Nested quantum search and structured problems

Nicolas J. Cerf,[1,2,3] Lov K. Grover,[4] and Colin P. Williams[2]
[1]*W. K. Kellogg Radiation Laboratory, California Institute of Technology, Pasadena, California 91125*
[2]*Information and Computing Technologies Research Section, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California 91109*
[3]*Ecole Polytechnique, Code Postale 165, Université Libre de Bruxelles, 1050 Brussels, Belgium*
[4]*3C-404A Bell Laboratories, 600 Mountain Avenue, Murray Hill, New Jersey 07974*
(Received 31 July 1998; revised manuscript received 21 October 1999; published 9 February 2000)

A quantum algorithm is known that solves an unstructured search problem in a number of iterations of order $\sqrt{d}$, where d is the dimension of the search space, whereas any classical algorithm necessarily scales as $O(d)$. It is shown here that an improved quantum search algorithm can be devised that exploits the structure of a tree search problem by *nesting* one quantum search within another. The average number of iterations required to find the solution of a typical hard instance of a constraint satisfaction problem is found to scale as $\sqrt{d^{\alpha}}$, with the constant $\alpha<1$ depending on the nesting depth and the type of problem considered. This corresponds to a square-root speedup over a classical nested search algorithm, of which our algorithm is the quantum counterpart. When applying a single nesting level to a problem with constraints of size 2 such as the graph coloring problem, α is estimated to be around 0.62.

PACS number(s): 03.67.Lx, 89.70.+c

I. INTRODUCTION

Over the past decade there has been steady progress in the development of quantum algorithms. Most attention has focused on the quantum algorithms for finding the factors of a composite integer [1,2] and for finding an item in an unsorted ''database'' [3,4]. These successes have inspired several researchers to look for quantum algorithms that can solve other challenging problems, such as decision problems [5] or combinatorial search problems [6], more efficiently than their classical counterparts.

The class of nondeterministic polynomial-time complete (NP-complete) problems includes the most common computational problems encountered in practice [7]. In particular, it includes scheduling, planning, combinatorial optimization, theorem proving, propositional satisfiability, and graph coloring. In addition to their ubiquity, NP-complete problems share a fortuitous kinship: any NP-complete problem can be mapped into any other NP-complete problem using only polynomial resources [7]. Thus any quantum algorithm that speeds up the solution of one NP-complete problem immediately leads to equally fast quantum algorithms for all NP-complete problems (up to the polynomial cost of translation). Unfortunately, NP-complete problems appear to be even harder than the integer factorization problem. Whereas, classically, the best-known algorithm for the latter problem scales only subexponentially [8], NP-complete problems are widely believed to be exponential [7]. Consequently, the demonstration that Shor's quantum algorithm [1,2] can factor an integer in a time that is bounded by a polynomial in the ''size'' of the integer (i.e., the number of bits needed to represent that integer), while remarkable, does not lead to a polynomial-time quantum algorithm for NP-complete problems, the existence of which being considered as highly improbable [9]. Moreover, it has proven to be very difficult to adapt Shor's algorithm to other computational applications.

By contrast, the unstructured quantum search algorithm [3,4] can be adapted quite readily to the service of solving NP-complete problems. As any candidate solution to an NP-complete problem can be tested for correctness in polynomial time, one simply has to embody this testing in an ''oracle'' (i.e., a black-box function that returns 1 if the candidate solution is correct, and zero otherwise), and apply the unstructured quantum search algorithm. Unfortunately, the speedup afforded by this algorithm is only $O(\sqrt{N})$, where N is the number of candidate solutions to be tested. For a typical NP-complete problem in which one has to find an assignment of one of b values to each of μ variables, the number of candidate solutions, b^{μ}, grows exponentially with μ. An unstructured classical algorithm would therefore take an average time $O(b^{\mu})$ to find the solution, whereas the unstructured quantum search algorithm would take $O(b^{\mu/2})$. Although this is an impressive speedup, one would still like to do better.

While there is now good evidence that for unstructured problems, the quantum search algorithm is optimal [9–11], these results have raised the question of whether faster quantum search algorithms might be found for problems that possess *structure* [6,12–14]. It so happens that NP-complete problems have such structure in the sense that one can often build up complete solutions (i.e., value assignments to all the variables) by extending so-called partial solutions (i.e., value assignments to a subset of the variables). This suggests that, rather than performing an unstructured quantum search among *all* candidate solutions, it may be possible to perform a quantum search among *partial* solutions in order to narrow a subsequent quantum search among their descendants. This is the approach presented in this paper, which, we believe, is applicable in all structured quantum searches.

We present an average-case complexity analysis of our algorithm, the average being taken across a *class* of problem instances, defined as a set of instances characterized by the same value of a parameter that is, roughly speaking, related to the difficulty of the problem (e.g., the average connectiv-

ity of the graph to be colored). Previous work in classical computer science has indeed identified the class of problem instances at which the hardest cases are most often encountered [15–19]. The motivation for investigating the average-case complexity of this class is that it is expected to give an estimate of the cost of a *typical* hard problem. In contrast, worst-case analyses can be misleading because they tend to focus on atypical problem instances. Similarly, naive average-case analyses can be misleading because the ensemble of problem instances over which the average is computed may contain, for example, an exceedingly large number of easy instances. Thus, the complexity analysis in terms of classes of problems that is carried out in this paper is believed to give a good tool for comparing the cost of solving typical hard problems (instead of that of an atypical maximally difficult problem) with different algorithms.

Our improved quantum search algorithm works by *nesting* one quantum search within another. Specifically, by performing a quantum search at a carefully selected level in the tree of partial solutions, we can narrow the effective quantum search among the candidate solutions so that the net computational cost is minimized. This allows us to find a solution to a constraint satisfaction problem in a time that grows, on average, as $O(b^{\alpha\mu/2})$ for hard problem instances, where $\alpha < 1$ is a constant depending on the type of problem considered. The resulting algorithm is the quantum counterpart of a *classical* nested search algorithm which scales as $O(b^{\alpha\mu})$, giving a quantum square-root speedup overall. This procedure corresponds to a *single* level of (classical or quantum) nesting, but it can be easily extended to several nesting levels. The constant α is then shown to decrease with an increasing nesting depth (i.e., an increasing number of nesting levels).

The outline of the paper is as follows. Section II introduces a simple classical tree search algorithm that exploits a problem structure to localize the search for solutions among the candidates. This is not intended to be a sophisticated tree search algorithm, but rather is aimed at providing a baseline against which our quantum algorithm can be compared. In Sec. III, we outline the standard unstructured quantum search algorithm [3,4]. We focus especially on the algorithm based on an arbitrary unitary search operator [20], as this is a key for implementing quantum nesting. Finally, Sec. IV describes the quantum tree search algorithm based on nesting, which is the quantum analog of the classical search algorithm presented in Sec. II. (The quantum search algorithm with several levels of nesting is briefly discussed in Appendix C.) We conclude by showing that the expected time to find the solution of a typical hard problem instance grows as $O(b^{\alpha\mu/2})$, that is, as the square root of the classical time. This result suggests a systematic technique for translating a nested classical search algorithm into a quantum one, giving rise a square-root speedup that can be useful to accelerate *efficient* classical search algorithms (instead of a simple exhaustive search, of no practical use). Such a square-root improvement was demonstrated independently in Ref. [21] in the special case of ''heuristics'' (i. e., probabilistic rule-of-thumb algorithms which tend to guide the search toward solutions).

II. NESTED CLASSICAL SEARCH ON STRUCTURED PROBLEMS

A. Structured search in trees

Many hard computational problems, such as propositional satisfiability, graph coloring, scheduling, planning, and combinatorial optimization, can be regarded as examples of so-called ''constraint satisfaction problems.'' Constraint satisfaction problems consist of a set of variables, each having a finite set of domain values, together with a set of logical relations (or ''constraints'') among the variables that are required to hold simultaneously. A solution is defined by a complete set of variable-value assignments such that every variable has some value, no variable is assigned conflicting values, and all the constraints are satisfied.

In such constraint satisfaction problems, there is often a degree of commonality between different nonsolutions. One typically finds, for example, that certain combinations of assignments of values to a subset of the variables are inconsistent (i.e., violate one or more of the constraints) and cannot, therefore, participate in any solution. These commonalities (several nonsolutions descending from the same inconsistent partial solution) can be exploited to focus the search for a solution. Thus, a classical *structured* search algorithm can find a solution to a constraint satisfaction problem in fewer steps than that required by a unstructured search, by avoiding regions of the search space that can be guaranteed to be devoid of solutions. Before investigating whether the problem structure can be exploited in a quantum search (see Sec. IV), we need to understand the circumstances under which knowledge of problem structure has the potential to be useful, classically. The key idea is that one can obtain complete solutions to a constraint satisfaction problem by systematically extending partial solutions (i.e., value assignment to a subset of the variables). Not all partial solutions are equally desirable however. A partial solution is ''good'' if it is consistent with all the constraints against which it may be tested. Conversely, a partial solution is ''nogood'' if it violates one or more such constraints. Sophisticated search algorithms work by incrementally extending good partial solutions and systematically terminating nogood partial solutions. This induces a natural treelike structure on the search space of partial solutions.

To give a concrete example of a tree search problem, we consider the *graph coloring problem* as depicted in Fig. 1. We have a graph that consists of μ nodes connected by e edges, with $0 \leqslant e \leqslant \mu(\mu-1)/2$. Each node must be assigned a color (out of b possible colors), so that any two nodes connected by an edge have different colors. More generally, for a constraint satisfaction problem, we are given a set of μ variables $(x_1, \ldots, x_\mu)$ which each must be assigned a value out of b possible values. This assignment must simultaneously satisfy a set of constraints, each involving k variables. In the special case of the graph coloring problem, there are e constraints of size $k=2$, since each edge imposes a constraint on the colors assigned to the pair of nodes it connects.

For each constraint, we define a *ground instance* as a specific assignment of a value to each of the variables in that

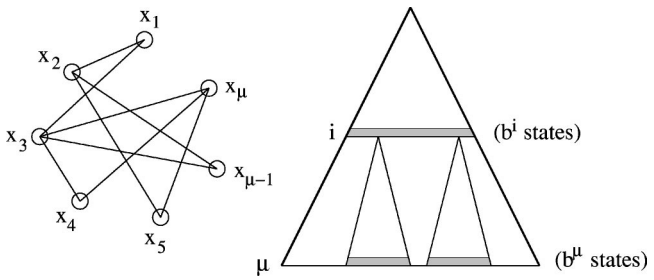

FIG. 1. Constraint satisfaction problem in which we must find an assignment to the μ variables $x_1, x_2, \ldots, x_\mu$. As an example, we picture the *graph coloring problem*, in which we have to assign one of b possible colors to each node of a graph so that every pair of nodes that are connected directly have different colors. The corresponding search tree is characterized by a depth μ and a branching ratio b. By looking at partial solutions at level i in the tree (the search space being of size b^i) and considering only the descendants at level μ of these partial solutions, one avoids having to search through the entire space at the bottom of the tree (of size b^μ).

constraint. For example, in the graph coloring problem, a ground instance corresponds to a color assignment to any two nodes connected by an edge. A ground instance is said to be *nogood* if it violates the corresponding constraint. An important variable in the following will be the number of nogood ground instances, denoted by ξ. Thus, for the graph coloring problem, the number of nogood ground instances $\xi = eb$ because each edge contributes exactly b nogood instances (for each edge, b pairs of identical colors are forbidden) and there are a total of e edges. (For other constraint satisfaction problems, ξ is in general only approximately proportional to the number of constraints because not all constraints need to have the same size.)

The search tree corresponding to this constraint satisfaction problem is also shown in Fig. 1. The ith level of the search tree enumerates all possible *partial* solutions involving a specific subset of i, out of the total μ, variables. The branching ratio in this tree, i.e., the number of children per node, is equal to b, the number of domain values of a variable. If an *unstructured* search algorithm is used to solve a constraint satisfaction problem, the number of steps required to find a good assignment at the bottom of the tree (or decide that there is no possible assignment satisfying all the constraints) scales as b^μ. Thus, a space of the order of the entire space of candidate solutions must be explored. As we will see, *structured* search algorithms work by pruning much of the search space, giving rise to a cost exponentially smaller than b^μ, though still exponential in the size of the problem μ.

In our analysis of the computational cost, we will be helped by an approximate model of tree search problems that greatly simplifies the calculation. Remarkably, many of the properties of search trees can be understood without precise knowledge of the problem. Indeed, it has been found empirically that the difficulty of solving a particular instance of a constraint satisfaction problem can be specified approximately by four parameters: the number of variables, μ, the number of values per variable, b; the number of variables per constraint, k; and the number of ground instances that are nogood, ξ [15–17]. Clearly, if ξ is small, there are generally many solutions satisfying the few constraints, so that the problem is easy to solve. Conversely, if ξ is large, the problem is in general overconstrained, and it is easy to find that it admits no solution. The problem is maximally hard in an intermediate range of values for ξ. In an effort to understand the observed variation in difficulty across different instances of NP-complete problems for fixed μ and b, it has been shown that the cost of finding a solution (or proving none exists) depends essentially on the parameter

$$\beta = \xi/\mu, \tag{1}$$

which characterizes the average number of nogood ground instances *per variable* [15,18,19]. Specifically, the problem solving difficulty exhibits a ubiquitous easy-hard-easy pattern, with the class of hard problem instances clustered around a critical value of β given, approximately, by

$$\beta_c = b^k \ln(b) \tag{2}$$

assuming here $b^k \gg 1$ for simplicity. This phenomenon, akin to a *phase transition* in physical systems [18,19], persists across many different sophisticated algorithms. (For a general reference on scaling in random graphs, see, e.g., Refs. [22,23].) The average case complexity for a fixed β (for a given class of problem instances) is therefore believed to be a more informative measure of computational complexity than either worst-case or naive average-case complexity. It is therefore the measure that we will use in the rest of this paper for estimating the scaling of the complexity of our improved quantum search algorithm (as well as the corresponding classical search algorithm).

B. Average computational complexity of a classical algorithm

Let us describe a simple classical algorithm for a tree search problem that exploits the structure of the problem by use of nesting. As pictured in Fig. 1, the key idea is to perform a preliminary search through a space of *partial* solutions in order to avoid a search through the entire space at the bottom of the tree. By definition, a partial solution at level i in the tree assigns values to a subset of i so-called *primary* variables $(x_1, \ldots, x_i)$, which we denote as A. The subset of *secondary* variables $(x_{i+1}, \ldots, x_\mu)$, denoted as B, corresponds to the variables to which we assign a value only when extending the partial solutions (i.e., when considering the descendants of the partial solutions). In general, any partial solution can be tested against a part of the constraints, namely, just those constraints involving the primary variables A. A partial solution that satisfies all these (testable) constraints can be viewed as a *could-be* solution in the sense that all solutions at the bottom of the tree (at level μ) must be descendants of could-be's. A classical search can be speeded up by terminating search along paths that are *not* descendants of a could-be, thereby avoiding to search through the entire space. The following algorithm can be used.

(i) Find a could-be solution at level i in the tree. For this purpose, repeatedly choose a random partial solution at level

i, until it satisfies the testable constraints.

(ii) For each could-be solution, check exhaustively (or by use of a random search) all its descendants at the bottom of the tree (level μ) for the presence of a possible solution.

This is clearly not a sophisticated algorithm. It amounts to nesting the search for a successful descendant at level μ into the search for a could-be solution at level *i*. Nevertheless, it does exploit the problem structure by using the knowledge gleaned from the search at level *i* to focus the search at level μ. By finding a quantum analog of this algorithm (cf. Sec. IV), we will, therefore, be able to address the impact of problem structure on quantum search.

Let us estimate the expected cost of running this algorithm. This cost consists essentially of three components: *n*, the cost of finding a consistent partial solution (a could-be) at level *i* in the tree; *m*, the cost of the subsequent search among its descendants at level μ; and *r*, the number of repetitions of this whole procedure before finding a solution. The search space for partial solutions (assignment of the primary variables *A*) is of size $d_A = b^i$. Assuming that there are n_A could-be solutions at level *i* (i.e., partial solutions that can lead to a solution), the probability of finding one of them by using a random search is thus n_A/d_A. Thus one needs of the order of

$$n \simeq d_A/n_A \tag{3}$$

iterations to find one could-be solution. The descendants of a could-be solution are obtained by assigning a value to the subset of *secondary* variables *B*, of size $d_B = b^{\mu - i}$ (each could-be solution has d_B descendants). Thus, searching through the entire space of descendants of a could-be solution requires, on average,

$$m \simeq d_B \tag{4}$$

iterations. If the problem admits a single solution, this whole procedure needs to be repeated n_A times, on average, since we have n_A could-be solutions. More generally, if the number of solutions of the problem is given by n_{AB}, this procedure must only be repeated

$$r \simeq n_A/n_{AB} \tag{5}$$

times in order to find a solution with a probability of order 1. Thus, the total number of iterations required to find a solution of an average instance is of the order of

$$T_c \simeq r(n+m) \simeq \frac{d_A + n_A d_B}{n_{AB}}. \tag{6}$$

This corresponds to an improvement over a naive *unstructured* search algorithm. Indeed, the cost of a naive algorithm that does not exploit structure is simply d_{AB}/n_{AB}, where $d_{AB} = d_A d_B = b^\mu$ is the dimension of the total search space.

The first term in the numerator of Eq. (6) corresponds to the search for could-be solutions in a space of partial solutions of size d_A (shaded area at level *i* in Fig. 1), while the second term corresponds to the search for actual solutions among all the descendants of the n_A could-be solutions, each of them having d_B descendants (shaded area at level μ in Fig. 1). The denominator in Eq. (6) accounts for a problem admitting more than one solution. We will see in Sec. IV that the quantum counterpart of Eq. (6) involves taking the square root of *n*, *m*, and *r*, which essentially results in a quantum-mechanical square-root speedup over this classical algorithm.

To make the estimate of this classical average-case complexity more quantitative, let $p(i)$ be the probability that a partial solution at level *i* is ''good'' (i.e., that it satisfies all the testable constraints). In Appendix A, we provide an approximate estimate of $p(i)$ for an average instance of a large problem ($\mu \gg 1$) with a fixed value of the parameter β. Recall that, if we want to preserve the difficulty while considering the limit of large problems, β must be kept constant. This is necessary for the complexity measure that we consider in this paper, as mentioned before. Thus, by making use of this estimate of $p(i)$, we can approximate the expected number of could-be solutions at the *i*th level, $n_A \simeq p(i) b^i$, and the expected number of solution at the bottom of the tree, $n_{AB} \simeq p(\mu) b^\mu$. Therefore, the average search time of the classical algorithm to find the first solution is approximately equal to

$$T_c(i) \simeq \frac{b^i + p(i) b^\mu}{p(\mu) b^\mu}. \tag{7}$$

This mean-field approximation of T_c is essentially the cost for finding one solution (which basically requires checking all the partial solutions for a could-be solution, and subsequently checking the descendants of all these could-be solutions) divided by the expected number of solutions at the bottom of the tree (of the order of one when $\beta = \beta_c$, i.e., for hard problems).

Equation (7) yields an approximate cost measure for our classical nested search algorithm as a function of the level of the ''cut,'' *i*. An important question now is where to cut the search tree? If one cuts the tree too high (searching for could-be solutions at small *i*), one is unlikely to learn anything useful as most partial solutions will probably be ''goods,'' allowing for little discrimination between solutions and nonsolutions. In other words, the second term in the numerator of Eq. (7) dominates since $p(i)$ is close to 1, i.e., there are many could-be solutions at level *i*. Searching for could-be solutions is thus fast (the space of primary variables is of size b^i only), but those partial solutions are of little use for singling out the actual solutions. The cost of the search among the descendants of those partial solutions is then high. Conversely, if one cuts the tree too deep (searching for could-be solutions at large *i*), although this would enhance discrimination between solutions and nonsolutions, the search space for the primary variables would be almost as large as the entire space. Then the first term dominates the scaling as the search for could-be solutions becomes time consuming. It is therefore apparent that, for a typical problem instance, there ought to be an optimal level at which to cut.

We can estimate the optimal level by finding the value of *i* that minimizes the classical computation time $T_c(i)$ for a

given value of μ, b, ξ, and k, using the functional form for $p(i)$ given in Appendix A. This is done in Appendix B, where we approximate the behavior of the location of the optimal cut level as a function of β. We then estimate the corresponding scaling of T_c for large problems ($\mu \gg 1$). The result is that the computation cost of running the classical nested search algorithm scales as

$$T_c \simeq b^{\alpha\mu} \tag{8}$$

for a search space of dimension $d=b^{\mu}$, where $\alpha<1$ is a constant depending on the problem considered. (More generally, α also depends on the number of nesting levels, but we have considered a single level of nesting in this section.) As we will see, the structured quantum search algorithm that we present in Sec. IV has a computational cost of order of $\sqrt{b^{\alpha\mu}}$, in agreement with the prevailing (but unproven) opinion that a square-root speedup is the best that can be achieved in *structured* search problems. The focus of this paper is to show explicitly *how* a quantum algorithm can be implemented that reaches this square-root speedup over the classical algorithm discussed above. Interestingly enough, the quantum complexity of our nested algorithm scales then as a power of the dimension of the search space $d=b^{\mu}$ that is less than 1/2. Structured quantum search therefore offers a significant speedup over both structured classical search and unstructured quantum search.

III. UNSTRUCTURED QUANTUM SEARCH

Let us first review the standard *unstructured* quantum search algorithm [3,4]. Consider a Hilbert space of dimension d in which each basis $|x\rangle$ state ($x=1,\ldots,d$) corresponds to a candidate solution of a search problem. Any search problem can be recast as the problem of finding the value(s) of x at which a black-box function $f(x)$, traditionally called an ''oracle,'' is equal to 1 (this function being zero elsewhere). We start the quantum search process from an arbitrary basis state $|s\rangle$, and the goal is to reach a solution (or target) basis state $|t\rangle$, with $f(t)=1$, in a shortest computation time. More precisely, if there is a single solution (or target state), the goal is to reach a state that has an amplitude of order 1 in $|t\rangle$, so that a measurement of this state gives the solution with a probability of order 1. (If there are r solutions, the goal is to reach a superposition of the states $|t\rangle$, each with an amplitude of order $r^{-1/2}$.)

The quantum search algorithm we discuss below is in fact an immediate extension of the original one [3,4], where an arbitrary unitary transformation is used instead of the Walsh-Hadamard transformation [20]. Assume that we have at our disposal a quantum circuit that performs a particular unitary operation U. If this operation connects the starting state $|s\rangle$ to the target state $|t\rangle$, i.e., $\langle t|U|s\rangle \neq 0$, then this operation can be used *classically* to find the target. Indeed, if we measure the system after applying U, the probability of obtaining the solution $|t\rangle$ is obviously $|\langle t|U|s\rangle|^2$. Thus, on average, we need to repeat this experiment $|\langle t|U|s\rangle|^{-2}$ times to find the solution with probability of order 1. We will show now that, using a quantum algorithm, it is possible to reach the target state $|t\rangle$ in a number of steps of order $|\langle t|U|s\rangle|^{-1}$ only, which represents a huge speedup provided that $|\langle t|U|s\rangle| \ll 1$ (this corresponds to the situation of interest where the search space is very large).

The idea behind a quantum search algorithm is to *postpone* the measurement, and keep a superposition of quantum states throughout the algorithm. Only at the end, a measurement is performed. Let us define the unitary operation

$$Q = -UI_sU^{\dagger}I_t = -U\,e^{i\pi P_s}\,U^{\dagger}\,e^{i\pi P_t}, \tag{9}$$

where $P_s=|s\rangle\langle s|$ and $P_t=|t\rangle\langle t|$ are projection operators on $|s\rangle$ and $|t\rangle$, respectively. The two unitary operators $I_s=\mathbb{1}-2P_s$ and $I_t=\mathbb{1}-2P_t$ perform a controlled-phase operation: applying I_s (or I_t) on a state $|x\rangle$ flips its phase if $x=s$ (or $x=t$), and leaves it unchanged otherwise. Note that the target state $|t\rangle$ is of course not available (it is what we are searching for). Instead, we have at our disposal the ''oracle'' quantum circuit that computes the function $f(x)$, and we can use it to implement the circuit for I_t: we have $I_t|x\rangle = (-1)^{f(x)}|x\rangle$ for all states $|x\rangle$. The circuit for I_s does not require the function $f(x)$ and is trivial.

The quantum search algorithm is based on iterating the operator Q starting from $U|s\rangle$, in order to *amplify* the target component $|t\rangle$. This quantum *amplitude amplification* [21] can be understood by noting that, after applying U to the starting state $|s\rangle$, the repeated applications of Q essentially rotate this state into the target state $|t\rangle$ with an angle that is *linear* in the number of iterations. More specifically, using $Q=-\mathbb{1}+2|t\rangle\langle t|+2U|s\rangle\langle s|U^{\dagger}-4U|s\rangle\langle s|U^{\dagger}|t\rangle\langle t|$, we can see that Q preserves the two-dimensional subspace spanned by $U|s\rangle$ and $|t\rangle$, namely,

$$Q\begin{pmatrix} U|s\rangle \\ |t\rangle \end{pmatrix} = \begin{pmatrix} 1-4\,|\langle t|U|s\rangle|^2 & 2\langle t|U|s\rangle \\ -2\langle t|U|s\rangle^* & 1 \end{pmatrix} \begin{pmatrix} U|s\rangle \\ |t\rangle \end{pmatrix}. \tag{10}$$

Therefore, at the limit of $|\langle t|U|s\rangle| \ll 1$, the states $U|s\rangle$ and $|t\rangle$ are almost orthogonal, and Q tends to an infinitesimal rotation matrix. (In fact, the transformation Q is always *exactly* a rotation if it is expressed in an orthogonal basis [24,25].) Indeed, keeping only the first-order terms in $u \equiv \langle t|U|s\rangle$, we obtain

$$\begin{aligned} Q\begin{pmatrix} U|s\rangle \\ |t\rangle \end{pmatrix} &\simeq \begin{pmatrix} 1 & 2u \\ -2u^* & 1 \end{pmatrix} \begin{pmatrix} U|s\rangle \\ |t\rangle \end{pmatrix} \\ &\simeq \exp\begin{pmatrix} 0 & 2u \\ -2u^* & 0 \end{pmatrix} \begin{pmatrix} U|s\rangle \\ |t\rangle \end{pmatrix}, \end{aligned} \tag{11}$$

so that Q is a rotation of angle $2u \ll 1$. We can then easily approximate Q^n in the subspace spanned by $U|s\rangle$ and $|t\rangle$:

$$Q^n\begin{pmatrix} U|s\rangle \\ |t\rangle \end{pmatrix} \simeq \exp\begin{pmatrix} 0 & 2nu \\ -2nu^* & 0 \end{pmatrix}\begin{pmatrix} U|s\rangle \\ |t\rangle \end{pmatrix} \simeq \begin{pmatrix} \cos(2n|u|) & \frac{u}{|u|}\sin(2n|u|) \\ -\frac{u^*}{|u|}\sin(2n|u|) & \cos(2n|u|) \end{pmatrix}\begin{pmatrix} U|s\rangle \\ |t\rangle \end{pmatrix}, \tag{12}$$

implying that the amplitude of the target state $|t\rangle$ after n iterations is

$$\langle t|Q^nU|s\rangle \simeq u\cos(2n|u|)+\frac{u}{|u|}\sin(2n|u|). \tag{13}$$

These last expressions are only asymptotically valid at the limit of small $|u|$. The exact expressions for Eqs. (12) and (13) in terms of Chebyshev polynomials can be found in Ref. [26].

Consider first the case of a small rotation angle. From Eq. (13), we see that if we iterate the application of Q on $U|s\rangle$, the amplitude of $|t\rangle$ grows approximately *linearly* with the number of iterations n, provided that the total angle $2n|u| \ll 1$:

$$\langle t|Q^nU|s\rangle \simeq (1+2n)\,\langle t|U|s\rangle. \tag{14}$$

Consequently, if we measure the system after n iterations, the probability $p(n)$ of finding the solution grows *quadratically* with n, as $p(n)\sim n^2|\langle t|U|s\rangle|^2$. This is a great improvement compared to the linear scaling of the classical algorithm consisting in repeating n times the measurement of $U|s\rangle$, namely, $p(n)\sim n|\langle t|U|s\rangle|^2$. This is the quadratic amplification effect provided by quantum mechanics.

Now, consider the goal of reaching the target state $|t\rangle$ using this operator Q. From Eq. (12) we see that, starting from the state $U|s\rangle$, we need to apply Q until we have rotated it by an angle of about $\pi/2$ in order to reach $|t\rangle$. At this time only, one measures the system and obtains the desired solution with a probability of order 1. The number of iterations required to rotate $U|s\rangle$ into the solution $|t\rangle$ is thus

$$n \simeq \frac{\pi}{4}\,|\langle t|U|s\rangle|^{-1}, \tag{15}$$

and scales as the *square root* of the classical time. It is worth noting that the amplitude of any state $|x\rangle$ orthogonal to the target $|t\rangle$ is given by

$$\langle x|Q^nU|s\rangle \simeq \cos(2n|u|)\,\langle x|U|s\rangle, \tag{16}$$

so that $\langle x|Q^nU|s\rangle \simeq \langle x|U|s\rangle$ for small angles. Thus the amplitude of nonsolutions is *not* amplified by applying Q repeatedly, so that the quantum search algorithm selectively amplifies the solutions only.

Thus, here we have described a general technique for achieving a quantum-mechanical square-root speedup of a search algorithm relying on *any* unitary transformation U [20]. The quantum search algorithm can be simply viewed as a rotation from $U|s\rangle$ to $|t\rangle$ based on the repeated operation of Q, followed by a measurement. In the above discussion, the search operator U can be arbitrary, *provided* it connects $|s\rangle$ and $|t\rangle$. In the case of an *unstructured* search problem, as we have no *a priori* knowledge about where the solution is located, the most natural choice for U is the Walsh-Hadamard transformation H [3,4]

$$H|x\rangle = \frac{1}{\sqrt{d}}\sum_{y=0}^{d-1}(-1)^{\overline{x}\cdot\overline{y}}|y\rangle, \tag{17}$$

where $\overline{x}\cdot\overline{y}=\sum_{i=0}^{d-1}x_iy_i(\mathrm{mod}2)$, with x_i (y_i) being the binary digits of x (y). [Here and below, we assume for simplicity that d is a power of 2. If it is not, then we need to choose for the size of the Hilbert space the nearest power of 2 that is larger than d and pad the function $f(x)$ with additional zeros.] Clearly, $U=H$ does not bias the search toward a particular candidate solution since $H|s\rangle$ has the same (squared) amplitude in all the candidate solutions, so that the search starts from a uniform distribution of all states. Applying $U=H$ to an arbitrary state in the computation basis, e.g., $|s\rangle=|00\ldots0\rangle$, we see that

$$\langle t|H|s\rangle = \pm 1/\sqrt{d} \tag{18}$$

for all possible target state $|t\rangle$. Thus, according to Eq. (15), the number of iterations in the quantum search algorithm relying on H is $O(\sqrt{d})$ [3,4], whereas a classical search algorithm obviously requires $O(d)$ steps. When there are multiple target states (the problem admits several solutions), it can be shown that the quantum computation time becomes $O(\sqrt{d/r})$, where r is the number of solutions [10]. The classical counterpart is then simply $O(d/r)$.

For a *structured* search problem, however, it is natural to use the knowledge of the structure in order to choose a better operator U or initial state $|s\rangle$. Indeed, if we have partial knowledge about where the solutions are, it seems that we can exploit it to *bias* the search in such a way that $U|s\rangle$ has larger amplitudes in states which are more probable to be solutions. It has been shown recently that an arbitrary (nonuniform) initial amplitude distribution $|s\rangle$ can be used with the standard quantum search algorithm, resulting in a $O(\sqrt{d/r})$ quantum computation time [27]. This seems to indicate, however, that the scaling remains in $O(\sqrt{d})$ even if we use our knowledge about the problem by biasing the initial distribution. In contrast, we will show in Sec. IV that the use of a *nested* quantum search algorithm results in a power law in d with an exponent that is *smaller* than 1/2. The central idea is that U is not fixed *a priori*, but is rather obtained ''dynamically'' by the quantum algorithm itself, depending on the particular instance. More specifically, the standard search algorithm is used to *construct* an effective search operator U (or a nonuniform initial distribution) which, itself, is nested within another quantum search algorithm. In other words, we apply quantum search ''recursively'': the overall operator $(-HI_sHI_t)^nH$ resulting from a nested search algorithm based on H is used as a better operator U for a quantum search at an upper level of hierarchy.

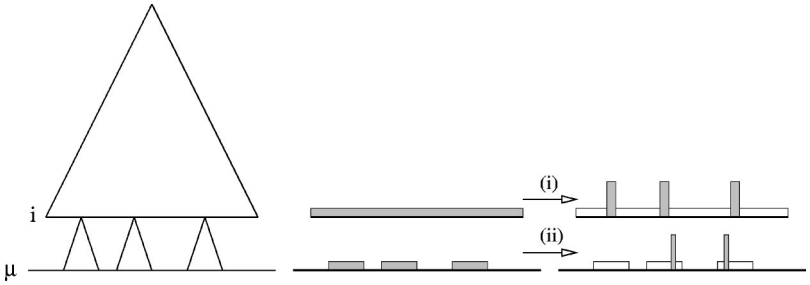

FIG. 2. Schematic representation of stages (i) and (ii) of the quantum algorithm. These stages partially amplify the solution states, and can be nested into a standard quantum search algorithm (iii) in order to speedup the amplification of the solutions.

IV. NESTED QUANTUM SEARCH ON STRUCTURED PROBLEMS

A. Core quantum algorithm

Assume that the Hilbert space of our search problem is the tensor product of two Hilbert spaces $\mathcal{H}_A$ and $\mathcal{H}_B$. As before, A denotes the set of primary variables, that is, the variables to which we assign a value in the first stage. The partial solutions correspond to definite values for these variables. Thus $\mathcal{H}_A$ represents the search space for partial solutions (of dimension d_A). The set of secondary variables, characterizing the extensions of partial solutions, is denoted by B, and the corresponding Hilbert space $\mathcal{H}_B$ is of dimension d_B. (Again, we assume here that d_A and d_B are powers of 2 to simplify the discussion.) Let us briefly describe the quantum algorithm with a single nesting level (the counterpart of the classical algorithm of Sec. II).

(i) The first stage consists of constructing a superposition (with equal amplitudes) of all the could-be solutions at level i by use of the standard unstructured search algorithm based on H.

(ii) Then one performs a subsequent quantum search in the subspace of the descendants of *all* the could-be partial solutions, simultaneously. This second stage is achieved by using the standard quantum search algorithm with, as an input, the *superposition* of could-be solutions resulting from the first stage. The overall yield of stages (i) and (ii) is a superposition of all states where the solutions have been partially amplified with respect to nonsolutions.

(iii) The final procedure consists of nesting stages (i) and (ii)—using them as a search operator U—inside a higher-level quantum search algorithm until the solutions are maximally amplified, at which point a measurement is performed. This is summarized in Fig. 2.

Let us now follow in more details the evolution of the quantum state by applying this quantum nested algorithm, and estimate the number of iterations required. The starting state of the search is denoted as $|s,s'\rangle$, where $|s\rangle$ (lying in $\mathcal{H}_A$) and $|s'\rangle$ (lying in $\mathcal{H}_B$) are just the initial state of two different parts of the same, single, quantum register which is large enough to hold all the potential solutions in the total search space (i.e., all the b^μ leaf nodes of the search tree at level μ). Register A stores the starting state at an intermediate level i in the tree, while register B stores the continuation of that state at level μ. In other words, A holds partial solutions and B their elaboration in the leaves of the tree.

(i) The first stage of the algorithm consists in a standard quantum search for *could-be* partial solutions $|c\rangle$ at level i, that is, states in subspace $\mathcal{H}_A$ that do not violate any (testable) constraint. We start from state $|s\rangle$ in subspace $\mathcal{H}_A$, and apply a quantum search based on the Walsh-Hadamard transformation H since we do not have *a priori* knowledge about the location of could-be solutions. Using

$$\langle c|H|s\rangle = \pm 1/\sqrt{d_A}, \tag{19}$$

we can perform an amplification of the components $|c\rangle$ based on $Q=-HI_sHI_c$ where

$$I_s = \exp(i\pi|s\rangle\langle s|), \tag{20}$$

$$I_c = \exp\left(i\pi \sum_{c\in C} |c\rangle\langle c|\right). \tag{21}$$

The states $|c\rangle$ correspond to the could-be partial solutions in $\mathcal{H}_A$ (assignment of the primary variables that could lead to a solution), and belong to the subset $C=\{c_1,\ldots,c_{n_A}\}$. We assume that there are n_A could-be partial solutions, with $1 \ll n_A \ll d_A$. The quadratic amplification of these could-be solutions, starting from $|s\rangle$, is reflected by

$$\langle c|Q^nH|s\rangle \simeq n\ \langle c|H|s\rangle \simeq n/\sqrt{d_A} \tag{22}$$

for a small rotation angle. Thus, applying Q sequentially, we can construct a superposition of all the could-be solutions $|c\rangle$, each with an amplitude of order $1/\sqrt{n_A}$. The required number of iterations of Q scales as

$$n \simeq \sqrt{d_A/n_A}. \tag{23}$$

This amplitude amplification process can be described equivalently in the joint Hilbert space $\mathcal{H}_A\otimes\mathcal{H}_B$, starting from the product state $|s,s'\rangle$, where $|s'\rangle$ denotes an arbitrary starting state in $\mathcal{H}_B$, and applying $(Q\otimes\mathbb{1})$ sequentially:

$$\langle c,s'|(Q\otimes\mathbb{1})^n(H\otimes\mathbb{1})|s,s'\rangle = \langle c|Q^nH|s\rangle \simeq n/\sqrt{d_A}. \tag{24}$$

Here and below, we use the convention that the left (right) term in a tensor product refers to subspace A (B).

(ii) The second stage of the algorithm is a standard quantum search for the secondary variables B in the subspace of the ''descendants'' of the could-be solutions that have been singled out in stage (i). As before, we can use the search operator H that connects extended could-be solutions $|c,s'\rangle$ to the actual solutions or target states $|t,t'\rangle$ in the joint Hilbert space:

$$\langle t,t'|(1\otimes H)|c,s'\rangle=\langle t|c\rangle\langle t'|H|s'\rangle=\pm\delta_{c,t}/\sqrt{d_B}. \tag{25}$$

Note that, this matrix element is non-vanishing only for could-be states $|c\rangle$ that lead to an actual solution. Define the operator $R=-(1\otimes HI_{s'}H)I_t$, with

$$I_{s'}=\exp(i\pi|s'\rangle\langle s'|), \tag{26}$$

$$I_t=\exp\left(i\pi\sum_{(t,t')\in T}|t,t'\rangle\langle t,t'|\right), \tag{27}$$

where T is the set of solutions $|t,t'\rangle$ at the bottom of the tree, and $\#(T)=n_{AB}$, i.e., the problem admits n_{AB} solutions. We can apply the operator R sequentially in order to amplify a target state $|t,t'\rangle$, namely,

$$\langle t,t'|R^m(1\otimes H)|c,s'\rangle\simeq\begin{cases} m\,\langle t,t'|(1\otimes H)|c,s'\rangle & \text{if } c=t \\ \langle t,t'|(1\otimes H)|c,s'\rangle & \text{if } c\neq t \end{cases} \tag{28}$$

for a small rotation angle. Note that, for a could-be state $|c\rangle$ that does not lead to a solution ($c\neq t$), we have $I_t|c,x\rangle=|c,x\rangle$ for all x, so that $R^m(1\otimes H)|c,s'\rangle=(-1\otimes HI_{s'}H)^m(1\otimes H)|c,s'\rangle=(1\otimes H)|c,s'\rangle$, and the matrix element is not amplified by m compared to the case $c=t$. In other words, no amplification occurs in the space of descendants of could-be partial solutions that do not lead to an actual solution. Thus Eq. (28) results in

$$\langle t,t'|R^m(1\otimes H)|c,s'\rangle\simeq\frac{m}{\sqrt{d_B}}\delta_{c,t}. \tag{29}$$

Assuming that, among the descendants of each could-be solution $|c,s'\rangle$, there is either zero or one solution, we need to iterate R of the order of

$$m\simeq\sqrt{d_B} \tag{30}$$

times in order to maximally amplify each solution. We then obtain a superposition of the solution states $|t,t'\rangle$, each with an amplitude of order $1/\sqrt{n_A}$. This can also be seen by combining Eqs. (24) and (29), and using the resolution of identity $1=\Sigma_{x,y}|x,y\rangle\langle x,y|$:

$$\begin{aligned}\langle t,t'|\underbrace{R^m(\mathbb{1}\otimes H)(Q\otimes\mathbb{1})^n(H\otimes\mathbb{1})}_{U}|s,s'\rangle &= \sum_{x,y}\langle t,t'|R^m(\mathbb{1}\otimes H)|x,y\rangle\langle x,y|(Q\otimes\mathbb{1})^n(H\otimes\mathbb{1})|s,s'\rangle \\ &= \langle t,t'|R^m(\mathbb{1}\otimes H)|t,s'\rangle\langle t,s'|(Q\otimes\mathbb{1})^n(H\otimes\mathbb{1})|s,s'\rangle \\ &\simeq (m/\sqrt{d_B})(n/\sqrt{d_A}) \\ &\simeq 1/\sqrt{n_A}\ .\end{aligned} \tag{31}$$

Thus, applying the operator Q^n followed by the operator R^m connects the starting state $|s,s'\rangle$ to each of the solutions $|t,t'\rangle$ of the problem with a matrix element of order $1/\sqrt{n_A}$.

(iii) The third stage consists of using the operator $U\equiv R^m(1\otimes H)(Q\otimes 1)^n(H\otimes 1)$ resulting from steps (i) and (ii) as a search operator for a higher-level quantum search algorithm, in order to further amplify the superposition of n_{AB} target (or solution) states $|t,t'\rangle$. The goal is thus to construct such a superposition where each solution has an amplitude of order $1/\sqrt{n_{AB}}$. As before, we can make use of the operator $S=-U(I_s\otimes I_{s'})U^\dagger I_t$ where I_s, $I_{s'}$, and I_t are defined in Eqs. (20), (26), and (27), in order to perform amplification according to the relation

$$\langle t,t'|S^rU|s,s'\rangle\simeq r\,\langle t,t'|U|s,s'\rangle\simeq r/\sqrt{n_A} \tag{32}$$

for a small rotation angle. The number of iterations of S required to maximally amplify the solutions is thus of the order of

$$r\simeq\sqrt{\frac{n_A}{n_{AB}}}. \tag{33}$$

This completes the algorithm. At this point, it is sufficient to perform a measurement of the amplified superposition of solutions. This yields one solution $|t,t'\rangle$ with a probability of order 1.

In Fig. 3, the quantum network that implements this nested quantum search algorithm is illustrated. Clearly, a sequence of two quantum search circuits (a search in the A space followed by a search in the B space) is *nested* into a global search circuit in the whole Hilbert space $\mathcal{H}_{AB}$. This can be interpreted as a ''dynamical'' choice of the search operator U that is used in the global quantum search. This quantum nesting is distinct from a procedure where one would try to choose an optimum U before running the quantum search by making use of the structure *classically* (i.e., by making several classical queries to the oracle in order to speedup the resulting quantum search). Here no measurement is involved, and structure is used at the quantum level.

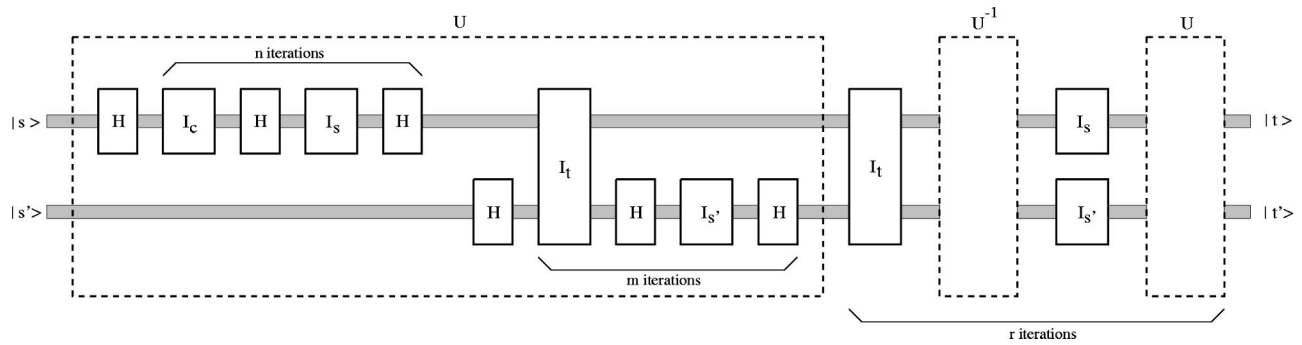

FIG. 3. Circuit implementing the nested quantum search algorithm (with a single level of nesting). The upper set of quantum variables, initially in state $|s\rangle$, corresponds to the primary variables *A*. The lower set of quantum variables, initially in states $|s'\rangle$, is associated with the secondary variables *B*. The quantum circuit makes use of controlled-phase gates $I_s=\exp(i\pi|s\rangle\langle s|)$, $I_{s'}=\exp(i\pi|s'\rangle\langle s'|)$, $I_c=\exp(i\pi\Sigma_{c\in C}|c\rangle\langle c|)$, and $I_t=\exp(i\pi\Sigma_{(t,t')\in T}|t,t'\rangle\langle t,t'|)$, and Walsh-Hadamard gates *H*. The entire operation of *U* (exhibited inside the dashed box) is repeated *r* times. Note that $U^{-1}=U^{\dagger}$ corresponds to same the circuit as *U* but read from right to left.

B. Quantum average-case complexity

Let us estimate the total number of iterations, or more precisely the number of times that a controlled-phase operator (I_t, which flips the phase of a solution, or I_c, which flips the phase of a could-be partial solution) is used. Since we need to repeat *r* times the operation *S*, which itself requires applying *n* times *Q* and *m* times *R*, for the quantum computation time we obtain

$$T_q \simeq r(n+m) \simeq \frac{\sqrt{d_A}+\sqrt{n_A d_B}}{\sqrt{n_{AB}}}. \tag{34}$$

This expression is the quantum counterpart of Eq. (6), and has the following interpretation. The first term in the numerator corresponds to a quantum search for the could-be partial solutions in space of size d_A. The second term is associated with a quantum search of actual solutions in the space of all the descendants of the n_A could-be solutions (each of them has a subspace of descendants of size d_B). The denominator accounts for the fact that the total number of iterations decreases with the square root of the number of solutions of the problem n_{AB}, as in the standard quantum search algorithm.

Let us now estimate the scaling of the computation time required by this quantum nested algorithm for a large search space. Remember that $\mu \gg 1$ is the number of variables (number of nodes for the graph coloring problem), and *b* is the number of values (colors) per variable. As before, if we ''cut'' the tree at level *i* (i.e., assigning a value to *i* variables out of μ defines a partial solution), we have $d_A=b^i$ and $d_B=b^{\mu-i}$. Also, we have $n_A \simeq p(i)b^i$, and $n_{AB} \simeq p(\mu)b^{\mu}$, where $p(i)$ is the probability of having a partial solution at level *i* that is ''good'' in a tree of height μ. [The quantity $p(\mu)$ is thus the probability of having a solution in the total search space.] We can reexpress the computation time as a function of *i*,

$$T_q(i) \simeq \frac{\sqrt{b^i}+\sqrt{p(i)b^{\mu}}}{\sqrt{p(\mu)b^{\mu}}}, \tag{35}$$

which is the quantum counterpart of Eq. (7). In order to determine the scaling of T_q, we use the estimate of $p(i)$ that is derived in Appendix A, namely,

$$p(i) \simeq b^{-\mu(\beta/\beta_c)(i/\mu)^k}, \tag{36}$$

where $\beta_c=b^k \ln(b)$, and *k* is the size of the constraint (i.e., number of variables involved in a constraint). Equation (36) is a good approximation of $p(i)$ when the dimension of the problem (or the number of variables μ) is large. Remember that, in order to keep the difficulty constant when increasing the size of the problem, we need to choose the number of constraints $\xi=\beta\mu$ for increasing μ. For the graph coloring problem, since $\xi=eb$ (where *e* being the number of edges and *b* the number of colors), this simply implies that the number of edges must grow linearly with the number of nodes for a fixed number of colors in order to preserve the difficulty, or, in other words, that the average connectivity must remain constant. In general, the constant β corresponds roughly to the average number of constraints *per variable*, and is a measure of the difficulty of the problem.

To understand this, note that $p(\mu) \simeq b^{-\mu(\beta/\beta_c)}$, so that the number of solutions at the bottom of the tree is $n(\mu) \simeq b^{\mu(1-\beta/\beta_c)}$. This implies that, if $\beta \simeq \beta_c$, we have $p(\mu) \simeq b^{-\mu}$, so that the problem admits of the order of one solution. It is therefore reasonable to consider that the hardest problems are found in the region where β is close to the critical value β_c, where one is searching for a single solution in the entire search space. (This is not rigorously true [18], but is a good enough characterization of hard problems for the purpose of this paper.) When $\beta<\beta_c$, however, there are less constraints and the problem admits more than one solution, on average. If $\beta>\beta_c$, the problem is overconstrained, and it typically becomes easier to check the nonexistence of a solution. Thus, in both cases, the difficulty of the average problem instances is lower.

Now, plugging Eq. (36) into Eq. (35), for the quantum computation time we obtain

$$T_q(i) \simeq \frac{\sqrt{b^i}+\sqrt{b^{\mu-\mu(\beta/\beta_c)(i/\mu)^k}}}{\sqrt{b^{\mu-\mu(\beta/\beta_c)}}}. \tag{37}$$

Defining the *reduced* level on the tree as $x=i/\mu$, i.e., the fraction of the height of the tree at which we exploit the structure of the problem, we have

$$T_q(x) \simeq \frac{a^x + a^{1-(\beta/\beta_c)x^k}}{a^{1-\beta/\beta_c}}, \tag{38}$$

where $a \equiv \sqrt{b^\mu}$. Now, we want to find the value of x that minimizes the computation time $T_q(x)$, so we have to solve

$$(\beta/\beta_c)\, k x^{k-1} = a^{(\beta/\beta_c)x^k + x - 1}. \tag{39}$$

For large μ (or large a), this equation approximately reduces to

$$(\beta/\beta_c)\, x^k + x - 1 = 0. \tag{40}$$

The solution x (with $0 \leqslant x \leqslant 1$) therefore corresponds to the reduced level for which $T_q(x)$ grows asymptotically ($\mu \to \infty$) with the smallest power in b. Note that this optimum x is such that both terms in the numerator of Eq. (37) grow with the same power in b (for large μ). This reflects that there is a particular fraction x of the height of the tree where it is optimal to ''cut,'' i.e., to look at partial solutions. The optimum computation time can then be written as

$$T_q \simeq \frac{2a^\alpha}{a^{1-\beta/\beta_c}} \simeq \frac{\sqrt{b^{\alpha\mu}}}{\sqrt{b^{\mu(1-\beta/\beta_c)}}}, \tag{41}$$

where the constant $\alpha < 1$ is defined as the solution x of Eq. (40). (We may ignore the prefactor 2 as it only yields an additive constant in the logarithm of the computation time.) Note that, for a search with several nesting levels, the constant $\alpha < x$, as shown in Appendix C. In fact, α can be shown to decrease with an increasing nesting depth (i.e., an increasing number of nesting levels).

Equation (41) implies that the scaling of the quantum search in a space of dimension $d = b^\mu$ is essentially $O(d^{\alpha/2})$ modulo the denominator (which simply accounts for the number of solutions). In contrast, the standard *unstructured* quantum search algorithm applied to this problem corresponds to $\alpha = x = 1$, with a computation time scaling as $T_q(\alpha = 1) = O(d^{1/2})$. This means that exploiting the structure in the quantum algorithm results in a decrease of the power in b by a coefficient α: the power 1/2 of the standard quantum search is reduced to $\alpha/2$ for this nested quantum search algorithm. Consider this result at $\beta = \beta_c$, i.e., when the difficulty of the problem is maximum for a given size μ. Then the nested algorithm essentially scales as

$$T_q \simeq d^{\alpha/2} = \sqrt{b^{\alpha\mu}}, \tag{42}$$

where $\alpha = x < 1$ with x being the solution of $x^k + x - 1 = 0$, and $d = b^\mu$ is the dimension of the search space. This represents a significant improvement over the scaling of the unstructured quantum search algorithm, $O(d^{1/2})$. Nevertheless, it must be emphasized that the speedup with respect to the computation time $O(d^\alpha)$ of the classical nested algorithm presented in Sec. II is exactly a square root (cf. Appendix B). This shows that our nested quantum search algorithm is the precise counterpart of this particular classical nondeterministic algorithm.

Consider the regime where $\beta < \beta_c$, i.e., there are fewer constraints and therefore more than one solution on average, so that the problem becomes easier to solve. For a given k, the solution x of Eq. (40) increases when β decreases, and tends to 1 for $\beta \to 0$. This means that we recover the *unstructured* quantum search algorithm in the limit where $\beta \to 0$. The denominator in Eq. (41) increases, and it is easy to check that the computation time

$$T_q \simeq \sqrt{b^{\mu(\alpha - 1 + \beta/\beta_c)}} \tag{43}$$

decreases when β decreases. As expected, the computation time of the nested algorithm approaches $O(\sqrt{d^{\beta/\beta_c}})$ as β tends to 0 (or $x \to 1$); that is, it reduces to the time of the standard unstructured quantum search algorithm at the limit $\beta \to 0$.

As an illustration of the scaling of our quantum algorithm, consider an average hard instance ($\beta = \beta_c$) of the graph coloring problem ($k = 2$). We must solve the linear equation of second order $x^2 + x - 1 = 0$, which yields $x = (-1 + \sqrt{5})/2 = 0.6180$. (When $k > 2$, the solution for x increases, and tends to 1 for large k.) This means that the level on the tree where it is optimal to use the structure is at about 62% of the total height of the tree, i.e., when assigning values to about 62% of the μ variables. In this case, the computation time of the quantum nested search algorithm scales as $O(d^{0.31})$, which is clearly an important computational gain compared to $O(d^{0.5})$.

Finally, it is worth comparing the scaling of our quantum algorithm with that of the best-known classical algorithm. For this comparison, here we consider another constraint satisfaction problem, the satisfiability problem of Boolean formulas in conjunctive normal form, or k-SAT problem. In this problem, one has to decide whether a given formula made of k clauses is satisfiable. (This problem is known to be NP-complete for $k \geqslant 3$.) The best known classical algorithm for 3-SAT has a worst-case running time that scales as $O(2^{0.446\mu})$ [28]. Applying our quantum nested search algorithm to it ($b = 2$, $k = 3$), we obtain $\alpha = x = 0.68$, so that the expected computation time scales as $O(2^{0.34\mu})$ for hard problem instances. This is compatible with our algorithm being better than (or comparable to) this best classical algorithm.

V. CONCLUSION

There is considerable interest in the possibility of using quantum computers to speedup the solution of NP-complete problems given the importance of these problems in complexity theory and their ubiquity among practical computational applications. This paper presents an attempt in this direction by showing that nesting the standard quantum search algorithm results in a faster quantum algorithm for structured search problems such as the constraint satisfaction problem than heretofore known. The key innovation is to cast the construction of solutions of the problem as a quan-

tum search through a tree of partial solutions, which narrows a subsequent quantum search at the next level in the search tree. The corresponding computation time scales exponentially with the problem size, but with a reduced coefficient that depends on the number of nesting levels and on the problem. The speedup that is achieved is a *square root* over the computation time of a corresponding classical nested search algorithm, therefore which represents the appropriate benchmark. Nevertheless, it is an *exponential* improvement with respect to the time needed to solve the problem by use of the standard unstructured quantum search algorithm. Moreover, for the 3-SAT problem, the computation time of the nested quantum search algorithm scales comparably to the worst-case running time of the best-known classical algorithm today. More generally, our result opens the possibility that a square-root quantum improvement could be achieved for any classical search method.

ACKNOWLEDGMENTS

N.J.C. was supported in part by the NSF under Grant Nos. PHY 94-12818 and PHY 94-20470, and by a grant from DARPA/ARO through the QUIC Program (No. DAAH04-96-1-3086). C.P.W. was supported by the NASA/JPL Center for Integrated Space Microsystems (Grant No. 100306-3R0U0), UPN-632 Program (Grant No. 100356-8AX24), and NASA Advanced Concepts (Grant No. 233-0NM71-0). This work was also supported by the Caltech President's Fund.

APPENDIX A: PROBABILITY OF A NODE IN A SEARCH TREE TO BE GOOD

Let us derive an approximate functional form for $p(i)$, the probability that a node at level i in the search tree is ''good.'' The derivation is complicated by the fact that the same problem instance can be easy or hard depending on the *order* in which the variables are assigned values. This is because it is possible that the constraints are such that a particular variable can only take one possible value. If this variable is examined early in the search process, the recognition that the value is highly constrained would permit a large fraction of the search space to be avoided. Conversely, if this variable is examined late in the search process, much of the tree might already have been developed, resulting in relatively little gain. However, the algorithm described in Sec. II is a naive algorithm that does *not* optimize the order in which the variables are assigned values. Thus we can compute the probability $p(i)$ for an average tree having a *random* variable ordering.

The simplest way to do this is to consider a *lattice* of partial solutions rather than a *tree* of partial solutions, because a lattice of partial solutions effectively encodes all possible variable orderings. In particular, the ith level of a lattice of partial solutions represents all possible subsets of i variables out of μ variables, assigned values in all possible combinations. Thus in a lattice there are $\binom{\mu}{i}b^i$ nodes at level i rather than the b^i nodes in a tree. So each level of the lattice encodes the information contained in $\binom{\mu}{i}$ different trees. As each constraint involves exactly k variables, and each variable can be assigned any one of its b allowed values, there are exactly b^k ''ground instances'' of each constraint. Moreover, as each constraint involves a different combination of k out of a possible μ variables, there can be at most $\binom{\mu}{k}$ constraints. Each ground instance of a constraint may be ''good'' or ''nogood,'' so the number of ground instances that are nogood, ξ, must be such that $0 \leqslant \xi \leqslant b^k\binom{\mu}{k}$. If ξ is small the problem typically has many solutions. If ξ is large the problem typically has few, or perhaps no, solutions. The exact placement of the ξ nogood ground instances is, of course, important in determining their ultimate pruning power.

Thus to estimate $p(i)$ in an *average* tree, we calculate the corresponding probability that a node in the lattice (which implicitly incorporates *all* trees) is nogood, conditional on there being ξ nogood nodes at level k. For a node at level i of the lattice to be good it must not sit above any of the ξ nogood nodes at level k. A node at level i of the lattice sits above $\binom{i}{k}$ nodes at level k. Thus, out of a total possible pool of $b^k\binom{\mu}{k}$ nodes at level k, we must exclude $\binom{i}{k}$ of them. However, we can pick the ξ nogood nodes from amongst the remaining nodes in any way whatsoever. Hence the probability that a node is good at level i, given that there are ξ nogood nodes at level k, is given by the ratio of the number of ways to pick the nogood nodes such that a particular node at level i is good, to the total number of ways of picking the ξ nogood nodes. As a consequence, the probability for a partial solution to be good at level i in a tree of height μ, and branching ratio b can be approximated as [16,17,19]

$$p(i) = \frac{\binom{b^k\binom{\mu}{k} - \binom{i}{k}}{\xi}}{\binom{b^k\binom{\mu}{k}}{\xi}}, \tag{A1}$$

where k is the size of the constraint (i.e., the number of variables involved in a constraint), and ξ is the number of nogood ground instances (or number of constraints). This approximation essentially relies on the assumption that the nogood ground instances at level k prune independently, so that the nogood partial solutions at level i are uncorrelated. In reality, the structure of the partial solution lattice implies that there are correlations among the nodes pruned by a given set of nogood nodes at level k.

Now we are interested in obtaining an approximate expression for $p(i)$ for large problems, i.e., when the number of variables μ is large. Recall that to scale a constraint satisfaction problem up, however, it is not sufficient to increase only μ. In addition, we also ought to increase the number of constraints so as to preserve the ''constrainedness per variable'' $\beta = \xi/\mu$. Thus, when we consider scaling our problems up, as we must do to assess the asymptotic behavior of the classical and quantum structured search algorithms, we have $\mu \to \infty$ and scale $\xi = \beta\mu$, keeping β, b and k constant. For graph coloring, this scaling corresponds to adding more edges to the graph as we allow the number of nodes to increase, while simultaneously keeping the average connectivity (number of edges per node) and the number of colors

fixed. We now make the assumption that $\xi \ll b^k \binom{\mu}{k}$ and $\xi \ll b^k \binom{\mu}{k} - \binom{i}{k}$, which is justified in the asymptotic regime. Using the Stirling formula, we have

$$\frac{\binom{M}{K}}{\binom{N}{K}} \simeq \frac{(M-K)^K}{(N-K)^K} \simeq \left(\frac{M}{N}\right)^K \tag{A2}$$

for large M and N, provided that $K \ll M,N$. This allows us to reexpress Eq. (A1) as

$$p(i) = \left(1 - b^{-k} \frac{\binom{i}{k}}{\binom{\mu}{k}}\right)^{\xi}. \tag{A3}$$

Now, assuming that $k \ll i$ and $k \ll \mu$, and reusing Eq. (A2), we have

$$p(i) = \left(1 - b^{-k} \left(\frac{i}{\mu}\right)^k\right)^{\xi} \tag{A4}$$

for large i and μ. Finally, assuming for simplicity that $b^k \gg 1$ and $(i/\mu)^k \ll 1$, we obtain

$$p(i) = b^{-\mu(\beta/\beta_c)\,(i/\mu)^k}, \tag{A5}$$

where the parameter $\beta = \xi/\mu$ measures the difficulty of the problem and $\beta_c = b^k \ln(b)$ is the critical value of this parameter.

Note that for $\beta = \beta_c$, we have $p(\mu) = b^{-\mu}$, that is, the problem admits a single solution on average. (This is actually how the critical point β_c is defined.) It is therefore reasonable to consider that β_c approximately characterizes the region of hard problems, as observed empirically.

APPENDIX B: AVERAGE-CASE COMPLEXITY OF THE CLASSICAL SEARCH

Plugging Eq. (A5) into Eq. (7), we obtain an approximate expression of the classical computation time needed to solve an average instance with fixed β,

$$T_c(i) \simeq \frac{b^i + b^{\mu - \mu(\beta/\beta_c)(i/\mu)^k}}{b^{\mu - \mu(\beta/\beta_c)}}, \tag{B1}$$

where the denominator is simply the expected number of solutions. Let us now find the level i where it is optimum to ''cut'' the tree. The value of i which minimizes $T_c(i)$ corresponds, for large μ, to the situation where both terms in the numerator grow with the same power of b, i.e., the solution of the equation $i = \mu - \mu(\beta/\beta_c)(i/\mu)^k$. Then one can show that the computation time approximately scales as

$$T_c \simeq \frac{2 b^{\alpha\mu}}{b^{\mu - \mu(\beta/\beta_c)}}, \tag{B2}$$

where the scaling coefficient $\alpha = x$ with $x = i/\mu$, the fraction of the height at which one cuts the tree, being the solution of Eq. (40) such that $0 \leqslant x \leqslant 1$. For hard problems, i.e., problems which admit a single solution on average ($\beta = \beta_c$), the classical time thus scales as

$$T_c \simeq \frac{2 d^{\alpha}}{d^{1 - \beta/\beta_c}} = O(d^{\alpha}) \tag{B3}$$

for a search space of dimension $d = b^{\mu}$. This represents a significant improvement over a classical search that does not exploit the structure, i.e., $T_c = O(d)$.

APPENDIX C: QUANTUM SEARCH WITH SEVERAL LEVELS OF NESTING

The quantum algorithm described in Sec. IV A relies on a single level of nesting. Indeed, the search at the bottom of the tree (level μ) is speeded up by making use of a search at level i which determines the partial solutions which are ''good.'' Only the candidate solutions which are descendants of these partial solutions are examined in the search at level μ. It should be realized that these good partial solutions at level i are selected, themselves, by a *naive* search: stage (i) indeed amounts to use the standard unstructured search based on H. In the corresponding classical nested algorithm, this amounts to select a random partial solution at level i and check whether it is good.

It is natural that both the classical and the quantum algorithms could be improved further if the search for good partial solutions at level i itself was made faster by making use of the structure of the upper part of the tree (by examining partial solutions at level j, with $j < i$, and considering only the descendants of the good ones). This leads to the notion of a search with several levels of nesting (i.e., a nesting depth larger than 1).

In order to analyze the scaling achieved by several levels of nesting, let us consider a search at level i which corresponds to the nth nesting level. We suppose that this search relies itself on a search at level j, where $j < i < \mu$, which corresponds therefore to the $(n+1)$th nesting level. Let $i = x_n \mu$ and $j = x_{n+1} \mu$, where x_n and x_{n+1} denote the reduced level on the tree at the nth and $(n+1)$th nesting level, respectively. Assume that the quantum computation cost at level j is given by

$$t(j) \simeq \frac{\sqrt{b^{\alpha_{n+1} j}}}{\sqrt{p(j) b^j}}, \tag{C1}$$

where α_{n+1} is the scaling coefficient at the $(n+1)$th level of nesting (level j in the tree). Using the structure at level j, the quantum computation cost at level i can be written as

$$t(i) \simeq \frac{\sqrt{p(j) b^j}\,[t(j) + \sqrt{b^{i-j}}]}{\sqrt{p(i) b^i}},$$

$$= \frac{\sqrt{b^{\alpha_{n+1} j}} + \sqrt{p(j) b^i}}{\sqrt{p(i) b^i}}. \tag{C2}$$

By optimizing j so that $t(i)$ is minimum, as before, we obtain $j=x_{n+1}\mu$, where x_{n+1} is a solution of

$$(\beta/\beta_c)\,x_{n+1}^{k}+\alpha_{n+1}\,x_{n+1}-x_n=0, \tag{C3}$$

with $0\leqslant x_{n+1}\leqslant 1$. Defining the scaling coefficient α_n by

$$\alpha_n x_n=\alpha_{n+1}x_{n+1}, \tag{C4}$$

we see that the corresponding computation cost at level i is given by

$$t(i)\simeq\frac{\sqrt{b^{\alpha_n i}}}{\sqrt{p(i)b^{i}}}. \tag{C5}$$

Thus, to determine the cost of the global algorithm, we need to solve the set of recurrence equations (C3) and (C4) for $n=0,1,\ldots,N-1$, where N is the nesting depth ($N=1$ corresponds to the algorithm described in Sec. IV A). The boundary conditions are $x_0=1$ (the upper level is a search for solutions at the bottom of the tree, i.e., at level μ) and $\alpha_N=1$ (the innermost search at the Nth level of nesting is supposed to be a naive search). These two conditions, together with the $2N$ recurrence relations, uniquely determine the variables $(x_0,x_1,\ldots x_N)$ and $(\alpha_0,\alpha_1,\ldots,\alpha_N)$. The overall scaling of the quantum search algorithm is $O(\sqrt{b^{\alpha_0\mu}})$, i.e., it is governed by α_0 (the constant that was denoted as α in Sec. IV B). Note that this entire calculation is also valid for a classical nested search with several levels of nesting, except for the square root. Thus the speedup of the multinested quantum search algorithm remains a square root if compared with the corresponding multinested classical search algorithm.

TABLE I. Reduced level x_n on the tree and corresponding scaling coefficient α_n at the nth level of nesting for the graph coloring problem ($k=2$) at $\beta=\beta_c$. The variable N denotes the nesting depth, and α_0 governs the scaling of the overall quantum (or classical) algorithm.

N	x_0	α_0	x_1	α_1	x_2	α_2	x_3	α_3
1	1.000	0.618	0.618	1.000	-	-	-	-
2	1.000	0.484	0.718	0.674	0.484	1.000	-	-
3	1.000	0.416	0.764	0.545	0.590	0.706	0.416	1.000

In Table I we show the values of the x_n's and α_n's for an average instance of maximum difficulty ($\beta=\beta_c$) of the graph coloring problem ($k=2$). The scaling coefficient α_0 decreases with an increasing nesting depth N, implying that the speedup over an unstructured search improves by adding further nesting levels. It should be emphasized, however, that the formalism used to estimate the scaling throughout this paper cannot be used for a large nesting depth N. Indeed, the derivation of $p(i)$ essentially neglects the correlations between partial solutions at any level in the tree which arise because of their sharing a same ancestor. Thus our cost estimate for the multinested algorithm is only valid provided that $N\ll\mu$ (the fact that $\alpha_0\to 0$ when $N\to\infty$ is meaningless).

[1] P. W. Shor, in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, edited by S. Goldwasser (IEEE Computer Society Press, New York, 1994), pp. 124–134.

[2] P. W. Shor, SIAM J. Comput. **26**, 1484 (1997).

[3] L. K. Grover, in *Proceedings of the 28th Annual Symposium on the Theory of Computing* (ACM Press, New York, 1996), pp. 212–219.

[4] L. K. Grover, Phys. Rev. Lett. **79**, 325 (1997).

[5] E. Farhi and S. Gutmann, Phys. Rev. A **58**, 915 (1998).

[6] T. Hogg, Phys. Rev. Lett. **80**, 2473 (1998).

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness* (Freeman, San Francisco, 1979).

[8] A. Lenstra and H. Lenstra, *The Development of the Number Field Sieve*, Lecture Notes in Mathematics Vol. 12, **1554** (Springer-Verlag, New York, 1993).

[9] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, SIAM J. Comput. **26**, 1510 (1997); e-print quant-ph/9701001.

[10] M. Boyer, G. Brassard, P. Hoyer, and A. Tapp, in *Proceedings of the 4th Workshop on Physics and Computation*, edited by T. Toffoli, M. Biafore, and J. Leao (New England Complex Systems Institute, Boston, 1996), p. 36; e-print quant-ph/9605034.

[11] C. Zalka, e-print quant-ph/9711070.

[12] T. Hogg, Physica D **120**, 102 (1998).

[13] E. Farhi and S. Gutmann, e-print quant-ph/9711035.

[14] L. K. Grover, in *Quantum Computing and Quantum Communications*, *Lecture Notes in Computer Science*, edited by C. P. Williams (Springer-Verlag, Berlin, 1999), Vol. 1509, pp. 126–139; also in Los Alamos e-print, quant-ph/9802035.

[15] P. Cheeseman, B. Kanefsky, and W. M. Taylor, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'91), Sydney* (Kauffman, 1991), pp. 331–337.

[16] C. P. Williams and T. Hogg, in *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92)* (AAAI Press, Menlo Park, CA, 1992), pp. 472–477.

[17] C. P. Williams and T. Hogg, in *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)* (AAAI Press, Menlo Park, CA, 1993), pp. 152–157.

[18] S. Kirkpatrick and B. Selman, Science **264**, 1297 (1994).

[19] C. P. Williams and T. Hogg, in *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI'94)* (AAAI Press, Menlo Park, CA, 1994), pp. 1310–1315.

[20] L. K. Grover, Phys. Rev. Lett. **80**, 4329 (1998).

[21] G. Brassard, P. Hoyer, and A. Tapp, e-print quant-ph/9805082.

[22] E. M. Palmer, *Graphical Evolution—an Introduction to the Theory of Random Graphs* (Wiley Interscience, New York, 1985).

[23] B. Bollobas, *Random Graphs* (Academic Press, New York, 1985).

[24] R. Jozsa, Los Alamos e-print, quant-ph/9901021.

[25] R. M. Gingrich, C. P. Williams, and N. J. Cerf, Phys. Rev. A

(to be published); e-print quant-ph/9904049.
[26] N. J. Cerf, L. K. Grover, and C. P. Williams, e-print quant-ph/9806078.
[27] D. Biron, O. Biham, E. Biham, M. Grassl, and D. A. Lidar, in *Quantum Computing and Quantum Communications, Lecture Notes in Computer Science Vol. 1509*, edited by C. P. Williams (Springer-Verlag, Berlin, 1999), pp. 140–147; e-print quant-ph/9801066.
[28] R. Paturi, P. Pudlak, M. E. Saks, and F. Zane, in *Proceedings of the 39th Annual Symposium on Foundations of Computer Science* (IEEE Computer Society, Los Alamitos, CA, 1998), pp. 628–637.