Discrete Optimization 8 (2011) 376–384

Contents lists available at ScienceDirect

Discrete Optimization

journal homepage: www.elsevier.com/locate/disopt

Note

Quantum annealing of the graph coloring problem

Olawale Titiloye, Alan Crispin *

School of Computing, Mathematics and Digital Technology, Manchester Metropolitan University, Manchester M1 5GD, UK

ARTICLE INFO

Article history:
Received 17 July 2010
Received in revised form 6 December 2010
Accepted 9 December 2010
Available online 7 January 2011

Keywords:
Quantum annealing
Simulated annealing
Graph coloring problem
Combinatorial optimization

ABSTRACT

Quantum annealing extends simulated annealing by introducing artificial quantum fluctuations. The path-integral Monte Carlo version chosen is population-based and designed to be implemented on a classical computer. Its first application to the graph coloring problem is presented in this paper. It is shown by experiments that quantum annealing can outperform classical thermal simulated annealing for this particular problem. Moreover, quantum annealing proved competitive when compared with the best algorithms on most of the difficult instances from the DIMACS benchmarks. The quantum annealing algorithm has even found that the well-known benchmark graph dsjc1000.9 has a chromatic number of at most 222. This is an improvement on its best upper-bound from a large body of literature.

1. Introduction

In the Graph Coloring Problem (GCP), we are given an undirected graph $G = (V, E)$ where V denotes the set of vertices and E the set of edges. An edge consists of a pair of adjacent vertices. With the minimum number of colors possible, each vertex should be assigned a color such that no vertices with a common edge have the same color. If K is a set of colors, then a (proper) coloring of G is a mapping $col : V \rightarrow K$ such that $col(u) \neq col(v)$, for any cases where u and v are adjacent. The minimum number of colors required to color G is known as the chromatic number $\chi(G)$ or χ. If V_j is the set of vertices with color j, the problem can also be viewed as the partitioning of V into k color classes $V_1, \ldots, V_k$ where $k = \chi(G)$, such that no two vertices in the same color class are adjacent. The GCP is an important combinatorial optimization problem with applications in several areas including register allocation [1], timetabling [2] and scheduling [3]. It is known to be very difficult and NP-hard in general. In practice, it is often sufficient to solve the relaxed problem of finding a coloring of G with k colors (or a k-coloring), where $\chi(G) \leq k \leq |V|$. One usually attempts to make k as close to $\chi(G)$ as possible, but achieving this is still NP-hard [4]. Exact algorithms [5] usually become impractical when presented with problems consisting of more than a hundred vertices, leading many researchers to concentrate on heuristic algorithms [6,7]. A number of heuristics have been used to find k-colorings with success. Some of these are based on simulated annealing [6,8], Tabu search [9], and hybrid evolutionary algorithms with specialized crossovers [10–13]. The GCP can be approached by first finding a k-coloring for a high estimate of $\chi(G)$, and then finding k-colorings for successively lower values of k, until this becomes impossible. This is a common approach that is also taken in this paper.

Quantum Annealing (QA) [14] is a relatively new heuristic which has been shown to be more effective than thermal Classical (simulated) Annealing (CA) [15] in solving some combinatorial optimization problems including the Traveling Salesman Problem (TSP) [16] and the Ising Spin Glass Problem [17]. CA is based on an analogy with statistical mechanics from classical physics. It involves attempting to find a global minimum of a cost function by introducing an artificial temperature parameter which is slowly lowered towards zero, during a Metropolis Monte Carlo simulation. This procedure aids in thermally overcoming barriers, and preventing the search from getting trapped in poor local minima. QA extends CA with

* Corresponding author. Tel.: +44 0 161 247 3571; fax: +44 0 161 247 1483.
E-mail addresses: olawale.titiloye@stu.mmu.ac.uk (O. Titiloye), a.crispin@mmu.ac.uk (A. Crispin).

1572-5286/$ – see front matter
doi:10.1016/j.disopt.2010.12.001

ideas from quantum mechanics. Artificial quantum fluctuations are applied with the aim of tunneling through barriers in the search for a global minimum [18]. To the best of our knowledge this is the first application of QA to the GCP. In this paper, we present a GCP solver based on QA (and called QA-col), and compare it with one based on CA (CA-col). In Section 2.1, a transformation is applied to a quantum system, so that it can be simulated feasibly as a population based search on a classical computer. The algorithms CA-col and QA-col are presented in Section 2.2, and efficiency concerns are addressed in Section 2.3. Section 3 contains experimental results which show that QA-col outperforms CA-col. QA-col also matches the quality of colorings found by the leading GCP algorithms for many graphs, and even finds a new result for one of them. Further discussion can be found in Section 4, followed by the conclusions in Section 5.

2. Formulating the GCP for CA and QA

The GCP is a combinatorial optimization problem that can be reduced to a series of decision problems concerning the existence of k-colorings, first for a value of k which is a high estimate of $\chi(G)$, and afterwards for decreasing values of k, until k-colorings can no longer be found. To differentiate from a k-coloring which we always take to mean one without any conflicts, we refer to an assignment of all vertices to colors as a configuration, denoted by ω. The set of all possible configurations is the search space Ω to be explored by CA and QA. Many heuristics including stochastic QA and CA are incomplete. This means that they are correct when their answer to a decision problem is 'yes', but may be wrong if their answer is no. Therefore the lowest value of k for which such an algorithm can find a k-coloring is only an upper-bound on the value of $\chi(G)$. We will concentrate on solving the k-coloring problem for a given k. A local search approach to finding a k-coloring might start off by randomly assigning colors to each of the vertices of the given graph using k colors. This would usually give rise to an initial configuration with a number of conflicting edges. For a k-coloring, the number of conflicting edges has to be equal to zero. We could attempt to improve the initial configuration by looking for color changes of vertices that result in a decrease in the total number of conflicts. A single color change is often termed a move, as this can be viewed as the transfer of a vertex from its current color class to another. With this approach, every vertex must be assigned to a color at all times. When using a local search with simple descent, any move that results in an increase in conflicts is rejected. This works in decreasing the number of conflicts, but only to a limited extent, because of the high likelihood of getting trapped in local minima. CA is commonly used to alleviate this situation by accepting apparently bad moves with some probability as a strategy for escaping local minima. In our formulation for CA, the cost function (or potential energy) to be minimized is $\mathcal{H}_{pot}$, and is given by the number of conflicting edges denoted by $C(\omega)$. The neighborhood of a configuration is the set of all configurations that can be reached from the current one by making a single move. The described method of adapting CA to the k-coloring problem was first suggested in [19] and has since been explored in [6,8].

In the case of QA, the cost function to be minimized is $\mathcal{H}_q = \mathcal{H}_{pot} + \mathcal{H}_{kin}$. The energy $\mathcal{H}_q$ is a quantum Hamiltonian composed of a classical potential energy $\mathcal{H}_{pot}$ and a kinetic one $\mathcal{H}_{kin}$. While CA uses only $\mathcal{H}_{pot}$ as a cost function, QA requires $\mathcal{H}_{kin}$ for the purpose of introducing artificial quantum fluctuations, which are used to try to escape local minima. Even though the k-coloring problem can be viewed as a decision problem, we approach it by optimizing the cost function $\mathcal{H}_{pot}$. In QA, we seek to minimize $\mathcal{H}_{pot}$ as a side effect of minimizing $\mathcal{H}_q$. A k-coloring is reached if and only if $\mathcal{H}_{pot}$ is zero. In order to define $\mathcal{H}_{kin}$ and apply QA to any combinatorial optimization problem, it is first represented as the ground state search of an Ising model [16,18]. For the k-coloring problem, this involves expressing a coloring configuration as a series of Boolean variables, such that when they are assigned the right values, the original problem is solved.

2.1. The k-coloring problem, the Ising model, and the Suzuki–Trotter transformation

An Ising model consists of a set of spins (or spin variables), each of which can only be in one of two states. Most combinatorial optimization problems can be expressed in terms of Boolean variables and hence spin variables [18]. Each of these spin variables usually takes on the value of either $+1$ or -1, also known as an up-spin and a down-spin respectively. Examples of how to express a combinatorial optimization problem as an Ising Model for QA can be found in [20] for Boolean satisfiability and also in [16] for the TSP where it was mentioned that domain-specific knowledge should be taken into consideration where possible. One could represent the k-coloring problem in terms of Boolean variables by having one variable each for all possible propositions that some vertex is currently assigned to a given color. We term this the absolute color mapping because it concentrates on the exact colors assigned to vertices. For reasons that will become clear shortly, the absolute color mapping is not a very good Ising model for the k-coloring problem. An alternative and more effective mapping is deferred till later in this section, after some relevant definitions have been given. For now, it will suffice to note that a coloring configuration can be fully represented in terms of spin variables, and that the values of these spin variables are dependent on the colors assigned to the vertices of the graph under consideration. The set of spin variables $S_1, S_2, \ldots, S_n$ or $\{S_i\}$ corresponds to a coloring configuration ω. Hence $\{S_i\}$ can be considered as an alternative representation of ω, interchangeable in many contexts. For example, the potential energy in CA can be expressed as $\mathcal{H}_{pot}(\{S_i\}) := \mathcal{H}_{pot}(\omega) = C(\omega)$.

Providing a deterministic solution to the minimization of $\mathcal{H}_q$ with quantum mechanics is possible, and has been done for some basic problems [21]. However, it is very computationally expensive, and results in intractability for most practical problems. For this reason, a Metropolis Monte Carlo scheme (as in CA) is used instead. This is implemented with stochastic dynamics using a pseudo-random number generator. Path-Integral Monte Carlo for Quantum Annealing (PIMC-QA) is a

stochastic implementation of QA that has proved to be successful on some combinatorial optimization problems [16,22]. In PIMC-QA, a quantum Hamiltonian is approximated by a classical one with the aid of a Suzuki–Trotter transformation [14,23,24]. This is possible because of an analogy with a standard Ising model in a transverse field [14]. The transformation maps the quantum Hamiltonian $\mathcal{H}_q$ to an effective classical Hamiltonian $\mathcal{H}$ similar to [20], and given by (1).

$$\mathcal{H} = \frac{1}{P}\sum_{\rho=1}^{P}\mathcal{H}_{pot}(\{S_{i,\rho}\}) - J_\Gamma\left(\sum_{\rho=1}^{P-1}\sum_{i}S_{i,\rho}S_{i,\rho+1} + \sum_{i}S_{i,1}S_{i,P}\right). \tag{1}$$

$\mathcal{H}$ can be viewed as consisting of P replicas $\{S_{i,\rho},\ \rho = 1,\ldots,P\}$ of the original classical Hamiltonian $\mathcal{H}_{pot}(\{S_i\})$, with an interaction of a combined kinetic energy between them [20]. $S_{i,\rho}$ denotes the ith spin of the ρth replica (or S_i of ρ). Generally, the bigger the value of P, the more the memory and computation needed [16]. In this paper we set $P = 10$ for all instances considered in Section 3. This is in line with hybrid evolutionary algorithms which also use a small population size [10]. While CA maintains a single configuration ω, QA requires a series of configurations $\omega_1, \omega_2, \ldots, \omega_P$ or $\{\omega_\rho\}$. The slight difference between (1) and the equivalent expression given in [20] is that we have added an extra term for the sum of products of spins involving the first and the last replica. This creates a symmetry in which every replica directly interacts with the replica behind and the one in front, with the connections between them resembling a cyclical structure. It will be seen in Section 2.3 that this allows for easier reasoning for efficient computations involving changes in $\mathcal{H}$. We verified by limited experiments with the more theoretically accurate Hamiltonian given in [20] that our extra term did not result in any noticeable negative impact on QA. For convenience we will sometimes denote $\{\omega_\rho\}$ by ϖ. Since $\mathcal{H}$ in (1) depends on ϖ, it will be expressed as $\mathcal{H}(\varpi)$ when this dependence needs to be emphasized. In Section 2.2, an algorithm is given which uses $\mathcal{H}$ at an effective quantum temperature $T_q = PT$ in a Monte Carlo simulation. The temperature at which each replica is simulated is given by T, and this is fixed throughout the annealing schedule. The term J_Γ is the coupling among the replicas given from [20] as:

$$J_\Gamma = -\frac{T}{2}\ln\tanh\left(\frac{\Gamma}{PT}\right) > 0 \tag{2}$$

where Γ is the tunneling field strength which is initially set to a high value and then slowly lowered at each Monte Carlo Step. J_Γ evolves inversely to Γ, so that as the strength of the quantum fluctuations decrease, the coupling term increases, causing the replicas to become more alike.

Now that the kinetic energy is given as the Γ dependent term in (1), it is easier to determine what makes a good choice of Ising model representation for the k-coloring problem. From (1), it can be observed that each of the P replicas possesses its own set of spin variables. The replicas interact with each other to make up the kinetic energy of the system. By attempting to minimize $\mathcal{H}$, the system is guided by a complex interaction between all replicas which produce an effect very different from simply running multiple copies of CA in parallel. Evolutionary algorithms are a well known approach also consisting of a population of individuals with interactions between them. The current best graph coloring algorithms pursue a hybrid approach with an evolutionary component and a local search one [10–13]. The idea that PIMC-QA can exhibit pseudo-evolutionary behavior has been pointed out by some QA researchers. For instance, Battaglia et al. [20] noted that "The P replicas can be seen as a population of individuals, the spin configuration of each replica as its genotype and the classical Hamiltonian as a fitness function". Since hybrid evolutionary algorithms for graph coloring have been around for at least a decade now, it is instructive to consider what domain-specific knowledge has been gathered that can help in making the best choice for representing a coloring configuration in terms of Boolean variables. Hybrid evolutionary algorithms commonly involve the maintenance of a population from which two or more individuals are repeatedly selected and passed on to a specialized crossover operator to produce an offspring. The offspring is then improved by a local search procedure, (usually Tabu [7,9]) and put back into the population by a replacement rule. The specialized crossover operator is designed with the aim of transmitting large color classes that are possessed by the parents, onto their offspring [12,13]. The authors of [10] noted that conventional genetic algorithm crossovers did not work well especially because of the redundancy of color naming inherent in these approaches which interfered with the transmission of useful properties from individuals to offspring. It is for this same reason that the absolute color mapping is a poor choice. The domain-specific knowledge for graph coloring which has been acquired after several years of research on hybrid evolutionary algorithms for graph coloring is that individuals in the population need to transmit color class information, rather than the absolute colors that have been assigned to vertices.

It would be advantageous if spin variables could be defined such that they take this principle into consideration. Fortunately this can be done. In order to represent the k-coloring problem as an Ising model, we define $\{S_i\}$ such that $S_i = +1$ if a pair of vertices denoted by $i := \langle u, v\rangle$ currently consists of differently colored vertices, that is $col(u) \neq col(v)$, otherwise $S_i = -1$. The vertices in each pair need not be adjacent and each unique pair in the graph is mapped to a separate spin variable. More precisely, if E' is the set of edges in G' (the complement of graph G), then each $i \in E \cup E'$ is associated with a variable S_i. Therefore the total number of spin variables for any graph is given by $(|V|(|V| - 1))/2$. To emphasize that i refers to a pair $\langle u, v\rangle$, we sometimes write $S_{i,\rho} := S_{\langle u,v\rangle,\rho}$. This notation will be useful in Section 2.3.2, where we provide fast schemes for computing changes in $\mathcal{H}$. The set of spin variables in each replica encompasses information about how vertices are being grouped into color classes relative to each other, without regard to the actual color given to the vertices. We term this choice a relative color mapping. The Γ-dependent kinetic energy term is a sum of products of spins which quantifies how similar the replica colorings are to each other. By incorporating this into the Hamiltonian to be minimized in PIMC-QA, replicas are able to implicitly transmit coloring information to each other.

2.2. The CA-col and QA-col algorithms for finding k-colorings

Algorithm 1 is an outline of CA-col. This algorithm is for the most part equivalent to an algorithm in [6].

Algorithm 1. CA-col: Graph coloring with CA

1: **Input:** Graph G, number of colors k, T_0 and *MaxSteps*
2: **Output:** Best coloring configuration found
3: Initialize a random configuration ω for graph G, $T = T_0$
4: **repeat**
5: **repeat**
6: Randomly select vertex v from the list of all vertices involved in conflicts
7: Move v to a new randomly selected color class to derive ω'
8: $\Delta\mathcal{H}_{pot} = \mathcal{H}_{pot}(\omega') - \mathcal{H}_{pot}(\omega)$
9: **if** $\Delta\mathcal{H}_{pot} < 0$ **then**
10: $\omega = \omega'$
11: **else**
12: With probability $\exp(-\Delta\mathcal{H}_{pot}/T)$, **set** $\omega = \omega'$
13: **until** iterations $= M.N$
14: $T = T - (T_0/MaxSteps)$
15: **until** termination condition.

Algorithm 2. QA-col: Graph coloring with QA

1: **Input:** Graph G, number of colors k, the number of replicas P, T_0, Γ_0 and *MaxSteps*
2: **Output:** Best coloring configuration found
3: Initialize $T = T_0$, $\Gamma = \Gamma_0$ and a set of P random coloring configurations $\{\omega_\rho\} := \varpi$
4: **repeat**
5: randomly shuffle the order of replicas
6: **for** $r = 1, \ldots, P$ **do**
7: select replica ρ in position r
8: **repeat**
9: Randomly select vertex v from the list of vertices involved in conflicts in ω_ρ
10: Only in ω_ρ, move v to a new randomly selected color class, to derive ω'_ρ and hence ϖ'
11: $\Delta\mathcal{H}_{pot} = \mathcal{H}_{pot}(\omega'_\rho) - \mathcal{H}_{pot}(\omega_\rho)$
12: $\Delta\mathcal{H} = \mathcal{H}(\varpi') - \mathcal{H}(\varpi)$
13: **if** $\Delta\mathcal{H}_{pot} < 0$ **or** $\Delta\mathcal{H} < 0$ **then**
14: $\varpi = \varpi'$
15: **else**
16: With probability $\exp(-\Delta\mathcal{H}/T)$, **set** $\varpi = \varpi'$
17: **until** iterations $= M.N$
18: **end for**
19: $\Gamma = \Gamma - (\Gamma_0/MaxSteps)$
20: **until** termination condition.

It was one of the earliest stochastic algorithms for graph coloring, and was considered successful in its time. It has since been surpassed by others, especially the hybrid evolutionary algorithms [10–13].

The outermost loop is controlled by the Temperature parameter T. We chose a linear annealing schedule consisting of the initial temperature T_0 and a maximum number of Monte Carlo steps (*MaxSteps*). The average neighborhood size can be estimated by $N = |V|.k$ as in [6]. Each Monte Carlo step for CA-col consists of a loop starting on line 5. Where M is a tunable multiplier, $M.N$ moves are attempted at each step after which the control parameter is decreased. Algorithm 2 (QA-col) is structurally similar to Algorithm 1, with a crucial difference of the presence of an additional loop for the replicas. Line 5 in Algorithm 2 requires some explanation. The replicas are always connected to each other in numerical order in the same way throughout the search for the purpose of spin products. For example, replica 3 is only ever connected to 2 and 4. The random shuffling specified is meant to only change the order in which replicas are selected for search. We found that this promotes diversity in the population of configurations. On line 13 of QA-col, in addition to checking if $\Delta\mathcal{H}$ is less than zero, we check if the conflicts in the current replica are reduced. This ensures that such opportunities are never missed. QA-col continues to make the next Monte Carlo Step each time line 19 is reached until the termination condition is satisfied. The tunable multiplier to the average neighborhood size was fixed at $M = 4$ for both CA-col and QA-col for all problem instances considered in the paper. The algorithms were made to terminate when a configuration without any conflicts was found, when a given time limit was reached, or when *MaxSteps* was exhausted, whichever occurred first.

2.3. Efficient techniques and data-structures for computationally expensive portions of CA-col and QA-col

We now present procedures for the fast computation of energy changes in both algorithms.

2.3.1. Efficient computation of changes in potential energy

The most visited point in the execution of CA-col is the calculation of $\Delta\mathcal{H}_{pot}$, which is the change in the number of conflicts that would occur if a chosen vertex v were to change its color. $\Delta\mathcal{H}_{pot}$ is also a critical aspect of QA-col. From the notation of Section 2, we can write $\Delta\mathcal{H}_{pot} = C(\omega') - C(\omega)$. The vertices adjacent to v (which we denote by $Adj(v)$) present the possibility of causing changes to the number of conflicts. Specifically, only the vertices in $Adj(v)$ which possess either the old or (the proposed) new color of v influence $\Delta\mathcal{H}_{pot}$. If V_α and V_β are the old and new color classes respectively, then $\Delta\mathcal{H}_{pot} = |Adj(v) \cap V_\beta| - |Adj(v) \cap V_\alpha|$. This is because the conflicts with adjacent vertices having color α would be resolved and new conflicts would be created with adjacent vertices having color β. Therefore $\Delta\mathcal{H}_{pot}$ can be computed by initializing it to zero, iterating through $Adj(v)$ while incrementing $\Delta\mathcal{H}_{pot}$ by one for each $u \in Adj(v) \cap V_\beta$, and decrementing it by one for each $u \in Adj(v) \cap V_\alpha$. This procedure yields a complexity of $O(|Adj(v)|)$. This idea was mentioned in [7] for checking the quality of a move in the Tabucol algorithm, after which a much more efficient method was presented. Tabucol is a graph coloring algorithm which uses the Tabu local search technique, and checking the quality of a move in Tabucol happens to be equivalent to determining $\Delta\mathcal{H}_{pot}$. We proceed to describe our version of the faster method from [7]. If a color frequency array F of dimensions $V \times k$ is defined and maintained such that $F(v, \theta) = |Adj(v) \cap V_\theta|$ for vertex v and color θ, then $\Delta\mathcal{H}_{pot} = F(v, \beta) - F(v, \alpha)$ can now be evaluated in constant time or $O(1)$. The array F is initialized at the beginning of the algorithm, and only needs to be updated any time a move is actually accepted. Updating F can be done in $O(|Adj(v)|)$ by taking each vertex $u \in Adj(v)$, incrementing $F(u, \beta)$ by one and decrementing $F(u, \alpha)$ by one. Since a lot more moves are usually attempted than accepted (as is evident in Section 3), this strategy is very efficient.

2.3.2. Efficient computation of changes in $\mathcal{H}$ in QA-col

To compute the $\Delta\mathcal{H}$ in QA-col efficiently, (1) is first re-written as $\mathcal{H} = PEterm - KEterm$ such that $PEterm$ refers to the expression involving the potential energy and $KEterm$ is the Γ dependent expression. It then follows that $\Delta\mathcal{H} = \Delta PEterm - \Delta KEterm$. Also, whenever $\Delta\mathcal{H}$ is being calculated, the move being attempted is always for the current replica only. Because the conflicts in all other replicas remain the same before and after the move, they cancel out in the calculation of $\Delta PEterm$, leaving only the change in conflicts in the current replica. Therefore $\Delta PEterm = \frac{1}{P}\Delta\mathcal{H}_{pot}$ and can therefore be determined as efficiently as in CA-col.

$\Delta KEterm$ is much more difficult to compute efficiently than $\Delta PEterm$. The first idea that needs to be recognized is that only the replicas directly next to the currently executing one can influence $\Delta KEterm$. Every replica ρ is involved in products of spins with only two others which we label b and f. Furthermore, only the products of spins involving vertices in V_α and V_β (the old and new color classes) of the current replica ρ can change whenever a move is made in ρ. These considerations mean that $\Delta KEterm$ can be computed in $O(|V_\alpha| + |V_\beta|)$. This can be achieved by defining I_$KEterm$ as an integer such that $KEterm = J_\Gamma I\_KEterm$, and then initializing ΔI_$KEterm$ to zero. From Section 2.1, $S_{\langle u,v\rangle,\rho}$ is $+1$ if $col(u) \neq col(v)$ in replica ρ, and -1 otherwise. Therefore, any time an attempt is made in ρ to change the color of v from α to β, then for each $u \in V_\alpha - \{v\}$, we increment ΔI_$KEterm$ by $\delta := 2(S_{\langle u,v\rangle,b} + S_{\langle u,v\rangle,f})$. Also, for each $u \in V_\beta$ we decrement ΔI_$KEterm$ by δ. After these operations, ΔI_$KEterm$ contains the correct value from which $\Delta KEterm$ can be calculated by multiplying J_Γ. The expression for δ was obtained by noting that if v moves from V_α to V_β, then for all $u \in V_\alpha - \{v\}$, the spin $S_{\langle u,v\rangle,\rho}$ will change from -1 to $+1$. The old spin products need to be replaced with the new ones. Therefore the expression to be added to ΔI_$KEterm$ for each u is $(+1)S_{\langle u,v\rangle,b} - (-1)S_{\langle u,v\rangle,b} + (+1)S_{\langle u,v\rangle,f} - (-1)S_{\langle u,v\rangle,f}$, which simplifies to the δ given. A similar reasoning for all $u \in V_\beta$ leads to the fact that the same δ should be subtracted from ΔI_$KEterm$ for each u.

I_KEterm can also be used as a measure of replica similarity for monitoring the progress of the search. We define $MaxI\_KEterm$ as the maximum value of I_KEterm. This maximum occurs when the configurations in all P replicas are identical, which is also when the set of spin variables in all replicas are identical. Therefore for a current configuration $\{\omega_\rho\}$ in QA-col, replica similarity is defined as $I\_KEterm/MaxI\_KEterm$.

The data structure holding all color classes of a particular replica is a list of disjoint sets of vertices. Because all possible contents of color classes have to be in the set of all vertices V, the addition, removal, membership checking and random access operations of a color class can be performed in $O(1)$. The constant time random access is particularly important in ensuring fast iteration over the color classes. Keeping an array representation of the coloring where the element $A(v)$ contains the value of $col(v)$, is also useful for storing configurations in a compact manner and for quickly checking the color of a vertex. The spins themselves should definitely not be stored in any data structure, as this is not needed.

If QA-col is given an instance with $|V| = 1000$ and $k = 222$ to solve, then the average color class size will be $|V|/k$ or in this case, about 5, thus making the time complexity of evaluating $\Delta KEterm$ particularly favorable. This typically happens with dense graphs, as they usually need a large number of colors. However, if we consider the other extreme with an instance involving a sparse graph with $|V| = 1000$ and $k = 20$, this results in a large average color class size of about 50. As the number of calculations of $\Delta\mathcal{H}$ could potentially run into billions, iterating over two color classes of average size 50 each time can result in a considerable slowdown. Fortunately, a further improvement can be made. The expression $\Delta\mathcal{H} = \Delta PEterm - \Delta KEterm$ is to be used to evaluate the value of $\exp(-\Delta\mathcal{H}/T)$ for comparison with a random number

$r \in [0, 1)$ to decide move acceptance. Smaller values of $\Delta\mathcal{H}$ increase the chance that a move will be accepted. The maximum value that ΔI_KEterm can have in the *current* configuration when changing the color of v from α to β is $4(|V_\alpha| + |V_\beta| - 1)$, and therefore computable in constant time. This expression which we denote by $UB\Delta I\_KEterm$ is an upper bound which has been derived by following the procedure given earlier in this section for calculating ΔI_KEterm, while setting spins such that a maximum value is obtained. For a given $\Delta PEterm$ and a given random number r, if $UB\Delta I\_KEterm$ is used in place of ΔI_KEterm to calculate $\Delta\mathcal{H}$ and a proposed move is rejected, then that move would still have been rejected had the actual value of ΔI_KEterm been used. If however the move is accepted, we can then calculate the actual ΔI_KEterm and derive the true $\Delta\mathcal{H}$. The move can then be finally accepted or rejected based on the same r. This means that an initial "quick evaluation" based on $UB\Delta I\_KEterm$ is able to filter out many attempted moves for rejection. Usually as much as 70%–99% of attempted moves can be eliminated this way depending on the instance. On average, this leads to a speed increase of about 100% or more in the overall algorithm, with instances involving the sparser graphs being the biggest beneficiaries. It must be stressed that the use of $UB\Delta I\_KEterm$ does not lead to approximations. Except for the difference in speed, the behavior of QA-col is the same with or without it.

Another important issue impacting on the efficiency of QA-col is the repeated calculation of the expensive exponential function exp. We tackle this situation by first expressing $\exp(-\Delta\mathcal{H}/T)$ as $\exp(\Delta KEterm/T)/\exp(\Delta PEterm/T)$. Because T always remains constant in QA-col, we can pre-compute all possible values of $\exp(\Delta PEterm/T)$ by noting that if v_{max} is a vertex with maximum degree, then $\Delta\mathcal{H}_{pot}$ can only ever take on integer values ranging from $-|Adj(v_{max})|$ to $+|Adj(v_{max})|$. Therefore an array of size $2|Adj(v_{max})| + 1$ is sufficient to hold a look-up table for all possible values of $\exp(\Delta PEterm/T)$. A similar idea for pre-computing values of $\exp(\Delta KEterm/T)$ can be derived by noting the maximum and minimum values of ΔI_*KEterm* from *any* move in *any* configuration. By substituting into $UB\Delta I\_KEterm$, $|V_\alpha| = |V|$ and $|V_\beta| = 0$ we derive the fact that all possible values of ΔI_KEterm are integers ranging from $-4(|V| - 1)$ to $+4(|V| - 1)$, and the size of the array needed for the look-up table is $8(|V| - 1) + 1$. Because the values of $\exp(\Delta KEterm/T)$ involve Γ, this second look-up table has to be re-calculated every time Γ is decremented. This turns out to be relatively inexpensive as the size of the look-up table is small compared to the number of moves attempted during each cycle. On average, a further 40% speed increase is realized in the overall QA-col algorithm when look-up tables are used for the potential and kinetic energy terms.

3. Experimental results

Both algorithms were implemented in MinGW C++ and run on a PC with a 3 GHz Intel processor and 3 GB of RAM with Windows XP. In order to measure their performances, we chose well known benchmarks from the second DIMACS competition [25]. Leighton graphs are named with the format le$X_\chi Y$, where X is the number of vertices, χ is the chromatic number and Y distinguishes between different graphs. Random graphs *dsjcX.Y* have X vertices, and each possible edge is connected with a probability of $Y/10$. Also included are two types of geometric random graphs with names of the form *dsjrX.Y* and *rX.Y* where X is the number of vertices and Y is a construction parameter. A "*c*" is added as a suffix when the complement of a graph is meant. Flat graphs have a name of the form flat X_χ_0. Once again X is the number of vertices and χ is the chromatic number. The "0" at the end refers to the fact that all vertices are incident to the same number of vertices. Finally there is a Latin square graph named latin_sqr_10. Graphs from DIMACS [25] have been used extensively in the literature for testing many GCP algorithms [10–13,26–29]. We chose the first basis of comparison between the QA-col and CA-col algorithms to be the relative speed. To estimate this, we recorded the average number of attempted moves (att-moves), the average number of moves, and the average time that each algorithm needed to find a solution. The second basis estimated robustness by noting the number of successful runs. The quality of colorings or lowest k reached was the third and final one. The lowest k reached is also our criterion for comparing QA-col to other algorithms in the literature. This is customary due to the widely varying experimental conditions.

The first three columns in both Tables 1 and 2 are for the names of the graphs and the best coloring found by any algorithm in the literature (k^*), the number of colors used by our algorithm, and the initial temperature, respectively. The rest of the columns are for *Maxsteps*, the average number of attempted moves, average number of moves, average time in seconds, and the frequency of success out of 10 runs. Each run used a different seed for the random number generator. For Table 2 only, there are also columns for Γ_0 and percentage of attempted moves with a quick evaluation hit (QE from Section 2.3.2). The parameters that needed to be set for each problem instance were T_0 and *MaxSteps* in CA-col, and T_q, Γ_0 and *MaxSteps* in QA-col. For a first application of QA to graph coloring, we decided to hand-tune the parameters. This allowed us to explore the relationship between the parameters for a wide variety of graphs. For all the DIMACS instances we tested, good choices for T_q and Γ_0 were found in the ranges $T_q \in (0, 1]$ and $\Gamma_0 \in (0, 3T_q]$ respectively. This assumes that we can always set *MaxSteps* to be large enough. Because the graphs used in our tests have widely varying structures, we conjecture that the optimal ranges for any arbitrary graph are very close to those we have derived empirically. Nevertheless, even if a k-coloring exists and QA-col is capable of finding it, simply choosing just any values within these ranges can cause a failure. Therefore extra information is needed in order to narrow down these intervals for particular instances. Ideally we want to operate with move acceptance ratios that are as low as possible, as this helps the efficiency of QA-col as seen from Section 2.3. It was observed that T_q was the major determinant of the initial acceptance ratio, and that the higher the edge density of the graph was, the smaller the optimal values of T_q and initial acceptance ratio needed to be. Initial acceptance ratios ranged from about 0.5%–10% on the instances we tested. Γ_0 is responsible for controlling replica similarity as described in Section 2.3.2, and larger values of Γ_0 are

Table 1
Results for CA-col with a 5 h time limit.

Graph (k^*)	k	T_0	*MaxSteps*	Att-moves	Moves	T (s)	Success
dsjc250.5 (28)	28	0.35	1.0×10^5	5.0×10^8	1.6×10^7	45	7/10
dsjc500.1 (12)	12	0.45	1.0×10^7	4.8×10^9	4.1×10^8	489	10/10
dsjc500.5 (48)	49	0.35	1.0×10^6	2.1×10^{10}	3.8×10^8	2117	8/10
	48	0.35	1.0×10^6	–	–	–	0/10
dsjc500.9 (126)	127	0.2	1.0×10^6	2.4×10^{10}	2.6×10^8	2330	10/10
	126	0.2	1.0×10^7	–	–	–	0/10
le450_15c (15)	15	0.6	1.0×10^6	6.3×10^8	6.3×10^7	73	10/10
le450_15d (15)	15	0.6	1.0×10^6	3.7×10^8	4.2×10^7	41	4/10
flat300_28_0 (28)	31	0.35	1.0×10^6	3.9×10^9	1.1×10^8	375	10/10

Table 2
Results for QA-col with a 5 h time limit.

Graph (k^*)	k	T_q	Γ_0	*MaxSteps*	Att-moves	Moves	QE (%)	T (s)	Success
dsjc250.5 (28)	28	0.35	0.75	1.0×10^4	6.1×10^7	2.1×10^6	95	8	10/10
dsjc500.1 (12)	12	0.45	1.3	1.0×10^6	4.5×10^8	3.8×10^7	88	82	10/10
dsjc500.5 (48)	49	0.35	0.65	1.0×10^5	4.3×10^8	9.3×10^6	96	63	10/10
	48	0.35	0.75	1.0×10^5	3.4×10^9	7.0×10^7	97	494	10/10
dsjc500.9 (126)	127	0.2	0.35	1.0×10^5	7.9×10^8	8.8×10^6	99	103	10/10
	126	0.2	0.38	1.0×10^6	9.9×10^9	1.1×10^8	99	1198	10/10
dsjc1000.1 (20)	20	0.44	1.1	1.0×10^6	9.1×10^9	3.8×10^8	85	1951	9/10
dsjc1000.5 (83)	84	0.36	0.68	2.0×10^7	1.8×10^{10}	2.8×10^8	97	2842	10/10
	83	0.36	0.72	5.0×10^7	8.2×10^{10}	1.0×10^9	98	12,773	9/10
dsjc1000.9 (223)	223	0.23	0.38	1.0×10^8	2.6×10^{10}	2.2×10^8	99	4100	8/10
	222	0.19	0.375	2.0×10^9	1.1×10^{11}	5.6×10^8	99	13,740	2/10
le450_15c (15)	15	0.6	1.6	1.0×10^5	1.9×10^7	1.9×10^6	83	4	10/10
le450_15d (15)	15	0.6	1.8	1.0×10^5	1.5×10^8	1.3×10^7	86	26	10/10
le450_25c (25)	26	0.3	0.48	1.0×10^5	5.5×10^7	2.2×10^6	74	9	10/10
	25	0.3	0.58	2.0×10^9	3.7×10^{10}	1.6×10^9	86	5592	2/10
le450_25d (25)	26	0.3	0.48	1.0×10^5	8.7×10^7	2.8×10^6	74	13	10/10
	25	0.3	0.59	1.0×10^7	6.7×10^{10}	3.2×10^9	87	10,069	1/10
flat300_28_0 (**28**)	31	0.35	0.75	1.0×10^5	1.5×10^8	4.7×10^6	95	19	10/10
flat1000_76_0 (82)	83	0.36	0.67	5.0×10^7	1.4×10^{10}	1.9×10^8	97	2250	10/10
	82	0.36	0.71	5.0×10^7	6.4×10^{10}	7.9×10^8	97	9802	7/10
r1000.5 (**234**)	239	0.11	0.07	1.0×10^5	2.3×10^{10}	5.7×10^8	85	5879	10/10
	238	0.11	0.07	1.0×10^8	3.7×10^{10}	8.9×10^8	86	9511	3/10
dsjr500.5 (122)	123	0.14	0.1	1.0×10^8	2.4×10^9	8.1×10^7	78	483	10/10
	122	0.15	0.1	1.0×10^8	1.8×10^9	5.6×10^7	73	370	2/10
dsjr500.1c (85)	85	0.25	0.55	1.0×10^6	3.4×10^9	6.1×10^7	97	525	10/10
r250.5 (65)	65	0.1	0.1	1.0×10^6	1.1×10^9	5.8×10^7	92	168	9/10
r1000.1c (98)	98	0.5	1.5	1.0×10^6	1.6×10^9	1.8×10^7	98	287	10/10
latin_sqr_10 (98)	98	0.45	0.9	1.0×10^7	9.0×10^9	9.9×10^7	98	1449	10/10

Colorings and log files from QA-col experiments are available at http://sites.google.com/site/olawaletitiloye/graphcoloring/qacol.

usually necessary for lower values of k for the same graph. A good Γ_0 for k can be incremented slightly and reused for $k-1$ as Table 2 shows. This means that quicker tuning can be performed by first looking for T_q and Γ_0 that solve larger values of k. We were also able to confirm the observation in [16] that a good value of T_q in QA was usually suitable for T_0 in CA and vice-versa.

In successful runs of either algorithm, only a fraction of *MaxSteps* usually gets used up before a solution is found. It is set to be very large for some difficult instances to create a slow annealing schedule. We also set *MaxSteps* in QA-col to be smaller than in CA-col by a factor of $P = 10$ to compensate for the difference in the rate of decrease of control parameters. Each run used a different seed of the pseudo-random number generator and if a solution was not found within 5 h then that particular run was terminated. By comparing Tables 1 and 2, it can be seen that for all solved instances, CA-col required more moves and more time in order to find a solution.

For graphs dsjc500.5 and dsjc500.9, CA-col failed to match QA-col in finding the best known results of 48 and 126 respectively. We checked that varying the parameters does not improve the quality of colorings that CA-col can obtain on these graphs. The results from CA-col were to be expected as one of the algorithms tested extensively in [6] was very similar and mainly differed in the annealing schedule used. The results obtained in [6] were comparable to that of CA-col as can be seen from Table 3. The algorithm in [26] is a variant of CA combined with some other heuristics. Its results are also listed in Table 3. After it was clear that QA-col was superior to CA-col on seven graphs, we decided to concentrate on testing QA-col alone on the other more difficult instances. The high values in the column for QE show how useful the quick evaluation is in preventing many attempted moves from being evaluated in the more expensive way. In Table 3, we compare QA-COL to nine of the most important algorithms in the literature, some of which are very recent. The 19 graphs considered include

Table 3
Comparison between QA-col and some other algorithms.

Graph (k^*)	QA -col	CA -col	[6] 1991	[26] 1996	[10] 1999	[27] 2000	[28] 2008	[11] 2008	[29] 2008	[12] 2010	[13] 2010
dsjc250.5 (28)	**28**	**28**	29	**28**	**28**	**28**	**28**	**28**	–	**28**	**28**
dsjc500.1 (12)	**12**	**12**	13	**12**	–	**12**	**12**	**12**	**12**	**12**	**12**
dsjc500.5 (48)	**48**	49	49	49	**48**	49	**48**	**48**	**48**	**48**	**48**
dsjc500.9 (126)	**126**	127	128	**126**	–	127	**126**	127	**126**	**126**	**126**
dsjc1000.1 (20)	**20**	–	21	21	**20**	21	**20**	**20**	**20**	**20**	**20**
dsjc1000.5 (83)	**83**	–	86	88	**83**	88	84	**83**	89	**83**	**83**
dsjc1000.9 (223)	**222**	–	226	226	224	228	224	224	225	223	223
le450_15c (15)	**15**	**15**	–	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**
le450_15d (15)	**15**	**15**	–	**15**	–	**15**	**15**	**15**	**15**	**15**	**15**
le450_25c (25)	**25**	–	–	**25**	26	26	26	**25**	**25**	**25**	**25**
le450_25d (25)	**25**	–	–	**25**	–	26	26	**25**	**25**	**25**	**25**
flat300_28_0 (28)	31	31	–	31	31	31	31	31	**28**	29	29
flat1000_76_0 (82)	**82**	–	–	89	83	87	84	**82**	87	**82**	**82**
r1000.5 (234)	238	–	–	241	–	237	–	**234**	247	245	237
dsjr500.5 (122)	**122**	–	124	123	–	**122**	125	**122**	125	**122**	**122**
dsjr500.1c (85)	**85**	–	**85**	**85**	–	**85**	86	**85**	**85**	**85**	**85**
r250.5 (65)	**65**	–	–	**65**	–	**65**	–	**65**	66	**65**	**65**
r1000.1c (98)	**98**	–	–	–	–	**98**	–	**98**	**98**	**98**	**98**
latin_sqr_10 (98)	**98**	–	–	**98**	–	99	104	101	–	99	**98**

some of the most difficult graphs from the DIMACS benchmarks. QA-col was able to obtain the best results ever found for all but two graphs. Moreover, we reach 222-colorings for the graph dsjc1000.9. This result had not previously been reported by any other algorithm in the literature.

4. Discussion

In addition to its pseudo-evolutionary behavior explained in Section 2.1, PIMC-QA does approximate a quantum-mechanical system. This means that another valid reason for QA-col's better performance over CA-col is the tunneling of QA through barriers, rather than having to thermally overcome them. This phenomenon has been known to guide QA into parts of the search space that would not usually be explored by CA [16,17]. Nevertheless PIMC-QA like any other population-based heuristic will be ineffective without an appropriate neighborhood function and a suitable definition of the interaction between individuals. Therefore, reasons for the effectiveness of QA-col naturally share a common ground with those of the powerful hybrid-evolutionary graph coloring algorithms recently described in [12,13] named MACOL and Evo-Div, respectively. In both of these, neighboring coloring configurations are obtained the same way as in QA-col, and the interaction between the individuals is also based on the principle from [10] of transmitting color class information. MACOL and Evo-Div both use Tabu as the underlying local search, and a way has been found to fix its parameters and still obtain acceptable performance for a wide variety of graphs. Therefore, standard versions of these algorithms which do not require any further tuning by the user are available. In this respect, MACOL and Evo-Div are superior to QA-col. It is likely that a future version of QA-col will include a procedure for automatically deriving good initial parameters by inspecting the invariants of the graphs such as edge density and average degree, and adaptively tuning the parameters during the search process if necessary. The results of the hand-tuning of QA-col suggest a strong link between graph invariants and the optimal parameters.

MACOL and Evo-Div define their own different multi-parent crossovers to improve on [10]. Also, each of them uses the standard set-theoretic partition distance information in their own way to encourage diversity in the population of individuals. The tunneling strength Γ controls diversity in QA-col but a future version is also likely to feature extra diversity measures based on the standard partition distance found in [12,13]. There is empirical evidence that the dynamics produced by all three algorithms MACOL, Evo-Div and QA-col are different. This follows from the fact that each algorithm finds particular graphs difficult which another finds easy. Comparisons between the three algorithms are facilitated by a common 5 hour time limit for most instances, and the usage of similar hardware. MACOL is the most effective of the three for flat300_28_0, reaching 29-colorings relatively easily. QA-col and the standard Evo-Div only reach 31-colorings, but a specially tuned Evo-Div can reach 29-colorings. QA-col has difficulties in consistently coloring Leighton graphs le450_25c and le450_25d optimally while this problem does not occur with MACOL and Evo-Div. QA-col performs better than MACOL on r1000.5, but Evo-Div is more effective than QA-col. Only QA-col reaches 222-colorings for dsjc1000.9, while MACOL reaches 223-colorings more easily than Evo-Div. Finally in the case of the latin_sq_10, MACOL only finds 99-colorings while the standard version of Evo-Div only reaches 100-colorings, but is able to reach 98-colorings when specially tuned and given 7.5 hours. In contrast QA-col takes an average of 30 min to find 98-colorings with a 100% success rate. Evo-div and QA-col are the only algorithms that replicate the 98-colorings first reported in [26]. When the overall results in Table 3 are considered, QA-col compares favorably with the best algorithms.

5. Conclusion

We have described QA-col, a first graph coloring algorithm based on PIMC-QA which is a population-based extension to simulated annealing inspired by quantum mechanics. With the aid of domain-specific knowledge, a meaningful interaction between individual replicas was defined in the form of a kinetic energy. QA-col is able to outperform classical simulated annealing and even find colorings of comparable quality to the best algorithms for many DIMACS graphs. QA is very likely to have a wider applicability to other combinatorial optimization problems and this paper shows that this could be a worthwhile investigation.

Acknowledgements

This work was supported by a Dalton Research Institute studentship towards the Ph.D. degree of the first author. We are grateful to two anonymous referees for their helpful comments and suggestions.

References

[1] F.C. Chow, J.L. Hennessy, The priority-based coloring approach to register allocation, ACM Transactions on Programming Languages and Systems 12 (4) (1990) 501–536.
[2] E.K. Burke, D.G. Elliman, R.F. Weare, A university timetabling system based on graph colouring and constraint manipulation, Journal of Research on Computing in Education 27 (1994) 1–18.
[3] F.T Leighton, A graph colouring algorithm for large scheduling problems, Journal of Research of the National Bureau of Standards 84 (1979) 489–506.
[4] M.R. Garey, D.S. Johnson, The complexity of near-optimal graph coloring, Journal of the ACM (JACM) 23 (1) (1976) 43–49.
[5] T.J. Sager, S. Lin, A pruning procedure for exact graph coloring, ORSA Journal on Computing 3 (3) (1991) 226–230.
[6] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning, Operations Research 39 (3) (1991) 378–406.
[7] P. Galinier, A. Hertz, A survey of local search methods for graph coloring, Computers and Operations Research 33 (6) (2006) 2547–2562.
[8] D. Fotakis, S.D. Likothanassis, S. Stefanakos, An evolutionary annealing approach to graph coloring, in: E.J.W. Boers, J. Gottlieb, P.L. Lanzi, R.E. Smith, S. Cagnoni, E. Hart, G.R. Raidl, H. Tijink (Eds.), Applications of Evolutionary Computing, EvoWorkshops, in: Lecture Notes in Computer Science, vol. 2037, Springer, 2001, pp. 120–129.
[9] A. Hertz, D. de Werra, Using Tabu search techniques for graph coloring, Computing 39 (1987) 345–351.
[10] P. Galinier, J.-K. Hao, Hybrid evolutionary algorithms for graph coloring, Journal of Combinatorial Optimization 3 (4) (1999) 379–397.
[11] E. Malaguti, M. Monaaci, P. Toth, A metaheuristic approach for the vertex coloring problem, INFORMS Journal on Computing 20 (2) (2008) 302–316.
[12] Z. Lü, J.-K. Hao, A memetic algorithm for graph coloring, European Journal of Operational Research 203 (1) (2010) 241–250.
[13] D.C. Porumbel, J.-K. Hao, P. Kuntz, An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring, Computers and Operations Research 37 (1) (2010) 1822–1832.
[14] A. Das, B.K. Chakrabarti, Quantum Annealing and quantum analog computation, Reviews of Modern Physics 80 (2008) 1061.
[15] S. Kirkpatrick, C. Gelatt, M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[16] R. Martoňák, G.E. Santoro, E. Tosatti, Quantum annealing of the traveling salesman problem, Physical Review E 70 (2004) 057701.
[17] G.E Santoro, R. Martoňák, E. Tosatti, R. Car, Theory of quantum annealing of an Ising spin glass, Science 295 (2002) 2427.
[18] S. Morita, H. Nishimori, Mathematical foundations of quantum annealing, Mathematical Physics 49 (2008) 12521.
[19] M. Chams, A. Hertz, D. de Werra, Some experiments with simulated annealing for coloring graphs, European Journal of Operational Research 32 (2) (1987) 260–266. Elsevier.
[20] D.A. Battaglia, G.E. Santoro, E. Tosatti, Optimization by quantum annealing: lessons from hard satisfiability problems, Physical Review E 71 (2005) 66707.
[21] L. Stella, G.E. Santoro, E. Tosatti, Optimization by quantum annealing: lessons from simple cases, Physical Review B 72 (2005) 014303.
[22] R. Martoňák, G.E. Santoro, E. Tosatti, Quantum annealing by the path integral Monte Carlo method: the two dimensional random Ising model, Physical Review B 66 (2002) 094203.
[23] M. Suzuki, Relationship between d-dimensional quantal spin systems and $(d + 1)$-dimensional Ising systems equivalence critical exponents and systematic approximants of the partition function and spin correlations, Progress of Theoretical Physics 56 (5) (1976) 1454–1469.
[24] H.F. Trotter, On the product of semi-groups of operators, Proceedings of the American Mathematical Society 10 (4) (1959) 545–551.
[25] D.S. Johnson, M. Trick, Cliques, colorings and satisfiability second DIMACS implementation challenge, in: D.S. Johnson, M.A. Trick (Eds.), Proceedings of the 2nd DIMACS Implementation Challenge, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, American Mathematical Society, 1996.
[26] C. Morgenstern, Distributed coloration neighborhood search, in: D.S. Johnson, M.A. Trick (Eds.), Proceedings of the 2nd DIMACS Implementation Challenge, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, American Mathematical Society, 1996, pp. 335–358.
[27] N. Funabiki, T. Higashino, A minimal-state processing search algorithm for graph coloring problems, IEICE Transaction Fundamentals E83-A (2000) 1420–1430.
[28] P. Galinier, A. Hertz, N. Zufferey, An adaptive memory algorithm for the K-coloring problem, Discrete Applied Mathematics 156 (2) (2008) 267–279.
[29] I. Blochliger, N. Zufferey, A graph coloring heuristic using partial solutions and a reactive Tabu scheme, Computers and Operations Research 35 (3) (2008) 960–975.