Finite Domain Constraint Satisfaction Using Quantum Computation

Ola Angelsmark*, Vilhelm Dahllöf**, and Peter Jonsson***

{olaan,vilda,petej}@ida.liu.se
Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden
Fax: +46/(0)13/284499

Abstract. We present a quantum algorithm for finite domain constraint solving, where the constraints have arity 2. It is complete and runs in $O((\lceil d/2 \rceil)^{n/2})$ time, where d is size of the domain of the variables and n the number of variables. For the case of $d = 3$ we provide a method to obtain an upper time bound of $O(8^{n/8}) \approx O(1.2968^n)$. Also for $d = 5$ the upper bound has been improved. Using this method in a slightly different way we can decide 3-colourability in $O(1.2185^n)$ time.

1 Introduction

Several algorithms for quantum computers have been presented recently, among them Shor's celebrated algorithm for finding prime factors of composite integers [12]. Another interesting result is the search algorithm presented by Grover [9] and improved by Boyer *et al.* [4]. Its input consists of a polynomial-time computable oracle function $f : \{0, \ldots, N-1\} \to \{0, 1\}$ and the algorithm decides whether there exists an x such that $f(x) = 1$ by only performing $O(\sqrt{N})$ evaluations of f. Obviously, any deterministic algorithm would need $O(N)$ evaluations of f to solve this problem. The algorithm has a constant probability of error, but it can easily be reduced to an arbitrarily low level by repeating the algorithm. The idea to use it for solving NP-complete problems has been proposed before – already Grover [9] discusses its possible use for solving propositional satisfiability (SAT) problems and comes up with an $O(2^{n/2})$ time algorithm. Another more elaborate example on Hamiltonian paths in graphs is given by Nielsen and Chuang [11]. We will focus on the well-known NP-complete *constraint satisfaction problem* (CSP) where we are given a set of variables that take their values from a finite domain and a set of constraints (*e.g.* relations) that constrain the values different variables can take – the question is whether the variables can

* The research is supported in part by CUGS – National Graduate School in Computer Science, Sweden.

** The research is supported by CUGS – National Graduate School in Computer Science, Sweden.

*** Partially supported by the *Swedish Research Council* (VR) under grant 221-2000-361.

be assigned values such that all constraints are satisfied. By (d, l)-CSP, we mean the CSP problem restricted to domains of size d and constraint arities less than or equal to l. We will exclusively consider *binary* problems where $l = 2$.

Now, assume that A is an $O(c^n)$-time $(c > 1)$ deterministic algorithm for some NP-complete problem L. It is obvious that A cannot immediately be combined with Grover's algorithm in order to obtain a $O(c^{n/2})$ quantum algorithm for L; after all, Grover's algorithm can only decide whether a given polynomial-time computable function $f : \{0, \ldots, N-1\} \rightarrow \{0, 1\}$ returns the value 1 for some input. In fact, it is typically as hard to construct a $O(c^{n/2})$ time quantum algorithm as constructing a $O(c^n)$ time deterministic algorithm for a given problem. Our basic idea is that given a problem instance, we construct a function f where N is 'small' and there exists an x such that $f(x) = 1$ if and only if the given instance is a member of the problem L. Consider, for example, a SAT problem over n variables $x_1, \ldots, x_n$. A straightforward proof of satisfiability is a string containing n bits where the ith bit corresponds to the value of variable x_i in some satisfying assignment. As was noted above, we can find such a string (if it exists) in $O(2^{n/2}) \approx O(1.4142^n)$ time on a quantum computer. If there was a shorter proof, say containing only $0.9n$ bits, we would immediately get a considerably faster $O(2^{0.45n}) \approx O(1.3660^n)$ time algorithm. Our algorithm for $(d, 2)$-CSP (Section 3) is a basic application of this 'short proof' approach where we use $\log_2(\lceil d/2 \rceil)$ bits for proving that the given instance is satisfiable. Our algorithm for 3-colourability (Section 5) elaborates upon this idea by adapting the proof system to the given instance; we prove that approximately $0.5702n$ bits are enough in this case.

In this paper, we show that $(d, 2)$-CSP can be solved in $O(\lceil d/2 \rceil^{n/2})$ time using quantum computation, where n is the number of variables in the instance. Furthermore, we give improved upper bounds for the cases $d = 3$ and $d = 5$ ($O(8^{n/8})$ and $O(8^{n/4})$, respectively) together with an algorithm for 3-colourability running in $O(1.2185^n)$ time. The best known deterministic algorithm for the 3-colourability problem [3] runs in $O(1.3289^n)$ so our algorithm has a considerably lower time complexity. Our results on $(d, 2)$-CSP are shown in Table 1 and compared with those found by others. Feder and Motwani [7] have a probabilistic algorithm, which runs in $O((d!)^{n/d})$ time. The algorithms given by Eppstein [6] runs in $O(1.3645^n)$ time for $d = 3$, and $O((0.4518d)^n)$ time for $d > 3$ (this algorithm is deterministic for $d \leq 4$ and probabilistic for $d > 4$). Comparing our results with those of Eppstein and Feder & Motwani, it seems that our results are close to the square-root of the old results as d grows. These algorithms are, of course, based on an entirely different computational model.

Cerf *et al.* [5] present a quantum algorithm for CSP and it runs in $O(d^{0.2080n})$ time for $(d, 2)$-CSP. Note, however, that this time complexity is not an upper bound for the worst case. Their analysis is made in the region where the number of constraints per variable is close to $d^l \cdot \log d$, *i.e.*, where the problem exhibits a *phase transition*. In Gent & Walsh [8], it is shown that many NP-complete problems have a different region where the problems are under-constrained and satisfiable but they can be orders of magnitude harder than those in the middle

	Feder & Motwani	Eppstein	Cerf *et al.*	New results
$(3,2)$-CSP	$O(1.8171^n)$	$O(1.3645^n)$	$O(1.2567^n)$	$O(1.2968^n)$
$(4,2)$-CSP	$O(2.2134^n)$	$O(1.8072^n)$	$O(1.3342^n)$	$O(1.4142^n)$
$(5,2)$-CSP	$O(2.6052^n)$	$O(2.2590^n)$	$O(1.3976^n)$	$O(1.6818^n)$
$(6,2)$-CSP	$O(2.9938^n)$	$O(2.7108^n)$	$O(1.4516^n)$	$O(1.7321^n)$
$(7,2)$-CSP	$O(3.3800^n)$	$O(3.1626^n)$	$O(1.4989^n)$	$O(2^n)$
$(8,2)$-CSP	$O(3.7644^n)$	$O(3.6144^n)$	$O(1.5411^n)$	$O(2^n)$
$(9,2)$-CSP	$O(4.1472^n)$	$O(4.0662^n)$	$O(1.5794^n)$	$O(2.2361^n)$
$(10,2)$-CSP	$O(4.5287^n)$	$O(4.5180^n)$	$O(1.6144^n)$	$O(2.2361^n)$
$(11,2)$-CSP	$O(4.9092^n)$	$O(4.9698^n)$	$O(1.6467^n)$	$O(2.4495^n)$
$(12,2)$-CSP	$O(5.2889^n)$	$O(5.4216^n)$	$O(1.6768^n)$	$O(2.4495^n)$
$(15,2)$-CSP	$O(6.4234^n)$	$O(6.7770^n)$	$O(1.7564^n)$	$O(2.8284^n)$
$(20,2)$-CSP	$O(8.3044^n)$	$O(9.0360^n)$	$O(1.8647^n)$	$O(3.1623^n)$

Table 1. Upper time bound for various algorithms for binary CSPs

of the phase transition. Also note that the very existence of phase transitions has been questioned, cf. [1]. Thus the complexity given in Cerf *et al.* cannot be directly compared with ours.

The structure of the paper is as follows: Section 2 contains some basic information on constraint satisfaction and quantum computation needed for the algorithm, which is presented and described in Section 3. Here we also find the main theorem of the paper, as well as a conjecture on the time complexity of the problem. In Section 4 we study two special cases of $(d,2)$-CSP, namely domain sizes 3 and 5, in order to corroborate our conjecture and to lay the foundations for the algorithm for deciding 3-colourability presented in Section 5.

2 Preliminaries

A *binary constraint satisfaction problem ((d,2)-*CSP*)* is a triple $\langle V, D, C\rangle$ with V a finite set of variables, D a finite domain of values, with $|D| = d$, and C a set of constraints $\{c_1, c_2, \ldots, c_q\}$. Each constraint $c_i \in C$ is a triple xRy where $x, y \in V$ and $R \subseteq D \times D$. A *solution* to a CSP instance, is a function $f : V \to D$, such that for each constraint xRy, $(f(x), f(y)) \in R$. Given a CSP instance, the computational problem is to decide whether the instance has a solution or not.

For a good introduction to quantum computation, we recommend [11]. In this paper, however, we only need to point out a few things. Given a polynomial-time computable function $f : \{0, \ldots, N-1\} \to \{0,1\}$, we can (in polynomial time) construct a polynomial size classical circuit representing the function f. This circuit can (once again, in polynomial time) be converted into a quantum circuit suitable for the quantum search algorithm as described by [9] and [4]. We will denote this search algorithm $QuantFind(A, N)$ where A is a description of the polynomial-time algorithm computing f. $QuantFind(A, N)$ will, with constant probability of error and in $O(\sqrt{N})$ time, return a number m such that $f(m) = 1$ (if such a number exists) and otherwise answer 'no'.

The algorithm presented in the next section relies on propositional logic. A *propositional variable* is a variable which represents the truth or falsehood of a proposition. A *2-SAT formula* is a sentence in conjunctive normal form consisting of the conjunction of a number of clauses, where each clause contains at most two literals. The *2-SAT problem* is to decide whether a given 2-SAT formula is satisfiable or not. The problem of finding a satisfying assignment for 2-SAT formulae is solvable in linear time [2].

3 Algorithm for $(d, 2)$-CSP

The main idea of our algorithm is to construct a polynomial-time computable function $f : \{0, \ldots, (d/2)^n - 1\} \to \{0, 1\}$ which answers 1 for some input iff there is a solution to the given CSP instance. Once f is found we employ it as an oracle for *QuantFind*. Our method builds on grouping the domain values into subsets of size 2 in order to obtain a shorter proof of satisfiability. This approach resembles Eppstein's randomized algorithm for $(d, 2)$-CSP; by dividing the domain into 4-element sets a number of $(4, 2)$-CSP instances are spawned for which he has an efficient algorithm. Since it is unlikely that k-SAT (assuming $P \neq NP$) is polynomial time solvable for $k > 2$, we are limited to subsets of size 2.

Theorem 1. *The $(d, 2)$-CSP can be solved in $O((\lceil d/2 \rceil)^{n/2})$ time, where d is the size of the domain of the variables, and n is the number of variables.*

Proof. Let $\Theta = (V, D, C)$ be a CSP instance such that $d = |D|$ is even; the case when d is odd will be considered below. The problem can be transformed to a number of instances of 2-SAT, $I_0, \ldots, I_{(d/2)^n - 1}$, such that there exists an instance I_k that has a solution iff Θ is satisfiable.

Each I_k contains $|V| \cdot d$ propositional variables x_d where $x \in V$ and $d \in D$. Our intended intepretation of these variables is that x_d is true if and only if x is assigned the value d.

The clauses are of two types: first, we have clauses for representing the constraints and these are the same for all instances I_k. Given a constraint $xRy \in C$, we represent it with a formula C_{xRy} as follows:

$$C_{xRy} = \bigwedge_{a,b \in D,\ (a,b) \notin R} (\neg x_a \vee \neg y_b).$$

For example, given the constraint $x \neq y$ where '$\neq$' $= \{(a, b) \mid a \neq b \wedge a, b \in D\}$, then the corresponding set of clauses prevents x and y from being assigned the same value.

The rest of the clauses will ensure that each variable takes exactly one value, *i.e.* that exactly one of x_a, $a \in D$, is assigned true for every $x \in V$. Assume that we want variable x to take either value a or value b; this can be guaranteed by the clauses

$$(x_a \vee x_b) \wedge (\neg x_a \vee \neg x_b) \wedge \bigwedge_{c \in D,\ c \neq a,\ c \neq b} (\neg x_c).$$

By using this construction, there are $d/2$ alternatives to consider per variable and we can construct $(d/2)^{|V|}$ instances that cover all possible assignments of the variables. If we combine these 'variable restricting' clauses with the clauses representing the constraints, it is obvious that if one of these instances is satisfiable, then Θ is satisfiable and vice-versa. It should be clear that we can enumerate these instances $I_0, \ldots, I_{(d/2)^n-1}$ in such a way that given a number m, we can in polynomial time construct I_m.

Finally, we construct a function $f : \{0, \ldots, (d/2)^n - 1\} \rightarrow \{0, 1\}$, computable in polynomial-time, such that $f(m) = 1$ if and only if I_m is satisfiable. By computing $QuantFind(f, (d/2)^n - 1)$, we can find out whether Θ is satisfiable or not in $O((d/2)^{n/2})$ time. Note that we tacitly ignore all polynomial factors in the analysis of exponential time algorithms.

If d is odd, then we add a dummy element to the domain and the algorithm will run in $O(((d+1)/2)^{n/2}))$ time. Consequently, the worst-case running time of the algorithm is in $O((\lceil d/2 \rceil)^{n/2})$. $\square$

Section 4 shows that there are cases when the upper limit on the time complexity of Q can be improved. It seems counter-intuitive that a domain with odd cardinality should be so much harder to solve than one with even cardinality. This leads us to the following plausible conjecture.

Conjecture 1 There exists a quantum algorithm which solves $(d, 2)$-CSP in time $O((d/2)^{n/2})$.

4 The special cases $|D| = 3$ and $|D| = 5$

We will now consider $(3, 2)$- and $(5, 2)$-CSP and construct algorithms faster than those presented in the previous section. Assume that $D = \{1, 2, 3\}$ and let us consider two variables at a time (instead of one at a time as in Section 3). Table 2 depicts this situation, with the variables x and y. It is easily seen that the second, fourth, and ninth columns, with boxed entries, *i.e.*, the clauses $(x_1 \vee x_2) \wedge (y_1 \vee y_3)$, $(x_1 \vee x_3) \wedge (y_1 \vee y_2)$ and $(x_2 \vee x_3) \wedge (y_2 \vee y_3)$, cover all assignments of values to the variables. Thus, by arbitrarily putting the variables in groups of two, we need only consider three different 2-SAT instances instead of four. By using ideas similar to those used in Section 3, this would yield an upper time bound of $O(1.3161^n)$.

However, we can also consider four variables at a time, the result of which is found in Proposition 1. This result will be used in Section 5 for constructing an algorithm for 3-colourability.

Proposition 1. $(3, 2)$-CSP *can be solved in* $O(8^{n/8}) \approx O(1.2968^n)$ *time.*

Proof. Assume $D = \{1, 2, 3\}$ and $V = \{p, q, r, s\}$. Analogously to the previous situation with two variables, we observe that we need the following eight cases to cover all possible assignments of values:

1. $(p = 1 \vee p = 2),\ (q = 1 \vee q = 2),\ (r = 1 \vee r = 2),\ (s = 1 \vee s = 2)$

	x_1, x_2	x_1, x_2	x_1, x_2	x_1, x_3	x_1, x_3	x_1, x_3	x_2, x_3	x_2, x_3	x_2, x_3
x y	y_1, y_2	y_1, y_3	y_2, y_3	y_1, y_2	y_1, y_3	y_2, y_3	y_1, y_2	y_1, y_3	y_2, y_3
1 1	X	X	-	X	X	-	-	-	-
1 2	X	-	X	X	-	X	-	-	-
1 3	-	X	X	-	X	X	-	-	-
2 1	X	X	-	-	-	-	X	X	-
2 2	X	-	X	-	-	-	X	-	X
2 3	-	X	X	-	-	-	-	X	X
3 1	-	-	-	X	X	-	X	X	-
3 2	-	-	-	X	-	X	X	-	X
3 3	-	-	-	-	X	X	-	X	X

Table 2. The special case of $|D| = 3$, with $D = \{1, 2, 3\}, V = \{x, y\}$. ($x_i, x_j$ is short for $x_i \vee x_j$)

2. $(p = 1 \vee p = 2),\ (q = 1 \vee q = 2),\ (r = 1 \vee r = 3),\ (s = 1 \vee s = 3)$
3. $(p = 1 \vee p = 2),\ (q = 1 \vee q = 3),\ (r = 2 \vee r = 3),\ (s = 2 \vee s = 3)$
4. $(p = 1 \vee p = 3),\ (q = 1 \vee q = 2),\ (r = 2 \vee r = 3),\ (s = 2 \vee s = 3)$
5. $(p = 1 \vee p = 3),\ (q = 1 \vee q = 3),\ (r = 1 \vee r = 2),\ (s = 1 \vee s = 2)$
6. $(p = 1 \vee p = 3),\ (q = 1 \vee q = 3),\ (r = 1 \vee r = 3),\ (s = 1 \vee s = 3)$
7. $(p = 2 \vee p = 3),\ (q = 2 \vee q = 3),\ (r = 1 \vee r = 2),\ (s = 1 \vee s = 3)$
8. $(p = 2 \vee p = 3),\ (q = 2 \vee q = 3),\ (r = 1 \vee r = 3),\ (s = 1 \vee s = 2)$

By performing a computer-assisted enumeration of all possible assignments, we found that at least 8 clauses are needed. Obviously, we can modify the function f (described in the previous section) so that it considers the variables in groups of four using the method described above. Thus we get 8 possibilities per four variables, and the call *QuantFind*$(f, 8^{n/4})$ will have a time complexity of $O(\sqrt{8^{n/4}})$. □

To further bolster Conjecture 1, we look at the case $|D| = 5$.

Proposition 2. $(5, 2)$-CSP *can be solved in* $O(8^{n/4}) \approx O(1.6818^n)$ *time*

Proof. Assume $D = \{1, 2, 3, 4, 5\}$, and $V = \{p, q\}$. The following eight cases are needed to cover all possible assignments of values, and it cannot be done with fewer:

1. $(p = 1 \vee p = 2),\ (q = 1 \vee q = 2)$
2. $(p = 1 \vee p = 2),\ (q = 1 \vee q = 3)$
3. $(p = 1 \vee p = 2),\ (q = 4 \vee q = 5)$
4. $(p = 1 \vee p = 3),\ (q = 1 \vee q = 2)$
5. $(p = 3 \vee p = 4),\ (q = 3 \vee q = 4)$
6. $(p = 3 \vee p = 5),\ (q = 3 \vee q = 5)$
7. $(p = 4 \vee p = 5),\ (q = 1 \vee q = 4)$
8. $(p = 4 \vee p = 5),\ (q = 2 \vee q = 5)$

Analogous to the proof of Proposition 1, we can change function f to reflect this new situation, and with the call *QuantFind*$(f, \sqrt{8^{n/2}})$, we get a time complexity of $O(8^{n/4}) \approx O(1.6818^n)$. □

In theory, we could examine cases with more variables and/or larger domains with the aid of a computer. However, the search space of the problem grows rapidly and it quickly becomes infeasible to enumerate the instances. For example, the case with 4 variables and domain size 5, gives 625 possible assignments. Each clause would satisfy 16 assignments, so we need at least 40 (possibly more) clauses to cover all possibilities. This implies that we need to consider at least $\binom{625}{40} \approx 2.34 \cdot 10^{63}$ possible choices of clauses, making the enumerative approach very time consuming.

5 Algorithm for 3-colourability

We will now present an $O(1.2185^n)$ time quantum algorithm for the 3-colourability problem. In order to obtain this time complexity (which is significantly lower than that for (3,2)-CSP), the algorithm begins by analysing the given graph and choose how to construct the oracle function depending on this analysis. Before we present our algorithm we need to give some more preliminaries:

An undirected *graph* G consists of a set $V(G)$ of *vertices*, and a set $E(G)$ of *edges*, where each element of $E(G)$ is an unordered pair of vertices. The *size* of a graph G, denoted $|G|$, is the number of vertices. An *independent set* S, is a set $S \subseteq V(G)$, such that for every pair $v, w \in S$, $\{v, w\} \notin E(G)$. A *3-colouring* of a graph G is a function $C : V(G) \rightarrow \{R, G, B\}$, such that, for all pairs $v, w \in V(G)$, if $C(v) = C(w)$, then $\{v, w\} \notin E(G)$; *viz.*, no adjacent vertices have the same colour. If G is a graph, and $S \subseteq V(G)$, the *subgraph of* G *induced by* S has the vertex set S and edge set $E(G|S) = \{\{u, v\} \in E(G) \mid u, v \in S\}$, and is denoted $G|S$. We write $G - S$ to denote the graph $G|(V - S)$.

A *matching* in a graph, is a set M of vertices such that each vertex $v \in M$ has an edge to one, and only one, other vertex in M. The maximum matching (wrt the number of matched vertices) of a graph is computable in polynomial time [10]. Let *Match*(G) be a function which computes a maximum matching of the graph G and returns a pair (U, M), where $U \subseteq V(G)$ contains the unmatched vertices and $M \subseteq E(G)$ the matched pairs.

We consider an arbitrary graph G with n vertices (where each vertex have an associated colour) and assume, without loss of generality, that it is connected. Our algorithm $3C$ works as follows: First we start by setting $(G_u, G_m) = Match(G)$. The algorithm then considers two cases: (1) $|G_u| < C \cdot |G|$; and (2) $|G_u| \geq C \cdot |G|$ where $C \approx 0.2806$ is the unique real solution to the equation $8^{C/4} \cdot 2^{(1-C)/2} = 3^{(1-C)/2}$ (the choice of C will be clear at the complexity analysis).

In the first case, we construct a polynomial-time computable function $a : \{0, \ldots, 8^{|G_u|/4} \cdot 2^{|G_m|/2} - 1\} \rightarrow \{0, 1\}$ such that there exists a value x such that $a(x) = 1$ if and only-if G is 3-colourable. The construction of a is described in Subsection 5.1. The second case is similar; we construct a function $b : \{0, \ldots, 3^{|G_m|/2}\} \rightarrow \{0, 1\}$ (see Subsection 5.2) with properties similar to function a. In both cases, *QuantFind* is employed to check whether the function returns 1 for some input. In Subsection 5.3, we show that the time complexity of the resulting algorithm is $O(1.2185^n)$.

5.1 Case 1: $|G_u| < C \cdot |G|$

Assume one transforms the graph into a (3,2)-CSP instance as described in Section 4 (*i.e.*, the vertices are treated as variables over the domain $\{R, G, B\}$ and an edge (x, y) corresponds to the constraint $x \neq y$). Solving this instance directly using Proposition 1 would lead to an algorithm running in $O(\sqrt{8^{|G|/4}}) = O(1.2968^{|G|})$ time. We improve this bound by treating the matched and unmatched vertices differently.

Consider a pair $x - y$ in G_m. Since x and y are connected by an edge we know that $x \neq y$ over the domain $\{R, G, B\}$. There are six allowed tuples (x, y) : $(R, G), (R, B), (G, R), (G, B), (B, R)$ and (B, G); and they are in fact covered by two cases: (1) $x = R \lor x = G$, $y = G \lor y = B$; and (2) $x = G \lor x = B$, $y = R \lor y = G$.

For the vertices in G_u, we arbitrarily group the vertices in sets of size 4. By Proposition 1, we need eight cases to cover all possibilities of such a set. It is now straightforward to construct a polynomial-time computable function $a : \{0, \ldots, 8^{|G_u|/4} \cdot 2^{|G_m|/2} - 1\} \rightarrow \{0, 1\}$ such that $a(m) = 1$ for some m iff G is 3-colourable.

5.2 Case 2: $|G_u| \geq C \cdot |G|$

We will now consider the function b (see Figure 1) that is used in the second case of $3C$. We show that $b(m) = 1$ iff G is 3-colourable.

Similarly to the vertices, we assume that each pair $p \in G_m$ have an associated colour. Let $H = G|G_m$. We define an $R\{G/B\}$ *assignment* of G as an assignment of colours to matched pairs such that any pair $x - y$ from G_m shall be assigned either $R - G/B$, $G/B - R$ or $G/B - G/B$. We say that we *extend* such an assignment to the whole of G iff for each of the vertices $v \in G_u$, we assign $v := R$ if it is not adjacent to any vertex with colour R; else we assign $v := G/B$. Observe that every vertex will have a colour after an extension since G is connected. We say that an extended R{G/B} assignment for G is *refineable* to a 3-colouring of G iff for each of the vertices v having colour G/B by the extended R{G/B} assignment, we can assign $v := G$ or $v := B$ in such a manner that we obtain a 3-colouring of G. We note that saying that we have an extended R{G/B} assignment for G which is refineable to a 3-colouring of G, is equivalent of saying that the assignment has the following properties:

P1. the vertices with colour R form an independent set;

P2. the induced subgraph of vertices with colour G/B is 2-colourable.

By the following lemma we establish that it suffices to investigate each R{G/B} assignment for H to find a 3-colouring of G.

Lemma 1. *G is 3-colourable iff there is an R{G/B} assignment for H which, after being extended to the whole of G, is refineable to a 3-colouring of G.*

Proof. One direction of the proof is trivial: if we have an R{G/B} assignment for H, extend it to the whole of G, and find that it is refineable to a 3-colouring then clearly G is 3-colourable.

The other direction: Assume $T : V(G) \to \{R, G, B\}$ is a 3-colouring of G. For each $h \in V(H)$, let h have the colour R if $T(h) = R$ and G/B otherwise; obviously, this a R{G/B} assignment of H. Extend this assignment to the whole of G. We continue to show that this R{G/B} assignment satisfies properties **P1** and **P2** and, in other words, that this assignment can be refined to a 3-colouring of G.

P1. Assume $(p, q) \in E(G)$ and both p and q have the colour R. If $p, q \in V(H)$, then $T(p) = T(q)$ which is a contradiction. If $p \in V(H)$ and $q \in G_u$ (or vice versa), then q would have been given the colour G/B in the extension of the initial assignment – a contradiction. Finally, both p and q cannot be members of G_u since G_u is an independent set.

P2. We show that if a vertex $p \in V(G)$ is assigned the colour G/B, then $T(p) \in \{G, B\}$. This proves that the induced subgraph of vertices with colour G/B is 2-colourable since T is a 3-colouring of G. Assume that p is coloured G/B and note that the claim trivially holds if $p \in V(H)$. If $p \in G_u$, then p is adjacent to a node q such that $T(q) = R$ which implies that $T(p) \in \{G, B\}$. □

Now, the idea behind function b is to carry out the tests described in Lemma 1 given a number that represents the R{G/B} assignment of the vertices of H. An enumeration of all assignments is trivial to construct and we have thus shown that there exists an m such that $b(m) = 1$ if and only if G is 3-colourable.

algorithm $b(m)$
assign colours to H according to m
extend the R{G/B} assignment of H to G
$V_R :=$ all elements of G having colour R
if V_R is not an independent set **then** return 0
$V_{G/B} :=$ all elements of G having colour G/B
if $G|V_{G/B}$ is not 2-colourable **then** return 0
else return 1

Fig. 1. The function b

5.3 Complexity analysis

We are now ready to analyse the complexity of algorithm $3C$ when applied to a graph containing n vertices. We will show that both Case 1 and 2 takes $O(1.2185^n)$ time which proves that $3C$ runs in $O(1.2185^n)$ time.

Case 1. When $|G_u| < C \cdot |G|$, the maximum number of assignment appears when $|G_u|$ is as large as possible, *i.e.*, $|G_u|$ is close to $C \cdot n$ (since $8^{C/4} \cdot 2^{(1-C)/2}$ is a strictly increasing function for $C > 0$). Thus, $8^{(C \cdot n)/4} \cdot 2^{(1-C)n/2} = 1.4847^n$ is an upper bound to the number of assignments, and the algorithm will run in $O(\sqrt{1.4847^n}) = O(1.2185^n)$ time.

Case 2. Given a graph of size $n = |G_u| + 2 \cdot |G_m|$, the worst case will arise when G_u is as small as possible, *i.e.*$|G_u| = C \cdot n$, since the number of possible R{G/B} assignments equals $3^{|G_m|} = 3^{(n-|G_u|)/2}$. So, the number of possible R{G/B} assignments is at most $3^{(1-C)n/2} \approx 1.4847^n$, and our algorithm runs in $O(1.2185^n)$ time in this case as well.

Acknowledgments

The authors wish to thank Johan Thapper for his useful comments.

References

1. D. Achlioptas, L. M. Kirousis, E. Kranakis, D. Krizanc, M. Molloy, and Y. Stamatiou. Random constraint satisfaction: A more accurate picture. *Constraints*, 6(4):329–344, 2001.
2. B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear time algorithm for testing the truth of certain quantified Boolean formulas. *Inf. Process. Lett.*, 8:121–123, 1979.
3. R. Beigel and D. Eppstein. 3-coloring in time $O(1.3289^n)$. ACM Computing Research Repository, June 2000.
4. M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. In *Proc. 4th Workshop on Physics and Computation*, pages 36–43, 1996.
5. N. J. Cerf, L. K. Grover, and C. P. Williams. Nested quantum search and NP-hard problems. *Applicable Algebra in Engineering, Communication and Computing*, 10(4/5):311–338, 2000.
6. D. Eppstein. Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In *Proc. 12th Annual Symposium on Discrete Algorithms (SODA)*, pages 329–337, 2001.
7. T. Feder and R. Motwani. Worst-case time bounds for coloring and satisfiability problems. Unpublished manuscript.
8. I. P. Gent and T. Walsh. The satisfiability constraint gap. *Artificial Intelligence*, 81(1–2):59–80, 1996.
9. L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proc. 28th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 212–219, 1996.
10. S. Micali and V. V. Vazirani. An $O(\sqrt{|v|} \cdot |e|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 10–16, IEEE Computer Society, 1980.
11. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.
12. P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.