An Evolutionary Algorithm With Guided Mutation for the Maximum Clique Problem

Qingfu Zhang*, Member, IEEE*, Jianyong Sun, and Edward Tsang*, Member, IEEE*

***Abstract*—Estimation of distribution algorithms sample new solutions (offspring) from a probability model which characterizes the distribution of promising solutions in the search space at each generation. The location information of solutions found so far (i.e., the actual positions of these solutions in the search space) is not directly used for generating offspring in most existing estimation of distribution algorithms. This paper introduces a new operator, called guided mutation. Guided mutation generates offspring through combination of global statistical information and the location information of solutions found so far. An evolutionary algorithm with guided mutation (EA/G) for the maximum clique problem is proposed in this paper. Besides guided mutation, EA/G adopts a strategy for searching different search areas in different search phases. Marchiori's heuristic is applied to each new solution to produce a maximal clique in EA/G. Experimental results show that EA/G outperforms the heuristic genetic algorithm of Marchiori (the best evolutionary algorithm reported so far) and a MIMIC algorithm on DIMACS benchmark graphs.**

***Index Terms*—Estimation of distribution algorithms, evolutionary algorithm, guided mutation, heuristics, hybrid genetic algorithm, maximum clique problem (MCP).**

I. Introduction

EVOLUTIONARY algorithms such as genetic algorithms (GAs), scatter search [1], and estimation of distribution algorithms (EDAs) [2]–[9] work with a population of solutions and combine them to generate new solutions (offspring), which may be further repaired or improved by other heuristic operators such as local search. Crossover and mutation are the main operators for generating offspring in conventional GAs. These two operators directly act on selected solutions (parents). The crossover operator often applies to a pair of parents and swaps parts of these two parents to produce offspring. The mutation operator randomly alters part of a parent solution to produce an offspring. Scatter search uses linear combinations of selected solutions with heuristic improvement and a rounding process for generating new solutions. The information about the locations of the solutions found so far (i.e., the actual positions of these solutions in the search space) are directly used in both GAs and scatter search. EDAs generate offspring in a quite different way. They maintain a probability model which characterizes the distribution of promising solutions at each iteration (i.e., generation). The probability model is updated, based on the global statistical information extracted from the current population. Offspring are generated by sampling from this model. The global information about the search space collected from the search so far is used to produce offspring in EDAs. However, the information about the locations of the solutions found so far is not directly used to guide the search. Very recently, Peña *et al.* [33] proposed a hybrid algorithm in which some offspring are generated by crossover and mutation while the others by EDA operators. Their preliminary experimental results were very promising.

In this paper, we propose a new offspring generating operator, called guided mutation, for evolutionary algorithms. Guided mutation can be regarded as a combination of the conventional mutation operator and the EDA offspring generating scheme. Guided by a probability model, guided mutation alters a parent solution to produce a new solution. The resultant solution can (hopefully) fall in or close to a promising area which is characterized by the probability model. Meanwhile, it directly takes a user-specified percentage of elements from its parent, which is often a better solution found so far. In such a way, the similarity between an offspring and its parent can be controlled to some extent.

A clique of a graph is a set of pairwise adjacent nodes. A maximal clique is a clique which is not a proper subset of any other clique. A maximum clique is a clique with the maximum cardinality (which is called the maximum clique number). A maximum clique is maximal but not *vice versa*. Given a graph, the maximum clique problem (MCP) is to find a maximum clique. The MCP is one of the best known problems of combinatorial optimization. In many applications such as coding theory, fault diagnosis in multiprocessor systems, constraint satisfaction, computer vision, and mobile networks, the underlying problem can be formulated as an instance of the MCP. The MCP is NP-complete. Moreover, there is no polynomial-time algorithm for approximating the maximum clique within a factor of $n^{1-\varepsilon}$ unless $P = \mathrm{NP}$ [10], where n is the number of the nodes of the graph. These facts indicate that the MCP is very difficult to solve. An excellent overview of the algorithms, complexity and applications of the MCP can be found in [11].

Due to the inherent difficulty and importance of the MCP, considerable efforts have been devoted to developing heuristics for it. The aim of heuristics is to find a maximal clique which is as large as possible. Since there is no sound theory about how and why heuristics work, the comparison of the performances of different heuristics is mainly based on extensive experimentation. A set of benchmark graphs from different applications have been collected for this purpose and these graphs are available at http://dimacs.rutgers.edu/Challenges/.

Manuscript received January 27, 2004; revised October 20, 2004. This work was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) under Grant GR/R64742/01.

The authors are with the Department of Computer Science, University of Essex, Wivenhood Park, Colchester CO4 3SQ, U.K. (e-mail: qzhang@essex.ac.uk; jysun@essex.ac.uk; edward@essex.ac.uk).

Digital Object Identifier 10.1109/TEVC.2004.840835

Several attempts have been made to solve the MCP with evolutionary algorithms. A scatter search algorithm for the MCP has been proposed by Cavique *et al.* [12]. The work of Cater and Park [13], [14] shows that a simple GA is not effective for the MCP, which implies that GAs need to be customized or to incorporate other techniques in order to improve their performances for this problem. In fact, the strengths and weaknesses of the simple GA have been studied theoretically (e.g., [3], [15]–[18], [34]). A fitness function with a graded penalty term for penalizing infeasible solutions [19], a modified genetic operator [20], [21], local search operator [22], [23] and problem-specific representation [24] have been used in GAs for the MCP. In the heuristic based genetic algorithm (HGA) proposed by Marchiori [25], [26], a naive greedy heuristic procedure is applied to every new solution (a subset of the nodes in the graph) generated by uniform crossover or mutation. This greedy heuristic first enlarges the subset of nodes by adding some nodes randomly selected, then reduces it to a clique and finally enlarges it to a maximal clique. The HGA algorithm of Marchiori outperforms other existing GAs and the scatter search of Cavique *et al.* [12] on the DIMACS benchmark graphs in terms of quality of solutions and speed [26].

This paper proposes an evolutionary algorithm with guided mutation (EA/G) for the MCP. The main features of EA/G are as follows.

- Guided mutation: New solutions are generated by applying the guided mutation operator to the best solution in the current population, the resultant solutions will be not far from the best solution found so far and fall into a promising area.
- Heuristic repair: Since a new solution generated by the guided mutation operator may not be a clique, the heuristic repair operator of Marchiori is applied to every new solution to generate a clique.
- Partitioning of the search space: The whole search space is divided into several search areas, the algorithm focuses on different search areas in different phases.

The rest of this paper is organized as follows. Section II introduces guided mutation. EA/G for the MCP is presented in Section III. Experimental study and comparisons are given in Section IV. We conclude the paper in Section V.

II. Guided Mutation Operator

One of the key issues in the design of evolutionary algorithms is how to generate offspring. The proximate optimality principle [27], an underlying assumption in most (if not all) heuristics, assumes that good solutions have similar structure. This assumption is reasonable for most real-world problems, e.g., the percentage of common edges in any two locally optimal solutions of a traveling salesman problem obtained by the Lin–Kernighan method is about 85% on average [28]. Based on this assumption, an ideal offspring generator should be able to produce a solution which is close to the best solutions found so far. Suppose the current population in an evolutionary algorithm with local search consists of the best locally optimal solutions found so far, a new solution generated by the conventional mutation is close (similar) to its parent, but may be far away from other better solutions since the mutation does not utilize any global information extracted from the current population. EDAs extract global statistical information from the previous search and then represent it as a probability model, which characterizes the distribution of promising solutions in the search space. New solutions are generated by sampling from this model. However, the location information of the locally optimal solutions found so far has not been directly used in EDAs, there is no mechanism to directly control the similarity between new solutions and a given solution. The idea behind the proposed operator which we call *guided mutation* is to combine the global statistical information and location information of the solutions found so far to overcome the shortcoming of GAs and EDAs.

Several different probability models have been introduced in EDAs for modeling the distribution of promising solutions. The univariate marginal distribution (UMD) model is the simplest one and has been used in univariate marginal distribution algorithm [3], population-based incremental learning [2], and compact GA [29]. Let the search space be $\Omega = \{0,1\}^n$, UMD model uses a probability vector $p = (p_1, \ldots, p_n) \in [0,1]^n$ to characterize the distribution of promising solutions in the search space, where p_i is the probability that the value of the ith position of a promising solution is one. The guided mutation operator uses a probability vector $p \in [0,1]^n$ to guide to mutate an $x \in \{0,1\}^n$ in the following way:

```
Guided mutation Operator $y = GM(p, x, \beta)$
Input: $p = (p_1, \ldots, p_n) \in [0,1]^n, x = (x_1, \ldots x_n) \in \{0,1\}^n$
  and $\beta \in [0,1]$.
Output: $y = (y_1, \ldots, y_n) \in \{0,1\}^n$.
For $i := 1$ to $n$ do
  Flip a coin with head probability $\beta$;
  If the head turns up, with probability $p_i$ set $y_i = 1$, otherwise
  set $y_i = 0$,
  Otherwise, $y_i := x_i$.
End For
```

Remark 1: In the above guided mutation operator, y_i is directly copied from the parent x or randomly sampled from the probability vector p. The larger β is, the more elements of y are sampled from the probability vector p. In other words, β, similar to the mutation rate in conventional mutation, controls the similarity between offspring and the parent, while the parent can be chosen from the best solutions found so far.

Remark 2: In the correlated mutation [30] for real vectors, the probability of generating an offspring in the steepest ascent direction is larger than in other directions. In the conventional mutation for binary strings, the probability of a vector y being generated from the parent vector x is entirely determined by the Hamming distance between x and y. The guided mutation operator can be regarded as a discrete counterpart of the correlated mutation. The probability vector p in the guided mutation can be learned and updated at each generation for modeling the distribution of promising solutions. Since some elements of the offspring y are sampled from the probability vector p, it can be expected that y should fall in or close to a promising area. Meanwhile, this sampling also provides diversity for the search afterwards.

III. Algorithm for the MCP

This section describes in detail the main components of the EA/G for the MCP. In EA/G, each potential solution to the MCP is encoded as a binary string. At each generation t, EA/G maintains a population $\mathrm{Pop}(t)$ of N binary strings and a probability vector p. The whole search procedure is divided into several phases, the search will focus on different search areas during different phases. New solutions are generated by the guided mutation operator or by sampling randomly from a particular search area.

A. Representation and Fitness

Given a graph $G = (V, E)$ where $V = \{1, 2, \dots, n\}$ is its node set and $E \subset V \times V$ is its edge set. A set of nodes $U \subset V$ is encoded as a string $x = (x_1, x_2, \dots, x_n) \in \{0,1\}^n$ where $x_i = 1$ if and only if node i is in U. Therefore, the search space is $\Omega = \{0,1\}^n$. The fitness of x is defined as its cardinality if x represents a clique. Since every new solution generated by the guided mutation will be repaired, there is no need to define fitness values for infeasible solutions.

B. Partitioning of the Search Space

The search space, $\Omega = \{0,1\}^n$ for the MCP can be divided into $n+1$ subspaces as follows:

$$\Omega = \Omega_0 \bigcup \Omega_1 \cdots \bigcup \Omega_n$$

where $\Omega_i = \{x = (x_1, x_2, \dots, x_n) : \sum_{j=1}^{n} x_j = i\}$. Obviously, for any Ω_i and Ω_j, we have $\Omega_i \bigcap \Omega_j = \emptyset$. Any string in Ω_i represents a set of nodes with cardinality i.

Our goal is to find a clique as large as possible. If we have found a maximal clique with cardinality ℓ, it is not necessary to explore subspaces $\Omega_i, 1 \le i \le \ell$ afterwards. Moveover, if ℓ was found after a considerable amount of effort, it is unwise to search these Ω_i with $i \gg \ell$, since according to the proximate optimality principle, it is often unlikely that the maximum clique number is far bigger than ℓ.

The proposed algorithm explores different subspaces in the search space during different search phases. At the beginning, a lower bound *low* of the maximum clique number is computed. The first search phase will be for the area $\bigcup_{i=\mathrm{low}+1}^{\mathrm{low}+\Delta} \Omega_i$, where Δ is a predefined integer number. If a maximal clique with a larger cardinality s is found, then the current search phase will end and the search area for next phase will be $\bigcup_{i=s+1}^{s+\Delta} \Omega_i$. In such a way, the search is focused on these promising Ω_i.

C. Repair Heuristic

We can induce a set of nodes from any string in $\{0,1\}^n$. This set of nodes, however, may not be a clique. To produce a clique, we use Marchiori's heuristic [25], [26] to repair it.

```
Repair Operator $S = \mathrm{Repair}(\alpha, U)$
Input: $U \subset V$: a set of nodes, $\alpha \in (0,1)$.
Output: $S$: a clique in $G$.
Step 1) Extraction
    Step 1.1) Set $W := U, S := U$.
    Step 1.2) If $W = \emptyset$, go to Step 2). Otherwise
              randomly pick a node $k$ from $W$ and
              remove $k$ from $W$.
    Step 1.3) Flip a coin with head probability $\alpha$. If the
              head turns up, remove $k$ from $S$, else
              remove from $S$ and $W$ all the nodes
              in $S$ that are not connected to $k$.
    Step 1.4) Go to Step 1.2).
Step 2) Extension
    Step 2.1) Set $W := V \backslash S$.
    Step 2.2) If $W = \emptyset$, Stop and return $S$. Otherwise
              randomly pick up a node $k$ from $W$ and
              remove it from $W$.
    Step 2.3) If node $k$ is connected to all nodes in $S$,
              then add $k$ to $S$.
    Step 2.4) Go to Step 2.2).
```

In the above repair heuristic operator, S will be a clique after Step 1). Generally speaking, α should be very small. Otherwise, S would be very small after Step 1) since many nodes in S may be removed. Step 2) simply extends S to a maximal clique.

D. Initialization and Update of the Probability Vector

The probability vector p for guided mutation (Section II above) needs to be initialized and updated in EA/G.

Suppose that the current population $\mathrm{Pop}(t)$ has N binary strings $x^j = (x_1^j, \dots, x_n^j), j = 1, 2, \dots, N, p = (p_1, \dots, p_n)$ will be initialized as

$$p_i = \frac{\sum_{j=1}^{N} x_i^j}{N} \tag{1}$$

at the beginning of each search phase or if the search needs to be restarted. p_i is the percentage of the binary strings with the value of the ith element being one in $\mathrm{Pop}(t)$. p can also be regarded as the center of $\mathrm{Pop}(t)$.

At each generation t in EA/G, some binary strings are selected from the current population $\mathrm{Pop}(t)$ to form the parent set $\mathrm{Parent}(t)$, which is then used for updating the probability vector p. Let $\mathrm{Parent}(t)$ contain M strings $y^j = (y_1^j, \dots, y_n^j), j = 1, 2, \dots, M$, the probability vector p is updated in the same way as in the PBIL algorithm [2]

$$p_i := (1-\lambda)p_i + \lambda \frac{\sum_{j=1}^{M} y_i^j}{M} \tag{2}$$

for $i = 1, 2, \dots, n$. $\lambda \in (0,1]$ is the learning rate. The bigger λ is, the more contribution of the strings in $\mathrm{Parent}(t)$ to the updated probability vector. Learning brings the probability vector p toward the center of the parent set.

E. Algorithm

EA/G works as follows.

```
Step 0) Set the following parameters: the population size $N$
        in the main algorithm ($N$ should be a even number),
        $\Delta$ used in the partitioning of the search space, $\lambda$
        used in the update of the probability vector, $\beta$ used
        in the guided mutation, and $\alpha$ used in the repair
        operator.
Step 1) Set $t := 0$, randomly pick an $x \in \Omega$ and apply the
        repair operator to $x$ to obtain a maximal clique $U$.
        Set $low := |U|$.
```

Step 2) Randomly pick N strings from $\cup_{i=\mathrm{low}+1}^{\mathrm{low}+\Delta}\Omega_i$ and apply the repair operator to each of them, the N resultant strings form $\mathrm{Pop}(t)$. Then, initialize the probability vector p by (1).

Step 3) Select the $N/2$ best strings from $\mathrm{Pop}(t)$ to form the parent set $\mathrm{Parent}(t)$, and then update the probability vector p by (2).

Step 4) Apply the guided mutation operator to the fittest string in $\mathrm{Parent}(t)$ $N/2$ times, and then apply the repair operator to the resultant strings to get $N/2$ cliques. Add these $N/2$ cliques to $\mathrm{Parent}(t)$ to form $\mathrm{Pop}(t+1)$. If the stopping condition is met, return the largest clique found so far.

Step 5) Set $t := t+1$. Let S be the largest clique in $\mathrm{Pop}(t)$. If $|S| > low$, set $low := |S|$ and go to Step 2).

Step 6) If all the strings in $\mathrm{Pop}(t)$ are identical, go to Step 2), else go to Step 3).

In Step 1), a lower bound *low* for the maximum clique number is obtained. The population $\mathrm{Pop}(t)$ and the probability vector p are initialized in Step 2) for the search on $\cup_{i=\mathrm{low}+1}^{\mathrm{low}+\Delta}\Omega_i$. Step 3) selects the fittest strings to become parents and then update p. In Step 4), $N/2$ strings are generated by applying the guided mutation operator to the fittest string in the current parent set. The mutated strings are then repaired. These resultant strings join their parents to form the population of the next generation. If a larger clique is found (Step 5), the search will move to a new area in the search space. If all the members in the current population are identical (Step 6), the search will be restarted to diversify the population.

The reasons for selecting only the largest clique (i.e., the fittest clique) to be mutated in Step 4) are twofold. First, it is desirable that the new cliques are similar to the largest clique found so far according to the proximate optimality principle. Second, the other cliques found so far will contribute to the new cliques via the probability vector p.

IV. Computational Experiments

In this section, we study EA/G experimentally and compare it with HGA [26], the best GA for the MCP, on the DIMACS benchmark problems. We also compare the performances of EA/G with MIMIC [31] on these test problems. EA/G has been implemented in C. As mentioned earlier, the parameter α in the repair operator should be very small. Following the suggestion in [26], we set $\alpha = 0.001$. Generally speaking, the lower bound obtained in Step 1) should not be very far from the maximum clique number according to the proximate optimality principle. Therefore, Δ should be set to a small number. Δ is set to 3 in all the experiments in this paper. We set the population size N to 10 as Marchiori did in HGA [26]. The algorithm stops after 20 000 calls of the repair operator. All the experiments were performed on AMD2400 (2000 MHz).

A. *Effects of λ and β*

Apart from α, Δ and N, the parameters in EA/G are λ and β. λ is the learning rate in the update of the probability vector p. β is used in the guided mutation operator for balancing the tradeoff between the global information and location information of solutions found so far.

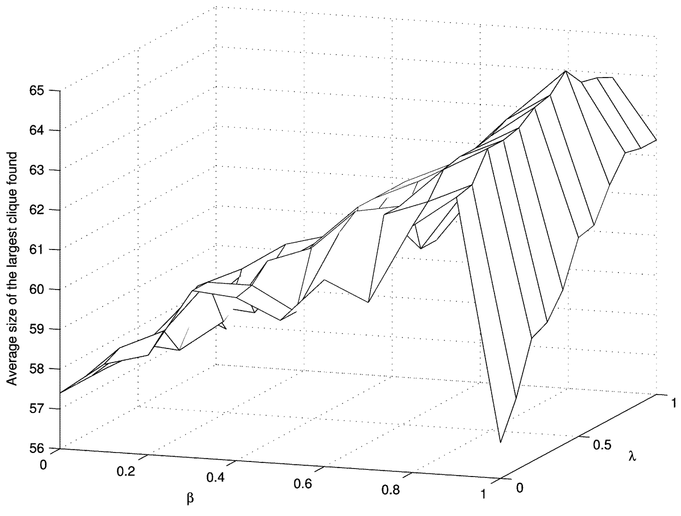

Fig. 1. Surface of the average sizes of the largest cliques found in ten runs on C1000.9.

To assess the effects of λ and β on the performance of EA/G, EA/G is tested on the DIMACS benchmark problem C1000.9 (random graph with 1000 nodes and a density of 0.9) for $\lambda = 0, 0.1, 0.2, \ldots, 1$ and $\beta = 0, 0.1, 0.2, \ldots, 1$. Fig. 1 shows the resulting surface of the average size of the largest cliques found in ten independent runs for different parameter settings of λ and β. Table I lists the corresponding data.

We conclude the following from the results shown in Fig. 1.

- When $\beta = 1$, no location information is utilized in the guided mutation operator. As can be seen, the algorithm with $\beta = 0.9$ performs much better than $\beta = 1$. This phenomenon clearly indicates that the location information does contribute positively to the performance of the algorithm.
- Given (2), the global statistical information collected in the past makes no contribution to the new probability vector when $\lambda = 1$. As λ decreases, the contribution of the old global statistical information will increase. Fig. 1 shows that the algorithm with $\lambda = 1$ is always worse than $\lambda = 0.7$. This suggests that the use of the global statistical information collected in the past can be beneficial.

B. *Contributions of the Main Components*

The main difference between EA/G and Marchiori's HGA is that EA/G uses the guided mutation operator instead of the uniform crossover operator for generating new offspring. Besides this, EA/G has its strategy for searching different areas in different search phases. A very natural question is whether the guided mutation and the partitioning of the search space have any positive contribution to the performance of the algorithm. To answer this question, we compare the performance of EA/G on C1000.9 with the following two algorithms.

- Marchiori's HGA: It uses the uniform crossover operator for generating offspring. Partitioning of the search space is not used in her algorithm.
- Evolutionary algorithm with uniform crossover and partitioning of the search space (EA/UX/P): It is the same as

TABLE I
EFFECT OF λ AND β: THE AVERAGE SIZE OF THE LARGEST CLIQUES FOUND FOR DIFFERENT PARAMETER SETTING

$\lambda\backslash\beta$	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.0	57.4	58.2	58.2	59.2	60.1	59.6	60.7	60.2	62.2	63.0	56.9
0.1	57.4	58.3	58.2	59.2	60.2	59.6	61.1	62.2	62.6	63.5	57.8
0.2	57.4	58.3	58.2	59.0	60.3	61.0	61.1	62.3	62.2	63.8	59.1
0.3	57.4	58.0	58.1	59.2	60.2	59.6	61.2	62.3	63.2	63.8	59.3
0.4	57.4	58.2	58.9	59.2	60.2	60.9	61.2	62.1	63.3	63.9	59.9
0.5	57.4	58.4	58.1	59.3	60.4	61.4	61.2	62.0	63.4	64.2	61.0
0.6	57.4	58.4	59.2	59.2	60.4	61.4	61.2	62.3	63.6	64.3	61.1
0.7	57.4	58.6	59.4	59.5	60.5	61.4	61.5	62.7	63.8	64.4	62.5
0.8	57.4	58.5	59.1	59.4	60.3	61.4	61.2	62.5	63.4	64.1	62.3
0.9	57.4	58.2	59.1	59.2	60.2	59.5	61.0	62.0	63.3	63.9	62.4
1.0	57.4	57.9	58.9	59.1	60.2	59.5	60.9	61.8	63.1	63.9	62.4

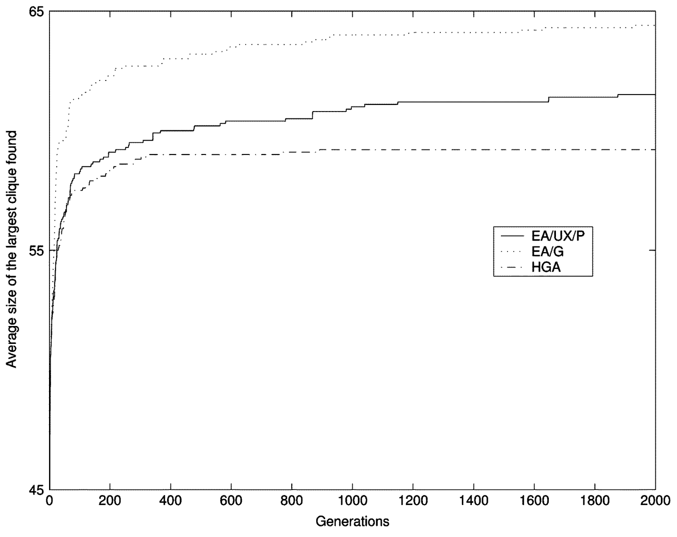

Fig. 2. Comparisons of EA/UX/P, EA/G, and HGA on C1000.9.

EA/G, except that it uses the uniform crossover instead of the guided mutation.

The population size for all these three algorithms is set to 10. λ is set to 0.7 and β to 0.9 in EA/G, based on the experimental results in Section IV-A. Ten independent runs for each algorithm were performed. The average size of the largest cliques found at each generation over the ten runs was recorded for each algorithm. Fig. 2 plots the evolution of the average size of the largest cliques for these three algorithms.

As can be seen, EA/UX/P performs better than Marchiori's HGA algorithm. Note that the main difference between these two algorithms is that EA/UX/P adopts the strategy for partioning the search space. Therefore, this experiment supports our claim that the partitioning of the search space could improve the performance of the algorithm.

Fig. 2 also shows that EA/G outperforms EA/UX/P, which indicates that the guided mutation operator performs better than the uniform crossover operator for this test problem. These results are not surprising since the uniform crossover operator only uses the location information of the selected parents while the guided mutation operator combines the global statistical information with the location information of the parents.

When $\lambda = 1$ and $\beta = 0$, EA/G generates its offspring in the same way as UMDA, while HGA uses uniform crossover [30]. We can observe from Figs. 1 and 2 that the performance of EA/G with $\lambda = 1$ and $\beta = 0$ is worse than that of HGA on C1000.9. Although it has been shown that the behaviors of a GA with uniform crossover and UMDA are very similar in the case of large populations [3], our observation suggests that the differences between UMDA and a GA with uniform crossover can be significant in the case of a small population. In uniform crossover, the Hamming distance between two offspring is always the same as that between their parents, and an offspring takes at least 50% elements from one of its parent. In contract, an offspring generated in UMDA can be far away from any members in the current population.[1]

We have recorded the number of the times that Step 2) was executed in EA/G for C1000.9. On average, Step 2) was executed 54 times, out of which about 12 times were due to that a larger clique was found in Step 5).

C. Comparisons With HGA

HGA [26] is the best evolutionary algorithm for the MCP reported so far. EA/G was compared with HGA on 37 DIMACS graphs. These graphs are the following:

- $\mathrm{C}x \cdot y$ and $\mathrm{DSJC}x \cdot y$: random graph of size x and density $0.y$;
- MANN: Steiner triple graph with up to 3321 nodes and 5 506 380 edges;
- $\mathrm{brock}x\_2$ and $\mathrm{brock}x\_4$: Brockington graph of size x;
- $\mathrm{gen}x\_\mathrm{p}0.9\_z$: Sanchis graph of size x;
- hamming: Hamming graph;
- keller: Keller graph with up 3361 nodes and 4 619 898 edges;
- $\mathrm{p\_hat}x - \mathrm{z}$: P-hat graphs of size x.

The two algorithms were run independently for ten times on each graph. The population size for both algorithms is ten. Both algorithms were terminated after 20 000 calls of the repair operator. λ is set to 0.7 and β to 0.9 in EA/G. Table II shows the experimental results including:

- *Best*: the size of the largest clique found in ten runs;
- *Avg*: the average size of the cliques found in ten runs;
- *std*: the standard deviation of the sizes of the cliques found in ten runs;
- *DIMACS*: the size of the largest clique found by the algorithms at the second DIMACS challenges;
- *time*: the run time (in seconds) of each algorithm;
- *t-test*: the result of the one-tailed t-test at the 0.05 significance level for the alternative hypothesis that the mean size of the cliques obtained by EA/G is larger than that obtained by HGA. t is the absolute value of the t statistic. sig. < 0.05 suggests that EA/G is better than HGA in terms of solution quality.

The running time of EA/G is about 16% longer than that of HGA mainly due to the computational overheads in the guided mutation operator and partitioning of the search space. However, this extra time is clearly compensated by the quality of the solutions produced. The t-test results suggest that EA/G outperforms HGA on 31 out of 37 graphs in terms of the mean size of

[1]For example, consider a population containing (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), and (0, 0, 0, 1), the probability vector in UMDA learned from this population will be (0.25, 0.25, 0.25, 0.25). Vector (1, 1, 1, 1) may be sampled from this probability vector as an offspring, whose Hamming distance from any members in the current population is 3. It is impossible for uniform crossover to produce such an offspring.

TABLE II

Comparison Results Between HGA and EA/G. *Best*: The Size of the Largest Clique Found, *Avg*: The Average Size of the Cliques Found, *std*: The Standard Deviation of the Sizes of the Cliques Found, *DIMACS*: The Size of the Largest Clique Found by the Algorithms, *time*: The Run Time (in Seconds) of Each Algorithm. *t-test*: The t-Test Results

Graph	HGA			EA/G			t-test		
	Avg(std)	Best	time	Avg(std)	Best	time	t	sig.	DIMACS
C125.9	33.8(0.4)	34	1.3	34.0(0.0)	34	1.5	1.500	0.084	34
C250.9	42.8(0.7)	44	2.4	44.0(0.0)	44	2.9	5.582	0.000	44
C500.9	52.2(1.6)	56	4.9	55.2(0.9)	56	5.7	6.018	0.000	57
C1000.9	61.6(2.1)	66	17.4	64.4(1.4)	**67**	21.2	6.532	0.000	68
C2000.9	68.2(2.4)	72	39.3	70.9(1.0)	72	45.2	3.693	0.003	78
DSJC500.5	12.2(0.4)	13	3.5	13.0(0.0)	13	4.7	4.583	0.000	14
DSJC1000.5	13.3(0.5)	14	9.2	14.5(0.3)	**15**	12.1	4.714	0.000	15
C2000.5	14.2(0.4)	15	23.6	14.9(0.7)	**16**	28.6	2.333	0.023	16
C4000.5	15.4(0.5)	16	50.8	16.1(0.3)	**17**	61.0	2.689	0.013	18
MANN_a27	125.6(0.5)	126	8.8	126.0(0.0)	126	12.1	2.449	0.018	126
MANN_a45	342.4(0.5)	343	74.3	343.7(0.7)	**345**	80.2	7.649	0.000	345
MANN_a81	1096.3(0.6)	1097	720.5	1097.2(0.6)	**1098**	829.5	2.602	0.025	1098
brock200_2	10.5(0.7)	12	1.4	12.0(0.0)	12	1.8	4.993	0.000	12
brock200_4	15.4(0.5)	16	1.4	16.5(0.5)	**17**	2.0	3.973	0.001	17
brock400_2	22.5(0.7)	24	3.0	24.7(0.4)	**25**	3.7	7.584	0.000	29
brock400_4	23.6(0.8)	25	3.0	25.1(2.6)	**33**	3.9	1.893	0.045	33
brock800_2	19.3(0.6)	20	7.6	20.1(0.4)	**21**	8.9	5.014	0.000	21
brock800_4	18.9(0.5)	20	7.8	19.9(0.5)	**21**	8.9	2.372	0.021	21
gen200_p0.9_44	39.7(1.6)	44	19.9	44.0(0.0)	44	25.1	5.763	0.000	44
gen200_p0.9_55	50.8(6.4)	55	24.0	55.0(0.0)	55	30.9	6.584	0.000	55
gen400_p0.9_55	49.7(1.2)	55	38.9	51.8(0.7)	55	45.2	6.866	0.000	55
gen400_p0.9_65	53.7(7.4)	65	41.8	65.0(0.0)	65	47.2	5.714	0.000	65
gen400_p0.9_75	60.2(12.1)	75	38.3	75.0(0.0)	75	46.4	4.572	0.000	75
hamming8-4	16.0(0.0)	16	1.8	16.0(0.0)	16	2.0	-	-	16
hamming10-4	37.7(1.9)	40	14.6	39.8(0.6)	40	16.7	3.083	0.006	40
keller4	11.0(0.0)	11	1.2	11.0(0.0)	11	1.5	-	-	11
keller5	26.0(0.8)	27	8.3	26.9(0.3)	27	10.7	3.250	0.005	27
keller6	51.8(1.5)	55	56.0	53.4(1.2)	**56**	63.1	3.491	0.017	59
p_hat300-1	8.0(0.0)	8	2.0	8.0(0.0)	8	2.4	-	-	8
p_hat300-2	25.0(0.0)	25	2.2	25.0(0.0)	25	2.3	-	-	25
p_hat300-3	34.6(0.9)	36	2.5	36.0(0.0)	36	2.7	3.674	0.003	36
p_hat700-1	9.8(0.9)	11	5.0	11.0(0.0)	11	6.6	4.811	0.000	11
p_hat700-2	43.5(0.8)	44	7.5	44.0(0.0)	44	8.9	1.809	0.052	44
p_hat700-3	60.4(1.0)	62	9.2	62.0(0.0)	62	13.1	7.236	0.000	62
p_hat1500-1	10.8(0.4)	11	14.5	11.1(0.3)	**12**	19.8	3.000	0.007	12
p_hat1500-2	63.8(1.0)	65	23.4	65.0(0.0)	65	28.9	4.993	0.000	65
p_hat1500-3	92.1(0.9)	94	28.8	93.7(0.5)	94	34.3	3.748	0.003	94

- the t-test has not been made for these graphs since both EA/G and HGA found the optimal solutions in each run.

the cliques found. On 13 graph instances, EA/G found larger cliques than HGA. Table II also shows that EA/G obtained the best known results on 30 instances.

D. Comparisons With MIMIC

In the probability model used in the guided mutation operator in EA/G, all the variables are treated independently of each other. Only part of the parent solution are altered according to this model and hence multivariate dependencies are processed up to some extent but in a rather random way. Advanced EDAs such as MIMIC [31], FDA [16], and BOA [6] can identify and utilize some dependence relationships among variables in a very systematical way. These EDAs are seemingly more powerful in solving a complicated optimization problem. However, unlike the guided mutation, these advanced EDAs cannot directly control the similarity between offspring and the best solutions found so far. Moreover, only a very limited number of dependence relationships can be considered in these EDAs due to computational complexity. This may hinder these pure EDAs from solving a hard problem. Besides, in the case of evolutionary algorithms with local search for a hard problem, since the distribution of locally optimal solutions are often very complicated, the approximation provided by any advanced EDAs may be very poor.

We compared the performances of EA/G with a MIMIC algorithm, which is the same as EA/G, except that it uses the MIMIC way to generate new solutions. MIMIC may be the simplest among the advanced EDAs considering variable dependencies. The computational complexity of the other advanced EDAs is prohibitively high for solving large-scale problems. This is why we chose MIMIC for the comparison.

The parameter setting for EA/G is the same as in Section IV-C. The population size for MIMIC is set as 100 (small populations may degrade the performance of advanced EDAs [32]). To have a fair comparison, both MIMIC and EA/G terminated after 20 000 calls of the repair operator. The two algorithms were run for ten times on each graph. The experimental results are given in Table III. The one tailed t-test results at the 0.05 significance level are also presented

TABLE III
COMPARISON RESULTS BETWEEN EA/MIMIC AND EA/G. *Best*: THE SIZE OF THE LARGEST CLIQUE FOUND, *Avg*: THE AVERAGE SIZE OF THE CLIQUES FOUND, *std*: THE STANDARD DEVIATION OF THE SIZES OF THE CLIQUES FOUND, *DIMACS*: THE SIZE OF THE LARGEST CLIQUE FOUND BY THE ALGORITHMS, *time*: THE RUN TIME (IN SECONDS) OF EACH ALGORITHM. *t-test*: THE t-TEST RESULTS

Graph	EA/MIMIC			EA/G			t-test	
	Avg(std)	Best	time	Avg(std)	Best	time	t	sig.
C125.9	34.0(0.0)	34	2.6	34.0(0.0)	34	1.5	-	-
C250.9	42.7(0.8)	44	8.4	44.0(0.0)	44	2.9	4.993	0.000
C500.9	52.7(0.8)	54	24.2	55.2(0.9)	56	5.7	5.839	0.000
C1000.9	60.8(1.4)	62	162.2	64.4(1.4)	**67**	21.2	5.823	0.000
C2000.9	61.8(1.0)	64	576.9	70.9(1.0)	72	45.2	19.858	0.000
DSJC500.5	12.2(0.4)	13	9.6	13.0(0.0)	13	4.7	6.000	0.000
DSJC1000.5	13.1(0.3)	14	27.8	14.5(0.3)	**15**	12.1	6.091	0.000
C2000.5	14.6(0.5)	15	413.7	14.9(0.7)	**16**	28.6	1.406	0.096
C4000.5	15.4(0.5)	16	395.4	16.1(0.3)	**17**	61.0	2.333	0.022
MANN_a27	125.2(0.4)	126	27.7	126.0(0.0)	126	12.1	6.000	0.000
MANN_a45	336.8(0.6)	338	189.1	343.7(0.7)	**345**	80.2	13.229	0.000
MANN_a81	1085.6(0.9)	1087	2669.2	1097.2(0.6)	**1098**	829.5	28.651	0.000
brock200_2	12.0(0.0)	12	4.4	12.0(0.0)	12	1.8	-	-
brock200_4	16.3(0.9)	17	5.8	16.5(0.5)	**17**	2.0	0.557	0.295
brock400_2	22.1(0.3)	23	10.7	24.7(0.4)	**25**	3.7	11.759	0.000
brock400_4	22.7(0.9)	23	24.3	25.1(2.6)	**33**	3.9	2.605	0.015
brock800_2	18.5(0.5)	19	63.7	20.1(0.4)	**21**	8.9	7.236	0.000
brock800_4	18.4(0.5)	19	52.8	19.9(0.5)	**21**	8.9	6.708	0.000
gen200_p0.9_44	38.4(0.7)	40	6.0	44.0(0.0)	44	2.1	25.372	0.000
gen200_p0.9_55	52.7(1.7)	55	6.8	55.0(0.0)	55	3.9	3.977	0.002
gen400_p0.9_55	49.5(0.5)	50	23.8	51.8(0.7)	55	4.2	10.776	0.000
gen400_p0.9_65	58.4(2.5)	61	22.6	65.0(0.0)	65	4.2	7.802	0.000
gen400_p0.9_75	67.8(2.7)	72	27.0	75.0(0.0)	75	4.4	8.072	0.000
hamming8-4	16.0(0.0)	16	1.0	16.0(0.0)	16	2.0	-	-
hamming10-4	33.6(0.5)	34	74.4	39.8(0.6)	40	16.7	24.855	0.000
keller4	11.0(0.0)	11	1.3	11.0(0.0)	11	1.5	-	-
keller5	24.6(0.5)	25	85.3	26.9(0.3)	27	10.7	10.776	0.000
keller6	46.1(1.2)	48	475.5	53.4(1.2)	**56**	63.1	10.590	0.000
p_hat300-1	8.0(0.0)	8	2.0	8.0(0.0)	8	2.4	-	-
p_hat300-2	25.0(0.0)	25	5.7	25.0(0.0)	25	2.3	-	-
p_hat300-3	33.8(0.4)	36	9.0	36.0(0.0)	36	2.7	16.500	0.000
p_hat700-1	10.2(0.4)	11	48.7	11.0(0.0)	11	6.6	6.000	0.000
p_hat700-2	42.5(0.8)	44	67.6	44.0(0.0)	44	8.9	5.582	0.000
p_hat700-3	60.9(0.3)	61	49.0	62.0(0.0)	62	13.1	11.000	0.000
p_hat1500-1	10.7(0.5)	11	213.3	11.1(0.3)	**12**	19.8	2.449	0.018
p_hat1500-2	64.1(0.5)	65	303.4	65.0(0.0)	65	28.9	5.014	0.000
p_hat1500-3	92.2(0.6)	93	332.5	93.7(0.5)	94	34.3	9.000	0.000

-: the t-test has not been made for these graphs since both EA/G and MIMIC found the optimal solutions in each run.

in Table III for the alternative hypothesis that the mean size of the cliques obtained by EA/G is larger than that obtained by MIMIC. In 29 instances (with sig < 0.05), EA/G is better than MIMIC in terms of the quality of the solutions. The running time of MIMIC is longer than that of EA/G. This is because that MIMIC needs additional computational overheads in searching for an optimal Markovian chain at each generation.

V. CONCLUSION

Conventional GAs use crossover and mutation for generating offspring, while EDAs sample offspring from a probability model. Crossover and mutation use the location information of the parent solutions, while EDAs are based on global information about the search space collected in the search process. We introduced the guided mutation operator for generating offspring in evolutionary algorithms in this paper. Guided by a probability model which characterizes the distribution of promising solutions in the search space, the guided mutation operator alters a parent solution to generate an offspring. Combining the global information and location information of the parent, the guided mutation operator attempts to generate promising solutions.

In this paper, we proposed EA/G, a hybrid evolutionary algorithm with guided mutation, for the MCP. Besides the guided mutation operator, EA/G adopts a strategy for searching different areas in different search phases. A comparison was made between EA/G and HGA, the best hybrid evolutionary algorithm for the MCP reported so far. Experimental results show that EA/G outperforms HGA. We also showed that the guided mutation operator and the partitioning of the search space do contribute positively to the performance of the algorithm. The experimental results show that EA/G performs better than MIMIC for the MCP, which suggests that location information should not be ignored and it is not sufficient to use a very limited number of dependence relationships for solving a hard optimization and search problem.

It should be pointed out that EA/G is not the best metaheuristic for the MCP. The best heuristic for this problem may be the reactive local search by Battiti and Protasi [35], which is about ten times faster than HGA and EA/G and can find the best known results in almost all DIMACS graphs. However, reactive local search is much more complicated and sophisticated than EA/G. The main purpose of this paper is to study two simple techniques, i.e., guided mutation and partitioning of the search space, and show how they can improve the performance of an evolutionary algorithm. In the future, we intend to refine EA/G and apply it to solve other hard optimization and search problems.

Acknowledgment

The authors are grateful to Dr. Marchiori for sending us the code of HGA. They also thank X. Yao, the anonymous reviewers, the Associate Editor, and P. Mills for their constructive comments and suggestions.

References

[1] F. Glover, "Heuristics for integer programming using surrogate constraints," *Dec. Sci.*, vol. 8, pp. 156–166, 1977.

[2] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," School of Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-94-163, 1994.

[3] H. Müehlenbein, "The equation for response to selection and its use for prediction," *Evol. Comput.*, vol. 5, pp. 303–346, 1998.

[4] H. Müehlenbein, T. Mahnig, and A. O. Rodriguez, "Schemata, distributions, and graphical models in evolutionary optimization," *J. Heuristics*, vol. 5, pp. 215–247, 1999.

[5] S. Baluja and S. Davies, "Fast probabilistic modeling for combinatorial optimization," in *Proc. 15th National/10th Conf. Artif. Intell./Innov. Appl. Artif. Intell.*, Madison, WI, 1998, pp. 469–476.

[6] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "BOA: The Bayesian optimization algorithm," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., 1999, pp. 525–532.

[7] B. T. Zhang, "A bayesian framework for evolutionary computation," in *Proc. 1999 Congr. Evol. Comput.*, 1999, pp. 722–228.

[8] P. A. N. Bosman and D. Thierens, "Expanding from discrete to continuous EDAs: The IDEA," in *Proc. Parallel Prob. Solving from Natutre (PPSN VI)*, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, Eds., Paris, France, pp. 767–776.

[9] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Norwell, MA: Kluwer, 2001.

[10] J. Hastad, "Clique is hard to approximate within $n^{1-\epsilon}$," in *Proc. 37th Annu. Symp. Found. Comput. Sci.*. Burlington, VT, Oct. 14–16, 1996, pp. 627–636.

[11] I. M. Bomze, M. Budinich, P. M. Paradalos, and M. Pelillo, "The maximum clique problem," in *Handbook of Combinatorial Optimization*, D.-Z. Du and P. M. Paradalos, Eds. Norwell, MA: Kluwer, 1999, vol. 4.

[12] L. Cavique, C. Rego, and I. Themido, "A scatter search algorithm for the maximum clique problem," Instituto Politecnico de Lisboa, Portugal, Tech. Rep. HCES-01-01, 2001.

[13] B. Carter and K. Park, "How good are genetic algorithms at finding large cliques: An experimental study," Comput. Sci. Dept., Boston Univ., Boston, MA, Tech. Rep. BU-CS-93-015, 1993.

[14] K. Park and B. Carter, "On the effectiveness of genetic search in combinatorial optimization," Comput. Sci. Dept., Boston Univ., Boston, MA, Tech. Rep. BU-CS-94-010, 1994.

[15] J. He and X. Yao, "From an individual to a population: An analysis of the first hitting time of population-based evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 6, no. 5, pp. 495–511, Oct. 2002.

[16] Q. Zhang, "On stability of fixed points of limit models of univariate marginal distribution algorithm and factorized distribution algorithm," *IEEE Trans. Evol. Comput.*, vol. 8, no. Feb., pp. 80–93, 2004.

[17] J. He and X. Yao, "A study of drift analysis for estimating computation time of evolutionary algorithms," *Natural Comput.*, vol. 3, no. 1, pp. 21–35, 2004.

[18] ——, "Toward an analytic framework for analyzing the computation time of evolutionary algorithms," *Artif. Intell.*, vol. 145, pp. 59–97, 2003.

[19] T. Bäck and S. Khuri, "An evolutionary heuristic for the maximum independent set problem," in *Proc. 1st IEEE Conf. Evol. Comput.*, 1994, pp. 531–535.

[20] M. Hifi, "A genetic algorithm-based heuristic for solving the weighted maximum independent set and some equivalent problems," *J. Oper. Res. Soc.*, vol. 48, pp. 612–622, 1997.

[21] A. S. Murthy, G. Parthasarathy, and V. U. K. Sastry, "Clique finding—A genetic approach," in *Proc. 1st IEEE Conf. Evol. Comput.*, 1994, pp. 18–21.

[22] T. N. Bui and P. H. Eppley, "A hybrid genetic algorithm for the maximum clique problem," in *Proc. 6th Int. Conf. Genetic Algorithms*, L. J. Eshelman, Ed., Pittsburgh, PA, July 1995, pp. 478–484.

[23] C. Fleurent and J. A. Ferland, "Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability," *SIAM J. Alg., Discr. Meth.*, vol. 3, pp. 584–591, 1996.

[24] J. A. Foster and T. Soule, "Using genetic algorithms to find maximum cliques," Dept. Comput. Sci., Univ. Idaho, Moscow, ID, Tech. Rep. LAL95-12, 1995.

[25] E. Marchiori, "A simple heuristic based genetic algorithm for the maximum clique problem," in *Proc. ACM Symp. Appl. Comput.*, 1998, pp. 366–373.

[26] ——, "Genetic, iterated and multistart local search for the maximum clique problem," in *Applications of Evolutionary Computing*. Berlin, Germany: Springer-Verlag, 2002, LNCS 2279, pp. 112–121.

[27] F. Glover and M. Laguna, *Tabu Search*. Norwell, MA: Kluwer, 1998.

[28] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," *Oper. Res.*, vol. 21, pp. 498–516, 1973.

[29] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 287–297, Nov. 1999.

[30] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Berlin, Germany: Springer-Verlag, 2003.

[31] J. S. De Bonet, C. L. Isbell, and P. Viola, "MIMIC: Finding optima by estimating probability densities," in *Advances in Neural Information Processing Systems*, M. C. Mozer, M. I. Jordan, and T. PetscheVol, Eds, MA: The MIT Press, 1997, pp. 424–431.

[32] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "Bayesian optimization algorithm, population sizing, and time to convergence," Illinois Genetic Algorithm Lab., Univ. Illinois Urbana–Champaign, Urbana, IL, Rep. No. 2 000 001, Jan. 2000.

[33] J. M. Peña, V. Robles, P. Larrañaga, V. Herves, F. Rosales, and M. S. Pérez, "GA-EDA: Hybrid evolutionary algorithm using genetic and estimation of distribution algorithms," in *Lecture Notes in Artificial Intelligence*, Ottawa, ON, Canada, May 2004, pp. 361–371. Proc. 17th Int. Conf. Ind. Eng. Appl. Artif. Intell. Expert Syst..

[34] D. Thierens, "Scalability problem of simple genetic algorithms," *Evol. Comput.*, vol. 7, pp. 331–352, 1999.

[35] R. Battiti and M. Protasi, "Reactive local search for the maximum clique problem," *Algorithmica*, vol. 29, no. 4, pp. 601–637, 2001.

Qingfu Zhang (M'01) received the B.Sc. degree in mathematics from Shanxi University, Shanxi, China, in 1984, the M.Sc. degree in applied mathematics and the Ph.D. in information engineering from Xidian University, Xi'an, China, in 1991 and 1994, respectively.

He has been a Lecturer in the Department of Computer Science, University of Essex, Colchester, U.K. since 2000. From 1994 to 2000, he worked in the National Laboratory of Parallel Processing and Computing, Changsha Institute of Technology, Hong Kong, China, the Hong Kong Polytechnic University, Hong Kong, the German National Research Center for Information Technology (now Fraunhofer–Gesellschaft), Sankt Augustin, Germany, and the University of Manchester Institute of Science and Technology, Manchester, U.K., as a Researcher. His main research areas are evolutionary computation, optimization, neural networks, and data analysis and their applications.

Jianyong Sun received the B.Sc. degree in computational mathematics from Xi'an Jiaotong University, Xi'an, China, in 1997. He is currently working towards the Ph.D. degree in computer science from the University of Essex, Colchester, U.K.

In 2000, he worked in the Department of Computer Science and Engineering, The Chinese University of Hong Kong, as a Research Assistant. From 2002 to 2003, he worked in the Department of Computer Science, University of Essex, as a Senior Research Officer. His main research areas are evolutionary computation, optimization, metaheuristics, and telecommunication networks.

Edward Tsang (M'04) received the B.B.A. degree in business administration from the Chinese University of Hong Kong, in 1997, and the M.Sc. and Ph.D. degrees in computer science from the University of Essex, Colchester, U.K., in 1984 and 1987, respectively.

He has broad interest in applied artificial intelligence, in particular, computational finance, scheduling, heuristic search, constraint satisfaction, and optimization. He is currently a Professor of Computer Science at the University of Essex, Colchester, U.K., where he leads the Computational Finance Group and Constraint Satisfaction and Optimization Group. He is also the Deputy Director of the Centre for Computational Finance and Economic Agents (CCFEA), an interdisciplinary centre.

Dr. Tsang chairs the Technical Committee for Computational Finance under the IEEE Computational Intelligence Society. Details of his work can be found at http://cswww.essex.ac.uk/CSP/edward.