R-EVO: A Reactive Evolutionary Algorithm for the Maximum Clique Problem

Mauro Brunato, *Member, IEEE,* and Roberto Battiti, *Fellow, IEEE*

***Abstract*—An evolutionary algorithm with guided mutation (EA/G) has been proposed recently for solving the maximum clique problem. In the framework of estimation-of-distribution algorithms, guided mutation uses a model distribution to generate offspring by combining the local information of solutions found so far with global statistical information. Each individual is then subjected to a Marchiori's repair heuristic, based on randomized extraction and greedy expansion, to ensure that it represents a legal clique. A novel reactive and evolutionary algorithm (R-EVO) proposed in this paper starts from the same evolutionary framework but considers more complex individuals, which modify tentative solutions by local search with memory, according to the reactive search optimization (RSO) principles. In particular, the estimated distribution is used to periodically initialize the state of each individual based on the previous statistical knowledge extracted from the population. We demonstrate that the combination of the estimation-of-distribution concept with RSO produces significantly better results than EA/G for many test instances and it is remarkably robust with respect to the setting of the algorithm parameters. R-EVO adopts a drastically simplified low-knowledge version of reactive local search (RLS), with a simple internal diversification mechanism based on tabu-search, with a prohibition parameter proportional to the estimated best clique size. R-EVO is competitive with the more complex full-knowledge RLS-EVO that adopts the original RLS algorithm. For most of the benchmark instances, the hybrid scheme version produces significantly better results than EA/G for comparable or a smaller central processing unit time.**

***Index Terms*—Estimation of distribution, guided mutation, maximum clique, reactive search optimization.**

I. Introduction

POPULATION-BASED heuristics can be made more efficient by working along different directions. One direction is the complexity of a single problem-solving entity [an individual in the population in genetic algorithm (GA) terms], another one is the number of individuals and the amount of their mutual interaction. For example, an individual may be very simple, leaving to mutation, crossover and selection, the work of exploring and exploiting the fitness surface, or it may become more complex, embodying for example repair procedures [2], or elements of local search, as in *memetic* algorithms [3], [4]. The interaction in the population can be indirect, based on the current fitness of the individuals which influences the reproduction, or more direct and based on both global and local information; see for example the "particle swarm" technique where each member of the swarm is updated based on the global best position and on the individual best [5]. Interaction through explicit statistical models of the promising solutions is advocated in the estimation of distribution algorithms (EDA), see, e.g., [6]–[10]. In a different community, methods to combine solutions have been designed with the term "scatter search" [11], [12], showing the advantages provided by intensification and diversification mechanisms that exploit adaptive memory, drawing on foundations that link scatter search to tabu search [13].

While a complete coverage of the tradeoffs between complexity of the individual population members and complexity of their interaction is beyond the scope of this paper and some useful historical references can be found in many papers cited in this article, this paper is motivated by a recent evolutionary algorithm with guided mutation for the maximum clique proposed in [14], where the authors obtain state-of-the-art results in the area of evolutionary algorithms for the problem, by improving upon Marchiori's results [2], [15] and upon an advanced EDA algorithm like mutual information-maximizing input clustering (MIMIC) [16].

The maximum clique (MC) problem in graphs is a paradigmatic combinatorial optimization problem with many relevant applications [17], including information retrieval, computer vision, and social network analysis. Recent interest includes computational biochemistry, bio-informatics, and genomics, see for example [18], [19]. Let $G = (V, E)$ be an undirected graph, $V = \{1, 2, \ldots, n\}$ its vertex set, $E \subseteq V \times V$ its edge set, and $G(S) = (S, E \cap S \times S)$ *the subgraph induced by* S, where S is a subset of V. A graph $G = (V, E)$ is *complete* if all its vertices are pairwise adjacent, i.e., $\forall i,\ j \in V,\ (i, j) \in E$. A *clique* K is a subset of V such that $G(K)$ is complete. The MC problem asks for a clique of maximum cardinality. The problem is nondeterministic polynomial-time hard and strong negative results have been shown about its approximability [20], making it an ideal test-bed for search heuristics.

The initial motivation of this paper was to assess whether the incorporation of reactive search optimization (RSO) ideas developed in [1], [21] into an evolutionary approach could lead to a competitive technique. Furthermore, we wanted to confirm whether the advantage of the technique persisted after radical simplifications of the RSO algorithm. The simplification has

Manuscript received May 9, 2007; revised August 28, 2008, January 30, 2009, May 8, 2009. This work was supported by the project Bionets (IST-027748), funded by the Future and Emerging Technologies Program of the European Commission.

The authors are with the Department of Information Engineering and Computer Science, University of Trento, Trento I-38100, Italy (e-mail: brunato@disi.unitn.it; battiti@disi.unitn.it).

Digital Object Identifier 10.1109/TEVC.2010.2043363

been motivated by a note [14], stating that, while more effective, "reactive local search is much more complicated and sophisticated than an evolutionary algorithm with guided mutation (EA/G)." We propose here a radically simplified reactive scheme, hybridized with an EDA approach, which maintains a significant advantage over EA/G, when both clique sizes and central processing unit (CPU) times are considered.

RSO, see the seminal papers [22], [23] and the book [1], advocates the use of *machine learning* to automate the parameter tuning process and make it an integral and fully documented part of the algorithm. Learning is performed on-line, and therefore, *task-dependent and local properties* of the configuration space can be used. In this way a single algorithmic framework maintains the flexibility to deal with related problems through an internal feedback loop that considers the previous history of the search. RSO is based on a human analogy of "learning on the job:" the more knowledge and experience are accumulated by the problem-solver, the better the problem-solving strategy becomes.

The main novelties introduced by the reactive and evolutionary algorithm (R-EVO) algorithm are given below.

1) Reactive and memetic evolution: Each individual is subjected to a short run of intelligent local search (prohibition-based reactive search) which considers the given individual as a starting point.
2) Use of estimation of distribution: While guided mutation is used to generate new individuals in EA/G, the model obtained by estimation of distribution is used to create new individuals in R-EVO.
3) Extreme simplification through the design of a low-knowledge version of the reactive local search algorithm: Instead, of the complete memorization of previous solutions, only the best cliques (up to a given tolerance parameter Δ for the size of the clique) are used to derive a probabilistic model of the fittest individuals and to self-tune the prohibition period on a specific instance.
4) Simplification of the model derivation: The model is not obtained through an exponentially weighted moving average but in a parameter-less manner from the best previous solutions (within tolerance parameter Δ).
5) Proposal of number of iterations and prohibition period scaled according to the best clique size estimated at run-time on a specific instance.

II. Evolutionary algorithms With Guided Mutation

EDA [6]–[10], [24], have been proposed in the framework of evolutionary computation for modeling promising solutions in a probabilistic manner, and for using such models to produce the next generation. A survey in [25] considers population-based probabilistic search algorithms based on "modeling promising solutions by estimating their probability distribution and using the model to guide the exploration of the search space." The main idea of model-based optimization is to create and maintain a *model* of the problem, whose aim is to provide some clues about the problem's solutions.

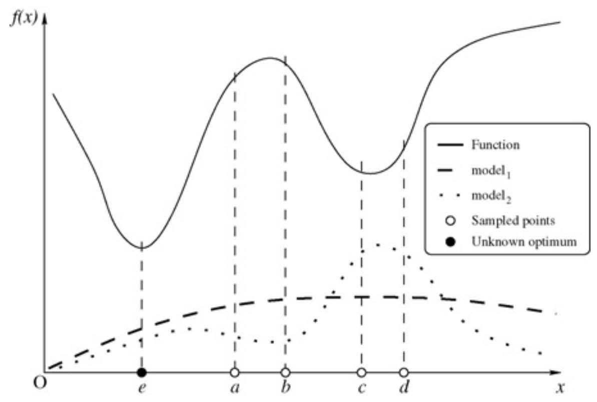

Fig. 1. Model-based search: one generates sample points from model$_1$ and updates the generative model to increase the probability for point with low cost values (see model$_2$). In pathological cases, optimal point *e* runs the risk of becoming more and more difficult to generate (figure adapted from [1]).

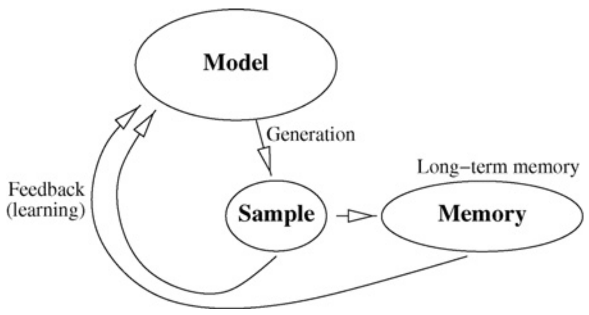

Fig. 2. Model-based architecture: a generative model is updated after learning from the last generated samples and the previous long-term memory (figure adapted from [1]).

If the problem is a function to be minimized, the model can be a *simplified version* of the function itself. When used to optimize functions of continuous variables, model-based optimization is related to *surrogate optimization*, where a surrogate function is used to generate new sample points instead of the original function, which is in some cases very costly to compute, see for example [26], and also connected to the *kriging* and *response surface* methodologies.

In more general settings, the model can summarize the relevant information obtained about a problem, for example in the form of a *probability distribution* defining the estimated likelihood of finding a good quality solution at a certain point. To solve a problem, we resort to the model in order to generate a candidate solution, then check it. The results of the check are then used to refine the model, so that the future generation is biased toward better and better candidate solutions. Clearly, for a model to be useful, it must provide as much information about the problem as possible, while being somehow "more tractable" (in a computational or analytical sense) than the problem itself.

Although model-based techniques can be used in both discrete and continuous domains, the latter case better supports our intuition. In Fig. 1, a function (continuous line) must be minimized. An initial model (the dashed line) provides a prior probability distribution for the minimum (in case of no prior knowledge, a uniform distribution can be assumed). Based on

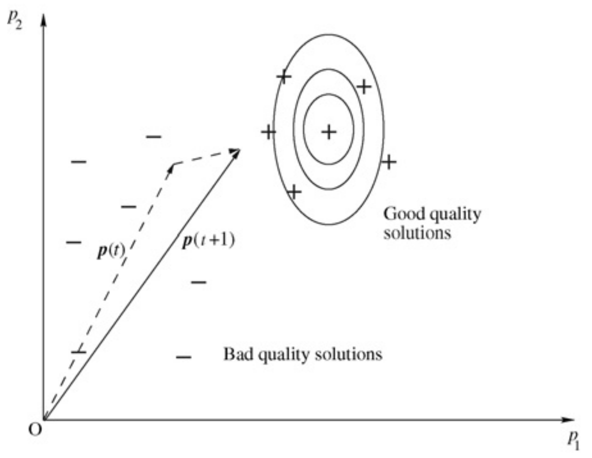

Fig. 3. PBIL: "Prototype" vector $\boldsymbol{p}$ gradually shifts toward good quality solutions (qualitative example in two dimensions).

this estimate, some candidate minima are generated (points *a*–*d*), and the corresponding function values are computed. The model is updated (dotted line) to take into account the latest findings: the global minimum is more likely to occur around *c* and *d*, rather than *a* and *b*. Further model-guided generations and tests will improve the distribution, eventually the region around the global minimum *e* will be discovered and a high probability density will be assigned to its surroundings. The same example also highlights a possible drawback of naïf applications of the technique: assigning a high probability to the neighborhood of *c* and *d* could lead to a negligible probability of selecting a point near *e*, so the global minimum would never be discovered. The emphasis is on *intensification* (or *exploitation*) of the search. This is why, in practice, the models are corrected to ensure a significant probability of generating points also in unexplored regions.

The scheme of a model-based search approach, see also [27], is presented in Fig. 2. Represented entities are:

1) a model used to generate sample solutions;
2) the last samples generated;
3) a memory containing previously accumulated knowledge about the problem (previous solutions and evaluations).

The process develops in an iterative way through a feedback loop where new candidates are generated by the model, and their evaluation—together with memory about past states—is used to improve the model itself in view of a new generation.

The design choices consist of defining a suitable generative model, and an appropriate learning rule to favor generation of superior models in the future steps. The simple model considered in this paper is as follows. The search space $\mathcal{X} = \{0, 1\}^n$ is the set of all binary strings of length n, the generation model is defined by an n-tuple of parameters

$$\boldsymbol{p} = (p_1, \ldots, p_n) \in [0, 1]^n$$

where p_i is the probability of producing 1 as the ith bit of the string and every bit is independently generated. One way to look at the model is to "remove genetics from the standard genetic algorithm" [7]: instead of maintaining implicitly a

```
function EA/G ($N$, $M$, $\Delta$, $\lambda$, $\beta$, $\alpha$)
    $t \leftarrow 0$;
    pick $x \in \Omega$;
    $Q_{\text{best}} \leftarrow$ Repair ($x$, $\alpha$);
    $S_{\text{best}} \leftarrow |U|$;
    restart $\leftarrow$ true;
    repeat
        if restart or all $x^i$'s are equal
            for $i \leftarrow 1, \ldots, N$
                pick $x \in \bigcup_{j=S_{\text{best}}+1}^{S_{\text{best}}+\Delta} \Omega_j$;
                $x^i \leftarrow$ Repair($x$, $\alpha$);
            for $i \leftarrow 1, \ldots, n$
                $p_i \leftarrow \sum_{j=1}^{N} x_i^j / N$;
            restart $\leftarrow$ false
        SortBySize ( $x^i$, $i = 1, \ldots, N$);
        for $i \leftarrow 1, \ldots, n$
            $p_i \leftarrow (1 - \lambda) p_i + \lambda \sum_{j=1}^{M} x_i^j / M$;
        for $i \leftarrow M + 1, \ldots, N$
            $x \leftarrow$ Mutate ( $x^1$, ( $p_i$), $\beta$);
            $x^i \leftarrow$ Repair ($x$, $\alpha$);
        if (size of fittest individual) $> S_{\text{best}}$
            $Q_{\text{best}} \leftarrow$ new maximum clique;
            $S_{\text{best}} \leftarrow |Q_{\text{best}}|$;
            restart $\leftarrow$ true;
        $t \leftarrow t + 1$
    until stopping condition
```

Fig. 4. EA/G algorithm for the MC problem.

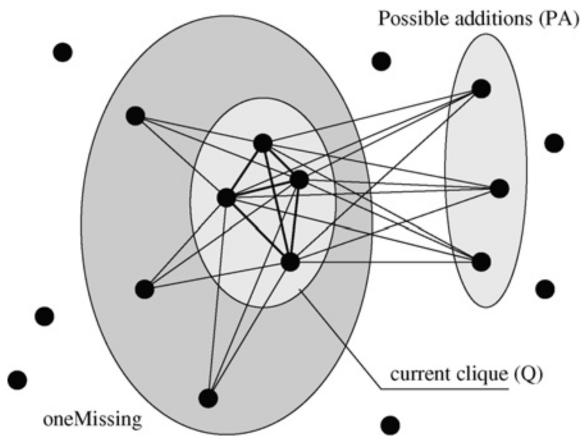

Fig. 5. Neighborhood of current clique.

T	Prohibition period
t	Iteration counter
t_R	Time of last reaction
Q	Current clique
Q_{best}	Best clique so far
S_{best}	Size of best clique so far
t_b	Time of last improvement
MemoryReaction	Modifies T based on visited clique history.
BestNeighbor	Generate best Q's neighboring clique (see Fig. 7)
Restart	Start with a new clique of size 1

```
1.  function RLS
2.    $t \leftarrow 0$; $T \leftarrow 1$; $t_R \leftarrow 0$;                         Initialization
3.    $Q \leftarrow \emptyset$; $Q_{\text{best}} \leftarrow \emptyset$; $S_{\text{best}} \leftarrow 0$; $t_b \leftarrow 0$;
4.    repeat                                                Iterative improvement
5.      $T \leftarrow$ MemoryReaction ($Q, T$);
6.      $Q \leftarrow$ BestNeighbor ($Q$);
7.      $t \leftarrow t + 1$;
8.      if $|Q| > S_{\text{best}}$
9.        $Q_{\text{best}} \leftarrow Q$;
10.       $S_{\text{best}} \leftarrow |Q|$;
11.       $t_b \leftarrow t$;
12.     if $t - \max\{t_b, t_R\} > A$
13.       $t_R \leftarrow t$;
14.       $Q \leftarrow$ Restart ();
15.   until $S_{\text{best}}$ is acceptable
16.     or maximum iterations reached;
```

Fig. 6. RLS algorithm for the MC problem.

statistic in a GA population, *statistics are maintained explicitly* in the vector $\boldsymbol{p}$.

The initial state of the model corresponds to indifference with respect to the bit values: $p_i = 0.5$, $i = 1, \ldots, n$. In the population-based incremental learning (PBIL) algorithm [7] the following steps are iterated:

```
1. Initialize $\boldsymbol{p}$;
2. repeat:
3.   Generate a sample set $S$ using the vector $\boldsymbol{p}$;
4.   Extract a fixed number $\bar{S}$ of the best solutions from $S$;
5.   for each sample $\boldsymbol{s} = (s_1, \ldots, s_n) \in \bar{S}$
6.     $\boldsymbol{p} \leftarrow (1 - \lambda)\boldsymbol{p} + \lambda \boldsymbol{s}$
```

where λ is a learning rate parameter (regulating exploration versus exploitation). The moving vector $\boldsymbol{p}$ can be seen as representing an exponentially weighted moving average of the best samples, a prototype vector placed in the middle of the cluster providing the recently found best quality solutions. As a parallel with machine learning literature, the update rule is similar to that used in learning vector quantization (see [28]). Variations include moving away from bad samples in addition to moving toward good ones. A schematic representation is shown in Fig. 3.

Estimates of probability densities for optimization considering possible dependencies in the form of pairwise conditional probabilities are studied in [16]. Their MIMIC technique aims at estimating a probability density for points with value below a given threshold (remember that the function is to be *minimized*). These more complex models have been considered for the MC problem in [14], which demonstrates inferior performance with respect to the simpler PBIL, in spite of added computational overhead.

The EA/G algorithm proposed in [14] for the MC problem is based on the following principles.

1) Use of the PBIL algorithm [6] to create a model of the fittest individuals created in the population.
2) Use of a guided mutation operator to produce the offspring. This is motivated by the "proximate optimality principle" [29] which assumes that good solutions tend

Q	Current clique
`PA`	Set of candidate nodes for addition to Q (see Fig. 5)
`type`	Type of the proposed move: can be `notFound`, `addMove` or `dropMove`
$D_{\max}$	Maximum degree of nodes in `PA`
$\Delta_{\max}$	Maximum increase in size of `PA` due to node removal from Q
v	Chosen node for insertion or removal
`IncrUpdate`	Modifies internal structure for fast computation.

```
function BestNeighbor ($Q$)
    type ← notFound;
    if {allowed $v \in$ PA} $\neq \emptyset$
        type ← addMove;
        $D_{\max} \leftarrow \max \deg_{G(\mathrm{PA})}(\{\text{allowed } v \in \mathrm{PA}\})$;
        pick $v \in \{\text{allowed } w \in \mathrm{PA} | \deg_{G(\mathrm{PA})}(w) = D_{\max}\}$;
    if type = notFound
        type ← dropMove;
        if {allowed $v \in Q$} $\neq \emptyset$
            $\Delta_{\max} \leftarrow \max\{\Delta_{\mathrm{PA}}[j] | j \in Q \wedge j \text{ allowed}\}$;
            pick $v \in \{\text{allowed } w \in Q | \Delta_{\mathrm{PA}}[w] = \Delta_{\max}\}$;
        else
            pick $v \in Q$;
    IncrUpdate ($v$, type);
    if type = addMove
        return $Q \cup \{v\}$
    else
        return $Q \setminus \{v\}$
```

Fig. 7. Search for the best neighboring configuration in RLS.

to have similar structures. Global statistical information is extracted through EDA from the previous search and represented as a probability model (a vector $\boldsymbol{p}$) characterizing the distribution of promising solutions in the search space. A new individual is moved in a stochastic manner toward the center of the model. In detail, for each bit of the binary string describing the individual, one flips a coin with head probability β. If head turns up, the specific bit i is set to one with probability p_i, to zero otherwise. If $\beta = 1$, the string is sampled from the probability model $\boldsymbol{p}$.

3) Use of a lower bound on the maximum clique (size of best clique found so far) to search for progressively larger cliques.
4) Use of Marchiori's repair heuristic [2] to create a legal clique (some of the internal connections can be missing in the individual created with guided mutation) and to extend it in a greedy manner until a maximal clique is reached.

Fig. 4 outlines the EA/G code. The algorithm accepts as input the population size N and some parameters described below. The guided mutation operator described above is implemented in function `Mutation`, while Marchiori's repair heuristic is performed by function `Repair`. The algorithm works by maintaining a population of size N. About the notation: S_{best} is the lower bound on the maximum clique size equal the size of the best clique found so far, and the population is partitioned into sets Ω_j

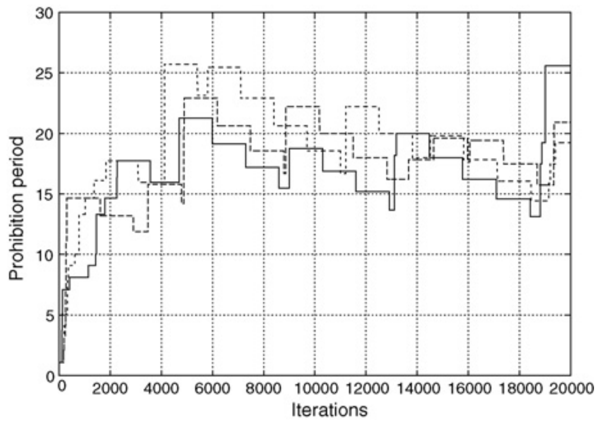

Fig. 8. Dynamic evolution of the prohibition $T(t)$ as a function of the iteration for three different runs on C1000.9 (with different random seeds).

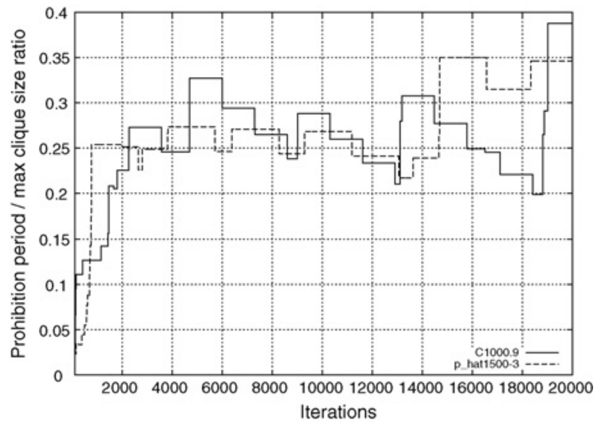

Fig. 9. Dynamic evolution of the prohibition $T(t)$ as a function of the iteration for runs on Q_{best}. y-values are ratios between prohibition and best clique size $T(t)/Q_{\text{best}}(t)$.

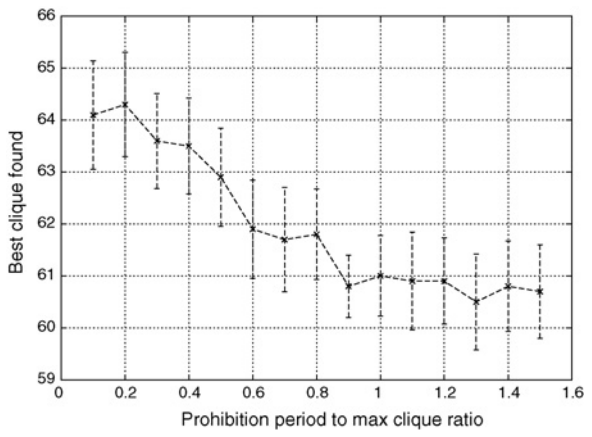

Fig. 10. Final Q_{best} results obtained after ten runs of s-RLS (simplified RLS) of 2000 iterations, as a function of the τ parameter on instance C1000.9. Error bars are at $\pm\sigma$.

containing individuals representing cliques of size equal to j.

At each step, the M (equal to $N/2$ in the cited paper) fittest individuals are kept, while the others are replaced by repaired mutations of the fittest ones. This is achieved in the pseudo-code by sorting individuals according to their fitness (line 15) and by using the first one, x^1, for generation of others (lines 18–20). Elements $x^2, \dots, x^M$ do not generate offspring, but participate to the PBIL model update (line 17). If a new population is to be generated (for instance, when a larger clique is found, or population converges to a single individual), new individuals are selected among individuals representing cliques of size $S_{\text{best}} + 1 \dots S_{\text{best}} + \Delta$, and the $\boldsymbol{p}$ distribution is reset (lines 9–13). The PBIL model is initialized at line 13 and updated by a moving average at line 17.

III. Reactive Search Optimization in a Population-Based Framework

A reactive local search (RLS) algorithm for the solution of the MC problem is proposed in [21], [30] and obtains state-of-the-art results in computational tests on the second discrete mathematics and theoretical computer science (DIMACS) implementation challenge.[1]

In local search algorithms for MC, the basic moves consist of the addition to or removal of single nodes from the current clique. A swap of nodes can be trivially decomposed into two separate moves. The local changes generate a search trajectory $X^{\{t\}}$, the current clique at different iterations t.

RLS is based on local search complemented by a feedback (history-sensitive) scheme to determine the amount of diversification. The reaction acts on the single parameter T that decides the temporary *prohibition* of selected moves in the neighborhood. In detail, given the prohibition parameter T, a just moved node (added to or dropped from the current clique) remains prohibited and cannot be moved for the next T iterations. We shall refer to nonprohibited nodes as *allowed nodes*.

Two sets are involved in the execution of basic moves: the set of the *possible additions* (`PA`) which contains nodes connected to all the elements of the clique, and the set of the *level neighbors* `oneMissing` containing the nodes connected to all *but* one element of the clique (see Fig. 5).

The RLS algorithm [21] is presented in Fig. 6. It consists of a local search loop (lines 4–16); every iteration calls the `BestNeighbor` function that returns the fittest neighboring configuration, given the current one, thus repeatedly moving from one configuration to a nearby one. The function `Best-Neighbor`, outlined in Fig. 7, alternates between expansion and plateau phases, and it selects the nodes among the allowed ones that have the highest degree in `PA`. The rationale is that in this manner future additions will be favored; if a node in `PA` is connected to the selected node it will remain in the set of possible additions also in the future step. In detail, the function searches for an allowed node within `PA` with the highest degree within `PA` itself (lines 4–6). If no such node is available, then it tries to remove an allowed node from the clique that would maximally increase `PA` (lines 10 and 11). If all such nodes are prohibited, then it proceeds by removing a random node from the clique (line 13).

The part that differentiates RLS from other local search mechanisms is function `MemoryReaction`, which maintains the history of the search by storing each visited clique Q (or

[1]http://dimacs.rutgers.edu/Challenges/.

```
function R-EVO ($N$)
    for $i \leftarrow 1, \dots, N$                                   Generate population
        $R_i \leftarrow$ new RLS searcher;
    for $i \leftarrow 1, \dots, n$                                   Initialize model uniformly
        $p_i \leftarrow 0.5$;
    repeat
        for $i \leftarrow 1, \dots, N$                               Iterate through solvers
            execute one step of $R_i$;
            $Q_i \leftarrow$ best clique of $R_i$;
        $S_{\max} \leftarrow \max\{|Q_i| : i = 1, \dots, N\}$
        $B \leftarrow \{Q_i : i = 1, \dots, N \wedge |Q_i| \geq S_{\max} - \Delta\}$;
        for $i \leftarrow 1, \dots, n$                               Update PBIL model
            $p_i \leftarrow (1 - \lambda)p_i + \lambda \sum_{Q \in B} \chi_Q(i)/|B|$;
    until $S_{\max}$ is acceptable
        or maximum iterations reached;
```

Fig. 11. RLS-EVO algorithm for the MC problem. The R-EVO simplified version only differs in the implementation of the RLS step, so the pseudo-code fits R-EVO as well.

a suitable fingerprint) into a dictionary structure, e.g., a hash table, together with some details, such as the number of times a given clique was found and the last time it has been generated. Such information is used to adjust the prohibition time T: if `MemoryReaction` detects that the same clique has been visited too often (a hint that the search is trapped inside a local minimum), it can try to improve differentiation by increasing the prohibition period T, thus forcing a larger Hamming distance from the current configuration in subsequent steps (for example a larger number of new nodes in the current clique). Restarts are executed only when the algorithm cannot improve the current configuration within a number of iterations A from either the last restart or the last improvement in clique size. Additional and more recent implementation details are described in [31].

A. Low-Knowledge Reactive Search Optimization Algorithm

Let us now come to the extreme simplification of the RLS algorithm. By analyzing the evolution of the prohibition period T during runs on different instances we noted a tendency of the T values during the run to grow as a function of the size of the maximum cliques contained in the graph.

The evolution of T for the graph C1000.9 on three different RLS runs is shown in Fig. 8. One observes a transient period when T grows from the initial value one to a value which then tends to remain approximately stable during the run (apart from random fluctuations caused by the reactive mechanism).

Fig. 9 shows the evolution of T for a couple of representative significant graphs. To make the approximate proportionality to the clique dimension clear we plot directly the ratio between the current T value and the size of the best clique found at iteration t, called $Q_{\text{best}}(t)$.

After confirming the hypothesis of a parameter T being adapted to approximately a fraction of $Q_{\text{best}}(t)$ we experimented with a simplified reactive scheme which sets the prohibition value $T(t)$ equal to $\tau \cdot Q_{\text{best}}(t)$. This version is called a *low-knowledge* reactive scheme because the only information used from the previous history of the search is given by the size of the best clique.

The average result obtained for Q_{best} in ten different runs are plotted in Fig. 10 for graph instance C1000.9, belonging to the DIMACS benchmark. It can be observed that a small value for the proportionality constant τ is associated to the best results obtained. Furthermore, one observes a robust behavior of the results as a function of τ, in particular when its value ranges approximately between 0.1 and 0.4 (in fact, error bars have a large overlap in this region). Following these experimental results, confirmed also on other graph families, we decided to fix $\tau = 0.2$ for all cases. While there is no reason to consider the chosen value for τ as "universal," we chose to fix it once and for all to avoid instance-level calibration.

The difference is coded in the `MemoryReaction` mechanism: the simplified version does not need to maintain a history structure, and the only action is to update the value of T according to the size of the largest clique found up to that moment.

B. Hybrid Evolutionary and Reactive Algorithms (R-EVO)

To distinguish the new and simplified version of RLS, the hybrid algorithm is called R-EVO, while the hybrid algorithm adopting the original RLS method is called RLS-EVO. Both algorithms have the same overall structure, shown in Fig. 11. The pool of searching individuals (R_i) is created with an empty clique, and the initial estimate $\boldsymbol{p}$ of the node distribution in the clique is set to a uniform value (each node having a 50% probability of appearing in a maximum clique).

Every iteration of the algorithm consists of a single iteration of each individual searcher (lines 7–9). After all searchers have

TABLE I
COMPARISON BETWEEN EA/G AND R-EVO FOR A FIXED NUMBER OF ITERATIONS

	EA/G			R-EVO (200 000)			t-test	
	Avg. (std)	Best	Time (s)	Avg. (std)	Best	Time (s)	t	Sig.
C125.9	34.0 (0.0)	34	1.3	34.0 (0.0)	34	0.464	–	–
C250.9	44.0 (0.0)	44	2.5	44.0 (0.0)	44	0.491	–	–
C500.9	55.2 (0.9)	56	4.8	57.0 (0.0)	57	0.719	6.325	0.000
C1000.9	64.4 (1.4)	67	18.0	67.3 (0.5)	68	1.189	6.169	0.000
C2000.9	70.9 (1.0)	72	38.4	75.8 (0.6)	77	2.900	13.287	0.000
DSJC500_5	13.0 (0.0)	13	4.0	13.0 (0.0)	13	1.206	–	–
DSJC1000_5	14.5 (0.3)	15	10.3	15.0 (0.0)	15	3.107	5.270	0.000
C2000.5	14.9 (0.7)	16	24.3	16.0 (0.0)	16	4.372	4.969	0.001
C4000.5	16.1 (0.3)	17	51.9	17.0 (0.0)	17	9.342	9.487	0.000
MANN_a27	126.0 (0.0)	126	10.3	125.6 (0.5)	126	0.651	2.530	0.026
MANN_a45	343.7 (0.7)	345	68.2	342.2 (0.4)	343	2.852	5.883	0.000
MANN_a81	1097.2 (0.6)	1098	705.1	1096.9 (0.6)	1098	5.754	1.118	0.208
brock200_2	12.0 (0.0)	12	1.5	11.5 (0.5)	12	0.844	3.162	0.009
brock200_4	16.5 (0.5)	17	1.7	16.1 (0.3)	17	0.500	2.169	0.044
brock400_2	24.7 (0.4)	25	3.1	25.0 (0.0)	25	0.722	2.372	0.034
brock400_4	25.1 (2.6)	33	3.3	25.8 (2.5)	33	1.158	0.614	0.323
brock800_2	20.1 (0.4)	21	7.6	21.0 (0.0)	21	1.430	7.115	0.000
brock800_4	19.9 (0.5)	21	7.6	21.0 (0.0)	21	2.253	6.957	0.000
gen200_p0.9_44	44.0 (0.0)	44	1.8	44.0 (0.0)	44	0.402	–	–
gen200_p0.9_55	55.0 (0.0)	55	3.3	55.0 (0.0)	55	0.586	–	–
gen400_p0.9_55	51.8 (0.7)	55	3.6	54.0 (1.1)	55	0.621	5.336	0.000
gen400_p0.9_65	65.0 (0.0)	65	3.6	65.0 (0.0)	65	0.940	–	–
gen400_p0.9_75	75.0 (0.0)	75	3.7	75.0 (0.0)	75	0.948	–	–
hamming8-4	16.0 (0.0)	16	1.7	16.0 (0.0)	16	0.597	–	–
hamming10-4	39.8 (0.6)	40	14.2	40.0 (0.0)	40	1.427	1.054	0.217
keller4	11.0 (0.0)	11	1.3	11.0 (0.0)	11	0.730	–	–
keller5	26.9 (0.3)	27	9.1	26.9 (0.3)	27	1.275	0.000	0.393
keller6	53.4 (1.2)	56	53.6	53.3 (0.7)	54	6.210	0.228	0.381
p_hat300-1	8.0 (0.0)	8	2.0	8.0 (0.0)	8	1.191	–	–
p_hat300-2	25.0 (0.0)	25	2.0	25.0 (0.0)	25	0.705	–	–
p_hat300-3	36.0 (0.0)	36	2.3	36.0 (0.0)	36	0.972	–	–
p_hat700-1	11.0 (0.0)	11	5.6	11.0 (0.0)	11	1.772	–	–
p_hat700-2	44.0 (0.0)	44	7.6	44.0 (0.0)	44	1.407	–	–
p_hat700-3	62.0 (0.0)	62	11.1	62.0 (0.0)	62	1.233	–	–
p_hat1500-1	11.1 (0.3)	12	16.8	11.7 (0.5)	12	3.933	3.254	0.006
p_hat1500-2	65.0 (0.0)	65	24.6	65.0 (0.0)	65	4.182	–	–
p_hat1500-3	93.7 (0.5)	94	29.2	94.0 (0.0)	94	2.408	1.897	0.072

been executed, the model is updated. The average is computed by counting, for each node i, how many searchers include node i in their maximum clique. In order to reduce noise, only searchers providing cliques whose size is comparable (within a tolerance $\Delta \in \mathbb{N}$) with the largest one are taken into account.

The model $\boldsymbol{p}$ is used within function `Restart` (see the RLS pseudo-code in Fig. 6) in order to build an initial clique with the most probable nodes. In detail, the initial clique is built in a greedy fashion, where candidate nodes at every step are selected with probability proportional to model values $\boldsymbol{p}$. The model is the only data that is globally shared by all searchers.

C. Complexity of the Proposed Solutions

The removal of the memory reaction structure makes R-EVO simpler than RLS-EVO from several points of view. The memory requirements of a single individual are greatly reduced because no history needs to be recorded. There is no need of implementing and maintaining history-related structures, the code is easier to implement and to maintain, and its memory footprint is smaller.

Experimental results shown in Section IV-C, on the other hand, suggest that the time complexity of the two algorithms, expressed as CPU time per iteration, is comparable: the history maintenance overhead is only expensive in terms of code size and implementation.

A comparison between the proposed algorithms and EA/G, however, is not straightforward: while the RLS-EVO code is more complex than EA/G because of the history structure, and hence its memory requirements, the EA/G and R-EVO techniques have similar memory footprints, i.e., the data structures to efficiently handle the clique maintenance or repair operations, plus the probabilistic model. Computational experiments in Section IV show that time complexity per iteration of both R-EVO and RLS-EVO is actually lower with

TABLE II
COMPARISON BETWEEN EA/G AND R-EVO FOR A NUMBER OF ITERATIONS PROPORTIONAL TO THE MAXIMUM DETECTED CLIQUE

	EA/G			R-EVO (20 000×clique size)			t-test	
	Avg. (std)	Best	Time (s)	Avg. (std)	Best	Time (s)	t	Sig.
C125.9	34.0 (0.0)	34	1.3	34.0 (0.0)	34	1.215	–	–
C250.9	44.0 (0.0)	44	2.5	44.0 (0.0)	44	3.237	–	–
C500.9	55.2 (0.9)	56	4.8	57.0 (0.0)	57	4.227	6.325	0.000
C1000.9	64.4(1.4)	67	18.0	68.0 (0.0)	68	10.165	8.132	0.000
C2000.9	70.9(1.0)	72	38.4	76.5 (0.5)	77	29.196	15.839	0.000
DSJC500_5	13.0 (0.0)	13	4.0	13.0 (0.0)	13	1.573	–	–
DSJC1000_5	14.5 (0.3)	15	10.3	15.0 (0.0)	15	3.732	5.270	0.000
C2000.5	14.9 (0.7)	16	24.3	16.0 (0.0)	16	6.833	4.969	0.001
C4000.5	16.1 (0.3)	17	51.9	17.1 (0.3)	18	15.729	7.454	0.000
MANN_a27	126.0 (0.0)	126	10.3	125.8 (0.4)	126	7.991	1.581	0.114
MANN_a45	343.7 (0.7)	345	68.2	342.5 (0.5)	343	46.859	4.411	0.000
MANN_a81	1097.2 (0.6)	1098	705.1	1096.7 (0.5)	1097	438.570	2.024	0.056
brock200_2	12.0 (0.0)	12	1.5	11.4 (0.5)	12	0.654	3.795	0.003
brock200_4	16.5 (0.5)	17	1.7	16.1 (0.3)	17	1.249	2.169	0.044
brock400_2	24.7 (0.4)	25	3.1	25.0 (0.0)	25	1.794	2.372	0.034
brock400_4	25.1(2.6)	33	3.3	25.0 (0.0)	25	2.965	0.122	0.385
brock800_2	20.1 (0.4)	21	7.6	21.0 (0.0)	21	3.725	7.115	0.000
brock800_4	19.9 (0.5)	21	7.6	21.0 (0.0)	21	2.977	6.957	0.000
gen200_p0.9_44	44.0 (0.0)	44	1.8	44.0 (0.0)	44	1.789	–	–
gen200_p0.9_55	55.0 (0.0)	55	3.3	55.0 (0.0)	55	2.026	–	–
gen400_p0.9_55	51.8 (0.7)	55	3.6	55.0 (0.0)	55	3.378	14.456	0.000
gen400_p0.9_65	65.0 (0.0)	65	3.6	65.0 (0.0)	65	3.744	–	–
gen400_p0.9_75	75.0 (0.0)	75	3.7	75.0 (0.0)	75	5.114	–	–
hamming8-4	16.0 (0.0)	16	1.7	16.0 (0.0)	16	1.814	–	–
hamming10-4	39.8 (0.6)	40	14.2	40.0 (0.0)	40	8.110	1.054	0.217
keller4	11.0 (0.0)	11	1.3	11.0 (0.0)	11	0.837	–	–
keller5	26.9 (0.3)	27	9.1	26.8 (0.4)	27	3.680	0.632	0.319
keller6	53.4 (1.2)	56	53.6	53.7 (0.7)	55	34.573	0.683	0.307
p_hat300-1	8.0 (0.0)	8	2.0	8.0 (0.0)	8	1.149	–	–
p_hat300-2	25.0 (0.0)	25	2.0	25.0 (0.0)	25	2.774	–	–
p_hat300-3	36.0 (0.0)	36	2.3	36.0 (0.0)	36	2.194	–	–
p_hat700-1	11.0 (0.0)	11	5.6	11.0 (0.0)	11	1.950	–	–
p_hat700-2	44.0 (0.0)	44	7.6	44.0 (0.0)	44	6.166	–	–
p_hat700-3	62.0 (0.0)	62	11.1	62.0 (0.0)	62	12.579	–	–
p_hat1500-1	11.1 (0.3)	12	16.8	11.8 (0.4)	12	5.181	4.427	0.000
p_hat1500-2	65.0 (0.0)	65	24.6	65.0 (0.0)	65	18.577	–	–
p_hat1500-3	93.7 (0.5)	94	29.2	94.0 (0.0)	94	24.735	1.897	0.072

respect to EA/G, although fundamental differences between the algorithms make such comparison hard to assess.

IV. COMPUTATIONAL EXPERIMENTS

All experiments have been performed with the following parameters, derived from [14]: population of $N = 10$ individuals, ten runs per problem instance, model depth $\Delta = 3$; the restart parameter A in Fig. 6 is $100 \cdot Q_{\text{best}}$. The EA/G data are obtained from [14], where $20,000$ calls of the repair operator per run were considered. The construction of a new individual in EA/G requires a sequence of node additions and removals, while an R-EVO iteration only performs one node addition or removal, so a direct comparison is not possible. To solve this problem, two series of experiments have been performed: one with a fixed but larger number of iterations, another with a number of iterations proportional to the maximum clique found. Both choices are motivated in the following sections.

Our experiments were executed on a Dual Xeon 3.4 GHz machine with 6 GB RAM and Linux 2.6 operating system. However, all tested algorithms were implemented as monolithic processes, so only a single processor core was in use at any time, and no CPU core parallelism has been exploited. To take hardware differences into account, some runs of EA/G have been reproduced on the same machine where we tested R-EVO. Execution time averages in our machine were systematically lower than those reported in [14] by 15%, so the EA/G time figures have been rescaled by multiplying them by a 0.85 speedup factor.

Following [14], we also chose to report execution times rather than number evaluations of clique size (our fitness function). The number of function evaluations, in fact, is mostly relevant when the fitness function is complex with respect to the heuristic execution time. If fitness is measured by clique size, simple optimizations (such as those described in [21]) allow to evaluate it incrementally with very few memory accesses. In such case, the time spent by the heuristic

TABLE III
INTERNAL COMPARISON BETWEEN A POPULATION OF TEN R-EVO SEARCHERS, A POPULATION OF TEN RLS-EVO SEARCHERS AND A MIXED POPULATION WITH FIVE INDIVIDUALS OF EACH TYPE

	R-EVO			RLS-EVO			Mix		
	Avg. (std)	Best	Time (s)	Avg. (std)	Best	Time (s)	Avg. (std)	Best	Time (s)
C125.9	34.0 (0.0)	34	1.215	34.0 (0.0)	34	1.147	34.0 (0.0)	34	1.509
C250.9	44.0 (0.0)	44	3.237	44.0 (0.0)	44	2.585	44.0 (0.0)	44	2.978
C500.9	57.0 (0.0)	57	4.227	57.0 (0.0)	57	5.252	57.0 (0.0)	57	6.231
C1000.9	68.0 (0.0)	68	10.165	67.9 (0.3)	68	8.471	67.9 (0.3)	68	13.337
C2000.9	76.5 (0.5)	77	29.196	76.2 (0.6)	77	18.116	76.2 (0.6)	77	29.219
DSJC500_5	13.0 (0.0)	13	1.573	13.0 (0.0)	13	1.489	13.0 (0.0)	13	2.247
DSJC1000_5	15.0 (0.0)	15	3.732	14.9 (0.3)	15	3.489	14.9 (0.3)	15	4.749
C2000.5	16.0 (0.0)	16	6.833	16.0 (0.0)	16	6.786	16.0 (0.0)	16	10.546
C4000.5	17.1 (0.3)	18	15.729	17.0 (0.0)	17	15.555	17.0 (0.0)	17	24.266
MANN_a27	125.8 (0.4)	126	7.991	125.7 (0.5)	126	8.034	125.7 (0.5)	126	12.267
MANN_a45	342.5 (0.5)	343	46.859	342.6 (0.5)	343	46.445	342.6 (0.5)	343	71.956
MANN_a81	1096.7 (0.5)	1097	438.570	1097.0 (0.5)	1098	424.238	1097.0 (0.5)	1098	688.045
brock200_2	11.4 (0.5)	12	0.654	12.0 (0.0)	12	0.994	12.0 (0.0)	12	0.985
brock200_4	16.1 (0.3)	17	1.249	16.6 (0.5)	17	0.821	16.6 (0.5)	17	1.161
brock400_2	25.0 (0.0)	25	1.794	25.0 (0.0)	25	1.715	25.0 (0.0)	25	2.663
brock400_4	25.0 (0.0)	25	2.965	25.8 (2.5)	33	1.961	25.8 (2.5)	33	2.753
brock800_2	21.0 (0.0)	21	3.725	21.0 (0.0)	21	3.845	21.0 (0.0)	21	4.566
brock800_4	21.0 (0.0)	21	2.977	21.0 (0.0)	21	2.965	21.0 (0.0)	21	4.591
gen200_p0.9_44	44.0 (0.0)	44	1.789	44.0 (0.0)	44	1.721	44.0 (0.0)	44	2.567
gen200_p0.9_55	55.0 (0.0)	55	2.026	55.0 (0.0)	55	2.233	55.0 (0.0)	55	3.305
gen400_p0.9_55	55.0 (0.0)	55	3.378	55.0 (0.0)	55	3.356	55.0 (0.0)	55	5.095
gen400_p0.9_65	65.0 (0.0)	65	3.744	65.0 (0.0)	65	4.140	65.0 (0.0)	65	6.190
gen400_p0.9_75	75.0 (0.0)	75	5.114	75.0 (0.0)	75	4.827	75.0 (0.0)	75	7.248
hamming8-4	16.0 (0.0)	16	1.814	16.0 (0.0)	16	1.404	16.0 (0.0)	16	1.417
hamming10-4	40.0 (0.0)	40	8.110	40.0 (0.0)	40	5.911	40.0 (0.0)	40	9.215
keller4	11.0 (0.0)	11	0.837	11.0 (0.0)	11	0.883	11.0 (0.0)	11	0.893
keller5	26.8 (0.4)	27	3.680	27.0 (0.0)	27	5.445	27.0 (0.0)	27	5.395
keller6	53.7 (0.7)	55	34.573	59.0 (0.0)	59	28.781	59.0 (0.0)	59	44.309
p_hat300-1	8.0 (0.0)	8	1.149	8.0 (0.0)	8	0.790	8.0 (0.0)	8	1.128
p_hat300-2	25.0 (0.0)	25	2.774	25.0 (0.0)	25	1.573	25.0 (0.0)	25	2.282
p_hat300-3	36.0 (0.0)	36	2.194	36.0 (0.0)	36	2.943	36.0 (0.0)	36	2.891
p_hat700-1	11.0 (0.0)	11	1.950	11.0 (0.0)	11	2.022	11.0 (0.0)	11	2.860
p_hat700-2	44.0 (0.0)	44	6.166	44.0 (0.0)	44	5.150	44.0 (0.0)	44	7.870
p_hat700-3	62.0 (0.0)	62	12.579	62.0 (0.0)	62	10.085	62.0 (0.0)	62	10.121
p_hat1500-1	11.8 (0.4)	12	5.181	11.6 (0.5)	12	5.056	11.6 (0.5)	12	6.613
p_hat1500-2	65.0 (0.0)	65	18.577	65.0 (0.0)	65	14.695	65.0 (0.0)	65	23.351
p_hat1500-3	94.0 (0.0)	94	24.735	94.0 (0.0)	94	19.649	94.0 (0.0)	94	30.629

for its bookkeeping operations (model maintenance, internal data structures) becomes significant.

A. Fixed Number of Iterations

The first series of tests was performed on R-EVO with 200 000 solver steps (20 000 steps per solver). The number of iterations has been chosen in order to approach the amount of work (in terms of adjacency matrix access operations) that the EA/G heuristic performs at every iteration. With this choice the execution times of R-EVO are significantly lower than those allowed for EA/G, therefore, the comparison is unbalanced in favor of EA/G.

Results are reported in Table I. The first set of columns reports the average maximum clique found (with the corresponding standard error), the overall maximum and the average execution time for ten runs of the EA/G algorithm; the second set of columns contains the corresponding results for the R-EVO algorithm. Finally, the results of a student's *t*-test for equality of means with unequal sample variance [32] has been applied in order to test the equality of the two clique size averages: the last column contains the significance of the null hypothesis, i.e., that the two distributions have the same average. Note that some tests could not be performed due to null variance in both algorithms.

The results show that the R-EVO algorithm has a significant superiority to EA/G in the case of dense random graphs (the `C*.9` and `DSJC*` lines), for example the average size is 7% better in `C2000.9`; for the small instances both algorithms always locate the maximum clique. The only `gen*`-type instance (random graphs embedding a known clique) which was not always solved by both algorithms is the 400-node graph with 55-node clique, where R-EVO outperforms EA/G. Also, the `p_hat1500-*` instances (random graphs with higher spread in node degree) and the `hamming*` instances

TABLE IV

Comparison Between EA/G, a Population of Ten HC-evo (Hill-Climbing) Searchers, a Population of Ten R-evo Searchers and the Sequential RLS Heuristic [21]

	EA/G			HC-evo			R-EVO			RLS		
	Avg. (std)	Best	Time (s)	Avg. (std)	Best	Time (s)	Avg. (std)	Best	Time (s)	Avg. (std)	Best	Time (s)
C125.9	34.0 (0.0)	34	1.3	34.0 (0.0)	34	0.798	34.0 (0.0)	34	1.215	34.0 (0.0)	34	0.835
C250.9	44.0 (0.0)	44	2.5	44.0 (0.0)	44	1.706	44.0 (0.0)	44	3.237	44.0 (0.0)	44	1.889
C500.9	55.2 (0.9)	56	4.8	56.9 (0.3)	57	3.540	57.0 (0.0)	57	4.227	57.0 (0.0)	57	3.101
C1000.9	64.4 (1.4)	67	18.0	65.5 (0.5)	66	7.257	68.0 (0.0)	68	10.165	68.0 (0.0)	68	6.320
C2000.9	70.9 (1.0)	72	38.4	73.9 (1.1)	76	15.338	76.5 (0.5)	77	29.196	78.0 (0.0)	78	18.022
DSJC500_5	13.0 (0.0)	13	4.0	12.5 (0.5)	13	1.331	13.0 (0.0)	13	1.573	13.0 (0.0)	13	1.023
DSJC1000_5	14.5 (0.3)	15	10.3	14.1 (0.3)	15	2.904	15.0 (0.0)	15	3.732	15.0 (0.0)	15	2.175
C2000.5	14.9 (0.7)	16	24.3	15.0 (0.0)	15	6.346	16.0 (0.0)	16	6.833	16.0 (0.0)	16	4.433
C4000.5	16.1 (0.3)	17	51.9	16.1 (0.3)	17	14.827	17.1 (0.3)	18	15.729	18.0 (0.0)	18	9.931
MANN_a27	126.0 (0.0)	126	10.3	125.8 (0.4)	126	6.372	125.8 (0.4)	126	7.991	126.0 (0.0)	126	6.302
MANN_a45	343.7 (0.7)	345	68.2	342.4 (0.5)	343	39.407	342.5 (0.5)	343	46.859	344.1 (0.3)	345	33.010
MANN_a81	1097.2 (0.6)	1098	705.1	1096.9 (0.6)	1098	358.437	1096.7 (0.5)	1097	438.570	1098.0 (0.0)	1098	131.420
brock200_2	12.0 (0.0)	12	1.5	10.6 (0.5)	11	0.509	11.4 (0.5)	12	0.654	12.0 (0.0)	12	0.423
brock200_4	16.5 (0.5)	17	1.7	15.9 (0.3)	16	0.663	16.1 (0.3)	17	1.249	17.0 (0.0)	17	0.638
brock400_2	24.7 (0.4)	25	3.1	24.2 (0.4)	25	1.647	25.0 (0.0)	25	1.794	29.0 (0.0)	29	0.988
brock400_4	25.1 (2.6)	33	3.3	25.0 (0.0)	25	1.623	25.0 (0.0)	25	2.965	33.0 (0.0)	33	1.233
brock800_2	20.1 (0.4)	21	7.6	20.0 (0.5)	21	2.777	21.0 (0.0)	21	3.725	21.0 (0.0)	21	2.190
brock800_4	19.9 (0.5)	21	7.6	20.2 (0.4)	21	2.690	21.0 (0.0)	21	2.977	21.0 (0.0)	21	1.504
gen200_p0.9_44	44.0 (0.0)	44	1.8	40.3 (1.3)	44	1.324	44.0 (0.0)	44	1.789	44.0 (0.0)	44	0.880
gen200_p0.9_55	55.0 (0.0)	55	3.3	55.0 (0.0)	55	1.734	55.0 (0.0)	55	2.026	55.0 (0.0)	55	1.573
gen400_p0.9_55	51.8 (0.7)	55	3.6	51.8 (0.4)	52	2.952	55.0 (0.0)	55	3.378	55.0 (0.0)	55	1.375
gen400_p0.9_65	65.0 (0.0)	65	3.6	55.9 (6.3)	65	2.969	65.0 (0.0)	65	3.744	65.0 (0.0)	65	1.755
gen400_p0.9_75	75.0 (0.0)	75	3.7	58.5 (11.4)	75	3.097	75.0 (0.0)	75	5.114	75.0 (0.0)	75	2.070
hamming8-4	16.0 (0.0)	16	1.7	16.0 (0.0)	16	0.741	16.0 (0.0)	16	1.814	16.0 (0.0)	16	1.113
hamming10-4	39.8 (0.6)	40	14.2	40.0 (0.0)	40	5.429	40.0 (0.0)	40	8.110	40.0 (0.0)	40	3.043
keller4	11.0 (0.0)	11	1.3	11.0 (0.0)	11	0.451	11.0 (0.0)	11	0.837	11.0 (0.0)	11	0.445
keller5	26.9 (0.3)	27	9.1	26.8 (0.6)	27	2.989	26.8 (0.4)	27	3.680	27.0 (0.0)	27	2.880
keller6	53.4 (1.2)	56	53.6	53.4 (1.3)	55	23.347	53.7 (0.7)	55	34.573	59.0 (0.0)	59	25.549
p_hat300-1	8.0 (0.0)	8	2.0	8.0 (0.0)	8	0.622	8.0 (0.0)	8	1.149	8.0 (0.0)	8	0.448
p_hat300-2	25.0 (0.0)	25	2.0	25.0 (0.0)	25	1.455	25.0 (0.0)	25	2.774	25.0 (0.0)	25	1.014
p_hat300-3	36.0 (0.0)	36	2.3	35.9 (0.3)	36	1.747	36.0 (0.0)	36	2.194	36.0 (0.0)	36	0.988
p_hat700-1	11.0 (0.0)	11	5.6	9.6 (1.0)	11	1.553	11.0 (0.0)	11	1.950	11.0 (0.0)	11	1.043
p_hat700-2	44.0 (0.0)	44	7.6	44.0 (0.0)	44	5.258	44.0 (0.0)	44	6.166	44.0 (0.0)	44	4.132
p_hat700-3	62.0 (0.0)	62	11.1	62.0 (0.0)	62	6.503	62.0 (0.0)	62	12.579	62.0 (0.0)	62	8.443
p_hat1500-1	11.1 (0.3)	12	16.8	11.0 (0.0)	11	3.888	11.8 (0.4)	12	5.181	12.0 (0.0)	12	4.100
p_hat1500-2	65.0 (0.0)	65	24.6	65.0 (0.0)	65	15.700	65.0 (0.0)	65	18.577	65.0 (0.0)	65	9.910
p_hat1500-3	93.7 (0.5)	94	29.2	94.0 (0.0)	94	19.688	94.0 (0.0)	94	24.735	94.0 (0.0)	94	11.845

(graphs of bit strings with connections between words if they are far apart) show a slight superiority of R-EVO in the cases that are not optimally solved by both algorithms.

The performance tends to be more difficult to assess in the Brockington–Culberson graphs (the `brock*` lines), where the best clique is hidden in a very effective manner so that intelligent techniques tend to be not competitive with respect to brute force local search (in fact, the camouflaging process is designed to *fool* intelligent techniques).

The R-EVO algorithm does not perform at the EA/G level on Steiner Triple Problem graphs (the `MANN*` lines), however, the t-test figures are uncertain in the larger case, where both techniques could find a (nonoptimal, however) 1098-node clique. Finally, while average cliques are comparable in the `keller6` instance, the ten EA/G runs find a larger clique.

B. Iterations Proportional to (Estimated) Maximum Clique Size

The CPU time differences in the previous series of experiments tended to increase on larger graphs, in particular on graphs with larger cliques. In order to attain a more fair comparison, we chose a different termination criterion for R-EVO by limiting the overall number of iterations to $20\,000 \times \max\{Q_i\}$, where Q_i is the maximum clique found by searcher R_i (see Fig. 11).

The results are reported on Table II. We can see that all cases where the null hypothesis is rejected with high confidence (significance column < 0.05) R-EVO outperforms EA/G, with the exception of some Steiner Triple Problem (`MANN*`) and Brockington–Culberson (`brock*`) instances. Note that time differences have greatly reduced; in particular, now large graphs are searched for a time that is in the same order of magnitude as EA/G.

C. Internal Comparison Between R-EVO and RLS-EVO

Another set of experiments has been devoted to a comparison between the full-fledged RLS-EVO heuristic and its much simpler low-knowledge R-EVO counterpart used in the previous experiments.

Results are shown in Table III. The simpler version shows some marginal performance degradation, but in many cases experimental variance is very large. For instance, the more complex RLS-EVO found the 33-node clique of `brock400_4` once, and the higher average is due to this single event. Execution times of the two heuristics are comparable; this means that the maintenance of the additional memory structure has no significant impact on the CPU effort needed to perform one iteration.

The last group of columns in Table III refers to a final experiment with a mixed population: five searchers implement the RLS-EVO algorithms, five implement R-EVO. The results show that in some cases this technique helps achieving the "best of both worlds," however, in other cases a slight degradation can be observed. As examples, the `brock400_4` and `keller6` cases show that the mixed technique can obtain a performance equal to the best of its components.

Finally, in order to better assess the extent to which the improvement with respect to EA/G can be attributed to the reactive tabu technique, Table IV shows some results obtained by removing the tabu mechanism, therefore, reducing the complexity of the individual searcher to a minimum hill climbing column (HC-EVO). The HC-EVO heuristic has a lower iteration time compared with R-EVO, every iteration just selects the best candidate node for insertion or removal. It can be observed that in some cases HC-EVO is less robust than EA/G in terms of solution quality, with a higher risk of missing the maximum clique. The graph instances where this disadvantage is more apparent are the Brockington–Culberson graphs (`brock`*nnn*) and some of the harder `gen`-type graph, where the naïf hill climbing strategy needs to be complemented with some more complex behavior in order to reach particularly isolated gobal optima. The same instances are known to be problematic also for the RLS technique when compared to other types of local search methods, see for instance [33].

A final observation, also shown in Table IV, concerns the comparison between the population-based techniques proposed in this paper and the original RLS heuristic as proposed in [21]. The original sequential heuristic is still faster. However, RLS is a much more memory-intensive technique, and we believe that comparisons with sequential techniques would be unfair toward population-based heuristics, which still show some disadvantages in certain combinatorial problems. The purpose of this paper is, in fact, the exploration of a hybrid approach which can lead to some important insight on the tradeoff between individual intelligence and number of actors, more than a brutal "horse race" aimed at the best possible performance.

V. Conclusion

The paper presented a hybrid algorithm that uses an evolutionary scheme in the framework of "estimation of distribution algorithms" to generate new individuals, which are then subjected to memetic evolution through a simplified RSO method [1]. In this manner, each individual in the population executes a short local search with prohibition. The prohibition period is determined in a simple reactive manner on a specific instance based on the estimated size of the maximum clique.

The results show that the proposed technique is competitive with respect to state-of-the-art evolutionary algorithms based on a similar population-based framework (the EA/G algorithm).

It is remarkable how a drastic simplification of the original RLS algorithm, complemented by the interaction of more population members through a model derived by EDA is capable of achieving results which are comparable to those obtained by a very complex individual searcher. Coupling a limited form of "intelligence" (actually a low-knowledge reactive local search technique) with an evolutionary scheme achieves state-of-the-art results on the MC problem.

Acknowledgment

The authors would like to thank Q. Zhang for sending the original software used in [14], and to acknowledge support by F. Mascia for developing part of the R-EVO software used in the experimental part.

References

[1] R. Battiti, M. Brunato, and F. Mascia, *Reactive Search and Intelligent Optimization* (Operations Research/Computer Science Interfaces Series 45). New York: Springer, Nov. 2008.

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

[3] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Toward memetic algorithms," Caltech Concurrent Computation Program, California Inst. of Technol., Pasadena, C3P Tech. Rep. 826, 1989.

[4] P. Moscato and C. Cotta, "A gentle introduction to memetic algorithms," in *Handbook of Metaheuristics*. Norwell, MA: Kluwer, 2003, pp. 105–144.

[5] M. Clerc and J. Kennedy, "The particle swarm: Explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58–73, Feb. 2002.

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

[7] S. Baluja and R. Caruana, "Removing the genetics from the standard genetic algorithm," School Comput. Sci., Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-CS-95-141, 1995.

[8] H. Mühlenbein, "The equation for response to selection and its use for prediction," *Evol. Comput.*, vol. 5, no. 3, pp. 303–346, 1998.

[9] M. Pelikan, D. Goldberg, and E. Cantu-Paz, "BOA: The Bayesian optimization algorithm," in *Proc. Genetic Evol. Comput. Conf.*, vol. 1. 1999, pp. 525–532.

[10] P. Larranaga, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Norwell, MA: Kluwer Academic, 2001.

[11] F. Glover, "Scatter search and star-paths: Beyond the genetic metaphor," *Operations Res. Spektrum*, vol. 17, nos. 2–3, pp. 125–138, 1995.

[12] F. Glover, M. Laguna, and R. Martí, "Scatter search," in *Advances in Evolutionary Computation: Theory and Applications*, A. Ghosh and S. Tsutsui, Eds. New York: Springer-Verlag, 2003, pp. 519–537.

[13] F. Glover, "Tabu search: Part I," *Oper. Res. Soc. Am. J. Comput.*, vol. 1, no. 3, pp. 190–260, 1989.

[14] Q. Zhang, J. Sun, and E. Tsang, "An evolutionary algorithm with guided mutation for the maximum clique problem," *IEEE Trans. Evol. Comput.*, vol. 9, no. 2, pp. 192–200, Apr. 2005.

[15] E. Marchiori, "Genetic, iterated and multistart local search for the maximum clique problem," in *Applications of Evolutionary Computing*, LNCS 2279, S. Cagnoni, *et al.*, Eds. Berlin, Germany: Springer, 2002, pp. 112–121.

[16] J. S. de Bonet, C. L. Isbell, Jr., and P. Viola, "MIMIC: Finding optima by estimating probability densities," in *Advances in Neural Information Processing Systems*, vol. 9, M. C. Mozer, M. I. Jordan, and T. Petsche, Eds. Cambridge, MA: MIT Press, 1997, p. 424.

[17] P. Pardalos and J. Xue, "The maximum clique problem," *J. Global Optimization*, vol. 4, no. 3, pp. 301–328, 1994.

[18] Y. Ji, X. Xu, and G. Stormo, "A graph theoretical approach to predict common RNA secondary structure motifs including pseudoknots in unaligned sequences," *Bioinformatics*, vol. 20, no. 10, pp. 1591–1602, 2004.

[19] S. Butenko and W. Wilhelm, "Clique-detection models in computational biochemistry and genomics," *Eur. J. Oper. Res.*, vol. 173, no. 1, pp. 1–17, 2006.

[20] J. Håstad, "Clique is hard to approximate within $n^{1-\epsilon}$," in *Proc. 37th Ann. IEEE Symp. Foundations Comput. Sci.*, 1996, pp. 627–636.

[21] R. Battiti and M. Protasi, "Reactive local search for the maximum clique problem," *Algorithmica*, vol. 29, no. 4, pp. 610–637, 2001.

[22] R. Battiti and G. Tecchiolli, "The reactive tabu search," *Oper. Res. Soc. Am. J. Comput.*, vol. 6, no. 2, pp. 126–140, 1994.

[23] R. Battiti and A. A. Bertossi, "Greedy, prohibition, and reactive heuristics for graph partitioning," *IEEE Trans. Comput.*, vol. 48, no. 4, pp. 361–385, Apr. 1999.

[24] H. Mühlenbein and G. Paaß, "From recombination of genes to the estimation of distributions I. binary parameters," in *Proc. 4th Parallel Problem Solving Nature,* 1996, pp. 178–187.

[25] M. Pelikan, D. Goldberg, and F. Lobo, "A survey of optimization by building and using probabilistic models," *Comput. Optimization Appl.*, vol. 21, no. 1, pp. 5–20, 2002.

[26] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. Global Optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[27] M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo, "Model-based search for combinatorial optimization," *Ann. Operations Res.*, vol. 131, nos. 1–4, pp. 373–395, 2004.

[28] J. Hertz, A. Krogh, and R. Palmer, *Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison-Wesley, 1991.

[29] F. Glover and M. Laguna, *Tabu Search*. Norwell, MA: Kluwer, 1997.

[30] R. Battiti and M. Protasi, "Reactive local search for the maximum clique problem," Int. Comput. Sci. Inst., Berkeley, CA, Tech. Rep. TR-95-052, Sep. 1995.

[31] R. Battiti and F. Mascia, "Reactive local search for maximum clique: A new implementation," Univ. Trento, Trento, Italy, Tech. Rep. DIT-07-018, May 2007.

[32] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*, 2nd ed. Cambridge, U.K.: Cambridge University Press, 1992, ch. 14, pp. 620–624.

[33] W. Pullan and H. H. Hoos, "Dynamic local search for the maximum clique problem," *J. Artif. Intell. Res.*, vol. 25, pp. 159–185, Feb. 2006.

Mauro Brunato received the B.S. and M.S. degrees in mathematics from the University of Padua, Padua, Italy, in 1994 and the Ph.D. degree in mathematics from the University of Trento, Trento, Italy, in 1999.

Currently, he is an Assistant Professor with the Department of Information Engineering and Computer Science, University of Trento. His current research interests include stochastic algorithms for hard combinatorial and continuous problems, and data visualization techniques for human-assisted optimization.

Roberto Battiti (F'09) received the B.S. and M.S. degrees in physics from the University of Trento, Trento, Italy, in 1985, and the Ph.D. degree in computation and neural systems from the California Institute of Technology, Pasadena, in 1990.

He started his research in the area of parallel computing and neural networks. He is currently a Professor of computer science with the Department of Information Engineering and Computer Science, University of Trento, where he is also the Director of the Machine Learning and Intelligent Optimization Laboratory. His current research interests include heuristic algorithms for optimization problems, in particular reactive search optimization algorithms for discrete optimization problems. He has made contributions to machine learning techniques and for intelligent optimization and neural networks.