Molecular Physics
Vol. 109, No. 5, 10 March 2011, 761–772

RESEARCH ARTICLE

Group leaders optimization algorithm

Anmer Daskin[a] and Sabre Kais[b]*

[a]*Department of Computer Science, Purdue University, West Lafayaette, IN 47907, USA;* [b]*Department of Chemistry and Birck Nanotechnology Center, Purdue University, West Lafayette, IN 47907, USA*

(*Received 11 November 2010; final version received 18 December 2010*)

We present a new global optimization algorithm in which the influence of the leaders in social groups is used as an inspiration for the evolutionary technique which is designed into a group architecture. To demonstrate the efficiency of the method, a standard suite of single and multi-dimensional optimization functions along with the energies and the geometric structures of Lennard-Jones clusters are given as well as the application of the algorithm on quantum circuit design problems. We show that as an improvement over previous methods, the algorithm scales as $N^{2.5}$ for the Lennard-Jones clusters of N-particles. In addition, an efficient circuit design is shown for a two-qubit Grover search algorithm which is a quantum algorithm providing quadratic speedup over the classical counterpart.

Keywords: optimization; quantum circuits; evolutionary algorithms; Lennard-Jones clusters

1. Introduction

Global optimization is one of the most important computational problems in science and engineering. Because of the complexity of optimization problems and the high dimension of the search space, in most cases, using linear or deterministic methods to solve them may not be a feasible way [1]. Wille and Vennik [2] argued that global optimization of a cluster of identical atoms interacting under two-body central forces, belong to the class of NP-hard problems. This means that as yet no polynomial time algorithm solving this problem is known. Recently, Adib [3] reexamined the computational complexity of the cluster optimization problem and suggested that the original NP level of complexity does not apply to pairwise potentials of physical interest, such as those that depend on the geometric distance between the particles. A geometric analogue of the original problem is formulated and new subproblems that bear more direct consequences to the numerical study of cluster optimization were suggested. However, the intractability of this subproblem remains unknown, suggesting the need for good heuristics.

Many optimization methods have been developed and these can be largely classified into two groups, deterministic and stochastic. Deterministic methods include variations on Newton's method such as discrete Newton, quasi Newton and truncated Newton [4,5], tunnelling method [6] and renormalization group methods [7]. Stochastic methods include, simulated annealing [8], quantum annealing [9], J-walking [10], tabu search [11], genetic algorithms (GA) [12] and basin-hoping approach [13,14]. More recent work on probabilistic techniques have been proposed to solve these optimization problems by observing nature and modelling social behaviours and characteristics, including GA, evolutionary algorithms (EA), such as the particle swarm optimization algorithm (PSO), and the Pivot methods [1,15–20].

Implementation of many of these algorithms on complex problems requires exhausting computational time and a growing need for more computer resources depending upon: the dimension, the solution space and the type of problem. The key to speed up the optimization process is reducing the number of computations in the algorithms while keeping the amount of iterations low and the success rate of the algorithms high. This paper introduces a new global optimization algorithm which reduces the optimization time, and is both simple and easy to implement. In the following sections, the inspiration and the implementation of the algorithm will be explained and test results will be given for some of the most famous optimization test problems; for the global optimization of the minimum energy structures of complex Lennard-Jones clusters; and for the quantum circuit design of the Grover search algorithm.

*Corresponding author. Email: kais@purdue.edu

ISSN 0026–8976 print/ISSN 1362–3028 online
© 2011 Taylor & Francis
DOI: 10.1080/00268976.2011.552444
http://www.informaworld.com

2. Inspirations and related works

Leaders in social groups affect other members of their groups by influencing either the number of members or each member intensively [21]. Therefore, the effect of group leaders inclines the groups to have uniform behaviour and characteristics similar to the leader. These new behaviours and characteristics may improve or worsen the quality of the members of a group. A leader represents the properties of its group. To become a leader for a group requires a person to have some better abilities than others in the group.

Similar ideas to using leaders or/and grouping solution population have been the inspiration for optimization algorithms such as Cooperative Co-evolutionary Genetic Algorithms (CCGA), Cooperative Co-evolutionary Algorithms (CCEAs) [22,23], and Parallel Evolutionary Algorithms (PEAs) [24]. However, instead of trying to fully simulate the influence of leaders on their peers in social groups by constructing a population which includes small, interacting groups with their leaders, most of these and other similar algorithms have attempted the decomposition of big and complex problems into subcomponents or divide the whole population into multiple subpopulations with a parallel structure. In CCGAs, as described by Potter and Jong [22], each species – which are evolved into subcomponents by using a standard genetic algorithm – represents a subcomponent of the potential solution, and each representative member of the species is used to form the complete solution of the optimization problem. In [25], a general architecture for the evolving co-adapted subcomponents was presented in order to apply evolutionary algorithms to complex problems. Therefore, instead of GA, the PSO Algorithm has been used for the subcomponents of Potter's CCGA structure by van den Bergh and Engelbrecht [23]. Briefly, the general framework of cooperative coevolutionary algorithms has three main steps: problem decomposition, subcomponent optimization, and subcomponent adaptation [26].

In PEAs the whole population forms in a distributed way and consists of a multiple subpopulation. Single-population master–slaves, multiple populations, fine-grained and hierarchical combinations are the main types of PEAs [27]. The proposed algorithm in this paper differs from the PEAs in that all members of the population are interacting and there is a mutual effect between the members of the population in addition to the leaders' effect on individuals in their groups. The sum of all these interactions forms the evolutionary technique. However, in PEAs, in most cases, the interaction between subpopulations is made with the migration of individuals and evolutionary techniques used for subpopulations can be independent from each other.

3. Group leaders optimization algorithm

3.1. *General idea*

Inspired by leaders in social groups and cooperative coevolutionary algorithms, we have designed a new global optimization algorithm in which there are separate groups and leaders for each group. Initially forming groups does not require members to have some similar characteristics. Instead, it is based on random selection. While CCGA and other similar algorithms decompose the solution space, and each group represents a solution for a part of the problem, in our algorithm each group tries to find a global solution by being under the influence of the group leaders which are the closest members of the groups to local or global minima. The leaders are those whose fitness values are the best in their groups, and a leader can lose its position after an iteration if another member in the same group then has a better fitness value.

Since in social networks, leaders have effects on their peers, thusly the algorithm uses the some portion of leaders while generating new group members. Hence, a leader, (in most cases a local optimum) dominates all other solution candidates (group members) surrounding it, and the members of a group come closer and resemble their leader more in each iteration. In this way, the algorithm is able to search the solution space between a leader and its group members thoroughly, and so is able to search the area for a local or a global optimum (or an approximation of it) in a fast way.

After a certain number of evolutions, it is obvious that the members may become too similar to their leaders. To maintain the diversity of the group, for each group, we transfer some variables from different groups by choosing them randomly. In addition to providing diversity, this one-way crossover helps a group to jump out of local minima and search new solution spaces.

3.2. *Algorithm steps*

In this section, algorithm steps are described with their reasons in sequence and are shown in Figures 1 and 2.

Step 1: Generate *p* number of population for each group randomly: The total population for each group is p, hence, the whole population is $n*p$ where n is the

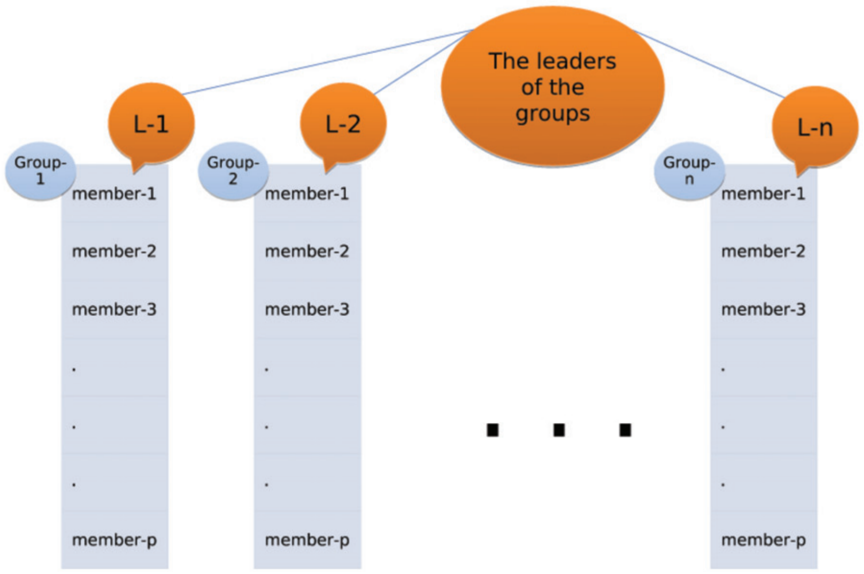
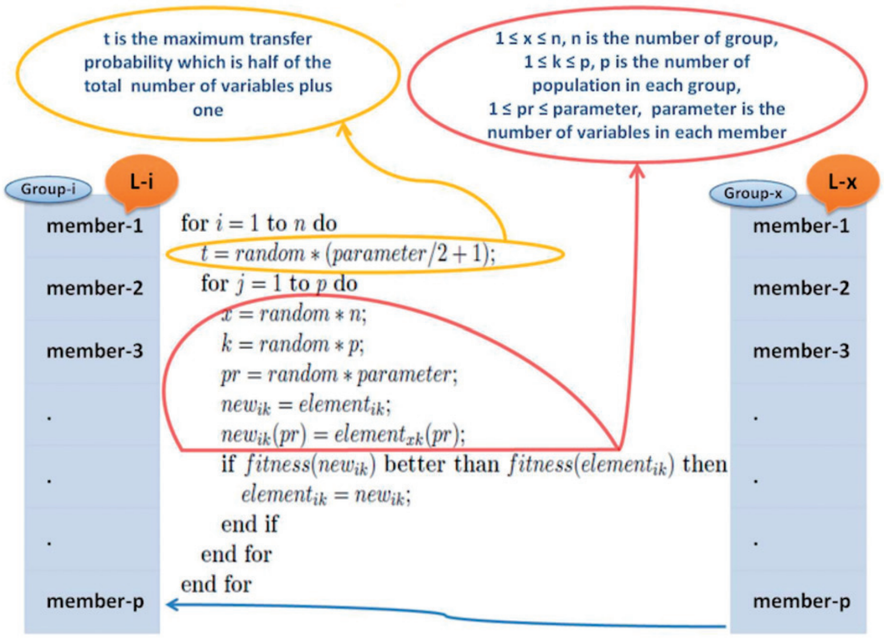


Figure 1. Steps 1–3 of the algorithm; groups consisting of *p* number of members are created, and their leaders are chosen based on the fitness values.


t is the maximum transfer probability which is half of the total number of variables plus one
1 ≤ x ≤ n, n is the number of group, 1 ≤ k ≤ p, p is the number of population in each group, 1 ≤ pr ≤ parameter, parameter is the number of variables in each member
Group-i
L-i
Group-x
L-x
member-1
member-2
member-3
member-p

```
for $i = 1$ to $n$ do
  $t = random * (parameter/2 + 1);$
  for $j = 1$ to $p$ do
    $x = random * n;$
    $k = random * p;$
    $pr = random * parameter;$
    $new_{ik} = element_{ik};$
    $new_{ik}(pr) = element_{xk}(pr);$
    if $fitness(new_{ik})$ better than $fitness(element_{ik})$ then
      $element_{ik} = new_{ik};$
    end if
  end for
end for
```

Figure 2. Step 5 of the algorithm: one way crossover: *pr*th variable of an element, member of the *i*th group, is replaced by *pr*th variable of the *k*th member of the *x*th group. The same operation is repeated *t* times for each group (maximum – half of the number of variables plus one – times for each group). The arrow at the bottom of the figure shows the direction of transfer.

number of groups. Creation of the groups and the members are totally random.

Step 2: Calculate fitness values for all members in all groups: All solution candidates, group members, are evaluated in the optimization problem and their fitness values are assigned.

Step 3: Determine the leaders for each group: Each group has one leader and the leaders are ones whose fitness values are the best within their respective groups. The algorithm steps 1–3 are shown in Figure 1.

Step 4: Mutation and recombination: Create new member by using the old one, its group leader, and a random element. If the new member has better fitness value than the old one, then replace the old one with the new one. Otherwise, keep the old member. For numerical problems, the expression simply reads;

$$new = r_1 * old + r_2 * leader + r_3 * random. \qquad (1)$$

In Equation (1), r_1, r_2, and r_3 are the rates determining the portions of old (current) member, leader, and random while generating the new population. Although in this paper, r_1, r_2, and r_3 will always sum to 1, it is not a requirement for the algorithm in general. For instance, let the current element be 0.9, the leader 1.8, and the generated random element 1.5, and suppose $r_1 = 0.8$, $r_2 = 0.19$, and $r_3 = 0.01$. In this case, the new element is equal to 1.077. Then fitness values of the old and the new element are checked. If fitness (1.077) is better than the fitness (0.9), then the old element is replaced by the new one. Pseudo-code for this step is as follows:

```
for $i = 1$ to $n$ do
  for $j = 1$ to $p$ do
    $new_{ij} = r_1 * member_{ij} + r_2 * L_i + r_3 * random$;
    if $fitness(new_{ij})$ better than $fitness(member_{ij})$
    then
      $member_{ij} = new_{ij}$;
    end if
  end for
end for
```

In the pseudo-code: n is the number of groups, p is the number of population in each group, r_1, r_2 and r_3 are the rates of the old value for the members of groups, leaders, and the random part. The expression in the inner loop is the general formula of recombination and mutation for numerical optimization problems.

Depending on the value of r_1 an element keeps its original characteristics, and depending on the value of r_2 it becomes more like its leader during iterations. Thus, in some cases, choosing the right values for r_1, r_2 and r_3 may play an important role for the algorithm to get better results during the optimization. However, choosing these parameters by obeying the property, r_3, $r_2 \leq 0.5 \leq r_1$ allows one to perform a thorough search of a solution space. Hence, this minimizes the effect of these parameters on the results.

The main benefit of this evolution is that the algorithm becomes able to search the solution space surrounding the leaders (which are possibly local or global minima). Therefore, this allows the population to converge upon global minima in a very fast way. The members of the groups are not subjected to a local minimization; however, an implementation of Lamarckian concepts of evolution for the local minimization [28] gives a more stable and efficient algorithm.

It is also important to note that Equation (1) looks similar to the updating equation of PSO [17]. The difference is that a member is always at its best position and the best position of a member is not saved in a parameter as is done in PSO, hence there is no information about the member's (or the particle's) previous positions (values).

Step 5: Parameter transfer from other groups (One way crossover): Choose random members starting from the first group, and then transfer some parameters by choosing another random member from another group. If this transfer makes a member have a better fitness value, then change the member, otherwise keep the original form. This process is shown in Figure 2 via pseudo code. This one-way crossover has similarities with the difference vector of Differential Evolution [29]. The difference is that the transfer operation is between the members which are in different groups. In this step, it is important to determine correct transfer rate, otherwise all populations may quickly become similar. In our experiments, transfer operation rate was taken t times [t is a random number between 1 and half of the number of total parameters (variables) plus one $(1 \leq t \leq (\sharp parameter/2) + 1)$] for each group (not for each member). And each time, only one parameter is transferred.

Step 6: Repeat step 3–step 5 number of given iteration times: Since each group looks for the solution in mostly different spaces, Group Leaders Optimization Algorithm (GLOA) is able to search different solution spaces simultaneously. We did not place any constraint for groups to only search in subspaces, so a few groups may also search the same places. However, this does not make them redundant as they allow GLOA to find different local or global minima within the same subspace. Since each group has a leader and the leaders direct the other members of the group in order to search the area between the leader and the members of the group, it is able to search for a good solution (around the leader of the group).

In addition to increasing the level of diversity of the groups, transferring some parameters (crossover) between groups allows the algorithm to direct the members of a group to search different spaces. Therefore, if a group has found some parameters correctly or very close to correct, then transferring parameters between groups allows other groups to get these parameters and find their solutions faster. Since only parametersare transferred which make the member have better fitness value, the spreading of a member who has a bad fitness value is avoided. In terms of optimization problems requiring highly diverse populations, choosing to do crossover and mutation-recombination steps without comparing fitness values may be wise and may improve the effectiveness of the algorithm.

4. Optimization results

The parameters for the algorithm have effects on the quality of results. As stated in the previous section, choosing r_3, r_2 less than and r_1 greater than 0.5 makes the algorithm more stable and minimizes the effects of these parameters. The number of groups and the population of the groups should be chosen to be large enough depending on the complexity of the problem. Therefore, while generating new elements, the rate of crossover between groups and the portions of elements, leaders, and random elements should be carefully adjusted for the type and the complexity of the optimization problem. For instance, if one takes crossover rate or the portion of the leaders too high in relation to the chosen group and population number, this may cause the whole population to become uniform very quickly. Hence, the algorithm may get stuck in a local solution, and not search the whole solution space. The algorithm was tested on different types of optimization problems, one-dimensional and multi-dimensional optimization test functions, and it was also used to find the minimum energy structures of Lennard-Jones clusters.

4.1. *Test functions*

While testing optimization algorithms on numerical problems, search domain and the number of iteration have crucial effects on the performance of the algorithms. In terms of the implementation, the number of groups and the population were taken the same for all numerical problems where the number of groups is 10, and the population number in each group is 25. Keeping the number of leaders the same for all numerical test functions which have different numbers of local minima shows the capability of the algorithm to escape from local minima is not highly related to the number of leaders. While results are shown in terms of function value and the number of iterations in graphs for single dimensional test problems, for multi-dimensional problems, they are presented in terms of the number of dimensions (up to 1000) and the minimum function value in each dimension.

Table 1. Parameters used for single dimensional test functions.

Parameters	Values
Number of groups	10
Number of population in each group	25
r_1 (the portion of element)	0.8
r_2 (the portion of leader)	0.2
r_3 (the portion of random)	0.0

4.1.1. *Single dimensional test problems*

For the single dimensional test functions, the parameters of the algorithm are shown in Table 1. In this table, $r_3 = 0$, that means that there is no randomness while generating new populations. We observed from our optimization that this makes GLOA converge to a global minima faster for single dimensional test problems.

Also, as shown in Figure 3, after the algorithm has gotten stuck in a local minima for some number of iterations where the minimum function values are not changing, it is still able to find the global minima at the end. This exemplifies the ability of the algorithm to jump out of the local minima and search for the global minima.

The first test function used in the optimization is the Beale Function [30] shown in Equation (2). The global minimum for this function is located at $f(x) = 0$, where $x_1 = 3$, $x_2 = 0.5$. The search domain was taken to be $[-100, 100]$, $-100 \leq (x_1, x_2) \leq 100$. Figure 3(a) shows the test result for the Beale Function.

$$f_{\text{Beale}}(x_1, x_2) = [1.5 - x_1(1 - x_2)]^2 + [2.25 - x_1(1 - x_2^2)]^2 + [2.625 - x_1(1 - x_2^3)]^2. \quad (2)$$

Secondly, the Easom Function [31] which is shown in Equation (3) and has a global minimum located at $f(x) = -1$, where $x_1 = \pi$, $x_2 = \pi$. The search domain for this function was taken as $[-100, 100]$, $-100 \leq (x_1, x_2) \leq 100$, in the implementation, and the result is presented in Figure 3(b).

$$f_{\text{Easom}}(x_1, x_2) = -\cos(x_1)\cos(x_2)\exp\big(-(x_1 - \pi)^2 - (x_2 - \pi)^2\big). \quad (3)$$

Another single dimensional test function is Goldstein–Price's Function [31], Equation (4). The global minimum for this function is at $f(x) = 3$, where $x_1 = 0$, $x_2 = -1$. The search domain used for this

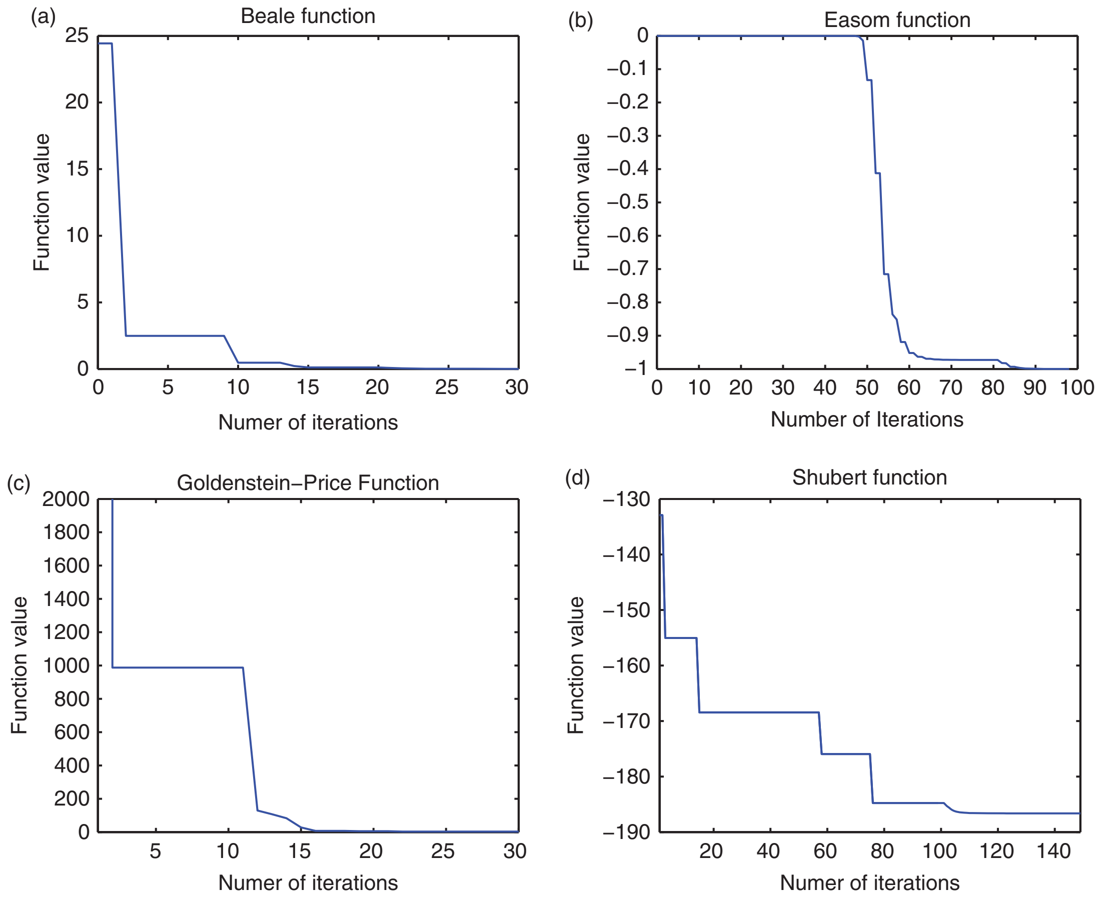


Figure 3. Results for single dimensional optimization test functions: x-axis represents the number of iterations and y-axis is the function value which is the result of the best member in the population. (a) The result for the Beale function. (b) The result for the Easom function. (c) The result for Goldein–Price's function. (d) The result for the Shubert function.

problem is $[-100, 100]$ $-100 \leq (x_1, x_2) \leq 100$, and the result is shown in Figure3(c).

$$f_{\mathrm{GP}}(x_1,x_2) = \left[1+(x_1+x_2+1)^2(19-14x_1+13x_1^2 - 14x_2+6x_1x_2+3x_2^2\right] * \left[30+(2x_1-3x_2)^2 \times (18-32x_1+12x_1^2+48x_2 - 36x_1x_2+27x_2^2\right]. \quad (4)$$

Shubert's Function [31] is the last single dimensional test function used in the experiments. This function, Equation (5), has a global minima at $f(x) = -186.7309$ for the search domain $[-10, 10]$, $-10 \leq (x_1, x_2) \leq 10$. Figure 3(d) shows the result for this function.

$$f_{\mathrm{Shubert}}(x_1,x_2) = \left(\sum_{i=1}^{5} i\cos[(i+1)x_1+i]\right) \times \left(\sum_{i=1}^{5}[i\cos(i+1)x_2+i]\right). \quad (5)$$

Table 2. Parameters used for multi-dimensional test functions.

Test function	r_1	r_2	r_3	Domain	Number of iteration
Griewank	0.8	0.19	0.01	$[-600, 600]$	1000
Ackley	0.8	0.19	0.01	$[-32.768, 32.768]$	1000
Rosenbrock	0.8	0.1	0.1	$[-100, 100]$	2000
Sphere	0.6	0.2	0.2	$[-100, 100]$	1000
Rastrigin	0.6	0.2	0.2	$[-10, 10]$	1500

4.1.2. *Multi-dimensional test functions*

Five multi-dimensional test functions were used as test cases. Each of these functions has different properties. The number of groups and populations are taken to be the same as in single dimensional functions (number of groups is 10 and number of populations in each group is 25). The other algorithm parameters and the number of iterations for these functions are given in Table 2.

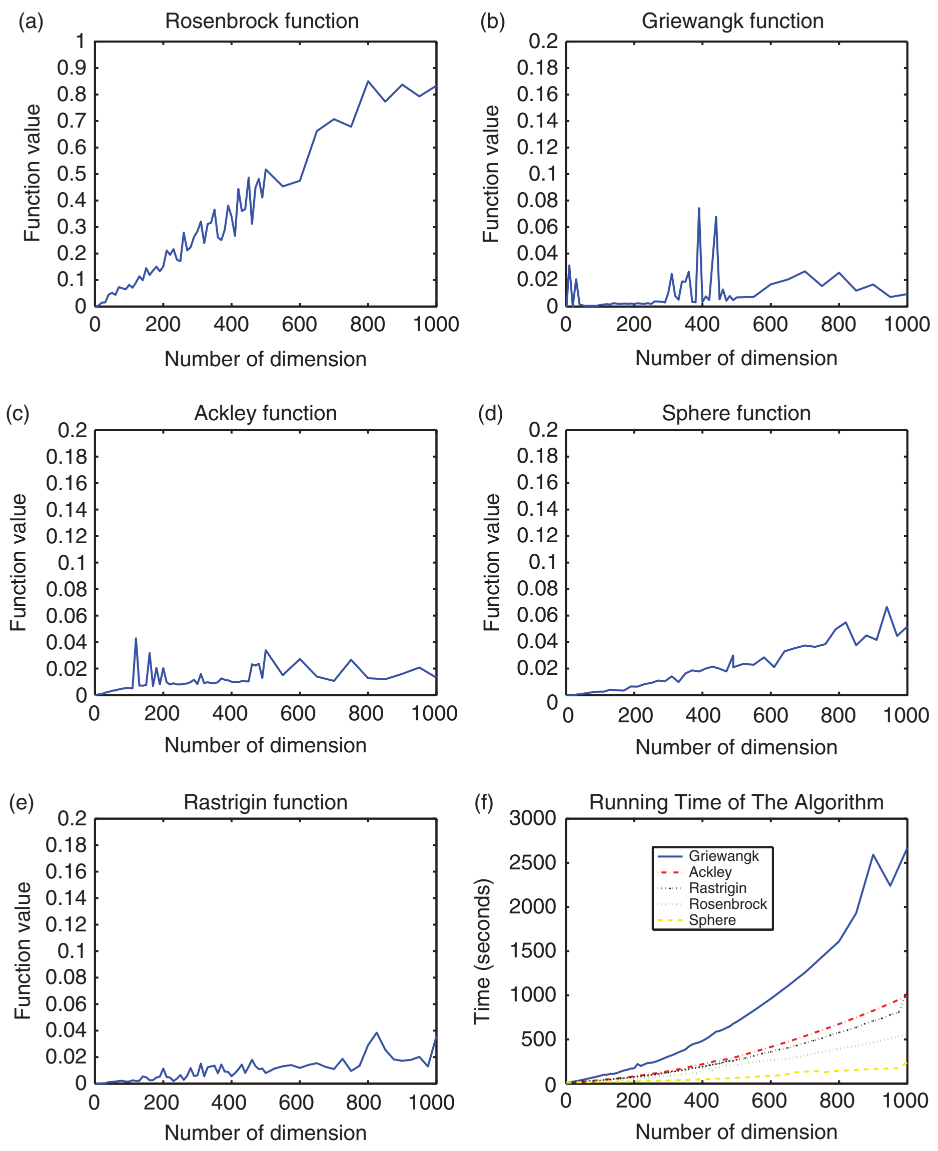


Figure 4. Results for multi-dimensional optimization test functions: for (a)–(e), the x-axis is the number of dimension, and the y-axis is the function value which is the result of the best member in the population. For (f), the x-axis is the number of dimension, and the y-axis is the running time of the algorithm. (a) This shows the result for Rosenbrock function; (b) is the result for Griewank function; (c) is the result for Ackley function; (d) is the result for Sphere function; (e) is the result for Rastrigin function; and (f) shows the CPU running time of the algorithm with respect to the dimension of the problems.

The number of iterations separately given for each function in the table were taken to be the same at all dimensions. The algorithm was implemented upon each multi-dimensional test function at sample dimensions, up to 1000. Therefore, for these optimizations, comparisons of running time of GLOA with the number of dimensions are shown in terms of seconds in Figure 4(f) (times in Figure 4(f) were taken from Matlab which was running on a laptop computer with 1.83 GHz Intel Core Duo CPU).

Multi-dimensional test functions that were used in test cases are as follows:

Rosenbrock's Banana Function [22,30–32]:

$$f_n(x) = \sum_{i=1}^{n-1}[100(x_i - x_{i+1})^2 + (x_i - 1)^2]. \tag{6}$$

The function value at $f_n(x) = 0$ is minimum for $x = (1, \ldots, 1)$. The optimization results of the algorithm for this function is shown in Figure 4(a) in which it is seen that the function values even at 1000 dimensions are always less than 1. That means the error (ϵ) is also less than 1, $\epsilon \leq 1$, for all dimensions. Since all multi-dimensional functions were tested for up to 1000 dimensions; for the following functions we will only give the function descriptions and the error terms.

Griewank's Function [22,32]:

$$f_n(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{i^{1/2}}\right). \tag{7}$$

This function is minimum at $f_n(x) = 0$, where $x = (0, 0, \ldots)$. The results are shown in Figure 4(b), and $\epsilon \leq 0.1$ for all dimensions.

Ackley's Function [22]:

$$f_n(x) = 20 + e - 20 \exp\left(-0.2\left(\frac{1}{n}\sum_{i=1}^{n} x_i^2\right)^{1/2}\right) - \exp\left(\sum_{i=1}^{n} \cos(2\pi x_i)\right). \tag{8}$$

At $f_n(x) = 0$, where $x = (0, 0, \ldots)$, the function value is minimum. The results are shown in Figure 4(c), and $\epsilon \leq 0.05$ for all dimensions.

Sphere Function [32]:

$$f_n(x) = \sum_{i=1}^{n} x_i^2. \tag{9}$$

The minimum of the function is located at $f_n(x) = 0$, where $x = (0, 0, \ldots)$. The results are shown in Figure 4(d), and $\epsilon \leq 0.1$ for all dimensions.

Rastrigin Function [22,23]:

$$f_n(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right). \tag{10}$$

At $f_n(x) = 0$, where $x = (0, 0, \ldots)$, the function value is minimum. The results are shown in Figure 4(e), and $\epsilon \leq 0.05$ for all dimensions.

4.2. *Lennard-Jones clusters*

The Lennard-Jones (LJ) potential describes the interaction of a pair of neutral particles. The total potential of interactions between N-particles is defined in Equation (11), and the pair potential is shown in Figure 5(a). Finding the minimum energy structure of interaction between pairs can be described as an optimization problem. The problem is to place the particles in 3D space such that their total interaction energy is at minimum.

Global optimization of LJ clusters is one of the most intensely studied molecular conformation problems since it is simple and accurate enough to describe real physical interacting systems. Moreover, the simulation results of LJ clusters can be directly compared with the laboratory results [11]. Because the number of local minima in the surface of an LJ cluster grows exponentially with N [33,34], the solution of the problem is very complex and difficult [2]. For instance, for the cluster of 13 particles, there are about 1500 local minima [35]. Thus, finding the minimum energy structure of LJ clusters is still an open and attractive research area and a good benchmark problem for the optimization algorithms [32].

The interaction potential is given by:

$$E = 4\epsilon \sum_{i<j}^{N} \left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right], \tag{11}$$

where r_{ij} is the distance between two particles. For simplicity we will take $\epsilon = 1$ and $\sigma = 1$ [14].

In the implementation, the solution of one cluster was used as initial guess for a larger cluster, this significantly shortened the optimization time (refer to Figure 5(b)). Here, the sum of algorithm parameters r_1 and r_2 is taken to be 1, while r_3 is set sufficiently small to locate the values of particles with high precision. Table 3 gives the values of the parameters used in the optimization. In addition, as the crossover between groups is done, the transferred elements are mutated. This mutation rate is determined also by the parameter r_3. This mutation does not change the transferred value dramatically; however, it helps the algorithm to decrease the number of iterations needed for small mutations.

The algorithm was run to locate the minimum energy structure for the clusters of 5–42, 38, 55, 65, and 75 particles. This running choice helped us to observe the growth of the computation time with respect to the number of particles. The optimization process was terminated if either the absolute error, which was defined as the absolute difference between the known lowest (reported in [36]) and the found energy level was less than 0.001, or the maximum number of iterations (3000) was reached. The algorithm has been able to find the lowest known energy structure with the absolute errors all of which are less than 0.1. The clusters of 38 and 75 particles are known to be difficult cases for optimization algorithms. For these cases the first encounter times of the algorithm from 100 random starting points (the previous structure is not used as an starting point) are shown in Figure 5(c) and (d) respectively for 38 and 75 particles. The computational time of the algorithm with respect to the cluster size

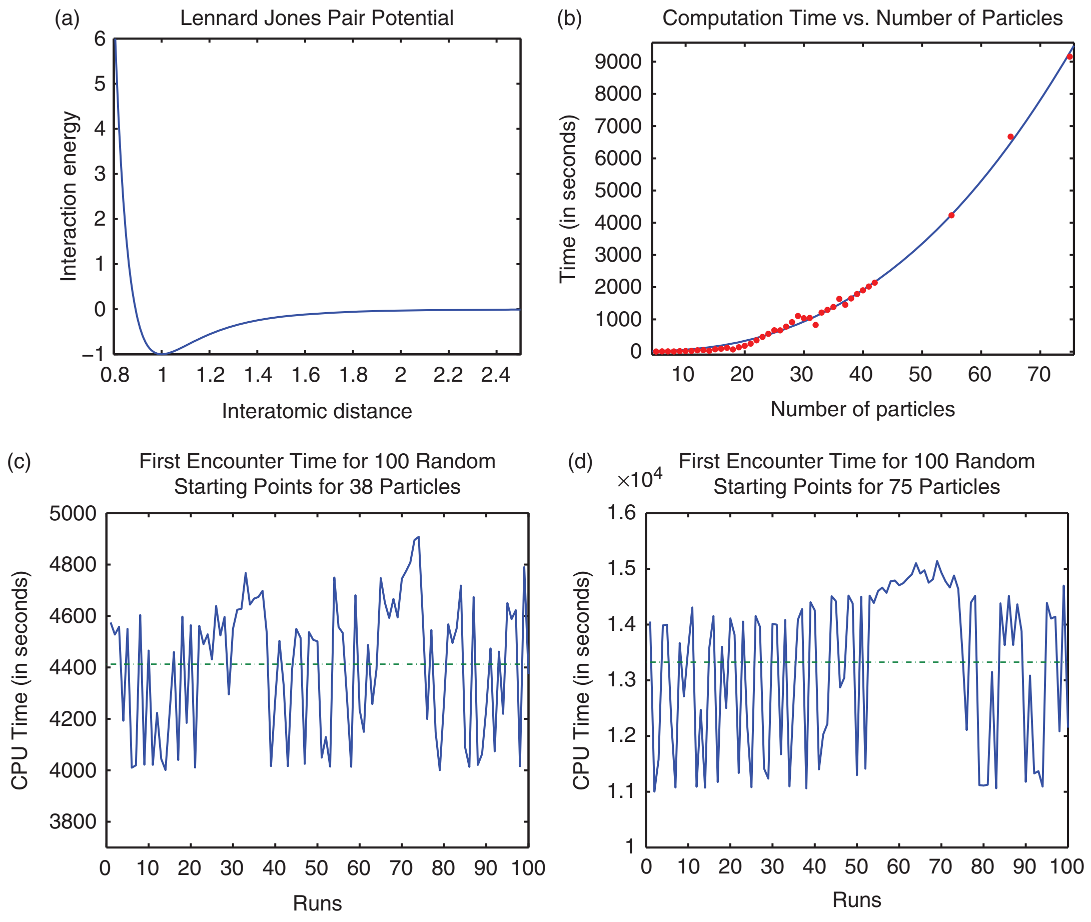


Figure 5. In (a), Lennard-Jones pair potential is shown for two particles which is the minimum at atomic distance 1. In (b), the dots are the measured CPU time for a cluster of different numbers of particles; the solid line which grows with the order of $O(n^{2.5})$ is the fitted curve for the measured data (n is the number of particles). Parts (c) and (d) show the first encounter time of the algorithm for 100 random starting points for the clusters of 38 and 75 particles.

Table 3. Parameters of the algorithm for the global optimization of LJ clusters.

Parameters	Values
Number of group	15
Number of population in each group	30
The domain of r_1	$[0.85, 0.95]$
The domain of r_2	$[0.15, 0.05]$
The domain of r_3	$[0.001, 0.0001]$
Initial generated population domain	$[-2, 2]$
Maximum number of iteration	3000

is given in Figure 5(b). Thus, the scaling of the algorithm with respect to the number of particles is $N^{2.5}$, which is an improvement over the previously reported scalings: $N^{4.7}$ for the genetic algorithm and $N^{2.9}$ for the pivot methods [16,37]. However, in terms of first encounter time, we see that GLOA in this present format is not as successful as the global optimization by basin-hoping for LJ clusters [14].

4.3. *Quantum circuit design*

In the field of quantum computation, finding quantum circuit designs by decomposing a given unitary matrix into a proper-minimum cost quantum gate sequence for the implementation of quantum algorithms and the simulation of molecular systems are of fundamental importance. Evolution of quantum circuits faces two major challenges: complex and huge search space and the high costs of simulating quantum circuits on classical computers. The optimization task involves not only finding the right sequence of gates, but the minimization of the cost of the sequence as well. In the circuit model of quantum computing, each computation or algorithm can be defined as a unitary matrix. Thus, the problem becomes the decomposition of given unitary matrices into a sequence of unitary operators which are describing the defined quantum gates. For this optimization problem, the objective function to be minimized is defined as follows:

$$y = \left|1 - \left(\alpha C + \frac{\beta}{Cost}\right)\right|, \tag{12}$$

where C is a value to determine the correctness of the circuit; $Cost$ is the implementation cost of the found circuit design; and α and β are weights to adjust the importance of the correctness and the cost of the circuit in the objective function. The correctness (C) is defined as

$$\left|\frac{Tr\left(U_g U_f^{\dagger}\right)}{N}\right|,$$

where U_g and U_f are the given and the found unitary matrices of order N, relatively; Tr(...) is the trace of a matrix; and N is the 2^n (n number of the qubits). When $U_g = U_f$, because all the diagonal elements of the product of U_g and U_f become ones, the correctness value is one (C is always in the range of one and zero). The cost of a circuit is defined as arbitrarily by considering the common implementation costs of quantum gates in different quantum computer models: the number of control qubits and the distance between the target and the control qubits of a gate in the circuits which involve many qubits. Hence, the cost of a control gate is determined by multiplying the distance (number of qubits) between the target and the control qubits of the gate by two, and the cost of a single gate is taken as one. The cost of a circuit is found by summing up the cost value of each quantum gate in the circuit.

In the optimization, α and β are considered as 0.9 and 0.1 which are the best choice among different alternatives to reduce the number of iterations and increase the correctness. The members in the population of the group structured algorithm are taken as genomes which represent a circuit as a numeric string describing gates and their target-control qubits and angles. The order of the gates in a genome represents their order in the circuit with respect to the string: the string, **2** 3 2 0.3 **3** 2 1 0.5, represents two quantum gates with the integers **2** and **3** related to their target and control qubits, and their angle values: 3 2 0.3 and 2 1 0.5, respectively.

As a test case, we use the Grover search algorithm [38] which is one of the advances quantum computing has brought on classical computing. It reduces the computational time of a brute force search from $O(N)$ to $O(N^{1/2})$. The algorithm can be described in four steps [39,40]:

(1) Start with an n-qubit initial state $|000\ldots0\rangle$.

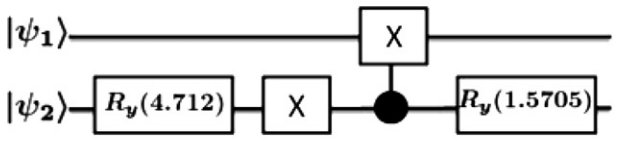


Figure 6. Quantum circuit design for the inversion about the average part of the Grover search algorithm.

(2) Put this initial state into the superposition by applying Hadamard (H) gates to the each qubit.

(3) $\lfloor(\pi/4)N^{1/2}\rfloor$ times:

- Apply the first operator U_f which is defined as:

$$U_f = I - |a\rangle\langle a|, \tag{13}$$

where a is the element that is being searched, and I is the identity. The act of U_f is to mark the element x if and only if $x = a$; the function, $f(x)$, is equal to 1 for $x = a$.

- Apply the second operator (the inversion about the average operator or diffusion operator) $U_{\psi^\perp}$ which is defined as:

$$U_{\psi^\perp} = 2|\psi\rangle\langle\psi| - I. \tag{14}$$

$U_{\psi^\perp}$ amplifies the amplitude of the state marked by the first operator and reduces the amplitudes of the rest. Thus, the probability of seeing the marked element at the end of measurement gets higher. The matrix **D** representing this operator is found as follows:

$$D_{ij} = \left\{\begin{array}{ll} \frac{N}{2}, & \text{if } i \neq j \\ -1 + \frac{N}{2}, & \text{if } i = j \end{array}\right\}. \tag{15}$$

(4) Measure the result.

The exact circuit design in Figure 6 for the second part (inversion about the average) of the Grover search algorithm (matrix elements of which are −0.5 in diagonal and 0.5 in the rest) is found with the objective function value 0.08 by applying the algorithm with the parameters given in Table 4.

5. Conclusions

In this paper we have presented the GLOA. The algorithm is quite efficient, very flexible, rarely gets trapped in local minima, does not require computationally expensive derivatives, and is quite easy

Table 4. Parameters of the algorithm for finding quantum circuits.

Parameters	Values
Number of group	15
Population in each group	25
r_1	0.8
r_2	0.1
r_3	0.1
Number of iteration	1000

to implement. Flexibility is further enhanced by the ability to incorporate any previous knowledge of the potential under investigation into the optimization. As a specific example, we used smaller (LJ) clusters as the starting point for the larger ones. Reducing the computational cost to scale approximately as $N^{2.5}$ is very promising to treat larger complex systems.

Recently, in a promising new direction, we demonstrate that a modified Grovers quantum algorithm can be applied to real problems of finding a global minimum using modest numbers of quantum bits [40]. Calculations of the global minimum of simple test functions and (LJ) clusters have been carried out on a quantum computer simulator using a modified Grovers algorithm. The number of function evaluations N reduced from $O(N)$ in classical simulation to $O(N^{1/2})$ in quantum simulation. We also show how the Grovers quantum algorithm can be combined with the classical Pivot method for global optimization to treat larger systems. We are currently investigating combining this new algorithm, the Group Leaders Optimization Algorithm, with the Grover's quantum algorithm for global optimization of complex systems. The algorithm defined here also allows us to design quantum circuits for further quantum algorithms and the simulations of molecular Hamiltonians such as the Hamiltonians of H_2O and H_2.

Therefore, because of the parallel group structure of the algorithm, it can be easily adapted to a parallel processing environment to improve the efficiency of the algorithm for hard problems. In that case, each of the groups can be run on a different machine and the parameter transfer between groups can be done through either a shared memory or a message-passing interface.

Acknowledgements

We would like to thank the Army Research Office (ARO) and the NSF Center for Quantum Information and Computation for Chemistry, award number CHE-1037992, for financial support of this project.

References

[1] T. Weise, *Global Optimization Algorithms – Theory and Application* (Thomas Weise, University of Kassel, Germany, 2007), <http://www.it-weise.de/projects/book.pdf>

[2] L.T. Wille and J. Vennik, J. Phys. A: Math. Gen. **18** (8), L419 (1985).

[3] A.B. Adib, J. Phys. A: Math. Gen. **38** (40), 8487 (2005).

[4] J.E. Dennis and R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations (Classics in Applied Mathematics)* (Society for Industrial Mathematics, Philadelphia, PA, 1996).

[5] J.C. Gilbert and C. Lemaréchal, Math. Program. **45** (3), 407 (1989).

[6] A.V. Levy and A. Montalvo, SIAM J. Sci. Stat. Comput. **6** (1), 15 (1985).

[7] D. Shalloway, J. Global Optim. **2**, 281 (1992).

[8] S. Kirkpatrick, J. Stat. Phys. **34**, 975 (1984).

[9] A.B. Finnila, M.A. Gomez, C. Sebenik, C. Stenson and J.D. Doll, Chem. Phys. Lett. **219** (5–6), 343 (1994).

[10] D.D. Frantz, D.L. Freeman and J.D. Doll, J. Chem. Phys. **93** (4), 2769 (1990).

[11] D. Cvijovicacute and J. Klinowski, Science **267** (5198), 664 (1995).

[12] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley Longman, Boston, MA, 1989).

[13] D.J. Wales and H.A. Scheraga, Science **285** (5432), 1368 (1999).

[14] D. Wales and J. Doye, J. Phys. Chem. A **101**, 5111 (1997).

[15] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms* (Oxford University Press, Oxford, 1996).

[16] P. Serra, A.F. Stanton and S. Kais, Phys. Rev. E **55** (1), 1162 (1997).

[17] J. Kennedy and R. Eberhart, in *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 4, August, 1995, pp. 1942–1948, <http://dx.doi.org/10.1109/ICNN.1995.488968>

[18] A.F. Stanton, R.E. Bleil and S. Kais, J. Comput. Chem. **18** (4), 594 (1997).

[19] P. Serra, A. Stanton, S. Kais and R. Bleil, J. Chem. Phys. **106** (17), 7170 (1997).

[20] P. Nigra and S. Kais, Chem. Phys. Lett. **305** (5–6), 433 (1999).

[21] B. Weinberg, in *2008–09 Mershon Center Research Projects (Institutions that Manage Violent Conflict)* (Mershon Center for International Security Studies, OH, USA) <http://hdl.handle.net/1811/36219>

[22] M.A. Potter and K.A.D. Jong, *A Cooperative Coevolutionary Approach to Function Optimization*, in Series *Lecture Notes in Computer Science* (Springer-Verlag, London, 1994), Vol. 866, pp. 249–257.

[23] F. van den Bergh and A.P. Engelbrecht, IEEE Trans. Evol. Comput. **8** (3), 225 (2004).

[24] E. Cantu-Paz, *Efficient and Accurate Parallel Genetic Algorithms* (Kluwer Academic Publishers, Norwell, MA, 2000).

[25] M.A. Potter and K.A. De Jong, Evol. Comput. **8** (1), 1 (2000).

[26] Z. Yang, K. Tang and X. Yao, Inf. Sci. (N. Y.) **178** (15), 2985 (2008).

[27] G. Winter, D. Geiner and B. Galvan, *Parallel Evolutionary Computation* (CEANI (Evolutionary Computation and Applications) Division of the Institute of Intelligent Systems and Numerical Applications in Engineering (IUSIANI) (University of Las Palmas de Gran Canaria, Spain , 2005).

[28] G.W. Turner, E. Tedesco, K.D.M. Harris, R.L. Johnston and B.M. Kariuki, Chem. Phys. Lett. **321** (3-4), 183 (2000).

[29] R. Storn and K. Price, J. Global Optim. **11**, 341 (1997).

[30] J.J. Moré, B.S. Garbow and K.E. Hillstrom, ACM Trans. Math. Software **7** (1), 17 (1981).

[31] R. Chelouah and P. Siarry, Eur. J. Oper. Res. **148** (2), 335 (2003).

[32] P.J. Angeline, in *EP '98: Proceedings of the 7th International Conference on Evolutionary Programming VII* (Springer-Verlag, London, 1998), pp. 601–610.

[33] F.H. Stillinger and T.A. Weber, Science **225** (4666), 983 (1984).

[34] D.J. Wales and J.P.K. Doye, J. Chem. Phys. **119**, 12409 (2003).

[35] J.P.K. Doye, M. Miller and D. Wales, J. Chem. Phys. **111**, 8417 (1999).

[36] D.J. Wales, J.P.K. Doye, A. Dullweber, M.P. Hodges, F.Y. Naumkin, F. Calvo, J. Hernández-Rojas and T.F. Middleton, *The Cambridge Cluster Database* <http://www-wales.ch.cam.ac.uk/CCD.html> (2010).

[37] S.K. Gregurick, M.H. Alexander and B. Hartke, J. Chem. Phys. **104** (7), 2684 (1996).

[38] L.K. Grover, *STOC '96: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing* (ACM, New York, 1996), pp. 212–219, Philadelphia, PA.

[39] M.A. Nielsen and I.L. Chuang, *Quantum Computation and Quantum Information*, 1st ed (Cambridge University Press, Cambridge, 2000).

[40] J. Zhu, Z. Huang and S. Kais, Mol. Phys. **107**, 2015 (2009).