A biased random-key genetic algorithm for the maximum quasi-clique problem

Bruno Q. Pinto[a,b], Celso C. Ribeiro[*b], Isabel Rosseti[b], Alexandre Plastino[b]

[a] *Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro, Uberlândia, MG 38411-104, Brazil*
[b] *Universidade Federal Fluminense, Institute of Computing, Niterói, RJ 24210-240, Brazil*

Abstract

Given a graph $G = (V, E)$ and a threshold $\gamma \in (0, 1]$, the maximum cardinality quasi-clique problem consists in finding a maximum cardinality subset C^* of the vertices in V such that the density of the graph induced in G by C^* is greater than or equal to the threshold γ. This problem is NP-hard, since it admits the maximum clique problem as a special case. It has a number of applications in data mining, e.g. in social networks or phone call graphs. In this work, we propose a biased random-key genetic algorithm for solving the maximum cardinality quasi-clique problem. Two alternative decoders are implemented for the biased random-key genetic algorithm and the corresponding algorithm variants are evaluated. Computational results show that the newly proposed approaches improve upon other existing heuristics for this problem in the literature. All input data for the test instances and all detailed numerical results are available from Mendeley.

Keywords: metaheuristics, biased random-key genetic algorithm, maximum quasi-clique problem, maximum clique problem, graph density

1. Introduction

Let $G = (V, E)$ be a graph defined by a vertex set V and an edge set $E \subseteq V \times V$. A graph $G' = (V', E')$ is a subgraph of G if $V' \subseteq V$ and $E' \subseteq E$, which is denoted by $G' \subseteq G$. The graph $G(V')$ induced in G by $V' \subseteq V$ is that with vertex set V' and edge set $E(V') \subseteq E$ formed by all edges of E with both ends in V'.

Email addresses: `bruno.queiroz@iftm.edu.br` (Bruno Q. Pinto), `celso@ic.uff.br` (Celso C. Ribeiro*), `rosseti@ic.uff.br` (Isabel Rosseti), `plastino@ic.uff.br` (Alexandre Plastino)

*Corresponding author

The density of graph G is given by $dens(G) = |E|/(|V| \times (|V| - 1)/2)$. The degree $deg_G(v)$ of a node $v \in V$ denotes the number of vertices in G that are adjacent to v.

A graph is complete if there is an edge connecting any pair of its vertices. A subset $C \subseteq V$ is a clique of G if the graph $G(C)$ induced in G by C is complete. Given a graph $G = (V, E)$, the *maximum clique problem* consists in finding a maximum cardinality clique of G. It was proved to be NP-hard by Karp (1972).

Given a graph $G = (V, E)$ and a threshold $\gamma \in (0, 1]$, a γ-clique is any subset $C \subseteq V$ such that the density of the subgraph $G(C)$ is greater than or equal to γ. A γ-clique C is maximal if there is no other γ-clique C' that strictly contains C. The *maximum quasi-clique problem* (MQCP) amounts to finding a maximum cardinality subset C^* of the vertices in V such that the density of the graph induced in G by C^* is greater than or equal to the threshold γ. This problem is also NP-hard, since it admits the maximum clique problem as a special case in which $\gamma = 1$, see (Pattillo et al., 2013). The problem has many applications and related clustering approaches include classifying molecular sequences in genome projects by using a linkage graph of their pairwise similarities (Brunato et al., 2008) and the analysis of massive communication data sets obtained from social networks or phone call graphs (Abello et al., 1999), as well as various data mining and graph mining applications.

A few heuristics for MQCP exist in the literature, based on well known approaches such as greedy randomized algorithms and their iterated extensions (Oliveira et al., 2013), stochastic local search (Brunato et al., 2008), and GRASP (Abello et al., 2002). In this work, we propose two variants of a biased random-key genetic algorithm for solving the maximum quasi-clique problem. The remainder of this article is organized as follows. Section 2 presents the formulation of the maximum quasi-clique problem and reviews exact solution approaches. Heuristics and related work are reviewed in Section 3. Section 4 gives the overall description of biased random-key genetic algorithms and their customization to the maximum quasi-clique problem. Section 5 describes the decoders and two variants of the biased random-key genetic algorithm, each of them based on a different decoder. Computational results are presented in Section 6. Concluding remarks are drawn in the last section. The computational experiments on dense graphs showed that the biased random-key genetic algorithm outperformed the best heuristic in the literature. The experiments on sparse graphs showed that the biased random-key genetic algorithm found results that are competitive with the mixed integer programming approaches in Veremyev et al. (2016).

2. Problem formulation

The maximum quasi-clique problem can be formulated by associating a binary variable x_i to each vertex of the graph (Pattillo et al., 2013):

$$x_i = \begin{cases} 1, & \text{if vertex } v_i \in V \text{ belongs to the solution,} \\ 0, & \text{otherwise.} \end{cases}$$

This formulation also considers a variable $y_{ij} = x_i \cdot x_j$ associated to each pair of vertices $i, j \in V$, with $i < j$, and is linearized as follows:

$$\max \sum_{i \in V} x_i \tag{1}$$

subject to:

$$\sum_{(i,j) \in E : i<j} y_{ij} \geq \gamma \cdot \sum_{i,j \in V : i<j} y_{ij} \tag{2}$$

$$y_{ij} \leq x_i, \qquad \forall i, j \in V, \quad i < j, \tag{3}$$

$$y_{ij} \leq x_j, \qquad \forall i, j \in V, \quad i < j, \tag{4}$$

$$y_{ij} \geq x_i + x_j - 1, \qquad \forall i, j \in V, \quad i < j, \tag{5}$$

$$x_i \in \{0, 1\}, \qquad \forall i \in V, \tag{6}$$

$$y_{ij} \geq 0, \qquad \forall i, j \in V, \quad i < j. \tag{7}$$

The objective function (1) maximizes the number of vertices in the solution. If two vertices i, j belong to a solution, then $x_i = x_j = 1$ and $y_{ij} = x_i \cdot x_j = 1$. If edge $(i, j) \in E$, then it contributes to the density of the quasi-clique. Constraint (2) ensures that the density of the solution is greater than or equal to γ. Constraints (3) and (4) ensure that any edge may contribute to the density of a solution only if both of its ends are chosen to belong to this solution. Constraints (5) ensure that any existing edge $(i, j) \in E$ will contribute to the solution if both of its ends are chosen. Constraints (6) and (7) impose the binary and nonnegativity requirements on the problem variables, respectively.

Veremyev et al. (2016) reported and compared four mixed integer programming formulations for the maximum quasi-clique problem in sparse graphs. Two algorithms based on the best formulations led to better results than the mixed integer programming formulation proposed in Pattillo et al. (2013), with all mixed integer programs solved using FICO Xpress-Optimizer (FICO, 2017) with the time limit of 3600 seconds. Ribeiro & Riveaux (2017) developed an exact algorithm based on a quasi-hereditary property and proposed a new upper bound that is used for pruning the search tree. Numerical results showed that their approach is competitive with the best integer programming approaches in the literature and that their new upper bound is consistently tighter than previously existing bounds.

3. Heuristics and related work

Some heuristics for the maximum quasi-clique problem exist in the literature, based on well known approaches such as greedy randomized algorithms and their iterated extensions (Oliveira et al., 2013), stochastic local search (Brunato et al., 2008), and GRASP (Abello et al., 2002).

The constructive heuristic HC3 (Oliveira et al., 2013) is an adaptation of the construction phase of the algorithm developed by Abello et al. (2002). It builds an initial solution, whose density γ_{temp} is greater than or equal to the threshold γ and may be decreased by the insertion of new vertices. At each iteration, it creates a candidate list of vertices (*CL*) that can be inserted into the current solution. A restricted candidate list (*RCL*) is built with the best candidates in *CL* and a vertex is randomly selected from *RCL* to be inserted in the current solution, until the candidate list becomes empty. A parameter α is used to define the size of the restricted candidate list. The criteria summarized below are used in this order to select the vertices that will be placed in *CL* by HC3 (Abello et al., 1999, 2002; Oliveira et al., 2013):

1. Vertex degree: this criterion is applied only once. The candidate list *CL* at the first iteration is formed by all vertices of the graph. The vertices with the largest degrees are inserted into *RCL* and one of them is randomly selected as the first vertex to be part of the solution.
2. Potential difference: the candidate list *CL* is formed by all vertices whose insertion in the current solution results in a new solution whose density is greater than or equal to the current density γ_{temp}. The vertices in *RCL* are those with the largest potential differences, i.e., those whose insertion increases maximally the density of the current solution.
3. Degree in the current solution: this last criterion is applied when there is no further vertex whose insertion in the current solution increases the density γ_{temp} of the current solution. The candidate list *CL* is formed by all neighbors of the current solution. The vertices in *CL* with the largest degrees are placed in *RCL*. The density of the resulting solution decreases with respect to the current density γ_{temp} whenever this selection criterion is applied.

Other constructive heuristics proposed by Oliveira et al. (2013) start from solutions generated by the greedy randomized heuristic HC3. They alternate between two phases: partial destruction of the current solution and reconstruction of a new feasible solution using a greedy randomized algorithm to complete the partially destroyed solution. Among some variants of this approach, the iterated greedy strategy (IG), used by Ruiz & Stützle (2006) to solve the permutation flowshop scheduling problem, consists in the repeated application of the process of partial destruction, followed by the reconstruction of a feasible solution.

The optimized iterated greedy heuristic (IG*) builds an initial solution using the constructive heuristic HC3 and repeatedly applies a destruction phase followed by a reconstruction phase that applies the HC3 heuristic. Two parameters δ and β are used in the destruction phase: parameter δ controls the fraction of the vertices of the current solution that will be removed, while parameter β determines the greediness of the removal process, by controlling the size of the restricted candidate list from where each vertex will be extracted.

The restarted optimized iterated greedy (RIG*) strategy repeatedly applies IG*, until the best solution found can not be improved (Oliveira et al., 2013). Furthermore, parameter δ is dynamically modified to diversify the fraction of the solution that is destroyed in the destruction phase of IG*. This strategy outperformed the others investigated in Oliveira et al. (2013).

4. Biased random-key genetic algorithms for maximum quasi-clique

Genetic algorithms with random keys, or random-key genetic algorithms (denoted by RKGA), were first introduced by Bean (1994) for combinatorial optimization problems whose solutions may be represented by permutation vectors. Solutions are represented as vectors of randomly generated real numbers called keys. A deterministic algorithm, called a decoder, takes as input a solution vector and associates with it a feasible solution of the combinatorial optimization problem, for which an objective value or fitness can be computed. Two parents are selected at random from the entire population to implement the crossover operation in the implementation of an RKGA. Parents are allowed to be selected for mating more than once in the same generation.

A biased random-key genetic algorithm (BRKGA) differs from an RKGA in the way parents are selected for crossover, see (Gonçalves & Resende, 2011) for a review. In a BRKGA, each element is generated combining one element selected at random from the elite solutions in the current population, while the other is a non-elite solution. The selection is said to be biased because one parent is always an elite solution and has a higher probability of passing its genes to the new generation.

In the following, we propose two variants of a BRKGA for MQCP, each of them using a different decoder. Both of them evolve a population of chromosomes that consists of vectors of real numbers in the interval $[0, 1)$ associated with the vertices of the graph G. Each chromosome is decoded by an algorithm that receives the vector of keys and builds a feasible solution for MQCP, i.e., the decoder returns a γ-clique as its output. The two decoders DECODER-HCB and DECODER-IG* will be described in the next section.

We used the parameterized uniform crossover scheme proposed in Spears & de Jong (1991) to combine two parent solutions and to produce an offspring. In this scheme, the offspring inherits each of its keys from the best fit of the two parents with a higher probability. The biased random-key genetic algorithm developed in this work does not make use of the standard mutation operator, where parts of the chromosomes are changed with small probability. Instead, the concept of mutants is used: mutant solutions are introduced in the population in each generation, randomly generated in the same way as in the initial population. Mutants play the same role of the mutation operator in traditional genetic algorithms, diversifying the search and helping the procedure to escape from locally optimal solutions (Brandão et al., 2015, 2017; Noronha et al., 2011).

The keys in the chromosome are randomly generated in the initial population. At each generation, the population is partitioned into two sets: *TOP* and *REST*. The size of the population is $|TOP| + |REST|$. Subset *TOP* contains the best solutions in the population. Subset *REST* is formed by two disjoint subsets: *MID* and *BOT*, with subset *BOT* being formed by the worst elements in the current population. As illustrated in Figure 1, the chromosomes in *TOP* are simply copied to the population of the next generation. The elements in *BOT* are replaced by newly created mutants that are placed in the new set *BOT*. The remaining elements of the new population are obtained by crossover, with one parent randomly chosen from *TOP* and the other from *REST*. This distinguishes a biased random-key genetic algorithm from the random-key genetic algorithm of Bean (1994), where both parents are selected at random from the entire population. Since a parent solution can be chosen for crossover more than once in a given generation, elite solutions have a higher probability of passing their random keys to the next generation. In this way, $|MID| = |REST| - |BOT|$ offspring solutions are created.

5. Decoders

The implementation of biased random-key genetic algorithms for the maximum quasi-clique problem made use of the C++ library brkgaAPI framework developed by Toso & Resende (2015). The instantiation of the framework shown in Figure 2 to some specific optimization problem requires exclusively the development of a class implementing the decoder for this problem. This is the only problem-dependent part of the tool. Other applications of this framework in the implementation of biased random-key genetic algorithms can be seen e.g. in (Brandão et al., 2015, 2017; Chaves et al., 2016; Gonçalves et al., 2014; Ribeiro et al., 2017; Ruiz et al., 2015).

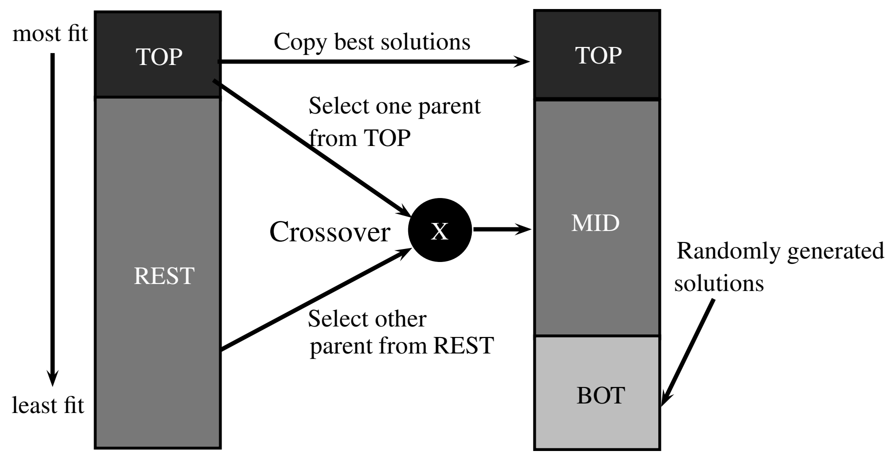

Figure 1: Population evolution between consecutive generations of a BRKGA.

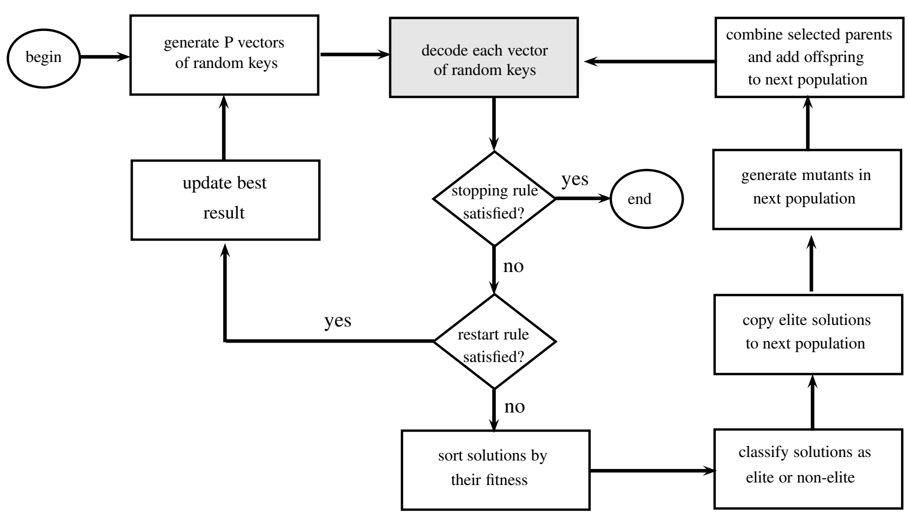

Figure 2: BRKGA framework.

According to Gonçalves et al. (2014), the BRKGA framework requires the following parameters: (a) the population size ($p = |TOP| + |REST|$); (b) the fraction *pe* of the population corresponding to the elite set *TOP*; (c) the fraction *pm* of the population corresponding to the mutant set *BOTTOM*; (d) the probability *rhoe* that the offspring inherits each of its keys from the best fit of the two parents; and (e) the number k of generations without improvement in the best solution until a restart is performed.

We developed two variants of a biased random-key genetic algorithm for solving MQCP: algorithm BRKGA-HCB makes use of the decoder DECODER-HCB based on the HCB constructive heuristic, which is an optimized implementation of the constructive heuristic HC3 (Oliveira et al., 2013), while algorithm BRKGA-IG* makes use of the decoder DECODER-IG* that applies the strategy IG* in the place of HCB. The heuristics and decoders are described next.

5.1. Constructive heuristic HC3

Algorithm 1 describes the constructive heuristic HC3, originally proposed in (Oliveira et al., 2013). It is slighted adapted from the construction phase of the algorithm developed by Abello et al. (2002). It takes as inputs the graph $G = (V, E)$, the threshold γ, and a parameter $\alpha \in [0, 1]$.

First, all vertices in V are assigned to the candidate list CL in line 1 and the restricted candidate list RCL is created in line 2, containing the $\max\{1, \alpha \cdot |CL|\}$ vertices with the largest degrees in CL. A vertex x is randomly selected from RCL in line 3 to initialize a solution S_{temp} in line 4. The density γ_{temp} of the graph $G(S_{temp})$ is set to 1 in line 5. The density γ_{temp} of $G(S_{temp})$ will be progressively reduced as new vertices are inserted into S_{temp} along the iterations of the loop in lines 6 to 34. The temporary solution S_{temp} is copied to S in line 7 and the candidate list CL is reset in line 8. The loop in lines 9 to 13 places in the candidate list CL the vertices of $V \setminus S$ that may be added to the current solution without reducing the density γ_{temp} of the corresponding γ-clique. If the candidate list CL is not empty, then the potential difference for each candidate vertex is computed in line 16 as proposed in (Abello et al., 2002). The restricted candidate list RCL is created in line 18, containing the $\max\{1, \alpha \cdot |CL|\}$ vertices with the largest potential differences in CL. Otherwise, in case the candidate list was empty, the third criterion is applied from line 19 to 30. The insertion of any new vertex in the current solution S_{temp} will lead to a reduction in the density γ_{temp} of the graph $G(S_{temp})$. A new candidate solution CL will be built in lines 20 to 24, containing all neighbors of the current solution S. If this new candidate list is also empty, then the algorithm stops and returns the current solution in line 26, since there are no more candidate vertices to be added to the current solution. Otherwise, the restricted candidate list RCL is created in line 28, containing the $\max\{1, \alpha \cdot |CL|\}$ vertices with the largest degrees

Algorithm 1 HC3(G, γ, α)

$CL \leftarrow V$
$RCL \leftarrow \{v \in CL : |\{v' \in CL : deg_G(v') \geq deg_G(v)\}| \leq \max\{1, \alpha \cdot |CL|\}\}$
Randomly select $x \in RCL$
$S_{temp} \leftarrow \{x\}$
$\gamma_{temp} \leftarrow 1$
while $\gamma_{temp} \geq \gamma$ **do**
  $S \leftarrow S_{temp}$
  $CL \leftarrow \emptyset$
  for all $v \in V \setminus S$ **do**
    if $\frac{|E(S)| + deg_{G(S)}(v)}{(|S|+1) \cdot |S|/2} \geq \gamma_{temp}$ **then**
      $CL \leftarrow CL \cup \{v\}$
    end if
  end for
  if $CL \neq \emptyset$ **then**
    for all $v \in CL$ **do**
      $dif_v \leftarrow deg_{G(CL)}(v) + |CL| \cdot (deg_{G(S)}(v) - \gamma_{temp} \cdot (|S| + 1))$
    end for
    $RCL \leftarrow \{v \in CL : |\{v' \in CL : dif(v') \geq dif(v)\}| \leq \max\{1, \alpha \cdot |CL|\}\}$
  else
    for all $v \in V \setminus S$ **do**
      if $deg_{G(S)}(v) > 0$ **then**
        $CL \leftarrow CL \cup \{v\}$
      end if
    end for
    if $CL = \emptyset$ **then**
      return S
    else
      $RCL \leftarrow \{v \in CL : |\{v' \in CL : deg_{G(S)}(v') \geq deg_{G(S)}(v)\}| \leq \max\{1, \alpha \cdot |CL|\}\}$
    end if
  end if
  Randomly select $x \in RCL$
  $S_{temp} \leftarrow S \cup \{x\}$
  $\gamma_{temp} \leftarrow |E(S_{temp})| / \binom{|S_{temp}|}{2}$
end while
return S

in CL. Since the restricted candidate list is not empty, a new vertex $x \in RCL$ is selected to be added to the current solution in line 31. The current solution S_{temp} and its density γ_{temp} are updated in lines 32 and 33, respectively, and a new iteration resumes. The algorithm returns the solution S in line 35.

5.2. Constructive heuristic HCB

Algorithm 2 HCB($G, \gamma, \alpha, minsize$)

$CL \leftarrow V$
$RCL \leftarrow \{v \in CL : |\{v' \in CL : deg_G(v') \geq deg_G(v)\}| \leq \max\{minsize, \alpha \cdot |CL|\}\}$
Randomly select $x \in RCL$
$S \leftarrow \{x\}$
while $CL \neq \emptyset$ **do**
    $CL \leftarrow \emptyset$
    for all $v \in V \setminus S$ **do**
        if $\frac{|E(S)|+deg_{G(S)}(v)}{(|S|+1)\cdot|S|/2} \geq \gamma$ **then**
            $CL \leftarrow CL \cup \{v\}$
        end if
    end for
    if $CL \neq \emptyset$ **then**
        for all $v \in CL$ **do**
            $dif_v \leftarrow deg_{G(CL)}(v) + |CL| \cdot (deg_{G(S)}(v) - \gamma \cdot (|S| + 1))$
        end for
        $RCL \leftarrow \{v \in CL : |\{v' \in CL : dif(v') \geq dif(v)\}| \leq \max\{minsize, \alpha \cdot |CL|\}\}$
        Randomly select $x \in RCL$
        $S \leftarrow S \cup \{x\}$
    end if
end while
return S

The constructive heuristic HCB introduced in this work is a variant of heuristic HC3 discussed in the previous section. While HC3 makes use of a control variable γ_{temp} to avoid that the number of candidates in RCL that satisfy the second criterion (potential differences) becomes very large, it will not be used by HCB since this is relevant only in the case of massive graphs (Abello et al., 2002). We also introduced a minimum size *minsize* for the restricted candidate list, to avoid that it becomes very small for small graphs.

The pseudo-code of Algorithm 2 presents the constructive heuristic HCB, which is basically a simplification of Algorithm 1 considering the two aspects above.

5.3. Decoder DECODER-HCB

Each solution of the maximum quasi-clique problem is associated with a set of $|V|$ random keys. Each random key is a real number in the range $[0, 1)$ and corresponds to a vertex of the graph. Each chromosome represented by a set of random keys is decoded by an algorithm (the decoder) that receives the keys and builds a feasible solution to MQCP. In other words, the decoder returns a γ-clique associated with the set of random keys.

Algorithm 3 DECODER-HCB$(G, \gamma, \alpha, minsize, S, r)$

```
CL ← V \ S
if S = ∅ then
    RCL ← {v ∈ CL : |{v′ ∈ CL : deg_G(v′) ≥ deg_G(v)}| ≤ max{minsize, α · |CL|}}
    x ← argmin{r_j : j ∈ RCL}
    S ← {x}
end if
while CL ≠ ∅ do
    CL ← ∅
    for all v ∈ V \ S do
        if (|E(S)| + deg_{G(S)}(v)) / ((|S|+1)·|S|/2) ≥ γ then
            CL ← CL ∪ {v}
        end if
    end for
    if CL ≠ ∅ then
        for all v ∈ CL do
            dif_v ← deg_{G(CL)}(v) + |CL| · (deg_{G(S)}(v) − γ · (|S| + 1))
        end for
        RCL ← {v ∈ CL : |{v′ ∈ CL : dif(v′) ≥ dif(v)}| ≤ max{minsize, α · |CL|}}
        x ← argmin{r_j : j ∈ RCL}
        S ← S ∪ {x}
    end if
end while
return S
```

Decoder DECODER-HCB to MQCP, whose pseudo-code is given by Algorithm 3, is based on and derived from the constructive heuristic HCB. It is used in two situations, as it will be described later in detail. First, to build a solution from scratch. Second, to complete (i.e., to reconstruct) a partially destroyed solution. In the second case, the decoder receives as an additional parameter a partial solution S formed by a non-empty list of vertices, while in the first case, $S = \emptyset$. DECODER-HCB decodes a population of random keys $r_j \in [0, 1)$, $j = 1, \dots, |V|$.

5.4. Decoder DECODER-IG*

The second decoder is an extension of DECODER-HCB proposed in the previous section. It is based on the constructive heuristic HCB and on the optimized iterated greedy heuristic IG* described in Section 3, but decodes a population formed by longer vectors of $2 \cdot |V|$ random keys $R_j \in [0, 1)$, $j = 1, \ldots, |V|, |V| + 1, \ldots, 2 \cdot |V|$ each. The first $|V|$ positions of each vector of random keys are used in the construction of the initial solution and in the reconstruction phase of the IG* strategy, while the the last $|V|$ positions of each vector of random keys are used in the destruction phase. The roles of parameters α, δ, and β are the same explained in Section 3 for strategy IG*.

Algorithm 4 starts by creating an initial solution S' in line 1, using the decoder DECODER-HCB and the first random keys $R_j, j = 1, \ldots, |V|$. This solution is copied to S in line 3. The loop in lines 2 to 10 repeats the partial destruction (vertex eliminations) followed by the reconstruction (vertex insertions) of the current solution, until no further improvements can be obtained.

The loop in lines 4 to 8 removes one by one the $\delta \cdot |S'|$ vertices that should be eliminated from the current solution. A restricted candidate list *RCL* of size $\max\{minsize, \beta \cdot |S'|\}$ is created in line 5, containing the vertices with the smallest degrees in $G(S')$. The vertex with the smallest random key $R_{|V|+j}$, $j \in RCL$, is selected from the restricted candidate list in line 6 and eliminated from the current solution in line 7.

The reconstruction phase is performed in line 9, where the current, partial solution S' is rebuilt by decoder DECODER-HCB, once again using the first random keys $R_j, j = 1, \ldots, |V|$. The decoder stops when the new solution obtained by destruction-reconstruction does not improve the incumbent S. The best solution S is returned in line 11.

6. Computational results

In this section, we address the effectiveness of the heuristics based on biased random-key genetic algorithms. We compare the results obtained with the two proposed BRKGA variants with those obtained by the original RIG* implementation of Oliveira et al. (2013) and by the BRKGA algorithm originally presented in (Pinto et al., 2015), which is a preliminary version of BRKGA-HCB. The first proposed variant is called BRKGA-HCB and makes use of the decoder DECODER-HCB described in Section 5.3, while the second one is denoted BRKGA-IG* and makes use of decoder DECODER-IG* proposed in Section 5.4. A restart strategy is incorporated into BRKGA-IG* and we show that it further improves the performance of the heuristic. We also report numerical experiments on sparse

Algorithm 4 DECODER-IG$^*(G, \gamma, \alpha, \delta, \beta, minsize, R)$

$S' \leftarrow$ DECODER-HCB$(G, \gamma, \alpha, minsize, \emptyset, R_j : j = 1, \ldots, |V|)$
repeat
  $S \leftarrow S'$
  for $k = 1$ to $\delta \cdot |S'|$ **do**
    $RCL \leftarrow \{v \in S' : |\{v' \in S' : deg_{G(S')}(v') \leq deg_{G(S')}(v)\}| \leq \max\{minsize, \beta \cdot |S'|\}\}$
    $x \leftarrow argmin\{R_{|V|+j} : j \in RCL\}$
    $S' \leftarrow S' \setminus \{x\}$
  end for
  $S' \leftarrow$ DECODER-HCB$(G, \gamma, \alpha, minsize, S', R_j : j = 1, \ldots, |V|)$
until $|S'| \leq |S|$ **or** graph $G(S')$ is not connected
return S

graph instances comparing BRKGA-IG* with the mixed integer programming approaches AlgF3 and AlgF4 of Veremyev et al. (2016).

Both algorithms BRKGA-HCB and BRKGA-IG* were implemented in C++ with the GNU GCC compiler C/C++ version 5.4.0. The experiments have been performed on a Lenovo i7-6500U computer with a 2.50 GHz CPU (maximum turbo frequency with 3.10 GHz) with 8 GB of RAM under the operating system Linux Ubuntu 16.04 LTS with parallel processing features disabled.

The numerical experiments on dense graphs involved 67 maximum clique instances of the Second DIMACS Implementation Challenge (DIMACS, 2016; Johnson & Trick, 1996) and 33 maximum clique instances of the Benchmarks with Hidden Optimum Solutions for Graph Problems (BHOSLIB, 2014; Pullan et al., 2011). The experiments on sparse graphs considered 12 instances obtained from the University of Florida Sparse Matrix Collection (Davis & Hu, 2011) that have also been used by Veremyev et al. (2016).

All the input data for the above test instances are available in Mendeley (see (Pinto et al., 2017)), together with the forthcoming detailed numerical results that will be reported along this section.

6.1. Tuning

The best parameters for algorithms BRKGA-HCB and BRKGA-IG* have been determined using the IRACE (López-Ibánez et al., 2011; Pérez Cáceres et al., 2014; Alfaro-Fernández et al., 2017) automatic tuning tool. In the first step of the tuning experiment, we determined the best values for parameters α and *minsize* used by the constructive heuristic HCB. In the second step, we looked for the best values for parameters δ and β used by algorithm BRKGA-IG*. Finally, in the third step,

Table 1: Test instances (20) used in the tuning experiment with IRACE for dense graphs.

Instance	$\lvert V \rvert$	$\lvert E \rvert$	threshold γ
brock200_4	200	13089	0.80
brock400_4	400	59765	0.80
brock800_4	800	207643	0.80
C2000.5	2000	999836	0.80
frb30-15-3	450	83216	0.95
frb35-17-3	595	148784	0.95
frb40-19-3	760	247325	0.95
frb45-21-3	945	387795	0.95
frb50-23-3	1150	579607	0.95
frb53-24-3	1272	714229	0.95
frb56-25-3	1400	869921	0.95
frb59-26-3	1534	1049729	0.95
gen200_p0.9_55	200	17910	0.99
gen400_p0.9_75	400	71820	0.99
p_hat300-3	300	33390	0.95
p_hat500-3	500	93800	0.95
p_hat700-3	700	183010	0.95
p_hat1500-1	1500	284923	0.95
p_hat1500-3	1500	847244	0.95
san400_0.9_1	400	71820	0.99

we sought the best values for parameters *p* (population size), *pe* (fraction of the population corresponding to the elite set), *pm* (fraction of the population corresponding to the mutant set), and *rhoe* (probability that the offspring inherits each of its keys from the best fit of the two parents) used by the biased random-key genetic algorithm, with the running times limited to one hour and considering the value ranges suggested by Gonçalves & Resende (2011).

The IRACE tuning experiment performed 1000 runs (Bouamama & Blum, 2017; Maschler et al., 2017) of each algorithm for 20 additional problem instances selected from different classes, as listed in Table 1: 12 instances from the Second DIMACS Implementation Challenge and eight BHOSLIB instances.

The value ranges considered by IRACE and the best parameter values identified by the tuning experiment are reported in Table 2.

6.2. *Experiments on dense graphs*

We considered 100 test problems for the comparative evaluation of the two biased random-key genetic algorithms BRKGA-HCB and BRKGA-IG* proposed in this work with the optimized restarted iterated greedy strategy RIG* (Oliveira et al., 2013) and with the preliminary BRKGA algorithm in (Pinto et al., 2015).

Table 2: Value ranges used by IRACE and best parameter values obtained after tuning.

Parameter	value ranges	BRKGA-HCB	BRKGA-IG*
α	0.01, 0.02, ..., 0.20	0.01	0.01
minsize	1, 2, 3, 4, 5, 6	3	3
β	0.01, 0.02, ..., 0.20	-	0.02
δ	0.01, 0.02, ..., 0.50	-	0.40
p	50, 51, ..., 100	64	91
pe	0.10, 0.11, ..., 0.25	0.22	0.13
pm	0.10, 0.11, ..., 0.30	0.15	0.22
rhoe	0.50, 0.51, ..., 0.80	0.63	0.78

None of these instances was used in the tuning experiments reported in Section 6.1. Each algorithm was run ten independent times for each instance using different seeds. Algorithm RIG* was made to stop after 100 iterations without improvement in the incumbent. The average time taken by algorithm RIG* over the ten runs for each problem was used as the stopping criterion for each of the BRKGA variants. Therefore, all algorithms are subject to exactly the same stopping criterion.

Tables 3, 4, 5, and 6 display the number of nodes $|V|$, the number of edges $|E|$, the density, and the threshold γ for each instance. In addition, for each instance and for each algorithm, these tables show the best and the average solution values over the ten runs. The last column in each table shows the running time in seconds observed for RIG*, which was used as the stopping criterion for the BRKGA variants. Cells highlighted in boldface indicate the algorithms that attained the best values for each heuristic. These tables show that the biased random-key genetic algorithm variants BRKGA-HCB and BRKGA-IG* found systematically better solutions than RIG*. In fact, while for only two (MANN_a27 and MANN_a45 – both of them being very dense graphs with density greater than 99%) out of the 100 instances RIG* found a solution that was not matched by at least one of the genetic algorithm variants, either BRKGA-HCB or BRKGA-IG* found solutions unmatched by RIG* for 64 test problems.

Table 3: Results for algorithms BRKGA, BRKGA-HCB, BRKGA-IG*, and RIG* - Part I.

					BRKGA		BRKGA-HCB		BRKGA-IG*		RIG*		running
Instance	$\|V\|$	$\|E\|$	dens. (%)	γ	best	average	best	average	best	average	best	average	time (s)
C125.9	125	6963	89.85	0.999	**34**	**34.00**	**34**	**34.00**	**34**	**34.00**	**34**	**34.00**	1.82
C250.9	250	27984	89.91	0.999	**44**	**44.00**	**44**	**44.00**	**44**	**44.00**	**44**	43.30	5.29
C500.9	500	112332	90.05	0.999	**57**	55.80	**57**	56.20	**57**	**56.70**	55	54.00	12.09
C1000.9	1000	450079	90.11	0.999	66	65.10	**67**	65.00	**67**	**65.80**	64	62.40	43.61
C2000.9	2000	1799532	90.02	0.999	73	71.60	72	71.30	**74**	**72.20**	71	68.70	95.67
C4000.5	4000	4000268	50.02	0.800	41	40.60	40	39.30	**44**	**42.60**	42	38.40	69.01
DSJC500.5	500	62624	50.20	0.800	33	32.20	**34**	32.40	**34**	**32.80**	31	30.20	5.93
DSJC1000.5	1000	249826	50.02	0.800	37	35.90	**38**	35.00	37	**36.20**	35	33.70	13.34
MANN_a9	45	918	92.73	0.999	**16**	**16.00**	**16**	**16.00**	**16**	**16.00**	**16**	**16.00**	0.24
MANN_a27	378	70551	99.01	0.999	133	132.20	133	132.40	133	132.40	**135**	**134.60**	37.21
MANN_a45	1035	533115	99.63	0.999	426	425.10	423	422.20	425	421.20	**439**	**437.60**	640.09
brock200_1	200	14834	74.54	0.800	**114**	113.30	**114**	113.70	**114**	**114.00**	**114**	113.00	8.17
brock200_2	200	9876	49.63	0.800	**24**	23.20	**24**	**24.00**	**24**	**24.00**	23	22.40	1.44
brock200_3	200	12048	60.54	0.800	**41**	40.30	**41**	**40.70**	**41**	**40.70**	40	39.20	2.70
brock400_1	400	59723	74.84	0.800	**189**	187.60	**189**	187.90	**189**	**189.00**	188	186.60	36.48
brock400_2	400	59786	74.92	0.800	185	184.40	**186**	184.80	**186**	**185.90**	185	183.20	32.45
brock400_3	400	59681	74.79	0.800	186	185.50	**187**	185.90	**187**	**186.10**	186	184.80	36.29
brock800_1	800	207505	64.93	0.800	92	90.70	92	89.50	**96**	**93.30**	87	85.10	31.84
brock800_2	800	208166	65.13	0.800	92	90.00	90	87.80	**93**	**91.70**	87	85.80	31.75
brock800_3	800	207333	64.87	0.800	90	89.40	90	88.30	**92**	**90.90**	87	84.50	26.73
c-fat200-1	200	1534	7.71	0.500	**30**	**30.00**	**30**	**30.00**	**30**	**30.00**	**30**	29.80	0.42
c-fat200-2	200	3235	16.26	0.500	**58**	**58.00**	**58**	**58.00**	**58**	**58.00**	**58**	**58.00**	0.92
c-fat200-5	200	8473	42.58	0.500	**148**	**148.00**	**148**	**148.00**	**148**	**148.00**	**148**	**148.00**	5.04
c-fat500-1	500	4459	3.57	0.500	**35**	**35.00**	**35**	**35.00**	**35**	**35.00**	**35**	34.20	0.66
c-fat500-2	500	9239	7.33	0.500	**66**	**66.00**	**66**	**66.00**	**66**	**66.00**	**66**	65.40	1.72
c-fat500-5	500	23191	18.59	0.500	**164**	**164.00**	**164**	**164.00**	**164**	**164.00**	**164**	163.50	7.86
c-fat500-10	500	46627	37.38	0.500	**324**	**324.00**	**324**	**324.00**	**324**	**324.00**	**324**	**324.00**	27.26
frb30-15-1	450	83198	82.35	0.950	57	56.30	**59**	57.20	**59**	**58.40**	56	54.70	11.39

Table 4: Results for algorithms BRKGA, BRKGA-HCB, BRKGA-IG*, and RIG* - Part II.

Instance	$\|V\|$	$\|E\|$	dens. (%)	γ	BRKGA best	BRKGA average	BRKGA-HCB best	BRKGA-HCB average	BRKGA-IG* best	BRKGA-IG* average	RIG* best	RIG* average	running time (s)
frb30-15-2	450	83151	82.31	0.950	57	56.20	57	56.00	**58**	**56.90**	55	53.60	10.82
frb30-15-4	450	83194	82.35	0.950	56	55.70	58	55.70	**60**	**57.20**	55	52.80	11.13
frb30-15-5	450	83231	82.39	0.950	58	56.70	58	56.90	**59**	**58.60**	56	54.20	12.32
frb35-17-1	595	148859	84.24	0.950	75	74.10	75	73.90	**78**	**76.60**	73	70.20	23.07
frb35-17-2	595	148868	84.24	0.950	72	70.90	**74**	70.50	**74**	**73.40**	70	68.40	20.95
frb35-17-4	595	148873	84.24	0.950	77	75.80	79	77.20	**80**	**78.60**	74	72.80	23.21
frb35-17-5	595	148572	84.07	0.950	77	75.10	78	75.50	**79**	**77.60**	73	71.30	24.84
frb40-19-1	760	247106	85.68	0.950	107	105.10	108	105.60	**111**	**109.70**	102	100.30	44.72
frb40-19-2	760	247157	85.69	0.950	98	96.40	98	95.50	**102**	**100.40**	96	92.00	41.34
frb40-19-4	760	246815	85.57	0.950	94	92.00	92	90.70	**95**	**94.00**	91	87.20	43.53
frb40-19-5	760	246801	85.57	0.950	98	96.00	96	94.10	**99**	**97.50**	92	89.70	43.92
frb45-21-1	945	386854	86.73	0.950	117	115.30	117	114.90	**121**	**120.00**	114	111.40	64.34
frb45-21-2	945	387416	86.86	0.950	114	112.60	114	111.30	**120**	**118.50**	112	108.30	56.12
frb45-21-4	945	387491	86.87	0.950	123	122.20	123	119.80	**128**	**126.00**	119	116.00	74.75
frb45-21-5	945	387461	86.87	0.950	116	113.70	114	112.30	**121**	**118.40**	111	109.20	70.98
frb50-23-1	1150	580603	87.88	0.950	152	148.80	148	146.20	**158**	**154.80**	146	143.40	123.42
frb50-23-2	1150	579824	87.76	0.950	152	147.70	151	145.80	**154**	**153.10**	147	144.00	107.99
frb50-23-4	1150	580417	87.85	0.950	149	145.60	145	142.70	**155**	**152.80**	143	141.40	103.64
frb50-23-5	1150	580640	87.89	0.950	152	149.20	149	146.10	**158**	**155.20**	149	144.80	117.98
frb53-24-1	1272	714129	88.34	0.950	190	186.10	186	183.10	**199**	**196.50**	191	184.70	159.12
frb53-24-2	1272	714067	88.34	0.950	162	160.30	160	157.90	**169**	**167.10**	159	157.10	136.73
frb53-24-4	1272	714048	88.33	0.950	172	167.90	169	165.60	**178**	**176.50**	170	163.90	143.50
frb53-24-5	1272	714130	88.34	0.950	158	157.00	155	153.10	**167**	**164.50**	158	153.50	120.71
frb56-25-1	1400	869624	88.80	0.950	212	208.90	210	207.90	**225**	**223.10**	216	211.00	217.61
frb56-25-2	1400	869899	88.83	0.950	200	198.50	201	196.20	**212**	**209.30**	200	196.50	224.00
frb56-25-4	1400	869262	88.76	0.950	187	183.60	182	178.40	**194**	**192.20**	183	177.40	182.62
frb56-25-5	1400	869699	88.81	0.950	189	186.40	187	184.00	**201**	**197.90**	187	184.30	192.85
frb59-26-1	1534	1049256	89.24	0.950	227	223.60	225	219.90	**239**	**237.40**	231	224.30	264.45

Table 5: Results for algorithms BRKGA, BRKGA-HCB, BRKGA-IG*, and RIG* - Part III.

					BRKGA		BRKGA-HCB		BRKGA-IG*		RIG*		running
Instance	$\lvert V\rvert$	$\lvert E\rvert$	dens. (%)	γ	best	average	best	average	best	average	best	average	time (s)
frb59-26-2	1534	1049648	89.27	0.950	226	221.70	220	216.90	**236**	**232.70**	226	220.60	234.40
frb59-26-4	1534	1048800	89.20	0.950	216	214.30	216	211.10	**227**	**225.60**	218	213.80	251.09
frb59-26-5	1534	1049829	89.29	0.950	210	207.90	205	203.40	**224**	**219.70**	210	204.70	198.77
frb100-40	4000	7425226	92.84	0.950	1819	1815.60	1819	1817.30	**1837**	1835.50	**1837**	**1836.00**	6530.86
gen200_p0.9_44	200	17910	90.00	0.999	40	40.00	40	40.00	**42**	**40.40**	40	39.80	3.91
gen400_p0.9_55	400	71820	90.00	0.999	52	51.40	**53**	52.20	**53**	**52.40**	51	50.70	10.47
gen400_p0.9_65	400	71820	90.00	0.999	52	50.80	55	53.20	**66**	**62.00**	51	49.20	9.24
hamming6-2	64	1824	90.48	0.950	**37**	**37.00**	**37**	**37.00**	**37**	**37.00**	**37**	**37.00**	0.64
hamming6-4	64	704	34.92	0.500	**32**	**32.00**	**32**	**32.00**	**32**	**32.00**	**32**	**32.00**	0.44
hamming8-2	256	31616	96.86	0.999	**129**	**129.00**	**129**	**129.00**	**129**	**129.00**	**129**	**129.00**	13.18
hamming8-4	256	20864	63.92	0.800	**71**	**71.00**	**71**	**71.00**	**71**	**71.00**	**71**	**71.00**	4.89
hamming10-2	1024	518656	99.02	0.999	**525**	**525.00**	**525**	**525.00**	**525**	**525.00**	**525**	**525.00**	505.75
hamming10-4	1024	434176	82.89	0.950	**82**	**81.60**	81	80.70	**82**	81.20	81	79.00	38.48
johnson8-2-4	28	210	55.56	0.800	**5**	**5.00**	**5**	**5.00**	**5**	**5.00**	**5**	**5.00**	0.06
johnson8-4-4	70	1855	76.81	0.800	**43**	42.80	**43**	**43.00**	**43**	**43.00**	**43**	**43.00**	1.03
johnson16-2-4	120	5460	76.47	0.800	**34**	**34.00**	**34**	**34.00**	**34**	**34.00**	**34**	**34.00**	1.12
johnson32-2-4	496	107880	87.88	0.950	**21**	**21.00**	**21**	**21.00**	**21**	**21.00**	**21**	**21.00**	4.59
keller4	171	9435	64.91	0.800	51	51.00	**54**	52.40	**54**	**54.00**	**54**	52.70	3.86
keller5	776	225990	75.15	0.800	**486**	**486.00**	**486**	**486.00**	**486**	**486.00**	**486**	**486.00**	110.42
keller6	3361	4619898	81.82	0.950	269	263.50	267	260.70	**279**	**275.80**	270	263.20	742.24
p_hat300-1	300	10933	24.38	0.500	**64**	63.10	**64**	63.70	**64**	**64.00**	63	62.30	8.05
p_hat300-2	300	21928	48.89	0.800	**114**	**114.00**	**114**	**114.00**	**114**	**114.00**	**114**	**114.00**	9.72
p_hat500-1	500	31569	25.31	0.500	**96**	95.80	**96**	**96.00**	**96**	**96.00**	95	94.60	20.39
p_hat500-2	500	62946	50.46	0.800	**211**	**211.00**	**211**	**211.00**	**211**	**211.00**	**211**	**211.00**	28.74
p_hat700-1	700	60999	24.93	0.500	**119**	117.90	118	117.50	**119**	**118.90**	118	116.40	36.10
p_hat700-2	700	121728	49.76	0.800	**288**	**288.00**	**288**	**288.00**	**288**	**288.00**	**288**	**288.00**	55.17
p_hat1000-1	1000	122253	24.48	0.500	144	143.20	144	142.20	**145**	**144.10**	142	141.10	60.64
p_hat1000-2	1000	244799	49.01	0.800	**385**	**385.00**	**385**	**385.00**	**385**	**385.00**	384	384.00	107.91

Table 6: Results for algorithms BRKGA, BRKGA-HCB, BRKGA-IG*, and RIG* - Part IV.

					BRKGA		BRKGA-HCB		BRKGA-IG*		RIG*		running
Instance	$\|V\|$	$\|E\|$	dens. (%)	γ	best	average	best	average	best	average	best	average	time (s)
p_hat1000-3	1000	371746	74.42	0.950	**210**	**210.00**	**210**	**210.00**	**210**	**210.00**	209	208.30	87.93
p_hat1500-2	1500	568960	50.61	0.800	**642**	**642.00**	**642**	**642.00**	**642**	**642.00**	**642**	**642.00**	274.25
san200_0.7_1	200	13930	70.00	0.950	**57**	**57.00**	**57**	**57.00**	**57**	**57.00**	**57**	55.80	3.91
san200_0.7_2	200	13930	70.00	0.950	**34**	**34.00**	**34**	33.60	**34**	**34.00**	**34**	**34.00**	2.43
san200_0.9_1	200	17910	90.00	0.990	74	70.40	**78**	**78.00**	**78**	**78.00**	**78**	77.00	6.76
san200_0.9_2	200	17910	90.00	0.999	55	50.40	58	58.00	**60**	**60.00**	55	46.20	3.97
san200_0.9_3	200	17910	90.00	0.999	37	36.90	42	39.40	**44**	**42.60**	37	36.30	2.80
san400_0.5_1	400	39900	50.00	0.500	**400**	**400.00**	**400**	**400.00**	**400**	**400.00**	**400**	**400.00**	29.40
san400_0.7_1	400	55860	70.00	0.950	**201**	**201.00**	**201**	**201.00**	**201**	**201.00**	**201**	**201.00**	23.28
san400_0.7_2	400	55860	70.00	0.950	**62**	**62.00**	**62**	**62.00**	**62**	**62.00**	**62**	**62.00**	8.03
san400_0.7_3	400	55860	70.00	0.950	**40**	37.20	39	36.80	**40**	**38.90**	38	36.40	6.90
san1000	1000	250500	50.15	0.800	**562**	**562.00**	**562**	**562.00**	**562**	**562.00**	**562**	**562.00**	126.67
sanr200_0.7	200	13868	69.69	0.800	**73**	72.50	72	72.00	**73**	**72.90**	72	72.00	4.74
sanr200_0.9	200	17863	89.76	0.950	91	91.00	**92**	91.50	**92**	**92.00**	91	90.90	7.80
sanr400_0.5	400	39984	50.11	0.800	**32**	31.40	**32**	31.80	**32**	**32.00**	31	30.00	4.64
sanr400_0.7	400	55869	70.01	0.950	30	29.30	**32**	31.20	**32**	**31.80**	30	28.70	5.01

Table 7 summarizes the following statistics resulting from the experiments reported in Tables 3, 4, 5, and 6:

- #*Best* is the number of instances for which a given heuristic found the best overall solution value. The higher its value, the better is the performance of the corresponding heuristic.

- #*BestAvg* is the number of instances for which a given heuristic found the best average solution value. The higher its value, the better is the performance of the corresponding heuristic.

- *SumBest* is the number of runs in which a given heuristic found the best overall solution value. The higher its value, the better is the performance of the corresponding heuristic.

- *AvgDevRuns* is the average relative deviation between the solution value found by a given heuristic over all runs of some instance and the best solution value obtained for this instance over all heuristics. The smaller its value, the better is the performance of the corresponding heuristic.

- *AvgDev* is the average relative deviation between the best solution value obtained by a given heuristic for some instance and the best solution value obtained for this instance over all heuristics. The smaller its value, the better is the performance of the corresponding heuristic.

- *AvgDevAvg* is the average relative deviation between the average solution value obtained by a given heuristic for some instance and the best average solution value obtained for this instance over all heuristics. The smaller its value, the better is the performance of the corresponding heuristic.

- *Score* is the sum over all instances of the number of approaches that provided a solution strictly better than that obtained by a given heuristic. The smaller the value of *Score*, the better is the performance of the corresponding heuristic.

- *ScoreAvg* is the sum over all instances of the number of approaches that found an average solution value strictly better than that obtained by a given heuristic for some instance. The smaller its value, the better is the performance of the corresponding heuristic.

The results in Table 7 show that variant BRKGA-IG* consistently obtains the best performance statistics over all criteria considered. BRKGA-IG* found 97 best solution values and 96 best average solution values out of the 100 instances.

Table 7: Comparative performance statistics for algorithms BRKGA, BRKGA-HCB, BRKGA-IG*, and RIG*.

	BRKGA	BRKGA-HCB	BRKGA-IG*	RIG*
#Best	44	52	**97**	36
#BestAvg	32	35	**96**	27
SumBest	356	404	**575**	283
AvgDevRuns (%)	3.33	3.29	**0.95**	4.99
AvgDev (%)	2.27	2.07	**0.07**	3.35
AvgDevAvg (%)	2.48	2.45	**0.06**	4.17
Score	84	87	**4**	158
ScoreM	102	119	**6**	203

BRKGA-IG* also obtained the solutions that led to the smallest values for all statistics reporting average relative deviations from the best solution values.

In the next experiment, we evaluate and compare the run time distributions (or time-to-target plots – or ttt-plots, for short) of algorithms BRKGA-HCB, BRKGA-IG*, and RIG*. Time-to-target plots display on the ordinate axis the probability that an algorithm will find a solution at least as good as a given target value within a given running time, shown on the abscissa axis. Run time distributions have also been advocated by Hoos & Stützle (1998) as a way to characterize the running times of stochastic local search algorithms for combinatorial optimization. In this experiment, the three algorithms were made to stop whenever a solution with cost smaller than or equal to a given target value was found. The heuristics were run 200 times each, with different initial seeds for the pseudo-random number generator. Next, the empirical probability distributions of the time taken by each heuristic to find the target value are plotted. To plot the empirical distribution for each heuristic, we followed the methodology proposed by Aiex et al. (2002, 2007). We associate a probability $p_i = (i - \frac{1}{2})/200$ with the i-th smallest running time t_i and plot the points (t_i, p_i), for $i = 1, \ldots, 200$. The more to the left is a plot, the better is the algorithm corresponding to it.

Time-to-target plots for instance san200_0.9_1 are shown in Figure 3, with the target set to 78, which corresponds to the best value in Table 6. Time-to-target plots for instance C500.9 are shown in Figure 4, with the target set to 56, which corresponds to the average solution value obtained by heuristic BRKGA-HCB in Table 3. Time-to-target plots for instance gen400_p0.9_65 are shown in Figure 5, with the target set to 51, which corresponds to the best value obtained by heuristic RIG* in Table 5. Also, time-to-target plots for instance frb30-15-4 are shown in Figure 6, with the target set to 56, which corresponds to the average solution value obtained by heuristic BRKGA-HCB in Table 4. These plots show that heuristic

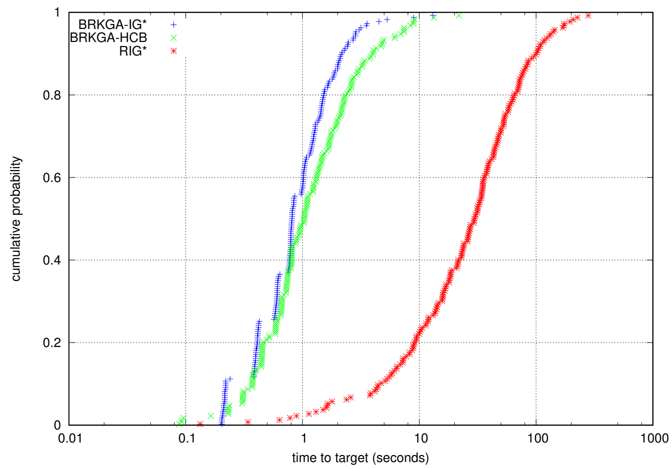

Figure 3: Runtime distributions for the instance san200_0.9_1 with the target value set at 78 (threshold $\gamma = 0.90$).

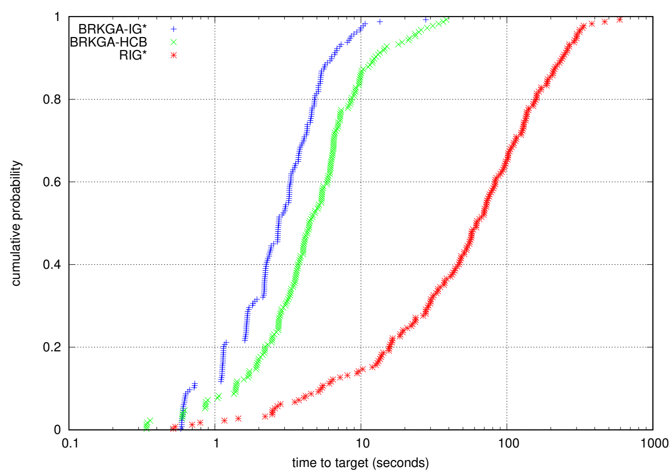

Figure 4: Runtime distributions for the instance C500.9 with the target value set at 56 (threshold $\gamma = 0.999$).

BRKGA-IG* is able to find with higher probability solutions as good as the target in smaller running times.

Figures 7 and 8 illustrate the evolution of the solution population along 100 generations of BRKGA-IG* for one execution of instances gen400_p0.9_65 and frb30-15-4, respectively. They show that the biased random-key genetic algorithm is able to continuously evolve the solution population and to improve the best so-

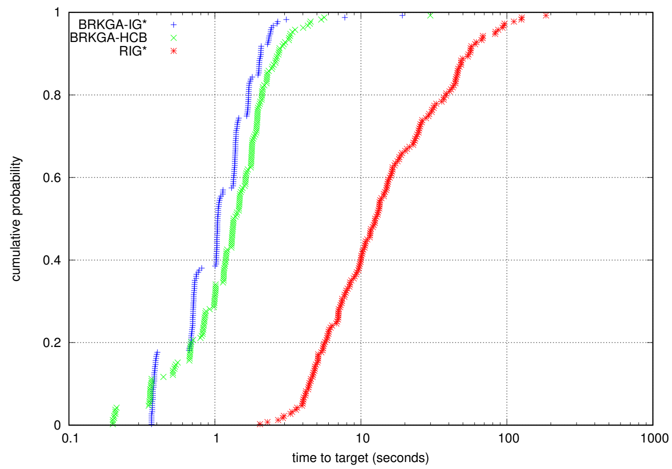

Figure 5: Runtime distributions for the instance gen400_p0.9_65 with the target value set at 51 (threshold $\gamma = 0.999$).

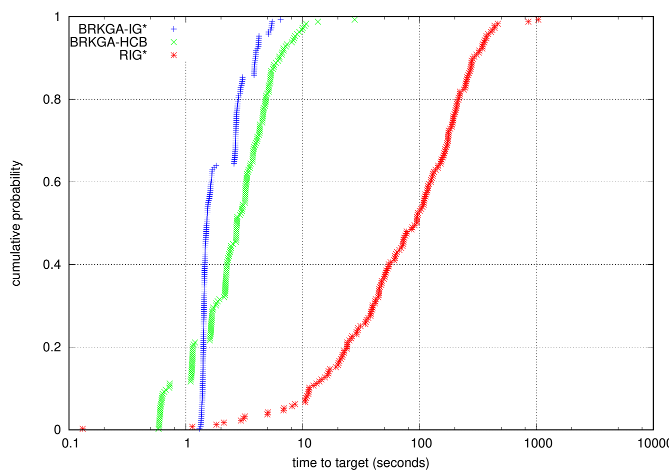

Figure 6: Runtime distributions for the instance frb30-15-4 with the target value set at 56 (threshold $\gamma = 0.95$).

lution value.

Figures 9 and 10 illustrate how the best solutions found by the three algorithms evolve along the first 3600 seconds of processing time, for the same two instances gen400_p0.9_65 and frb30-15-4, respectively. They show that BRKGA-IG* systematically finds better solutions faster than the other algorithms. The best solution obtained by BRKGA-IG* is better than that found by RIG* and BRKGA-HCB

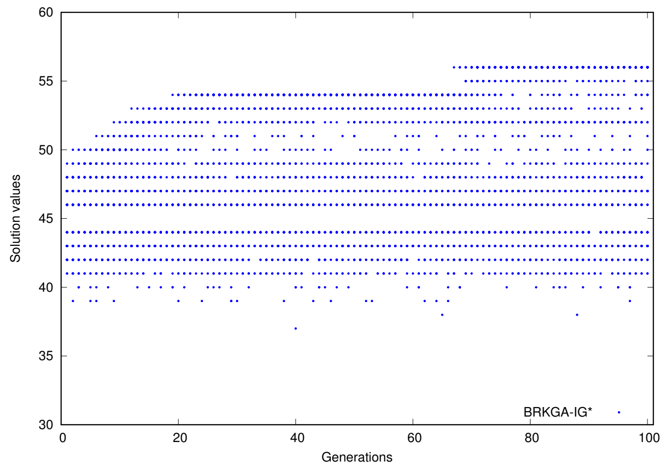

Figure 7: Population evolution for instance gen400_p0.9_65: best value found by BRKGA-IG* after 33.3 seconds (100 generations) is 56 (threshold $\gamma = 0.999$).

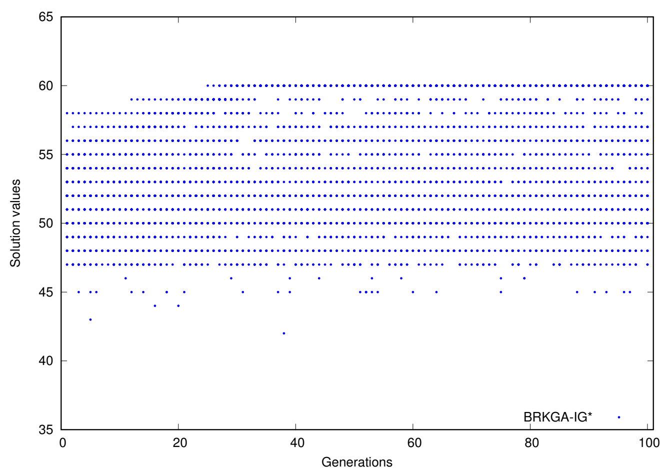

Figure 8: Population evolution for instance frb30-15-4: best value found by BRKGA-IG* after 147.99 seconds (100 generations) is 60 (threshold $\gamma = 0.95$).

most of the time along the runs displayed in these figures.

Restart strategies are able to reduce the running times to reach target solution values. We applied to heuristic BRKGA-IG* the same type of restart(κ) strategy discussed in Resende & Ribeiro (2011, 2016) (see also (Interian & Ribeiro, 2017)), in which the population is entirely renewed after κ generations have been performed without improvement in the best solution found. We evaluated the per-

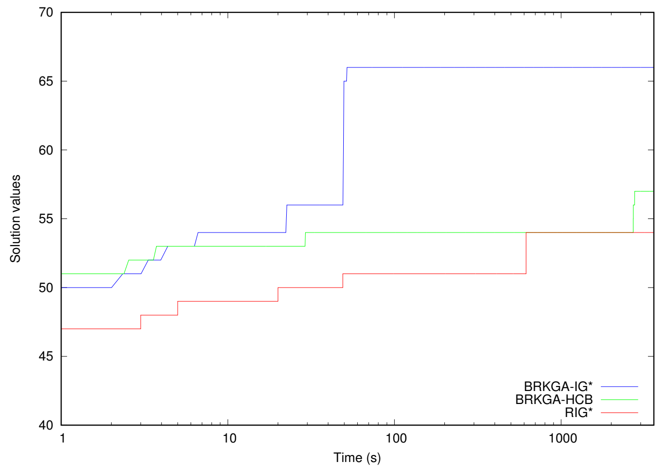

Figure 9: Evolution of the best solution value found by BRKGA-IG* and the other algorithms along the 3600 first seconds of running time for instance gen400_p0.9_65: best solution value obtained by BRKGA-IG* is 66 (threshold $\gamma = 0.999$).

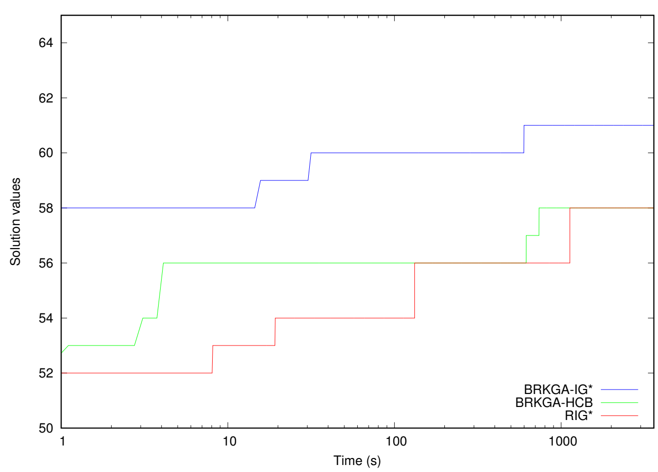

Figure 10: Evolution of the best value found by BRKGA-IG* and the other algorithms along the 3600 first seconds of running time for instance frb30-15-4: best solution value obtained by BRKGA-IG* is 61 (threshold $\gamma = 0.95$).

formance of restart(κ) strategies for $\kappa = 100, 200, 500$. Time-to-target plots for instance C500.9 are displayed in Figure 11 with 200 runs of each strategy restart(κ). Each run was limited to 1000 seconds and the target value was set to 57. Figure 11 shows that RIG* failed to reach the target within the time limit in 71 runs and BRKGA-HCB failed to reach the target within the time limit in 4 runs. For this

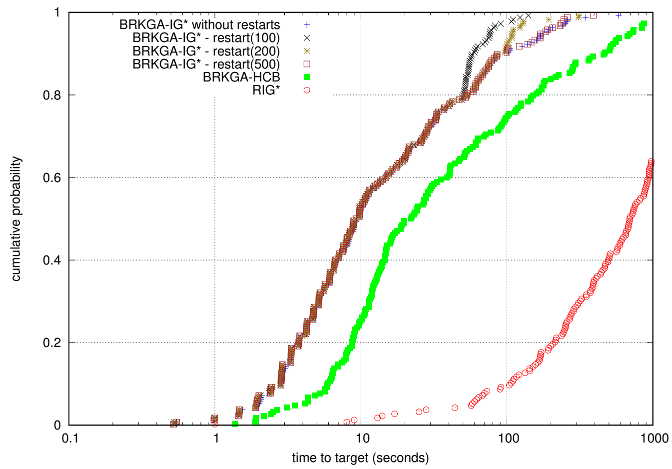

Figure 11: Runtime distributions for heuristic BRKGA-IG* with restart(κ) strategies on instance C500.9 with the running time limited to 1000 seconds and target value set at 57 (threshold $\gamma = 0.999$).

instance, strategy restart(100) presented the best results, i.e., the leftmost plots.

The next experiment addresses the behavior of heuristic BRKGA-IG* with harder target values. Figure 12 displays time-to-target plots for all variants of BRKGA-IG*, with and without restarts, on instance san200_0.9_1 with the target value set at 79. In this experiment, each algorithm variant was run 200 times, with the running time limited to 10000 seconds. Again, BRKGA-IG* with restart(100) presented the best behavior.

Resende & Ribeiro (2011, 2016) observed that the effect of restart strategies can be mainly noticed in the longest runs. As an example, Table 8 illustrates the results obtained by the restart strategies on instance san200_0.9_1, considering 200 runs of each algorithm variant with the target value set to 79. We consider the column corresponding to the fourth quartile in this table, whose entries correspond to those in the heavy tails of the runtime distributions. The restart strategies affect all quartiles of the distributions, which is a desirable result. Compared to the strategy without restarts, the restart(100) strategy was able to reduce not only the average running time in the fourth quartile, but also in the other quartiles. The best results for each quartile are highlighted in boldface. Strategy BRKGA-IG* with restart(100) clearly outperformed all other variants tested, with the smallest average running times. We notice that BRKGA-IG* without restarts failed to reach the target within the time limit in 23 runs.

Figure 13 displays time-to-target plots for BRKGA-IG* with restart (100) strategy on instance frb30-15-4 with the running time limited to 10000 seconds as the target value increases from 55 to 61. Figure 14 shows the average running time (in

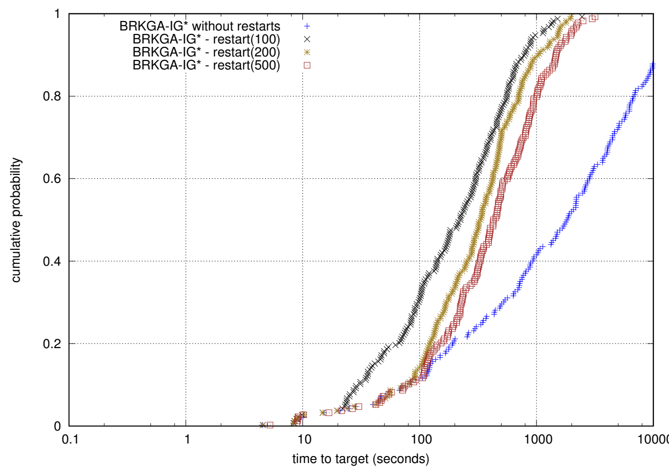

Figure 12: Runtime distributions for heuristic BRKGA-IG* with restart(κ) strategies on instance san200_0.9_1 with the running time limited to 10000 seconds and target value set at 79 (threshold $\gamma = 0.90$).

Table 8: Summary of computational results for each strategy on instance san200_0.9_1. Each run was made to stop when a solution as good as the target solution value 79 was found (threshold $\gamma = 0.90$). For each strategy, the table shows the distribution of the running times by quartile. For each quartile, the table gives the average running times in seconds over all runs in that quartile. The average running times over the 200 runs are also given for each strategy.

	Average running times in quartile (seconds)				
Strategy	1st	2nd	3rd	4th	average
BRKGA-IG* without restarts	118.48	906.97	3341.26	-	-
BRKGA-IG* with restart(100)	**39.51**	**133.93**	**330.57**	**799.44**	**325.86**
BRKGA-IG* with restart(200)	81.07	232.08	438.30	1038.03	447.37
BRKGA-IG* with restart(500)	95.23	305.23	621.06	1452.35	618.47

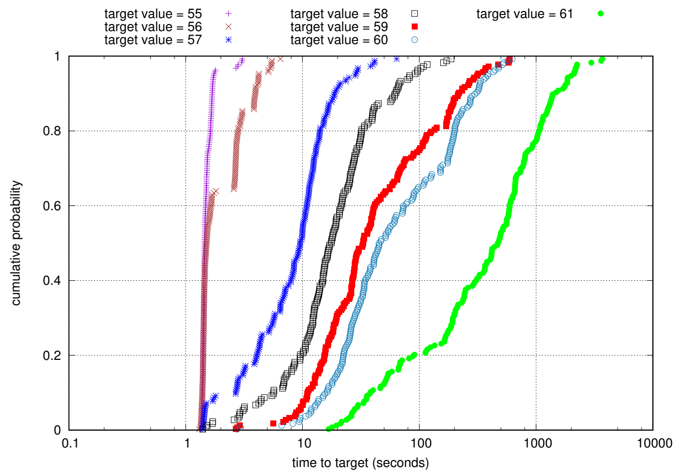

Figure 13: Runtime distributions for heuristic BRKGA-IG* with restart(100) strategy on instance frb30-15-4 with the running time limited to 10000 seconds as the target value set increases from 55 to 61 (threshold $\gamma = 0.95$).

seconds) over 200 runs for each target. As expected, the running time grows fast as the target increases towards the optimal value.

The experiments reported in this section showed that the biased random-key genetic algorithm BRKGA-IG* with the restart(100) strategy obtained the best results among all tested variants.

6.3. Experiments on sparse graphs

In Section 6.2, we concluded that variant BRKGA-IG* with strategy restart(100) presented the best numerical results on dense graphs. In this section, we compare this approach with algorithms AlgF3 and AlgF4, originally proposed in Veremyev et al. (2016). Since the original source codes of AlgF3 and AlgF4 were not available and the experiments reported in Veremyev et al. (2016) have been made on a different processor, we should consider an approximate scale ratio to compare the speed of the two processors. Regarding the single-rating performance, since the algorithms were tested on single-thread environments, the corresponding ratio for the two CPU models is $1067/1646 \approx 0.65$, according to the PassMark benchmark (PassMark, 2018).

The performance of algorithms to the maximum quasi-clique problem is very sensitive to the density of the input graph. Since heuristic BRKGA-IG* did not performed well for sparse graphs with the parameter $\alpha = 0.01$ set as determined in the experiment reported in Section 6.1, we performed a new tuning experiment for this parameter for the case of sparse graphs. Table 9 displays the characteristics of the

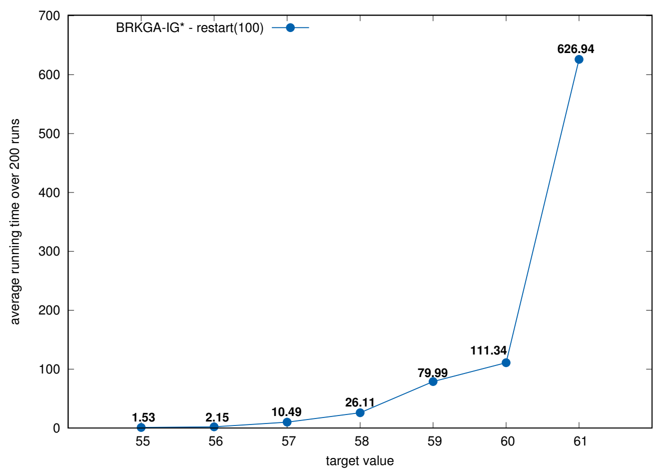

Figure 14: Average running times (seconds) for heuristic BRKGA-IG* with restart(100) strategy on instance frb30-15-4 with the running time limited to 10000 seconds as the target value set ranges from 55 to 61 (threshold $\gamma = 0.95$).

nine instances of the University of Florida Sparse Matrix Collection (Davis & Hu, 2011) used for tuning parameter α in the range $0.01, 0.02, \ldots, 0.20$. The heuristic was run 1000 times, with all other parameters fixed as before. The experiment determined $\alpha = 0.09$ as the most appropriate value for this parameter.

Algorithms AlgF3, AlgF4, and BRKGA-IG* with restart(100) were compared on six sparse instances with two values of the threshold γ for each of them. Each algorithm was made to stop when a solution with cardinality given by the bound $\omega_\gamma(G)$ was reached (Veremyev et al., 2016). The running times for each algorithm are reported in Table 10. The running times for algorithms AlgF3 and AlgF4 are those reported by Veremyev et al. (2016), multiplied by the scale factor 0.65. The

Table 9: Test instances used in the tuning experiment with IRACE for sparse graphs.

Instance	$\lvert V\rvert$	$\lvert E\rvert$	threshold γ
CA-GrQc	5242	14496	0.4
CA-GrQc	5242	14496	0.6
CA-GrQc	5242	14496	0.8
Harvard500	500	2636	0.4
Harvard500	500	2636	0.6
Harvard500	500	2636	0.8
USAir97	332	2126	0.4
USAir97	332	2126	0.6
USAir97	332	2126	0.8

Table 10: Comparative running times in seconds for algorithms AlgF3 (original times multiplied by 0.65) and AlgF4 (original times multiplied by 0.65), and BRKGA-IG* with restart(100).

						running times (seconds)		
Instance	$\|V\|$	$\|E\|$	dens. (%)	γ	$\omega_\gamma(G)$	AlgF3	AlgF4	BRKGA-IG* restart(100)
Harvard500	500	2043	1.64	0.9	23	9.11	10.08	0.07
Harvard500	500	2043	1.64	0.5	37	36.62	11.57	1.21
CA-GrQc	5242	14496	0.10	0.9	49	200.85	207.44	2.34
CA-GrQc	5242	14496	0.10	0.5	81	328.24	406.12	–
USAir97	332	2126	3.87	0.9	35	13.26	4.42	0.22
USAir97	332	2126	3.87	0.5	67	10.40	1.88	0.53
Email	1133	5451	0.85	0.9	13	–	345.41	0.17
Email	1133	5451	0.85	0.5	25	809.25	–	1.48
SmallW	396	994	1.27	0.9	11	4.74	1.88	0.06
SmallW	396	994	1.27	0.5	28	3.71	1.76	0.28
Erdos971	429	1312	1.43	0.9	8	13.00	2.47	0.05
Erdos971	429	1312	1.43	0.5	23	3.97	478.59	0.23

last column gives the average running time over ten runs of BRKGA-IG* with restart(100) for $\alpha = 0.09$, limited to one hour for each run. The running times of algorithms AlgF3 and AlgF4 were also limited to one hour for each instance. Blank cells correspond to cases where none of the ten runs reached a solution of cardinality $\omega_\gamma(G)$. For all other cells, all ten runs found a γ-clique with $\omega_\gamma(G)$ vertices. These results show that the biased random-key genetic algorithm BRKGA-IG* with the restart(100) strategy proposed in this work was faster than AlgF3 and AlgF4 for all but one of the test instances (CA-GrQc with the threshold $\gamma = 0.5$), for which it was not able to find a solution as good as the target in less than one hour of computation in any of the ten runs. This was most likely due to the very small density of this instance, for which AlgF3 and AlgF4 also took very long.

7. Concluding remarks

Given a graph $G = (V, E)$ and a threshold $\gamma \in (0, 1]$, the maximum cardinality quasi-clique problem consists in finding a maximum subset C^* of the vertices in V such that the density of the graph induced in G by C^* is greater than or equal to the threshold γ.

In this work, we proposed a biased random-key genetic algorithm for finding approximate solutions to the maximum quasi-clique problem, using two different decoders. The decoder based on an optimized iterated greedy constructive heuristic led to the best numerical results. We also showed that the use of a restart strategy significantly contributed to improve the robustness and the efficiency of the algorithm. The resulting BRKGA-IG* heuristic with restart(100) strategy outperformed different variants of the algorithm, as well as the restarted optimized iterated greedy (RIG*) construction/destruction heuristic that originally reported the best results in the literature for dense graphs.

In addition, the newly proposed BRKGA-IG* with restart(100) approach was also compared with the exact algorithms AlgF3 and AlgF4 of Veremyev et al. (2016) used as a heuristics with time limits on their running times. Also in this case, BRKGA-IG* with restart(100) applied to sparse graphs outperformed both mixed integer programming approaches, finding target solution values in much smaller running times.

All the input data for the test instances used in the experiments reported in this work are available in Mendeley (see (Pinto et al., 2017)), together with the resulting detailed numerical results.

Acknowledgement

Work of Celso C. Ribeiro was partially supported by CNPq research grant 303958/2015-4 and by FAPERJ research grant E-26/201.198/2014. Work of Alexandre Plastino was partially supported by CNPq research grant 308369/2015-7.

References

Abello, J., Pardalos, P. M., & Resende, M. G. C. (1999). On maximum clique problems in very large graphs. In J. M. Abello, & J. S. Vitter (Eds.), *External Memory Algorithms* (pp. 119–130). American Mathematical Society.

Abello, J., Resende, M., & Sudarsky, S. (2002). Massive quasi-clique detection. In S. Rajsbaum (Ed.), *LATIN 2002: Proceedings of the 5th Latin American Symposium on Theoretical Informatics* (pp. 598–612). Berlin: Springer volume 2286 of *Lecture Notes in Computer Science*.

Aiex, R., Resende, M., & Ribeiro, C. C. (2002). Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, *8*, 343–373.

Aiex, R., Resende, M., & Ribeiro, C. C. (2007). TTTPLOTS: A Perl program to create time-to-target plots. *Optimization Letters*, *1*, 355–366.

Alfaro-Fernández, P., Ruiz, R., Pagnozzi, F., & Stützle, T. (2017). Exploring automatic algorithm design for the hybrid flowshop problem. In *12th Metaheuristics International Conference* (pp. 201–203). Barcelona.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, *6*, 154–160.

BHOSLIB (2014). Benchmarks with Hidden Optimum Solutions for Graph Problems. http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm. Last accessed on November 14, 2017.

Bouamama, S., & Blum, C. (2017). A population-based iterated greedy algorithm for the knapsack problem with setup. In *12th Metaheuristics International Conference* (pp. 558–565). Barcelona.

Brandão, J. S., Noronha, T. F., Resende, M. G. C., & Ribeiro, C. C. (2015). A biased random-key genetic algorithm for single-round divisible load scheduling. *International Transactions Operational Research*, *22*, 823–839.

Brandão, J. S., Noronha, T. F., Resende, M. G. C., & Ribeiro, C. C. (2017). A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems. *International Transactions Operational Research*, *27*, 1061–1077.

Brunato, M., Hoos, H., & Battiti, R. (2008). On effectively finding maximal quasi-cliques in graphs. In V. Maniezzo, R. Battiti, & J.-P. Watson (Eds.), *Learning and Intelligent Optimization: Second International Conference, LION 2007* (pp. 41–55). Berlin: Springer volume 5313 of *Lecture Notes in Computer Science*.

Chaves, A. A., Lorena, L., Senne, E., & Resende, M. G. C. (2016). Hybrid method with cs and brkga applied to the minimization of tool switches problem. *Computers & Operations Research*, *67*, 174 – 183.

Davis, T. A., & Hu, Y. (2011). The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, *38*, 1:1–1:25.

DIMACS (2016). DIMACS Implementation Challenges. http://dimacs.rutgers.edu/Challenges/. Last access on November 14, 2017.

FICO (2017). FICO Xpress Optimization Suite 7.6. http://www.fico.com/en/products/fico-xpress-optimization-suite. Last access on November 14, 2017.

Gonçalves, J. F., & Resende, M. G. C. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, *17*, 487–525.

Gonçalves, J. F., Resende, M. G. C., & Toso, R. F. (2014). An experimental comparison of biased and unbiased random-key genetic algorithms. *Pesquisa Operacional*, *34*, 143 – 164.

Hoos, H., & Stützle, T. (1998). Evaluation of Las Vegas algorithms - Pitfalls and remedies. In G. Cooper, & S. Moral (Eds.), *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence* (pp. 238–245). Madison.

Interian, R., & Ribeiro, C. (2017). A GRASP heuristic using path-relinking and restarts for the Steiner traveling salesman problem. *International Transactions in Operational Research*, *24*, 1307–1323.

Johnson, D. J., & Trick, M. A. (Eds.) (1996). *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge* volume 26 of *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*. Boston: American Mathematical Society.

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller, & J. W. Thatcher (Eds.), *Complexity of Computer Computations* (pp. 85–103). Boston: Springer.

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2011). The IRACE package: Iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium.

Maschler, J., Hackl, T., Riedler, M., & Raidl, G. R. (2017). Enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem. In *12th Metaheuristics International Conference* (pp. 118–127). Barcelona.

Noronha, T. F., Resende, M. G. C., & Ribeiro, C. C. (2011). A biased random-key genetic algorithm for routing and wavelength assignment. *Journal of Global Optimization*, *50*, 503–518.

Oliveira, A. B., Plastino, A., & Ribeiro, C. C. (2013). Construction heuristics for the maximum cardinality quasi-clique problem. In *Abstracts of the 10th Metaheuristics International Conference (MIC 2013)* (p. 84). Singapore.

PassMark (2018). CPU Benchmarks. https://www.cpubenchmark.net/compare.php. Last access on March 14, 2018.

Pattillo, J., Veremyev, A., Butenko, S., & Boginski, V. (2013). On the maximum quasi-clique problem. *Discrete Applied Mathematics*, *161*, 244–257.

Pérez Cáceres, L., López-Ibáñez, M., & Stützle, T. (2014). An analysis of parameters of IRACE. In *Proceedings of the 14th European Conference on Evolutionary Computation in Combinatorial Optimization* (pp. 37–48). Berlin: Springer volume 8600 of *Lecture Notes in Computer Science*.

Pinto, B. Q., Plastino, A., Ribeiro, C. C., & Rosseti, I. (2015). A biased random-key genetic algorithm to the maximum cardinality quasi-clique problem. In *Abstracts of the 11th Metaheuristics International Conference*. Agadir.

Pinto, B. Q., Ribeiro, C. C., Rosseti, I., & Plastino, A. (2017). Input data and detailed numerical results for 'A biased random-key genetic algorithm for the maximum quasi-clique problem'. https://data.mendeley.com/datasets/khdncrwb5s/1. doi:`10.17632/khdncrwb5s.1`. Last accessed on November 14, 2017.

Pullan, W., Mascia, F., & Brunato, M. (2011). Cooperating local search for the maximum clique problem. *Journal of Heuristics*, *17*, 181–199.

Resende, M. G. C., & Ribeiro, C. C. (2011). Restart strategies for GRASP with path-relinking heuristics. *Optimization Letters*, *5*, 467–478.

Resende, M. G. C., & Ribeiro, C. C. (2016). *Optimization by GRASP*. Boston: Springer.

Ribeiro, A. G., Oliveira, B. B., Carravilla, M. A., & Oliveira, J. F. (2017). A biased random-key genetic algorithm for the car rental vehicle-reservation assignment problem. In *12th Metaheuristics International Conference* (pp. 301–310). Barcelona.

Ribeiro, C. C., & Riveaux, J. A. (2017). An exact algorithm for the maximum quasi-clique problem. Submitted for publication.

Ruiz, E., Albareda-Sambola, M., Fernández, E., & Resende, M. G. (2015). A biased random-key genetic algorithm for the capacitated minimum spanning tree problem. *Computers & Operations Research*, *57*, 95 – 108.

Ruiz, R., & Stützle, T. (2006). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *177*, 2033–2049.

Spears, W., & de Jong, K. (1991). On the virtues of parameterized uniform crossover. In R. Belew, & L. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 230–236). San Mateo: Morgan Kaufman.

Toso, R. F., & Resende, M. G. C. (2015). A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, *30*, 81–93.

Veremyev, A., Prokopyev, O. A., Butenko, S., & Pasiliao, E. L. (2016). Exact MIP-based approaches for finding maximum quasi-cliques and dense subgraphs. *Computational Optimization and Applications*, *64*, 177–214.