RAIRO-Oper. Res. 55 (2021) S741–S763
https://doi.org/10.1051/ro/2020003

RAIRO Operations Research
www.rairo-ro.org

A BRKGA-BASED MATHEURISTIC FOR THE MAXIMUM QUASI-CLIQUE PROBLEM WITH AN EXACT LOCAL SEARCH STRATEGY

Bruno Q. Pinto[1], Celso C. Ribeiro[2,*], José A. Riveaux[2] and Isabel Rosseti[2]

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 has a number of applications in data mining, *e.g.*, in social networks or phone call graphs. We propose a matheuristic for solving the maximum cardinality quasi-clique problem, based on the hybridization of a biased random-key genetic algorithm (BRKGA) with an exact local search strategy. The newly proposed approach is compared with a pure biased random-key genetic algorithm, which was the best heuristic in the literature at the time of writing. Computational results show that the hybrid BRKGA outperforms the pure BRKGA.

Mathematics Subject Classification. 05C69, 05C85, 68T20, 90C27, 90C59.

Received June 10, 2019. Accepted January 4, 2020.

1. Introduction

Let $G = (V, E)$ be a graph defined by a vertex set V and an edge set $E \subseteq V \times V$. G is a complete graph if there is an edge in E connecting every two different vertices in 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 formed by all edges of E with both ends in V'. For any $V' \subseteq V$, the subset $E(V') \subseteq E$ is formed by all edges of E with both ends in V'. In other words, $E(V')$ is the edge set of the graph induced in G by V'.

The density of graph G is given by $dens(G) = |E|/(|V| \times (|V| - 1)/2)$. For any $v \in V$, the degree $deg_G(v)$ denotes the number of vertices in G that are adjacent to v. In addition, for any $v \in V$ and any $V' \subseteq V$, we denote by $deg_G(v, V')$ the number of vertices of V' that are adjacent to v in G.

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* (MCP) consists in finding a maximum cardinality clique of G. It was proved to be NP-hard by [15].

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

Keywords. Maximum quasi-clique problem, maximum clique problem, biased random-key genetic algorithm, metaheuristics, matheuristics.

[1] Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro, Uberlândia, MG 38411-104, Brazil.
[2] Institute of Computing, Universidade Federal Fluminense, Niterói, RJ 24210-346, Brazil.
*Corresponding author: celso@ic.uff.br

Article published by EDP Sciences

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 was proved to be NP-hard in [21]. 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 [8] and the analysis of massive telecommunication data sets obtained from social networks or phone call graphs [1], as well as various other data mining and graph mining applications. The reader is also referred to Pastukhov *et al.* [20] for other examples of real-life applications.

A few heuristics for MQCP exist in the literature, based on well known approaches such as greedy randomized algorithms and their iterated extensions [18], stochastic local search [8], and GRASP [1]. Pinto *et al.* [23] proposed a biased random-key genetic algorithm for finding approximate solutions to the maximum cardinality quasi-clique problem, using two different decoders. The resulting BRKGA-IG* heuristic strategy achieved the best performance and outperformed the restarted optimized iterated greedy (RIG*) construction/destruction heuristic of Oliveira *et al.* [18]. BRKGA-IG* was also compared with the exact algorithms AlgF3 and AlgF4 of Veremyev *et al.* [31] used as heuristics with time limits on their running times. BRKGA-IG* applied to sparse graphs also outperformed the mixed integer programming approaches, finding target solution values in much smaller running times.

Ribeiro and Riveaux [25] proposed the exact enumeration algorithm *QClique* to solve the maximum quasi-clique problem, based on a quasi-hereditary property. They also 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 formulations in [21, 31] solved by CPLEX and with the branch-and-bound algorithm proposed by Pajouh *et al.* [19], in terms of both solution quality and running time. In general, algorithm *QClique* showed the best performance on random instances with respect to the best existing MIP models, obtaining the best solutions and significantly smaller running times for several instances. Again, it showed the best performance overall for several literature instances such as hamming6-4, johnson8-2-4, brock200-2, brock200-3, keller4, and p-hat300-1. The numerical results also showed that performed significantly better than AlgF3 for ten out of the 16 large sparse instances, in particular for larger values of the threshold γ. Although for the other six instances AlgF3 was faster, for some instances AlgF3 was not even capable to solve the linear relaxation at the root node of the search tree within the time limit of one hour.

We show that a variant of the exact enumeration algorithm *QClique* proposed by Ribeiro and Riveaux [25] can be hybridized with the biased random-key genetic algorithm BRKGA-IG* developed by Pinto *et al.* [23] as a local search strategy to improve the quality of the solutions created by the decoder. This paper is organized as follows. Section 2 presents the problem formulation. Section 3 introduces biased random-key genetic algorithms and describes their customization to the maximum quasi-clique problem. Section 4 describes in detail the decoder DECODER-IG* previously used in the implementation of the biased random-key genetic algorithm for the maximum quasi-clique problem. The new decoder DECODER-LSQClique using an exact local search strategy is presented in Section 5, giving rise to a matheuristic for solving the maximum cardinality quasi-clique problem based on a biased random-key genetic algorithm. The exact local search algorithm *LSQClique* used by the new decoder is described in detail in Section 6. Numerical results are reported in Section 7. Concluding remarks are drawn in the last section.

2. Problem formulation and related work

The maximum quasi-clique problem can be formulated by associating a binary variable x_i to each vertex of the graph [21]:

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

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

$$\max \sum_{i \in V} x_i \tag{2.1}$$

subject to:

$$\sum_{[i,j] \in E : i<j} y_{ij} \geq \gamma \cdot \sum_{i,j \in V : i<j} y_{ij}, \tag{2.2}$$

$$y_{ij} \leq x_i, \quad \forall i, j \in V, \quad i < j, \tag{2.3}$$

$$y_{ij} \leq x_j, \quad \forall i, j \in V, \quad i < j, \tag{2.4}$$

$$y_{ij} \geq x_i + x_j - 1, \quad i, j = 1, \ldots, n, \quad i < j, \tag{2.5}$$

$$x_i \in \{0, 1\}, \quad \forall i \in V, \tag{2.6}$$

$$y_{ij} \geq 0, \quad \forall i, j \in V, \quad i < j. \tag{2.7}$$

The objective function (2.1) 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 (2.2) ensures that the density of the solution is greater than or equal to γ. Constraints (2.3) and (2.4) 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 (2.5) ensure that any existing edge $[i, j] \in E$ will contribute to the solution if both of its ends are chosen. Constraints (2.6) and (2.7) impose the binary and non-negativity requirements on the problem variables, respectively.

Veremyev *et al.* [31] 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 [21], with all mixed integer programs solved using FICO Xpress-Optimizer [10] with the time limit of 3600 s.

Ribeiro and Riveaux [25] developed the exact algorithm *QClique* based on a quasi-hereditary proposition and proposed a new upper bound that is used for pruning the search tree. Numerical results showed that their approach is competitive and outperforms the best integer programming approaches in the literature. The new upper bound is consistently tighter than previously existing bounds. The cuts lead to considerable reductions in the number of visited nodes of the search tree. Consider the case of instance Erdos992, when 5 293 928 cuts are generated. The number of visited nodes of the search tree is reduced from 31 238 995 to 5 837 764, *i.e.*, to only 18.68% of the original number. The running time is reduced from 1511.70 to 426.79 s, *i.e.*, to 28.17% of the original time.

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

In a biased random-key genetic algorithm (BRKGA), each solution is represented by a vector of randomly generated real numbers called keys. A deterministic algorithm, called a decoder, takes as input a solution represented by a vector of random keys and associates with it a feasible solution of the combinatorial optimization problem at hand, for which an objective value or fitness can be computed. Selection in a BRKGA is said to be biased because one of the two parents selected for mating in each crossover operation is always an elite solution and has a higher probability of passing its genes to the new generation, while the other is a non-elite solution. The reader is referred to Gonçalves and Resende [11] and Resende and Ribeiro [24] for complete reviews about biased random-key genetic algorithms and their applications.

Two variants of a BRKGA for MQCP have been developed in [23], each of them using a different decoder. The algorithms evolve a population that is formed by vectors of real-valued components in the range $[0, 1)$, each component being associated with one of the vertices of the graph G. Each vector is decoded by a deterministic

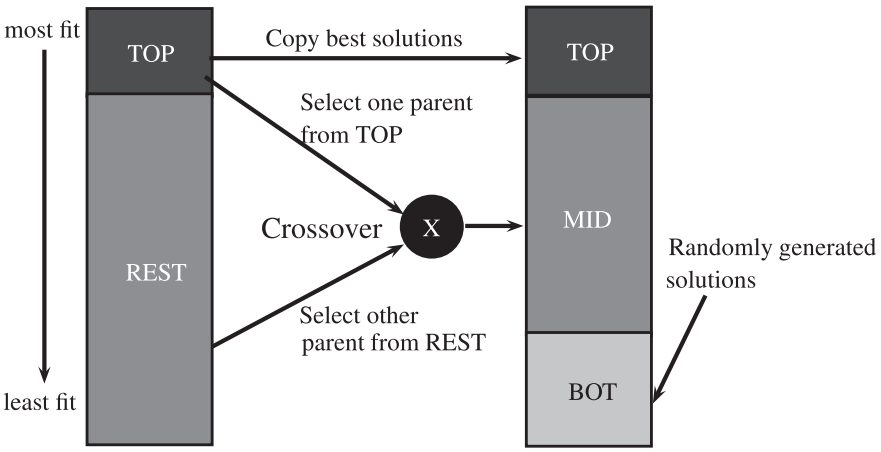


FIGURE 1. Population evolution between consecutive generations of a BRKGA [23].

algorithm that builds a feasible solution for MQCP, *i.e.*, a γ-clique. The two decoders DECODER-HCB and DECODER-IG* originally presented in [23] are briefly summarized in Section 4.

The parametric uniform crossover of Spears and de Jong [29] is used to combine two mates: the offspring inherits with higher probability each of its keys from the best fit of the two parents. Instead of the standard mutation operator, the concept of mutants is used: randomly generated mutant solutions are introduced in the population in each generation. 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 (see also [6, 7, 17]).

The initial population is randomly generated. The population is partitioned into two subsets at each generation: TOP and $REST$. Subset TOP contains the elite solutions, being formed by the best solutions in the population of the current generation. Subset $REST$ is decomposed in two subsets: MID and BOT, with subset BOT being formed by the worst elements in the population of the current generation. The size of the population is $|TOP| + |REST|$.

The population evolves from one generation to the next as illustrated in Figure 1. First, the solutions in TOP are simply copied to the population of the next generation. The elements in BOT are replaced by new randomly created mutants that are placed in the new set BOT. The remaining $|MID| = |REST| - |BOT|$ solutions of the new population are obtained by crossover, with one parent randomly chosen from TOP and the other from $REST$. This is the main difference between a BRKGA and the random-key genetic algorithm of Bean [4], where both parents are randomly selected from the entire population: since a solution can be chosen for mating more than once in any given generation, elite solutions have a higher probability of passing their random keys to the next generation.

The variants of the BRKGA for the maximum quasi-clique problem were implemented with the C++ library brkgaAPI developed by Toso and Resende [30], which is a framework illustrated in Figure 2 for the development of biased random-key genetic algorithms. Its instantiation to some specific optimization problem requires only the development of a class implementing the decoder for the problem at hand, since this is the only problem-dependent part of the tool.

The brkgaAPI framework requires the following parameters [12]: (a) the population size $p = |TOP|+|REST|$; (b) the fraction pe of the population corresponding to the elite subset TOP; (c) the fraction pm of the population corresponding to the mutant subset BOT; (d) the probability $rhoe$ that the offspring inherits the keys from the best fit parent; and (e) the number k of generations without improvement in the best solution until a restart is performed.

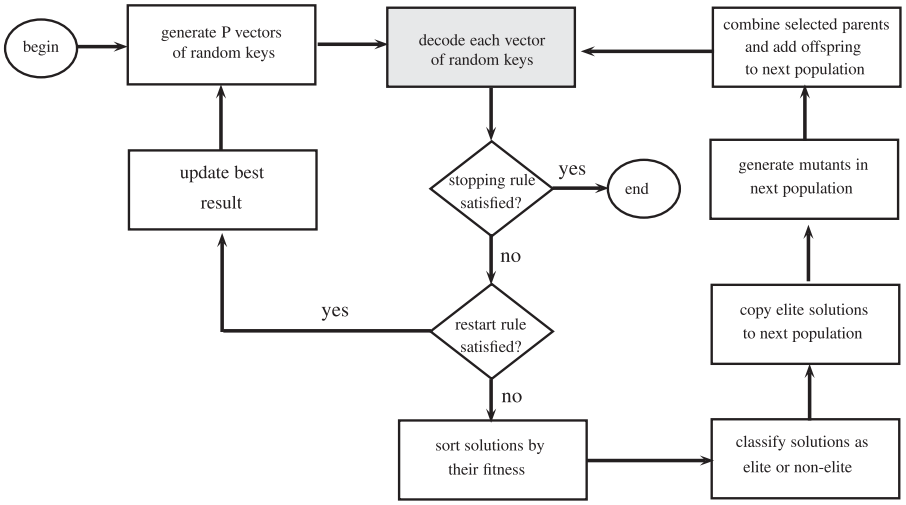


FIGURE 2. BRKGA framework [23].

In the remainder of this work, we consider and compare two variants of a biased random-key genetic algorithm for solving MQCP, based on two different decoders. Decoder DECODER-IG* was originally proposed by Pinto *et al.* [23] and will be summarized in the next section. The new decoder, named DECODER-LSQClique and proposed in this work, is based on the exact enumeration algorithm of Ribeiro and Riveaux [25] and will be presented in Section 5.

4. DECODER DECODER-IG*

Decoder DECODER-HCB was firstly presented in [23] and finds its roots in the construction phase of the GRASP approach of Abello *et al.* [1]. It takes as inputs the graph $G = [V, E]$, the threshold γ, the parameter α used to create the restricted candidate lists, the parameter *minsize* that defines the minimum size of the restricted candidate lists, and the subset S of vertices in the initial solution. The decoder also receives as parameters the random keys $R_j \in [0, 1), j = 1, \ldots, |V|$, each of them corresponding to a vertex of the graph. Each vector of random keys is decoded by an algorithm that builds a feasible solution to MQCP, *i.e.*, a γ-clique. The pseudo-code of this decoder appears in Algorithm 1.

Algorithm 1 may be used in two situations. First, to build a solution from scratch. Second, to complete (*i.e.*, to reconstruct) a partially destroyed solution. For the second case, the decoder receives as an additional parameter a partial solution formed by a non-empty list S of vertices.

The second decoder, named DECODER-IG*, was also originally presented in [23] and is an extension of DECODER-HCB. It follows the iterated greedy scheme, using the constructive heuristic HCB. The population is formed by longer vectors of $2 \cdot |V|$ random keys $R_j \in [0, 1)$ each, with $j = 1, \ldots, |V|, |V| + 1, \ldots, 2 \cdot |V|$. The first $|V|$ random keys are used in the construction of the initial solution and in the reconstruction phase, while the last $|V|$ random keys are used in the destruction phase. Algorithm 2 takes two additional parameters as inputs: δ and β. 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 role played by each parameter is explained with more detail in [23]. The pseudo-code of this decoder appears in Algorithm 2.

The algorithm starts in line 1 building the initial solution S' with decoder DECODER-HCB and the first $|V|$ random keys $R_j, j = 1, \ldots, |V|$. Lines 2 to 10 implement the loop that repeats the partial destruction (vertex eliminations) followed by the reconstruction (vertex insertions) of the current solution, until no further

Algorithm 1. DECODER-HCB$(G, \gamma, \alpha, minsize, S, R)$.

1: $CL \leftarrow V \setminus S$
2: **if** $S = \emptyset$ **then**
3: $\quad RCL \leftarrow \{v \in CL : |\{v' \in CL : deg_G(v') \geq deg_G(v)\}| \leq \max\{minsize, \lfloor \alpha \cdot |CL| \rfloor\}\}$
4: $\quad x \leftarrow argmin\{R_j : j \in RCL\}$
5: $\quad S \leftarrow \{x\}$
6: **end if**
7: **while** $CL \neq \emptyset$ **do**
8: $\quad CL \leftarrow \emptyset$
9: $\quad$ **for all** $v \in V \setminus S$ **do**
10: $\quad\quad$ **if** $\frac{2 \cdot (|E(S)| + deg_G(v,S))}{|S|(|S|+1)} \geq \gamma$ **then**
11: $\quad\quad\quad CL \leftarrow CL \cup \{v\}$
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: $\quad$ **if** $CL \neq \emptyset$ **then**
15: $\quad\quad$ **for all** $v \in CL$ **do**
16: $\quad\quad\quad dif(v) \leftarrow deg_G(v, CL) + |CL| \cdot (deg_G(v, S) - \gamma \cdot (|S| + 1))$
17: $\quad\quad$ **end for**
18: $\quad\quad RCL \leftarrow \{v \in CL : |\{v' \in CL : dif(v') \geq dif(v)\}| \leq \max\{minsize, \lfloor \alpha \cdot |CL| \rfloor\}\}$
19: $\quad\quad x \leftarrow argmin\{R_j : j \in RCL\}$
20: $\quad\quad S \leftarrow S \cup \{x\}$
21: $\quad$ **end if**
22: **end while**

improvements can be obtained. The current solution S' is copied to S in line 3. Lines 4 to 8 implement the loop that removes one by one $\lfloor \delta \cdot |S'| \rfloor$ vertices that should be eliminated from the current solution. In each step, in line 5 a restricted candidate list RCL of size $\max\{minsize, \lfloor \beta \cdot |S'| \rfloor\}$ is created containing the vertices with the smallest degrees in $G(S')$. The vertex with the smallest random key is selected from RCL in line 6 and eliminated from the current solution in line 7. Line 9 performs the reconstruction phase: the current partial solution S' is rebuilt by decoder DECODER-HCB using the first $|V|$ random keys. The loop is interrupted in line 10 when the new solution S' obtained by destruction-reconstruction does not improve the incumbent S or the graph $G(S')$ is not connected; otherwise a new iteration resumes.

Algorithm 2. DECODER-IG$^*(G, \gamma, \alpha, \delta, \beta, minsize, R)$.

1: $S' \leftarrow$ DECODER-HCB$(G, \gamma, \alpha, minsize, \emptyset, R_j : j = 1, \ldots, |V|)$
2: **repeat**
3: $\quad S \leftarrow S'$
4: $\quad$ **for** $k = 1$ to $\lfloor \delta \cdot |S'| \rfloor$ **do**
5: $\quad\quad RCL \leftarrow \{v \in S' : |\{v' \in S' : deg_G(v', S') \leq deg_G(v, S')\}| \leq \max\{minsize, \lfloor \beta \cdot |S'| \rfloor\}\}$
6: $\quad\quad x \leftarrow argmin\{R_{|V|+j} : j \in RCL\}$
7: $\quad\quad S' \leftarrow S' \setminus \{x\}$
8: $\quad$ **end for**
9: $\quad S' \leftarrow$ DECODER-HCB$(G, \gamma, \alpha, minsize, S', R_j : j = 1, \ldots, |V|)$
10: **until** $|S'| \leq |S|$ **or** graph $G(S')$ is not connected

5. New decoder based on exact local search

The principle of the new decoder proposed in this section consists in replacing the reconstruction phase of decoder DECODER-IG* (*i.e.*, line 9 of Algorithm 2) by an exact algorithm that finds the maximum quasi-

clique that can be reconstructed by DECODER-HCB starting from the remaining set of vertices S' with $\alpha = 1$. A slightly modified version of the *QClique* algorithm of C.C. Ribeiro and J.A. Riveaux [25] is used with this goal.

Algorithm 3 describes the pseudo-code of decoder DECODER-LSQClique. The algorithm takes as inputs the same parameters as DECODER-IG*, with the exception of an extended vector with $2 \cdot |V| + 1$ random keys $R_j^+, j = 1, \ldots, 2 \cdot |V|, 2 \cdot |V| + 1$ and one additional frequency parameter ρ: the exact search algorithm will be applied whenever $R_{2\cdot|V|+1}^+ \leq \rho$, otherwise decoder DECODER-HCB will be used to reconstruct the solution. We note that, if $\rho = 1$ the exact search algorithm will always be executed. If $\rho = 0$, it will never be executed and DECODER-LSQClique will behave exactly as DECODER-IG*. We observe that by appropriately tuning and setting the best value for ρ one can optimize the performance of this decoder.

DECODER-LSQClique starts by creating an initial solution S' in line 1, using DECODER-HCB and the random keys $R_j, j = 1, \ldots, |V|$. The loop in lines 2 to 14 repeats the partial destruction (vertex eliminations) followed by the reconstruction (vertex insertions) of the current solution, until no further improvements can be obtained. The current solution S' is copied to S in line 3. The loop in lines 4 to 8 removes one by one the $\lfloor \delta \cdot |S'| \rfloor$ vertices that should be eliminated from the current solution. A restricted candidate RCL of size $\max\{minsize, \lfloor \beta \cdot |CL| \rfloor\}$ 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 starts in line 9. If the random key $R_{2\cdot|V|+1}^+$ is smaller than or equal to parameter ρ and $G(S')$ is a γ-clique, then the partial solution S' is extended in line 10 by an exact local search algorithm and a new solution S^* is obtained. Otherwise, solution S' is rebuilt by DECODER-HCB in line 12, once again using the first random keys $R_j^+, j = 1, \ldots, |V|$. The loop is interrupted in line 14 when the new solution S' obtained by destruction-reconstruction does not improve the incumbent S or the graph $G(S')$ is not connected; otherwise a new iteration resumes. The best solution is returned in S.

Algorithm 3. DECODER-LSQClique$(G, \gamma, \alpha, \delta, \beta, minsize, S, R^+, \rho)$.

1: $S' \leftarrow$ DECODER-HCB$(G, \gamma, \alpha, minsize, \emptyset, R_j^+, j = 1, \ldots, |V|)$
2: **repeat**
3: $\quad S \leftarrow S'$
4: $\quad$ **for** $k = 1$ to $\lfloor \delta \cdot |S'| \rfloor$ **do**
5: $\quad\quad RCL \leftarrow \{v \in S' : |\{v' \in S' : deg_G(v', S') \leq deg_G(v, S')\}| \leq \max\{minsize, \lfloor \beta \cdot |S'| \rfloor\}\}$
6: $\quad\quad x \leftarrow argmin\{R_{|V|+j} : j \in RCL\}$
7: $\quad\quad S' \leftarrow S' \setminus \{x\}$
8: $\quad$ **end for**
9: $\quad$ **if** $R_{2\cdot|V|+1}^+ \leq \rho$ **and** $dens(G(S')) \geq \gamma$ **then**
10: $\quad\quad LSQClique(G, S', \gamma, |S|, S^*)$
11: $\quad$ **else**
12: $\quad\quad S' \leftarrow$ DECODER-HCB$(G, \gamma, \alpha, minsize, S', R_j^+ : j = 1, \ldots, |V|)$
13: $\quad$ **end if**
14: **until** $|S'| \leq |S|$ **or** graph $G(S')$ is not connected

6. Exact algorithm for new decoder

Algorithm *LSQClique* described in this section is an exact algorithm for the maximum quasi-clique problem. This variant may investigate multiple permutations of the same solution along the search tree, because it does not make use of the quasi-heredity proposition [25]. It will be used as a variant of algorithm *QClique* in line 10 of Algorithm 3 to perform the exact local search.

Algorithm 4. *LSQClique*(G, S', γ, LB, S^*).

1: $S^* \leftarrow S'$
2: $i \leftarrow |S'|$
3: **if** a solution of size k containing all vertices of S' does not exist for some $k = i+1, \ldots, LB+1$ **then return**
4: $CL \leftarrow \{v \in V \setminus S' : dens(G(S' \cup \{v\})) \geq \gamma\}$
5: **if** $CL = \emptyset$ **then return**
6: **for all** $j \in CL$ **do**
7: $\quad$ *LSQClique*$(G, S' \cup \{j\}, \gamma, \max\{LB, |S'|+1\}, S'')$
8: $\quad$ **if** $|S''| > |S^*|$ **then**
9: $\quad\quad S^* \leftarrow S''$
10: $\quad\quad LB \leftarrow \max\{LB, |S^*|\}$
11: $\quad$ **end if**
12: **end for**

Algorithm 4 works directly with the initial solution represented by the set of vertices S'. It takes as inputs the graph G, the threshold γ, the partial solution S', and the initial lower bound $LB = |S|$, returning the best solution found S^* also as a parameter. Line 1 of the pseudo-code copies to S^* the current partial solution S'. Line 2 sets i with the size of the current solution S'. If a solution of size k containing all vertices in S' does not exist for some value of $k = i+1, \ldots, LB+1$, then the algorithm returns in line 3 (pruning).

The candidate list CL created in line 4 is formed by every vertex $v \in V \setminus S' : dens(G(S' \cup \{v\})) \geq \gamma$, as for DECODER-HCB. If the candidate list is empty, then the algorithm returns in line 5.

The loop in lines 6 to 12 attempts to extend the current solution S' by each vertex j in the candidate list CL. The vertices in the candidate list are taken in non increasing order of their degrees $deg_G(j, S')$. For each of them, line 7 recursively invokes the algorithm for a new, extended partial solution $S' \cup \{j\}$ and a new, updated lower bound $\max\{LB, |S'|+1\}$, returning a new solution S''. Line 8 checks if the new solution S'' improves then current best S^*. If this is the case, the current best and the lower bound are updated in lines 9 and 10, respectively. When the recursion terminates, S^* contains the best solution obtained by the algorithm.

The pruning strategy discards the current node of the enumeration tree in line 3 if a solution S of size k containing all vertices of S' can not exist for some $k = i+1, \ldots, LB+1$. This is done by calculating an upper bound to the number of edges in $G(S)$ and checking if the number of edges in a γ-clique is greater than this upper bound. We follow a similar idea to that in [25].

Let $E(S', S \setminus S')$ denote the set of edges in G between the vertices of S' and $S \setminus S'$. If S is a γ-clique, then the following condition holds and gives a lower bound to $|E(S)|$:

$$|E(S)| = |E(S') \cup E(S', S \setminus S') \cup E(S \setminus S')| \geq \gamma \cdot \binom{k}{2}. \tag{6.1}$$

Since the sets in equation (6.1) are disjoint,

$$|E(S)| = |E(S')| + |E(S', S \setminus S')| + |E(S \setminus S')|. \tag{6.2}$$

Since $|E(S')|$ is already known, in order to calculate an upper bound to $|E(S)|$ we need to calculate an upper bound to $|E(S', S \setminus S')| + |E(S \setminus S')|$.

Let V' be formed by the $k-i$ vertices of $V \setminus S'$ with the maximum number of adjacent vertices in G that belong to S'. V' can be determined in polynomial time $O(|V|)$: first sort the vertices in the non increasing order of their degrees $deg_G(v, S)$, then select the $k-i$ first vertices. Then, an upper bound to $|E(S', S \setminus S')|$ is given by $|E(S', V')|$.

Now, let us define V'' as the set formed by the $k-i$ vertices of $V \setminus S'$ with maximum degree in G. Then, the following condition holds:

$$\sum_{v \in S \setminus S'} deg_G(v) \leq \sum_{v \in V''} deg_G(v). \tag{6.3}$$

Following a reasoning similar to that in [25], we obtain:

$$|E(S', S \setminus S')| + |E(S \setminus S')| \\ \leq \sum_{v \in V'} deg_G(v, S') + \min \left\{ \frac{\sum_{v \in V''} deg_G(v) - \sum_{v \in V'} deg_G(v, S')}{2}, \binom{k-i}{2} \right\}. \tag{6.4}$$

Therefore, if the inequality below is not satisfied for at least one value of $k = i+1, \ldots, LB+1$, then a solution of size $LB + 1$ does not exist and we can prune the current node of the search tree:

$$|E(S')| + \sum_{v \in V'} deg_G(v, S') + \min \left\{ \frac{\sum_{v \in V''} deg_G(v) - \sum_{v \in V'} deg_G(v, S')}{2}, \binom{k-i}{2} \right\} \\ \geq \gamma \cdot \binom{k}{2}. \tag{6.5}$$

7. Computational results

7.1. Experimental setting

All algorithms were implemented using version 19.14.26429.4 of the Microsoft® C/C++ Optimizing Compiler for x64. The computational experiments have been performed on an Intel Core i5-5200 processor with 2.20 GHz and 8 GB of RAM running under Windows 10.

The newly proposed algorithm BRKGA-LSQClique was compared with the original BRKGA-IG* heuristic of [23], which was the best heuristic for the maximum quasi-clique problem at the time of writing. The best parameters for algorithm BRKGA-IG* were determined using the automatic tuning tool IRACE [16, 22]. The same parameter settings were used for algorithm BRKGA-LSQClique, except for the parameter δ that controls the number of vertices to be destroyed. In order to enforce that the fraction of nodes eliminated in the destruction phase be smaller for dense graphs, DECODER-LSQClique empirically sets $\delta = 1 - dens(G)$, if $dens(G) < 0.8$; $\delta = 2 \cdot (1 - dens(G))$, otherwise.

DECODER-LSQClique makes use of an additional parameter ρ to establish whether the exact local search should be applied or not to some solution. Again, we empirically set the probability that exact local search is applied at $\rho = \max\{1 - dens(G), 0.80\}$.

Implementation notice. An additional parameter *maxnodes* was used to enforce a maximum limit to the number of nodes of the search tree generated by the exact algorithm *LSQClique*. In case the search tree generated by *LSQClique* reaches this maximum limit of nodes, then the algorithm returns the incumbent before an optimal solution is obtained. For this reason, an additional application of the reconstruction procedure

$$S' \leftarrow \text{DECODER-HCB}(G, \gamma, \alpha, minsize, S^*, R_j^+ : j = 1, \ldots, |V|)$$

should be enforced after line 10 of Algorithm 3 to further improve the current solution. This parameter *maxnodes* was set at 220 using the IRACE tool.

All tested value ranges and the best parameter settings are shown in Table 1.

7.2. Numerical results

Pinto *et al.* [23] presented numerical results for experiments on 67 DIMACS instances [14, 28], 33 maximum clique instances of the Benchmarks with Hidden Optimum Solutions for Graph Problems (BHOSLIB) [5, 26, 27], and six sparse instances from the University of Florida Sparse Matrix Collection [9]. In the experiments reported in the section, we eliminated all instances with less than 400 vertices or with density greater than or equal to 0.99, therefore excluding 29 instances from [14, 28] and two instances from [9] that were used in [23]. We also considered three large sparse instances from [9] that were used by Veremyev *et al.* [31] (Geom, EVA and

TABLE 1. Parameters settings: parameter *maxnodes* was set using IRACE in this work.

Parameter	Value ranges	BRKGA-IG*	BRKGA-LSQClique
p	50, 51, ..., 100	91	91
p_e	0.10, 0.11, ..., 0.25	0.13	0.13
p_m	0.10, 0.11, ..., 0.30	0.22	0.22
rhoe	0.50, 0.51, ..., 0.80	0.78	0.78
α	0.01, 0.02, ..., 0.20	0.01	0.01
minsize	1, 2, 3, 4, 5, 6	3	3
δ	0.01, 0.02, ..., 0.50	0.40	$\begin{cases} 1 - dens(G); \text{ if } dens(G) < 0.8 \\ 2 \cdot (1 - dens(G)); \text{ otherwise} \end{cases}$
β	0.01, 0.02, ..., 0.20	0.02	0.02
maxnodes	100, 110, ..., 300	–	220
ρ	–	–	$\max\{1 - dens(G), 0.80\}$

Notes. Parameters ρ and δ were empirically set in this work. All other parameters were set using IRACE by Pinto *et al.* [23].

PGPgiantcompo) and instance vsp_p0291_seymourl_iiasa used by Ribeiro and Riveaux [25]. Table 2 describes the characteristics of each instance.

In the first experiment, for each instance and each value of the threshold γ, we performed 30 runs of each algorithm BRKGA-IG* and BRKGA-LSQClique until a target solution value was found. For each instance and value of γ, the target was determined as follows. We run both BRKGA-IG* and BRKGA-LSQClique for 100 generations or ten minutes, whatever happened first, and selected the target as the best among the solution values found by the two algorithms. For some instances for which this value revealed itself to be easy, we progressively increased the target until a sufficiently hard value was reached.

We discarded the results for the combinations of instance and value of γ for which both algorithm obtained the target solution value in less than two generations on average over the 30 runs. Therefore, Tables 3–9 display the results for 150 combinations of different values of γ with 24 of the 38 DIMACS instances, with all 33 BHOSLIB instances, and with four of the eight sparse instances that were not completely discarded. Each run was limited to ten minutes of execution, except for the instances in Table 9, whose runs were limited to 30 minutes of execution. For each instance and value of γ, the tables show the target `look4` considered in the experiment, and the results obtained by each algorithm BRKGA-IG* and BRKGA-LSQClique. For each algorithm, the table presents the average and best sizes of the solution found over the 30 runs, the average running time in seconds and the average generation in which the target was found over the runs that reached the target value (#look4). Whenever any of the two methods failed to find a solution as good as the target value `look4` within the time limit for all instances in the group of same density, we indicate it by "> 600 (0)" or "> 1800 (0)".

The rows highlighted in dark gray (▓) correspond to the cases where BRKGA-IG* reached the target `look4` more often or, when both of them reached the target the same number of times, it was faster than BRKGA-LSQClique in terms of the average time to target. Rows in light gray (▒) are those for which BRKGA-LSQClique performed better in terms of the same criteria. For all other cases, both algorithms found the target in the first generation in all runs.

For each instance and each value of γ in Tables 3–9, the two algorithms are compared primarily on the number of runs in which the target solution value was found and, whenever there is a tie, on the average time taken to find the target solution value over the successful runs. The plot in Figure 3 displays the comparison between algorithms BRKGA-IG* and BRKGA-LSQClique. For each value of γ, it indicates the fraction of the number of instances for which each algorithm outperforms the other. Considering all 150 combinations of instances and values of γ, BRKGA-LSQClique outperformed the original BRKGA-IG* in 63.33% of the cases in terms of the time-to-target solution value.

TABLE 2. Description of the test instances.

Instance	$\|V\|$	$\|E\|$	Density	Instance	$\|V\|$	$\|E\|$	Density
san400_0.5_1	400	39 900	0.50	frb45-21-1	945	386 854	0.87
sanr400_0.5	400	39 984	0.50	frb45-21-2	945	387 416	0.87
san400_0.7_1	400	55 860	0.70	frb45-21-5	945	387 461	0.87
san400_0.7_2	400	55 860	0.70	frb45-21-4	945	387 491	0.87
san400_0.7_3	400	55 860	0.70	p_hat1000-1	1000	122 253	0.24
sanr400_0.7	400	55 869	0.70	p_hat1000-2	1000	244 799	0.49
brock400_3	400	59 681	0.75	DSJC1000.5	1000	249 826	0.50
brock400_1	400	59 723	0.75	san1000	1000	250 500	0.50
brock400_2	400	59 786	0.75	p_hat1000-3	1000	371 746	0.74
gen400_p0.9_55	400	71 820	0.90	C1000.9	1000	450 079	0.90
gen400_p0.9_65	400	71 820	0.90	hamming10-4	1024	434 176	0.83
frb30-15-2	450	83 151	0.82	email	1133	5451	0.01
frb30-15-4	450	83 194	0.82	frb50-23-2	1150	579 824	0.88
frb30-15-1	450	83 198	0.82	frb50-23-4	1150	580 417	0.88
frb30-15-5	450	83 231	0.82	frb50-23-1	1150	580 603	0.88
Erdos971	472	1314	0.01	frb50-23-5	1150	580 640	0.88
johnson32-2-4	496	107 880	0.88	frb53-24-4	1272	714 048	0.88
Harvard500	500	2043	0.02	frb53-24-2	1272	714 067	0.88
c-fat500-1	500	4459	0.04	frb53-24-1	1272	714 129	0.88
c-fat500-2	500	9139	0.07	frb53-24-5	1272	714 130	0.88
c-fat500-5	500	23 191	0.19	frb56-25-4	1400	869 262	0.89
p_hat500-1	500	31 569	0.25	frb56-25-1	1400	869 624	0.89
c-fat500-10	500	46 627	0.37	frb56-25-5	1400	869 699	0.89
DSJC500.5	500	62 624	0.50	frb56-25-2	1400	869 899	0.89
p_hat500-2	500	62 946	0.50	p_hat1500-2	1500	568 960	0.51
p_hat500-3	500	93 800	0.75	frb59-26-4	1534	1 048 800	0.89
C500.9	500	112 332	0.90	frb59-26-1	1534	1 049 256	0.89
frb35-17-5	595	148 572	0.84	frb59-26-2	1534	1 049 648	0.89
frb35-17-1	595	148 859	0.84	frb59-26-5	1534	1 049 829	0.89
frb35-17-2	595	148 868	0.84	C2000.9	2000	1 799 532	0.90
frb35-17-4	595	148 873	0.84	keller6	3361	4 619 898	0.82
p_hat700-1	700	60 999	0.25	C4000.5	4000	4 000 268	0.50
p_hat700-2	700	121 728	0.50	frb100-40	4000	7 425 226	0.93
frb40-19-5	760	246 801	0.86	CA-GrQc	5242	14 496	$1.1/10^3$
frb40-19-4	760	246 815	0.86	Geom	7343	11 898	$1.9/10^4$
frb40-19-1	760	247 106	0.86	EVA	8497	6711	$3.4/10^4$
frb40-19-2	760	247 157	0.86	vsp_p0291-	10 498	53 868	$9.8/10^4$
keller5	776	225 990	0.75	_seymourl_iiasa			
brock800_3	800	207 333	0.65	PGPgiant-	10 680	24 316	$4.3/10^4$
brock800_1	800	207 505	0.65	compo			
brock800_2	800	208 166	0.65				

In the next experiment, we evaluate and compare the run time distributions (or time-to-target plots) of algorithms BRKGA-IG* and BRKGA-LSQClique for some instances. 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 and Stützle [13] as a way to characterize the running times of stochastic local search algorithms for combinatorial optimization problems. In this experiment, the two algorithms were made to stop whenever a solution with cost greater than or equal to a given target value was found. The targets are the same used in the first experiment.

TABLE 3. First experiment: 30 runs of each instance and algorithm limited to 600 s of running time each.

			BRKGA-IG*				BRKGA-LSQClique			
Instance	γ	`look4`	Avg.	Best	Time (#`look4`)	Avg. gen.	Avg.	Best	Time (#`look4`)	Avg. gen.
sanr400_0.5	0.999	13	13.00	13	6.21(30)	49.33	13.00	13	6.67(30)	25.97
	0.80	32	32.00	32	1.99(30)	3.20	32.00	32	1.46(30)	1.70
san400_0.7_1	0.999	40	40.00	40	0.26(30)	1.77	40.00	40	0.54(30)	2.47
san400_0.7_3	0.999	20	21.30	22	141.01(29)	1153.10	20.30	22	245.54(20)	920.95
sanr400_0.7	0.999	21	21.00	21	0.39(30)	3.03	21.00	21	0.87(30)	3.87
	0.90	46	46.00	46	4.88(30)	9.43	46.00	46	3.60(30)	5.00
	0.80	113	113.00	113	5.24(30)	7.50	113.00	113	3.31(30)	3.40
brock400_3	0.999	25	25.00	25	1.25(30)	10.47	25.00	25	1.57(30)	9.10
	0.90	61	61.00	61	29.52(30)	56.03	61.00	61	14.58(30)	20.93
	0.80	187	187.00	187	29.11(30)	39.60	187.00	187	6.09(30)	2.47
brock400_1	0.999	25	25.00	25	101.51(30)	852.43	24.83	25	294.50(25)	1504.56
	0.90	60	60.00	60	23.60(30)	47.30	60.00	60	10.78(30)	14.37
	0.80	189	189.00	189	2.62(30)	3.10	189.00	189	1.84(30)	1.27
brock400_2	0.999	25	25.00	25	1.57(30)	13.10	25.00	25	4.81(30)	22.43
	0.90	61	61.00	61	1.88(30)	3.67	61.00	61	2.30(30)	3.23
	0.80	186	186.00	186	4.50(30)	6.00	186.00	186	1.74(30)	1.63
gen400_p0.9_55	0.999	54	53.50	54	340.34(15)	2554.47	53.10	54	212.16(3)	1498.67
gen400_p0.9_65	0.999	66	66.00	66	8.73(30)	64.47	66.00	66	26.00(30)	149.10
frb30-15-2	0.999	29	29.13	30	13.16(30)	84.63	29.03	30	24.96(30)	119.60
	0.95	58	58.00	58	11.38(30)	19.40	58.00	58	7.96(30)	11.60
	0.90	138	137.40	138	276.64(12)	318.17	137.83	138	221.01(25)	196.72
frb30-15-4	0.999	29	29.00	29	58.10(30)	394.97	29.00	29	75.50(30)	332.63
	0.95	61	60.80	61	243.11(24)	364.58	60.80	61	170.38(24)	184.75
	0.90	138	137.70	138	255.28(21)	303.10	137.97	138	172.72(29)	150.90

TABLE 4. First experiment: 30 runs of each instance and algorithm limited to 600 s of running time each.

			BRKGA-IG*				BRKGA-LSQClique			
Instance	γ	`look4`	Avg.	Best	Time (#`look4`)	Avg. gen.	Avg.	Best	Time (#`look4`)	Avg. gen.
frb30-15-1	0.999	29	29.00	29	18.90(30)	126.47	29.00	29	34.37(30)	171.73
	0.95	59	59.00	59	7.31(30)	13.07	59.00	59	7.20(30)	9.80
	0.90	139	139.00	139	25.30(30)	31.47	139.00	139	11.57(30)	11.37
frb30-15-5	0.999	29	29.00	29	55.16(30)	375.47	29.00	29	141.80(30)	694.60
	0.95	60	60.00	60	99.01(30)	157.43	60.00	60	41.69(30)	47.63
	0.90	134	134.00	134	19.61(30)	24.60	134.00	134	11.39(30)	11.57
DSJC500.5	0.90	21	21.00	21	3.04(30)	4.87	21.00	21	2.89(30)	3.10
	0.80	34	34.00	34	20.66(30)	21.70	34.00	34	10.85(30)	10.13
p_hat500-3	0.999	51	51.00	51	5.07(30)	22.57	51.00	51	8.14(30)	23.63
C500.9	0.999	57	57.00	57	10.79(30)	49.40	57.00	57	11.12(30)	55.10
	0.95	157	157.00	157	8.72(30)	10.17	157.00	157	4.01(30)	4.83
frb35-17-5	0.999	33	33.03	34	15.21(30)	60.90	33.00	33	24.78(30)	87.80
	0.95	79	79.00	79	86.15(30)	69.50	79.03	80	33.77(30)	23.07
	0.90	225	224.53	225	389.49(16)	252.63	225.00	225	109.48(30)	71.33
frb35-17-1	0.999	33	33.00	33	41.66(30)	168.80	33.00	33	108.88(30)	372.70
	0.95	78	77.97	78	112.96(29)	92.87	78.00	78	67.17(30)	47.77
	0.90	216	216.00	216	6.75(30)	3.97	216.00	216	4.31(30)	2.30
frb35-17-2	0.999	33	33.00	33	21.08(30)	85.87	33.00	33	32.79(30)	115.53
	0.95	75	75.10	76	81.34(30)	63.47	75.10	76	33.02(30)	23.03
	0.90	207	207.00	207	100.84(30)	60.60	207.00	207	43.63(30)	23.40
frb35-17-4	0.999	33	33.00	33	47.44(30)	187.57	33.00	33	77.14(30)	247.93
	0.95	79	79.07	80	23.49(30)	18.93	79.17	81	13.27(30)	9.27
	0.90	238	237.77	238	258.20(23)	170.00	238.00	238	155.42(30)	98.00
p_hat700-1	0.999	11	11.00	11	4.10(30)	23.13	11.00	11	8.93(30)	14.77
	0.80	22	22.00	22	1.81(30)	2.07	22.00	22	1.56(30)	1.27

TABLE 5. First experiment: 30 runs of each instance and algorithm limited to 600 s of running time each.

			BRKGA-IG*				BRKGA-LSQClique			
Instance	γ	`look4`	Avg.	Best	Time (#`look4`)	Avg. gen.	Avg.	Best	Time (#`look4`)	Avg. gen.
frb40-19-5	0.999	38	38.00	38	99.55(30)	254.43	37.97	38	120.16(29)	282.14
	0.95	100	99.50	100	264.97(15)	115.07	99.97	100	123.23(29)	51.97
	0.90	333	332.80	333	219.13(24)	87.38	333.00	333	32.61(30)	13.17
frb40-19-4	0.999	38	37.83	39	145.57(24)	370.67	37.60	38	252.25(18)	573.72
	0.95	95	95.00	95	53.74(30)	24.57	95.20	96	22.56(30)	9.33
	0.90	292	291.10	292	395.01(4)	160.50	291.73	292	229.11(22)	90.50
frb40-19-1	0.999	37	37.00	37	27.10(30)	68.90	37.03	38	32.11(30)	73.73
	0.95	114	112.63	114	504.90(2)	232.00	113.30	114	340.15(11)	136.91
frb40-19-2	0.999	37	37.00	37	15.78(30)	40.13	37.00	37	23.98(30)	55.23
	0.95	103	102.90	104	178.76(26)	82.04	103.07	104	71.23(30)	30.57
	0.90	363	363.00	363	16.89(30)	6.80	363.00	363	6.77(30)	2.83
keller5	0.90	132	132.00	132	92.79(30)	41.03	132.00	132	111.25(30)	45.77
brock800_3	0.999	21	21.00	21	11.83(30)	18.67	21.00	21	15.97(30)	24.17
	0.90	42	41.97	42	190.77(29)	79.76	42.00	42	28.47(30)	12.30
	0.80	93	92.73	93	237.31(22)	78.91	93.00	93	59.67(30)	20.27
brock800_1	0.999	21	21.00	21	50.07(30)	108.77	21.00	21	91.67(30)	135.43
	0.90	42	41.87	42	143.39(26)	54.88	42.00	42	68.18(30)	26.67
	0.80	96	95.77	96	254.22(23)	80.22	96.00	96	58.27(30)	19.03
brock800_2	0.999	21	21.00	21	54.91(30)	106.90	21.00	21	131.81(30)	189.13
	0.90	42	41.93	42	182.78(28)	70.07	42.00	43	98.38(29)	39.59
	0.80	94	94.03	95	94.59(30)	30.87	94.00	94	33.99(30)	11.27
frb45-21-1	0.999	41	41.07	42	12.20(30)	18.97	41.03	42	26.03(30)	37.90
	0.95	121	121.33	123	43.20(30)	10.37	121.57	124	15.65(30)	3.63
	0.90	508	508.00	508	35.54(30)	10.27	508.00	508	7.10(30)	2.20

TABLE 6. First experiment: 30 runs of each instance and algorithm limited to 600 s of running time each.

			BRKGA-IG*				BRKGA-LSQClique			
Instance	γ	`look4`	Avg.	Best	Time (#`look4`)	Avg. gen.	Avg.	Best	Time (#`look4`)	Avg. gen.
frb45-21-2	0.999	41	41.07	42	15.26(30)	23.63	41.03	42	41.90(30)	62.47
	0.95	120	120.23	122	71.53(30)	16.93	120.33	122	39.44(30)	9.83
	0.90	491	490.17	491	291.03(5)	73.80	491.00	491	151.37(30)	42.80
frb45-21-5	0.999	42	41.90	43	195.14(26)	302.19	41.57	42	186.09(17)	274.47
	0.95	121	120.93	123	177.89(27)	44.82	121.20	122	96.31(30)	23.33
	0.90	508	507.13	508	176.41(4)	43.75	508.00	508	61.36(30)	16.97
frb45-21-4	0.999	42	42.00	42	125.88(30)	189.07	41.93	42	183.97(28)	260.39
	0.95	128	128.00	128	126.04(30)	30.83	128.33	130	43.94(30)	9.67
	0.90	554	554.00	554	17.76(30)	5.00	554.00	554	3.44(30)	1.00
p_hat1000-1	0.90	15	15.00	15	2.48(30)	3.60	15.00	15	2.05(30)	1.57
	0.80	23	23.00	23	48.94(30)	22.67	23.00	23	70.60(30)	32.37
DSJC1000.5	0.999	15	14.67	15	265.43(20)	349.20	14.73	15	180.49(22)	180.36
	0.90	24	23.30	24	254.91(9)	82.22	23.63	24	203.97(19)	71.05
	0.80	39	38.87	40	242.46(27)	31.59	39.03	40	41.22(30)	7.03
C1000.9	0.999	67	67.10	68	126.51(30)	60.97	67.00	68	170.53(27)	103.26
	0.95	217	217.30	219	128.51(30)	27.30	217.43	219	36.15(30)	6.37
hamming10-4	0.999	40	40.00	40	30.99(30)	34.53	40.00	40	29.45(30)	31.53
	0.95	83	82.10	83	249.13(3)	60.67	83.00	83	73.87(30)	22.17
frb50-23-2	0.999	46	46.03	47	27.28(30)	14.53	46.00	46	61.16(30)	33.60
	0.95	158	156.13	158	397.67(6)	40.17	158.13	159	193.64(30)	24.40
	0.90	780	779.93	780	154.18(28)	20.07	780.00	780	18.71(30)	2.80
frb50-23-4	0.999	47	47.00	47	159.94(30)	85.43	46.77	47	185.04(23)	99.87
	0.95	157	156.43	158	350.15(19)	36.11	157.40	160	154.65(29)	20.97
	0.90	772	771.00	771	0.00(0)	–	771.60	772	217.30(18)	33.61

TABLE 7. First experiment: 30 runs of each instance and algorithm limited to 600 s of running time each.

			BRKGA-IG*				BRKGA-LSQClique			
Instance	γ	`look4`	Avg.	Best	Time (#`look4`)	Avg. gen.	Avg.	Best	Time (#`look4`)	Avg. gen.
frb50-23-1	0.999	46	46.03	47	29.85(30)	15.97	46.03	47	70.75(30)	38.87
	0.95	157	157.30	159	295.15(28)	28.46	157.53	160	54.86(30)	6.63
	0.90	797	796.03	797	87.44(1)	11.00	797.00	797	52.13(30)	8.13
frb50-23-5	0.999	46	46.10	47	26.44(30)	13.60	46.13	47	54.29(30)	29.43
	0.95	160	158.53	162	356.84(6)	33.50	160.43	162	201.76(30)	26.43
	0.90	788	788.00	788	77.23(30)	10.00	788.00	788	8.17(30)	1.17
frb53-24-4	0.999	50	49.00	50	283.98(5)	91.40	48.70	50	317.79(4)	107.75
	0.95	178	177.83	179	298.29(22)	19.18	178.40	180	78.62(30)	5.80
frb53-24-2	0.999	50	49.43	51	189.84(13)	64.38	49.30	50	216.89(12)	77.08
	0.95	172	169.80	172	382.90(5)	26.20	172.07	174	202.27(29)	18.00
	0.90	926	925.07	926	285.68(2)	25.00	925.90	926	123.69(27)	13.00
frb53-24-1	0.999	50	49.10	50	366.36(6)	123.83	48.77	49	>600(0)	0.00
	0.95	199	198.57	200	336.91(17)	20.88	199.50	201	142.60(30)	10.93
	0.90	978	977.43	978	272.78(13)	27.46	978.00	978	23.79(30)	2.63
frb53-24-5	0.999	49	48.97	49	165.88(29)	54.83	48.80	50	194.90(23)	65.26
	0.95	166	166.50	168	302.86(26)	20.58	167.13	171	69.86(30)	5.50
	0.90	894	893.00	893	>600(0)	–	894.00	894	125.81(30)	11.70
frb56-25-4	0.999	51	51.07	52	92.58(30)	25.20	51.00	52	131.59(29)	37.00
	0.95	195	193.33	196	221.82(1)	10.00	195.33	197	128.24(30)	7.47
frb56-25-1	0.999	53	51.63	53	236.63(4)	66.00	51.20	53	188.83(1)	56.00
	0.95	225	224.30	225	242.23(15)	10.80	225.33	227	69.21(30)	3.27
	0.90	1161	1161.00	1161	52.68(30)	4.27	1161.00	1161	11.05(30)	1.00
frb56-25-5	0.999	52	51.53	53	240.08(15)	66.40	51.23	52	165.91(8)	45.50
	0.95	201	199.10	202	515.94(6)	23.33	201.50	203	188.53(30)	11.41

TABLE 8. First experiment: 30 runs of each instance and algorithm limited to 600 s of running time each.

			BRKGA-IG*				BRKGA-LSQClique			
Instance	γ	`look4`	Avg.	Best	Time (#`look4`)	Avg. gen.	Avg.	Best	Time (#`look4`)	Avg. gen.
frb56-25-2	0.999	52	51.43	53	280.08(12)	74.17	51.03	52	152.36(4)	42.75
	0.95	213	210.63	212	0.00(0)	0.00	213.80	217	198.41(29)	12.00
frb59-26-4	0.999	55	54.33	55	228.03(11)	52.45	53.73	55	322.46(4)	78.50
	0.95	230	228.27	232	340.83(6)	13.50	230.70	233	203.91(30)	10.87
frb59-26-1	0.999	55	54.17	56	346.35(7)	81.00	53.57	55	242.13(2)	55.50
	0.95	240	239.03	241	318.16(10)	11.30	240.90	244	156.25(30)	6.83
frb59-26-2	0.999	54	53.83	54	199.91(25)	46.44	53.60	54	201.10(18)	49.06
	0.95	235	234.57	237	351.97(16)	13.25	235.80	238	102.22(30)	4.37
frb59-26-5	0.999	54	53.80	55	198.84(22)	46.32	53.50	54	282.46(15)	68.40
	0.95	221	220.83	223	297.89(21)	13.91	221.90	224	89.80(30)	3.73

TABLE 9. First experiment: 30 runs of each instance and algorithm limited to 600 s of running time each.

			BRKGA-IG*				BRKGA-LSQClique			
Instance	γ	`look4+`	Avg.	Best	Time (#`look4`)	Avg. gen.	Avg.	Best	Time (#`look4`)	Avg. gen.
C2000.9-999	0.999	74	74.07	75	535.71(30)	54.13	73.90	75	599.70(26)	71.65
	0.95	276	272.30	275	0.00(0)	0.00	276.63	279	434.55(30)	13.47
keller6	0.999	54	53.50	55	845.64(18)	36.78	53.33	55	662.55(16)	29.94
C4000.5	0.999	17	16.93	17	464.20(28)	19.00	16.97	18	355.98(28)	14.79
	0.90	27	26.83	27	761.12(25)	6.76	27.07	28	116.51(30)	1.20
	0.80	48	43.60	47	>600(0)	–	48.10	49	449.38(29)	5.35
frb100-40	0.999	86	85.90	87	779.23(23)	13.04	85.03	86	774.81(8)	16.00
	0.95	1835	1834.97	1835	885.90(28)	5.29	1835.57	1837	206.73(30)	1.13
EVA	0.90	4	4.00	4	1.69(30)	6.30	4.00	4	1.78(30)	2.73
	0.60	7	7.00	7	3.40(30)	3.17	7.00	7	2.15(30)	1.10
Geom	0.90	23	23.00	23	3.99(30)	8.17	23.00	23	1.94(30)	1.00
	0.70	30	30.00	30	36.86(30)	3.53	30.00	30	6.97(30)	1.13
	0.60	36	35.53	36	607.54(16)	32.00	36.00	36	420.33(30)	28.70
	0.50	44	43.80	44	486.85(24)	17.79	44.00	44	18.50(30)	1.00
vsp_p0291_sey-	0.60	13	12.97	13	360.60(29)	31.52	13.00	13	9.49(30)	1.07
mourl_iiasa	0.50	16	16.00	16	84.82(30)	5.27	16.00	16	13.56(30)	1.00
PGPgiantcompo	0.80	48	48.00	48	24.78(30)	2.40	48.00	48	9.09(30)	1.03
	0.60	60	60.00	60	50.92(30)	2.23	60.00	60	21.59(30)	1.07
	0.50	73	73.00	73	173.76(30)	4.33	73.00	73	26.58(30)	1.00

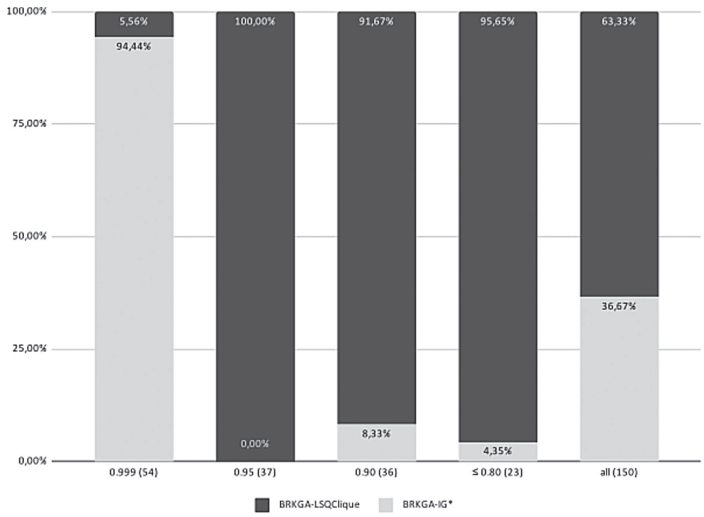


FIGURE 3. Comparison between algorithms BRKGA-IG* and BRKGA-LSQClique: fraction of the number of instances on which each algorithm outperforms the other for each value of γ. Considering all 150 instances and values of γ, BRKGA-LSQClique outperformed the original BRKGA-IG* in 63.33% of the cases in terms of the time-to-target solution value.

Each heuristic was run 200 times for each value of the threshold γ. Next, the empirical probability distributions of the time taken by each heuristic to find the target solution value are plotted. To plot the empirical distribution for each heuristic, we followed the methodology proposed by Aiex *et al.* [2, 3]. 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.

Figures 4 and 5 illustrate the time-to-target plots for instance brock400_2 with $\gamma = 0.999$, 0.90, and 0.80 and for instance PGPgiantcompo with $\gamma = 0.90$, 0.80, and 0.50, respectively. We recall from Table 3 that BRKGA-IG* obtained better results for brock400_2 with $\gamma = 0.90$. The plots in these figures show that BRKGA-IG* performed better for $\gamma = 0.999$, with BRKGA-LSQClique becoming progressively better as γ decreases.

8. Concluding remarks

In this work, we showed that the exact enumeration algorithm *QClique* proposed by Ribeiro and Riveaux [25] can be hybridized with the biased random-key genetic algorithm BRKGA-IG* developed by Pinto *et al.* [23] as a local search strategy to improve the quality of the solutions created by the decoder. The new decoder using an exact local search strategy gives rise to a matheuristic for solving the maximum cardinality quasi-clique problem based on a biased random-key genetic algorithm.

Computational experiments were performed on a set of benchmark instances, considering different thresholds.

The numerical results showed that although BRKGA-IG* performed better for $\gamma = 0.999$, algorithm BRKGA-LSQClique becomes consistently better for smaller values of γ and considerably outperformed BRKGA-IG* in terms of the time-to-target solution value in 63.33% of the 150 combinations of instances and values of γ.

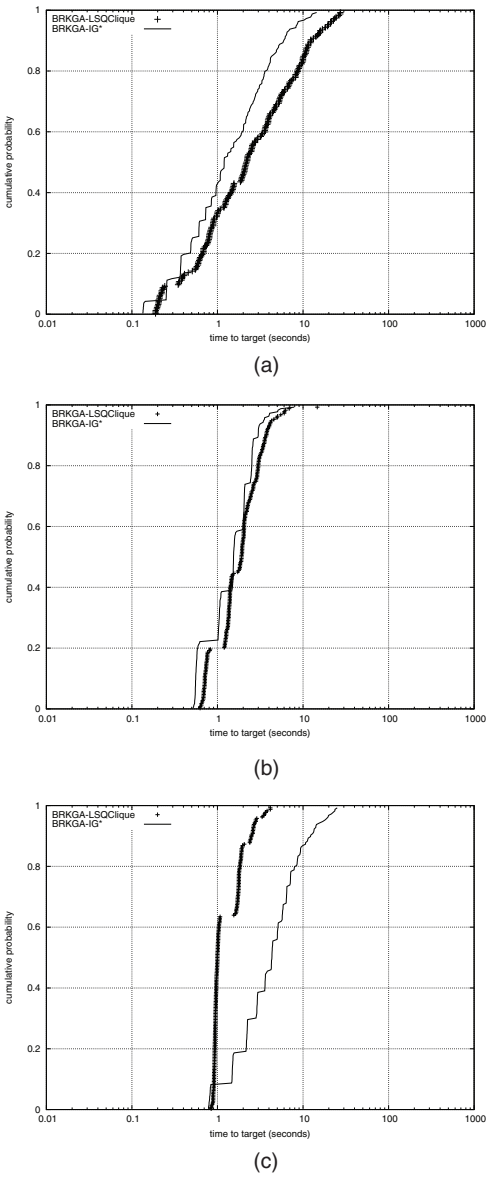
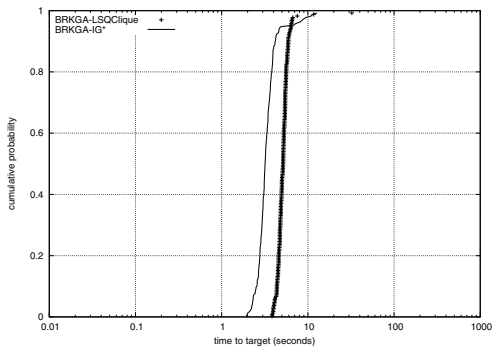
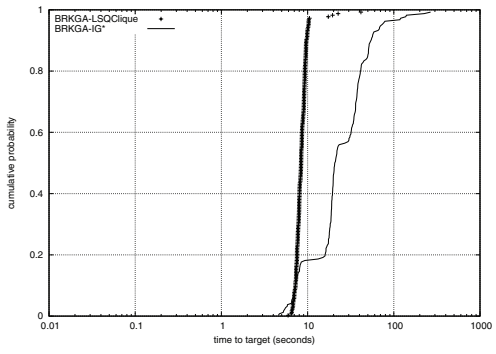


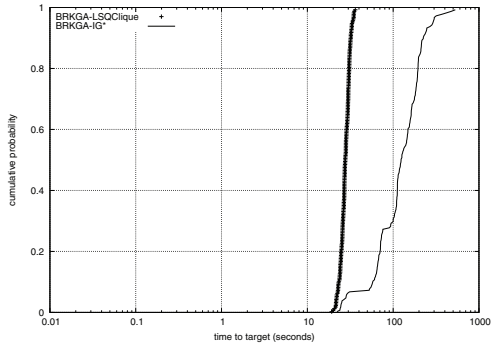
FIGURE 4. Time-to-target plots for instance brock400_2. (a) $\gamma = 0.999$. (b) $\gamma = 0.90$. (c) $\gamma = 0.80$.



(a)



(b)



(c)

FIGURE 5. Time-to-target plots for instance PGPgiantcompo. (a) $\gamma = 0.90$. (b) $\gamma = 0.80$. (c) $\gamma = 0.50$.

Acknowledgements. The authors are thankful to three anonymous referees for their very constructive remarks that substantially contributed to improve the quality and the readability of this article. Work of Celso C. Ribeiro was partially supported by CNPq research grants 303958/2015-4 and 425778/2016-9 and by FAPERJ research grant E-26/202.854/2017. Work of José Angel Riveaux was supported by a CNPq scholarship. Work of Isabel Rosseti was partially supported by CNPq research grant 310624/2018-5. This work was also partially sponsored by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), under Finance Code 001.

References

[1] J. Abello, M. Resende and S. Sudarsky, Massive quasi-clique detection, edited by J. Abello and J. Vitter. In: *Proceedings of the 5th Latin American Symposium on the Theory of Informatics*. Springer-Verlag Berlin Heidelberg (2002) 598–612.

[2] R. Aiex, M. Resende and C.C. Ribeiro, Probability distribution of solution time in GRASP: an experimental investigation. *J. Heuristics* **8** (2002) 343–373.

[3] R. Aiex, M. Resende and C.C. Ribeiro, TTTPLOTS: a Perl program to create time-to-target plots. *Optim. Lett.* **1** (2007) 355–366.

[4] J.C. Bean, Genetic algorithms and random keys for sequencing and optimization. *ORSA J. Comput.* **2** (1994) 154–160.

[5] BHOSLIB, Benchmarks with hidden optimum solutions for graph problems. http://networkrepository.com/ (2004). Online reference, last visited on June 7, 2018.

[6] J.S. Brandão, T.F. Noronha, M.G.C. Resende and C.C. Ribeiro, A biased random-key genetic algorithm for single-round divisible load scheduling. *Int. Trans. Oper. Res.* **22** (2015) 823–839.

[7] J.S. Brandão, T.F. Noronha, M.G.C. Resende and C.C. Ribeiro, A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems. *Int. Trans. Oper. Res.* **27** (2017) 1061–1077.

[8] M. Brunato, H. Hoos and R. Battiti, On effectively finding maximal quasi-cliques in graphs, Learning and Intelligent Optimization edited by V. Maniezzo, R. Battiti and J.-P. Watson. In: Vol. 5313 of *Lecture Notes in Computer Science*. Springer, Berlin (2008) 41–55.

[9] T.A. Davis and Y. Hu, The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.* **38** (2011) 1–25.

[10] FICO, FICO Xpress Optimization Suite 7.6. http://www.fico.com/en/products/fico-xpress-optimization-suite (2017).

[11] J.F. Gonçalves and M.G.C. Resende, Biased random-key genetic algorithms for combinatorial optimization. *J. Heuristics* **17** (2011) 487–525.

[12] J.F. Gonçalves, M.G.C. Resende and R.F. Toso, Biased and unbiased random key genetic algorithms: an experimental analysis. In: *Abstracts of the 10th Metaheuristics International Conference*. Singapore (2013).

[13] H. Hoos and T. Stützle, Evaluation of Las Vegas algorithms – Pitfalls and remedies, edited by G. Cooper and S. Moral. In: *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence. Madison* (1998) 238–245.

[14] D.S. Johnson, Cliques, coloring, and satisfiability: second DIMACS implementation challenge. In: Vol. 26 of *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence (1996).

[15] R.M. Karp, Reducibility among combinatorial problems. In: Complexity of Computer Computations, edited by R.E. Miller and J.W. Thatcher. Plenum, New York (1972) 85–103.

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

[17] T.F. Noronha, M.G.C. Resende and C.C. Ribeiro, A biased random-key genetic algorithm for routing and wavelength assignment. *J. Global Optim.* **50** (2011) 503–518.

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

[19] F.M. Pajouh, Z. Miao and B. Balasundaram, A branch-and-bound approach for maximum quasi-cliques. *Ann. Oper. Res.* **216** (2014) 145–161.

[20] G. Pastukhov, A. Veremyev, V. Boginski and O.A. Prokopyev, On maximum degree-based γ-quasi-clique problem: complexity and exact approaches. *Networks* **71** (2018) 136–152.

[21] J. Pattillo, A. Veremyev, S. Butenko and V. Boginski, On the maximum quasi-clique problem. *Discrete Appl. Math.* **161** (2013) 244–257.

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

[23] B.Q. Pinto, C.C. Ribeiro, I. Rosseti and A. Plastino, A biased random-key genetic algorithm for solving the maximum quasi-clique problem. *Eur. J. Oper. Res.* **271** (2018) 849–865.

[24] M.G.C. Resende and C.C. Ribeiro, Biased-random key genetic algorithms: an advanced tutorial. In: Proceedings of the 2016 Genetic and Evolutionary Computation Conference – GECCO'16 Companion Volume. Association for Computing Machinery, Denver (2016) 483–514.

[25] C.C. Ribeiro and J.A. Riveaux, An exact algorithm for the maximum quasi-clique problem. *Int. Trans. Oper. Res.* **26** (2019) 2199–2229.

[26] R.A. Rossi and N.K. Ahmed, Coloring large complex networks. *Soc. Network Anal. Min.* **4** (2014) 1–37.

[27] R.A. Rossi and N.K. Ahmed, The network data repository with interactive graph analytics and visualization. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (2015) 4292–4293.

[28] E.C. Sewell, An improved algorithm for exact graph coloring, Cliques, Coloring, and Satisfiability, edited by D.S. Johnson and M.A. Trick. In: Vol. 26 of *2nd DIMACS Implementation Challenge. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society (1996) 359–373.

[29] W. Spears and K. de Jong, On the virtues of parameterized uniform crossover, edited by R. Belew and L. Booker. *Proceedings of the Fourth International Conference on Genetic Algorithms. Morgan Kaufman, San Mateo* (1991) 230–236.

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

[31] A. Veremyev, O.A. Prokopyev, S. Butenko, and E.L. Pasiliao, Exact MIP-based approaches for finding maximum quasi-clique and dense subgraph. *Comput. Optim. App.* **64** (2016) 177–214.