operation can be decomposed as an array of three gates: NOT : $b \mapsto b+1$, C-NOT : $(a,b) \mapsto (a, b+a)$, C-C-NOT : $(a,b,c) \mapsto (a,b,c+ab)$, where a, b and c are bits and $+$ is addition modulo 2. These gates permute the set of possible states of 1, 2, and 3 bits, respectively. Thus any classical computation can be implemented as a sequence of permutations acting on the possible states of the classical computer—n-bit strings, $b_1 \dots b_n$—each of which acts non-trivially only on a subset of 1, 2, or 3 bits.

In an analogous formulation of quantum computation, the possible states are complex linear combinations (*superpositions*) of n-bit strings, $\sum a_{b_1 \dots b_n} |b_1 \dots b_n\rangle$. Here $|\cdot\rangle$ is the standard physics notation for a vector in $(\mathbb{C}^2)^{\otimes n}$, in which we have chosen a basis $\{|b_1 \cdots b_n\rangle \mid b_i \in \{0,1\}\}$. That is, $\mathbb{C}^2$ has a basis

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Thus if

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

then

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle. \end{aligned}$$

When we write $|b_1 \dots b_n\rangle$ where $b_i \in \{0,1\}$, we mean the column vector of size 2^n whose rows are indexed by bits strings $c_1 \dots c_n \in \{0,1\}^n$ such that there is a 1 in the row indexed by the bit string $b_1 \dots b_n$ and all other entries are 0. The states $\sum a_{b_1 \dots b_n} |b_1 \dots b_n\rangle$ should be thought of as being analogous to the states of a classical *probabilistic* computer, which are probability distributions over n-bit strings, *i.e.*, linear combinations $\sum p_{b_1 \dots b_n} \mathtt{b_1} \dots \mathtt{b_n}$, where the coefficients are probabilities, so $\sum p_{b_i \dots b_n} = 1$. In quantum mechanics, it is the *norm-squares* of the components that are probabilities, so states satisfy $\sum |a_{b_1 \dots b_n}|^2 = 1$. This condition allows *measurement* of the state to be described as a choice of basis (which in this paper will always be this *computational* basis), and the *outcome* of a measurement to be one of the basis vectors, with probability given by the norm-squared of that component. We will also need the notation $\langle \cdot |$ to indicate a vector in the dual space to $(\mathbb{C}^2)^{\otimes n}$.

Any quantum computation can be implemented as a sequence of unitary transformations acting on $(\mathbb{C}^2)^{\otimes n}$, each of which acts non-trivially only on 1 or 2 of the factors (*qubits*) in this tensor product. That only such 1 and 2 qubit *quantum*

gate operations are necessary follows from the result that arbitrary unitary transformations of a single qubit, together with the linear extension of C-NOT to a unitary transformation of $\mathbb{C}^2 \otimes \mathbb{C}^2$, are universal for quantum computation (Adelman et. al. 1997). Notice that since permutations are unitary transformations, if we also allow a C-C-NOT quantum gate, *i.e.*, the linear extension of C-C-NOT to a unitary transformation of $\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2$, every (reversible) classical computation is trivially, exactly, a quantum computation. Without this 3 qubit gate we must rely on the universality theorem to ensure that every (reversible) classical computation can be approximated efficiently (*i.e.*, with poly(n) overhead) by a quantum computation. But the remarkable discoveries of Deutsch (Deutsch 1989), Simon (Simon 1997), Bernstein and Vazirani (Bernstein and Vazirani 1997), Shor (Shor 1994), and Grover (Grover 1996) show that in some cases *fewer* quantum gates are required to transform the quantum computer into a state that upon measurement returns the correct bit string with probability greater than $2/3$ (any probability bounded above $1/2$ is polynomially equivalent, by repeating the quantum computation multiple times and choosing the most frequent outcome). We will explain exactly how this works in the case of Grover's quantum search algorithm, but first we describe in detail the classical problems to which we will apply it.

ASP search problems

A search problem is a set $\mathcal{P}$ of finite instances such that, given any instance $\Pi \in \mathcal{P}$, there is a set S_Π of solutions to $\mathcal{P}$ for instance Π, where S_Π can be empty. For example, the search problem might be to find Hamiltonian paths in a graph. In this case, the set of instances of the problem is the set of all finite graphs. Given any instance, *i.e.*, a graph Γ, S_Γ is the set of all Hamiltonian paths of Γ. We say that an algorithm solves the search problem $\mathcal{P}$ if it returns a solution $S \in S_\Pi$ whenever S_Π is non-empty and it returns the string "empty" otherwise. We say that a search problem $\mathcal{P}$ *is in NP* if there is such an algorithm which can be computed by a non-deterministic polynomial time Turing machine. We say that search problem $\mathcal{P}$ *is solved by a uniform logic program* if there exists a single logic program $P_\mathcal{P}$, a polynomial time extensional data base transformation function $edb_\mathcal{P}(\cdot)$, and a polynomial time solution decoding function $sol_\mathcal{P}(\cdot,\cdot)$, such that for every instance $\Pi \in \mathcal{P}$,

1. $edb_\mathcal{P}(\Pi)$ is a finite set of facts, *i.e.*, clauses with empty bodies and no variables, and

2. whenever S_Π is not empty, $sol_\mathcal{P}(\Pi,\cdot)$ maps the set of stable models of $edb_\mathcal{P}(\Pi) \cup P_\mathcal{P}$ onto the set of solutions S_Π of Π, and

3. whenever S_Π is empty, $edb_\mathcal{P}(\Pi) \cup P_\mathcal{P}$ has no stable models.

We note that decision problems can be viewed as special cases of search problems. Schlipf's result (Schlipf 1995) shows, in fact, that the class of *decision* problems in NP is captured precisely by uniform logic programs. Specifically he proved that a decision problem is solved by a uniform logic program if and only if it is in NP. Marek and Remmel showed that Schlipf's result can be extended to all NP *search* problems. That is, Marek and Remmel showed that there is a single logic program $P_{\mathcal{T}uring}$ that is capable of simulating polynomial time nondeterministic Turing machines in the sense that given any polynomial time nondeterministic Turing machine M, any input σ, and any run-time polynomial $p(x)$, there is a set of facts $edb_{\mathcal{T}uring,p,\sigma}(M)$ such that a stable model of $P_{\mathcal{T}uring} \cup edb_{\mathcal{T}uring,p,\sigma}(M)$ codes an accepting computation of M started with input σ that terminates in $p(|\sigma|)$ or fewer steps and any such accepting computation of M is coded by some stable model of $P_{\mathcal{T}uring} \cup edb_{\mathcal{T}uring,p,\sigma}(M)$. This result shows that logic programs without function symbols under the stable logic semantics capture all NP-search problems. The converse, that a search problem computed by a uniform logic program P is an NP-search problem, is obvious since one can compute a stable model s of a program by guessing s and then doing a polynomial time check to verify that s is a stable model of the program.

In this paper we will consider only so-called DATALOG$^\neg$ programs. These consist of clauses of the form

$$p(\overline{X}) \leftarrow q_1(\overline{X}), \ldots, q_m(\overline{X}), \neg\, r_1(\overline{X}), \ldots, \neg\, r_n(\overline{X}), \quad (1)$$

where $p, q_1, \ldots, q_m, r_1, \ldots, r_n$ are atoms, possibly with variables and/or constants. Here we abuse notation by writing $p(\overline{X})$ to mean that the variables that occur in the predicate p are contained in $\overline{X}$. A program is a finite set P of clauses of the form (1). We assume that the underlying language $\mathcal{L}_P$ of any given program P is determined by the constants and predicate symbols that occur in the program. Thus the Herbrand universe U_P of P is just the set of all constant terms occurring in P, and the Herbrand base H_P of P is the set of all ground atoms of the language $\mathcal{L}_P$. Since there are no function symbols in our programs, both the Herbrand universe and the Herbrand base of the program are finite.

A ground instance of a clause C of the form (1) is the result of a simultaneous substitution of constants for variables occurring in C. Given a program P, P_g is the propositional program consisting of all ground substitutions of clauses of P. Given a propositional program P and a set S included in its Herbrand base, H_P, the *Gelfond-Lifschitz transformation of P relative to S*, is the program $\mathrm{GL}(P,S)$ arising from P as follows. First, eliminate all clauses C in P such that for some j, $1 \leq j \leq n$, $r_j \in M$. Second, in any remaining clause, eliminate all negated atoms. The resulting set of clauses forms a program, $\mathrm{GL}(P,S)$, which is a Horn program and hence possesses a least model M_S. We say that S is a *stable model of the propositional program P* if $S = M_S$. Finally, given any program P with variables, we say that S is a stable model of a program P if S is a stable model of the propositional program P_g.

Finding a unique stable model

Finding a stable model for an ASP program P is a search problem among the $2^{|H_P|}$ possible models for an $S \subset H_P$

such that $S = M_S$. The simplest classical algorithm over which there is no known worst-case improvement is

1. choose $S \subset H_P$;
2. construct the Gelfond-Lifschitz transform $\mathrm{GL}(P, S)$;
3. find the least model M_S of $\mathrm{GL}(P, S)$ by iterating the one-step provability operator $T_{\mathrm{GL}(P,S)}$ on $\emptyset$; (This can be done in linear time, see (Dowling and Gallier 1984).)
4. if $S = M_S$, return S, else repeat.

Each step here is polynomial in the problem size, but in the worst case the steps must be repeated $O(2^n)$ times, where $n = |H_P|$.

Notice that we could formulate each iteration of this algorithm as an evaluation of a function $f_P : \mathcal{P}(H_P) \to \{0, 1\}$ on $S \in \mathcal{P}(H_P)$ (the power set of H_P), where

$$f_P(S) = \begin{cases} 1 & \text{if } S = M_S; \\ 0 & \text{otherwise,} \end{cases}$$

and the problem is to identify an argument at which f_P takes the value 1. If we encode $\mathcal{P}(H_P)$ as n-bit strings, then we can write $f_P : \{0, 1\}^n \to \{0, 1\}$. Consider the function from $\{0, 1\}^n \times \{0, 1\}$ to itself defined by $(x, b) \mapsto \big(x, b + f_P(x)\big)$, which is a permutation. Since the collection of all $|x, b\rangle$ is a basis (the computational basis) of the vector space $V = (\mathbb{C}^2)^{\otimes n} \otimes \mathbb{C}^2$, this permutation induces a linear map U_{f_P} from V to V, which satisfies

$$U_{f_P}|x, b\rangle = |x, b + f_P(x)\rangle.$$

Since U_{f_P} permutes the computational basis, the *same* (polynomial size) reversible gate array that computes this permutation classically can be also be used to implement the unitary map U_{f_P}. We will set the last qubit to be in the state $|-\rangle = \big(|0\rangle - |1\rangle\big)/\sqrt{2}$ so that this unitary transformation takes a particularly simple form:

$$U_{f_P}|x, -\rangle = (-1)^{f_P(x)}|x, -\rangle.$$

Thus we can abuse notation slightly and write

$$U_{f_P}|x\rangle = (-1)^{f_P(x)}|x\rangle.$$

More generally, we can define U_g similarly, for *any* function $g : \{0, 1\}^n \to \{0, 1\}$.

We need one more definition before we can describe the application of Grover's algorithm (Grover 1996) to this problem. Let

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

denote the discrete Fourier transform on $\mathbb{C}^2$. In the quantum computing literature this is usually called *the Hadamard transform*. Notice that

$$H|0\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big);$$

this is used in Step 2 below.

Grover's algorithm consists of the following steps:

1. prepare the state
$$|0 \ldots 0\rangle \in (\mathbb{C}^2)^{\otimes n};$$
2. apply $H^{\otimes n}$ to obtain the state
$$\frac{1}{\sqrt{2^n}} \sum_x |x\rangle;$$
3. repeat T times:
   (a) apply U_{f_P};
   (b) apply $-H^{\otimes n} U_{\delta_0} H^{\otimes n}$; ($\delta_0(x) = 1$ if $x = 0$, and vanishes otherwise)
4. measure the state and return the outcome x of the measurement.

A simple analysis determines the appropriate value of T in Grover's algorithm. Assume first that P has a unique stable model S, as, for example, is the case in stratified programs (Apt et. al. 1988). Then U_{f_P} acts as a reflection in the hyperplane orthogonal to $|s\rangle$, where s is the n-bit string encoding S. Similarly, since U_{δ_0} is a reflection in the hyperplane orthogonal to $|0 \ldots 0\rangle$, and $H^{\otimes n}$ maps $|0 \ldots 0\rangle$ to the equal superposition state obtained in Step 2, the unitary transformation $-H^{\otimes n} U_{\delta_0} H^{\otimes n}$ is a reflection in the hyperplane orthogonal to the equal superposition vector. Recall that the product of two hyperplane reflections is a rotation in the plane spanned by the vectors orthogonal to the hyperplanes, through twice the angle between them. In this case that angle is $\theta = \arcsin(1/\sqrt{2^n})$, so each iteration of Step 3 rotates the state by 2θ towards $|s\rangle$, and choosing

$$T = \left\lfloor \frac{\pi/2 - \theta}{2\theta} \right\rceil \sim \frac{\pi}{4}\sqrt{2^n} \quad \text{as } n \to \infty$$

ensures that when the state is measured in Step 4, it returns the encoding of the stable model with probability at least $\cos^2 \theta \sim 1 - 2^{-n}$. But it does so with only $O(\sqrt{2^n})$ iterations, quadratically faster than the classical algorithm at the beginning of this section.

Finding one of multiple stable models

Of course, in general there are multiple stable models for an ASP program. Suppose P has $k \in \mathbb{N}$ stable models, and we apply Grover's algorithm as described in the previous section. Now U_{f_P} is reflection through the hyperplane orthogonal to $\sum_{x \text{ stable}} |x\rangle$, so the two reflections in Step 3 create a rotation by 2θ where

$$\sin\theta = \frac{1}{\sqrt{k}} \sum_{x \text{ stable}} \langle x| \cdot \frac{1}{\sqrt{2^n}} \sum_x |x\rangle = \sqrt{\frac{k}{2^n}}.$$

Thus, if we knew the number of stable models, k, we could iterate Step 3 $T \sim \frac{\pi}{4}\sqrt{2^n/k}$ times, after which the state would be within an angle θ of $\sum_{x \text{ stable}} |x\rangle/\sqrt{k}$. Then measuring the state in Step 4 would, with probability $1 - O(2^{-n})$, return one of the stable models, each with equal probability.

Unfortunately, we do not know how many stable models there are *a priori*. Brassard, Høyer, Mosca and Tapp have

shown, however, that even when the number of solutions to Grover's search problem is unknown, one can still be found with expected number of iterations $O(\sqrt{2^n/k})$ (Brassard et. al. 2002). Their algorithm is based on the observation that picking a random number of iterations of Step 3 would correspond to picking an angle $0 \leq \phi < 2\pi$ at random, where ϕ is the angle between the final state and $\sum_{x \text{ stable}} |x\rangle/\sqrt{k}$. A measurement returns the encoding of a stable model with probability $\cos^2 \phi$, so for random choices of ϕ,

$$E(\text{successful measurement}) \approx \int_0^{2\pi} \cos^2 \phi \, \mathrm{d}\phi = \frac{1}{2}.$$

Since we can check efficiently whether the outcome is a stable model, we could repeat this procedure r times and reduce the failure probability to $O(2^{-r})$, without increasing the computational complexity. Of course, we cannot really pick a random positive integer number of iterations. Nevertheless, this observation suggests the following algorithm:

1. let $M = 1$;
2. pick T uniformly at random from the integers in $[1, M]$;
3. run Grover's algorithm with this value of T;
4. if outcome is stable model, stop, else let $M = cM$ and repeat from Step 2.

For $1 < c < 2$, Brassard, Høyer, Mosca and Tapp's result (Brassard et. al. 2002) shows that this algorithm succeeds after an expected $O(\sqrt{2^n/k})$ total number of Grover iterations. Thus there is a quadratic improvement in this case as well as in the unique stable model case.

Discussion

We have shown that Grover's algorithm and Brassard, Høyer, Mosca and Tapp's generalization can be applied to the problem of finding stable models for ASP programs and that these quantum algorithms run quadratically faster than existing classical solvers in the worst case. Since Answer Set Programming is a general formalism for NP-search problems, this result shows how to solve any such problem with a quantum computer. Used in this way, ASP would be a general "quantum programming language".

Of course, since Shor's quantum algorithm for factoring provides a superpolynomial improvement over the best classical algorithm known, the hope is for more than a quadratic improvement in other problems. Evidence from highly structured search problems (Hunziker and Meyer 2002) suggests that generalizing f_P to return more information about a possible model S than only whether it is M_S or not could provide a greater improvement.

A similar idea motivates quantum adiabatic algorithms (Farhi et. al. 2001). These utilize a Hermitian matrix (the *Hamiltonian*) that acts on potential solutions of the problem encoded as quantum states. Applied to 3-SAT, for example, this matrix is constructed from the clauses of the problem instance in such a way that quantum states encoding truth assignments are eigenstates with eigenvalues that are positively proportional to the number of clauses violated. The algorithm is designed to evolve an initial state into a quantum state that has minimum eigenvalue, *i.e.*, corresponds to a truth assignment with the smallest number of violated clauses.

This idea has also been applied within the discrete time model of quantum computation described in this paper. Hogg describes this idea as the application of 'quantum heuristics' (Hogg 2000). Just as in the corresponding quantum adiabatic algorithm, his algorithm applies a multiplicative phase associated with each truth assignment, one that increases linearly with the number of violated clauses. This multiplicative phase generalizes the $(-1)^{f_P}$ phase that appears in Grover's algorithm.

Numerical results for simulations of these quantum algorithms indicate that search complexity may be reduced to $O(2^{n/6})$ (Hogg 2000), or possibly even to poly(n) (Farhi et. al. 2001; Hogg 2000) by using this extra information beyond whether a truth assignment is satisfying or not. Thus, in our context of finding stable models, we are currently investigating the use of functions (beyond the indicator function for $S = M_S$) that can be computed in polynomial time, *e.g.*, some function of $|S| - |M_S|$. We anticipate numerical results hinting at similar reduction in complexity.

Acknowledgements

This work was supported in part by the National Security Agency (NSA) and Advanced Research and Development Activity (ARDA) under Army Research Office (ARO) Grant No. DAAD19-01-1-0520.

References

L. M. Adelman, J. Demarrais and M.-D. A. Huang. Quantum computability. *SIAM J. Comput.* 26:1524–1540, 1997.

K. R. Apt. Logic programming. in *Handbook of Theoretical Computer Science* Elsevier 475–574, 1990.

K. R. Apt, H. A. Blair and A. Walker, Towards a theory of declarative knowledge. in J. Minker, ed., *Foundations of Deductive Databases and Logic Programming* Morgan Kaufmann, 89–148, 1988.

C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Develop.* 17:525-532, 1973.

C. H. Bennett, E. Bernstein, G. Brassard and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.* 26: 1510–1523, 1997.

D. Bernstein and U. Vazirani. Quantum complexity theory. in *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, San Diego, CA, 16–18 May 1993, New York ACM: 11–20, 1993 and
Quantum complexity theory. *SIAM J. Comput.* 26: 1411–1473, 1997.

G. Brassard, P. Høyer, M. Mosca and A. Tapp. Quantum amplitude amplification and estimation. `quant-ph/0005055`; in S. J. Lomonaco, Jr. and H. E. Brandt, eds., *Quantum Computation and Information*, *Contemporary Mathematics*, 305: 53–74, 2002.

M. Cadoli and L. Palipoli. Circumscribing datalog: expressive power and complexity. *Theor. Comput. Science* 193: 215–244, 1988.

P. Cholewiński, W. Marek and M. Truszczyński. Default reasoning system DeReS. in *Proceedings of KR-96* (Morgan Kaufmann) 518–528, 1998.

E. Dantsin, T. Eiter, G. Gottlob and A. Voronkov. Complexity and expressive power of logic programming. Proc. 12-th IEEE International Conf. on Computational Complexity (IEEE Computer Society Press) 82–101, 1997 and *ACM Computing Surveys*, 33(3): 374–425, 2001.

D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. Roy. Soc. London Ser. A* 400: 97–117, 1985.

D. Deutsch. Quantum computational networks. *Proc. Roy. Soc. London Ser. A* 425: 73–90, 1989.

W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Logic Programming* 3:267–284, 1984.

T. Eiter, M. Fink, G. Sabbatini and H. Tompits. A framework for declarative update specifications in logic programs. in *Proceedings of the 17th International Joint Conference on Artificial Intelligence* Seattle, WA, 4–10 August 2001 (San Francisco: Morgan Kauffman) 649–654, 2001.

T. Eiter, N. Leone, C. Mateis, G. Pfeifer and F. Scarcello. The KR system dlv: progress report, comparisons, and benchmarks. in *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning* 406–417, 1998.

E. Farhi, J. Goldstone, S. Gutmann and M. Sipser, "Quantum computation by adiabatic evolution", `quant-ph/0001106`, 2001.

R. Feynman. Simulating physics with computers. *Internat. J. Theoret. Phys.* 21: 467–488, 1982.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness* (W. H. Freeman), 1979.

M. Gelfond and V. Lifschitz. The stable semantics for logic programs. in *Proceedings of the 5th International Symposium on Logic Programming* (Cambridge, MA: MIT Press) 1070–1080, 1988.

Gottlob and Koch 2002 GottlobKoch] G. Gottlob and Ch. Koch. Monadic datalog and the expressive power of languages for web information extraction. in *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* Madison, WI, 3–6 June 2002 (New York: ACM Press) 17–28, 2002.

L. Grover. A fast quantum mechanical algorithm for database search. in *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, Philadelphia, PA, 22–24 May 1996 (New York: ACM) 212–219, 1996 and Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.* 79: 325–328, 1997.

K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. in T. Eiter, W. Faber and M. Truszczyński, eds., *Proceedings of the Sixth International Conference on Logic Programming and Nonmonotonic Reasoning* Vienna, Austria, 17–19 September 2001, Lecture Notes in Artificial Intelligence (Springer), 2173 200–212, 2001.

M. Hirvensalo, *Quantum Computing* Springer, 2001.

T. Hogg. Quantum search heuristics. *Phys. Rev. A* 61 052311/1–7, 2000.

M. Hunziker and D. A. Meyer. Quantum algorithms for highly structured search problems. *Quantum Inform. Processing* 1:145–154, 2002.

J. Jaffar and M. J. Maher. Constraint logic programming: a survey. *J. Logic Programming* 19:503–581, 1994.

V. Lifschitz. Action languages, answer sets and planning. in *The Logic Programming Paradigm*, Series Artificial Intelligence (Springer), 357–373, 1999.

V. W. Marek and J. B. Remmel. On the expressibility of stable logic programming. *Theory and Practice of Logic Programming* 3: 551–567, 2003.

W. Marek and M. Truszczyński. Autoepistemic logic. *J. ACM* 38: 588–619, 1991.

V. W. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. in *The Logic Programming Paradigm*, Series Artificial Intelligence (Springer-Verlag), 375–398, 1999.

K. Marriott and P. J. Stuckey. *Programming with Constraints: An Introduction*, MIT Press, 1998.

I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. in *Proceedings of the Workshop on Computational Aspects of Nonmonotonic Reasoning*, 72–79, 1998.

I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. in *Proceedings of JICSLP-96* MIT Press, 1996.

I. Niemelä and P. Simons. Smodels — an implementation of the stable model and well-founded semantics for normal logic programs. in *The 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, Dagstuhl, Germany, 1997, Springer Lecture Notes in Computer Science 1265, 420–429, 1997.

J. Schlipf. The expressive powers of the logic programming semantics. *J. Comput. Systems Science* 51: 64–86, 1995.

P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. in S. Goldwasser, ed., *Proceedings of the 35th Symposium on Foundations of Computer Science*, Santa Fe, NM, 20–22 November 1994 (Los Alamitos, CA: IEEE Computer Society Press 1994) 124–134;
and
Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* 26: 1484–1509, 1997.

D. Simon. On the power of quantum computation. in S. Goldwasser, ed., *Proceedings of the 35th Symposium on Foundations of Computer Science*, Santa Fe, NM, 20–22 November 1994 (Los Alamitos, CA: IEEE Computer Society Press 1994) 116–123;
and

On the power of quantum computation. *SIAM J. Comput.* **26**:1474–1483, 1997.

T. Soininen, I. Niemelä, J. Tiihonen and R. Sulonen. Representing configuration knowledge with weight constraint rules. in *Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning* papers from the AAAI Spring Symposium, Stanford, CA, 26–28 March 2001, AAAI Technical Report SS-01-01, AAAI Press, 195–201, 2001.

T. Toffoli. Bicontinuous extensions of invertible combinatorial functions. *Math. Systems Theory* 14: 13–23, 1981.

J.D. Ullman. *Principles of Database and Knowledge-Base Systems* Computer Science Press, 1988.