RAIRO-Oper. Res. 57 (2023) 1559–1578
https://doi.org/10.1051/ro/2023081

RAIRO Operations Research
www.rairo-ro.org

A BIASED RANDOM-KEY GENETIC ALGORITHM FOR THE CHORDAL COMPLETION PROBLEM

Samuel E. Silva, Celso C. Ribeiro and Uéverton dos Santos Souza*

Abstract. A graph is chordal if all its cycles of length greater than or equal to four contain a chord, *i.e.*, an edge connecting two nonconsecutive vertices of the cycle. Given a graph $G = (V, E)$, the chordal completion problem consists in finding the minimum set of edges to be added to G to obtain a chordal graph. It has applications in sparse linear systems, database management and computer vision programming. In this article, we developed a biased random-key genetic algorithm (BRKGA) for solving the chordal completion problem, based on the strategy of manipulating permutations that represent perfect elimination orderings of triangulations. Computational results show that the proposed heuristic improve the results of the constructive heuristics fill-in and min-degree. We also developed a strategy for injecting externally constructed feasible solutions coded as random keys into the initial population of the BRKGA that significantly improves the solutions obtained and may benefit other implementations of biased random-key genetic algorithms.

Mathematics Subject Classification. 05C85, 90-05.

Received March 21, 2023. Accepted May 30, 2023.

1. Introduction

1.1. Problem description

Let $G = (V, E)$ be a simple graph defined by a vertex set V and an edge set $E \subseteq V \times V$. $V(G)$ defines the vertex set of G. $E(G)$ defines the edge set of G. A *walk* in G is a sequence of vertices where any two consecutive vertices are adjacent, *i.e.*, share a common edge in E. A *path* is a walk where each vertex occurs exactly once. A cycle C in G consists of a sequence of vertices $v_1, v_2, \ldots, v_k$ that form a path, where v_1 and v_k (*i.e.*, the first and last vertices of the sequence) are also adjacent. In this case, $E(C) = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k), (v_k, v_1)\}$ is the edge set of the cycle C. A *chord* of a cycle C is any edge $e \in E \setminus E(C)$ that connects two vertices of C. An *induced* (or *chordless*) cycle in a graph G is a cycle without chords.

A graph G is *chordal* when all its induced cycles have exactly three vertices, *i.e.*, every cycle of length greater than or equal to four of G has at least one chord. The class of chordal graphs is one of the most studied graph classes in algorithmic graph theory, with practical applications in fields such as computer vision [16].

Keywords. chordal completion, Chordal graphs, Biased random-key genetic algorithm, BRKGA, Metaheuristics, Fill-in, Combinatorial optimization.

Universidade Federal Fluminense, Institute of Computing, Niterói, RJ 24210-240, Brazil.
*Corresponding author: ueverton@ic.uff.br

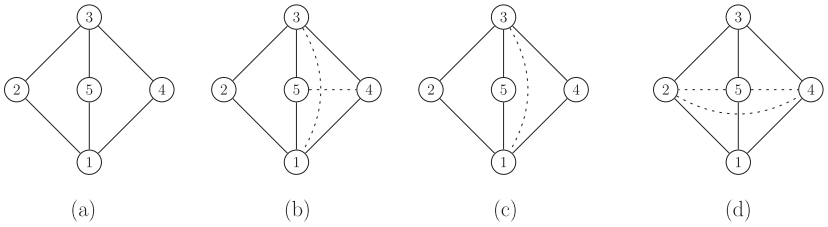

FIGURE 1. Possible minimal and non-minimal triangulations of graph G. (a) Graph G, (b) $E' = \{(1,3),(4,5)\}$, (c) $E' = \{(1,3)\}$ and (d) $E' = \{(2,5),(2,4),(4,5)\}$.

A *triangulation* (or *chordal completion*) of the graph $G = (V, E)$ is a set $E' \subseteq V \times V$ such that the graph $G' = (V, E \cup E')$ is chordal. Figure 1 shows an example. Figure 1(a) illustrates the original graph G. Figure 1(b) displays a minimal triangulation with two edges (1,3) and (4,5). Figure 1(c) shows a minimum triangulation formed by the edge (1,3) that makes the original graph chordal. Finally, Figure 1(d) shows a triangulation formed by three edges (2,5), (2,4), and (4,5) that also make G chordal but not is minimal.

The *chordal completion problem* (CCP) (also known as the *minimum fill-in problem*) consists in finding the minimum cardinality set of edges to be added to a simple graph to make it chordal. In other words, given a graph $G = (V, E)$ and denoting by $\mathcal{C}_\mathcal{G}$ the family of chordal graphs G' such that $V(G) = V(G')$ and $E(G) \subseteq E(G')$, the chordal completion problem problem consists in determining

$$\min_{G' \in \mathcal{C}_\mathcal{G}} |E(G') \setminus E(G)|, \tag{1}$$

or, alternatively,

$$\min_{G' \in \mathcal{C}_\mathcal{G}} |E(G')|. \tag{2}$$

CCP is a classical combinatorial optimization problem. It has many applications, *e.g.*, in optimizing Gaussian elimination on sparse symmetric matrices [27], computational phylogenetics [12], database management [4], sparse matrix computation [26], artificial intelligence [33], semidefinite and nonlinear optimization (in particular for exploiting the sparsity of linear and nonlinear constraint matrices [30,37,48]), and in several other contexts; see the survey in [29]. It was one of the open problems in the first edition of Garey and Johnson's monograph [23] on NP-completeness [28] and was proved to be NP-hard in [49].

1.2. Related work

In [3], it was described an integer programming formulation for CCP. Most of the work in the literature has focused on classes of graphs for which polynomial-time algorithms exist [5,11,14,31]. From the point of view of parameterized complexity, CCP is fixed-parameter tractable when parameterized by the number k of edges that need to be added to make the input graph chordal [13]. The fastest parameterized algorithm for this parameter has a subexponential running time of $2^{o(k)} + O(k^2 nm)$ [20]. The problem also admits a polynomial kernel with respect to this parameter and the current best bound on the kernel size is $O(k^2)$ [38]. The best exact algorithms for CCP are dynamic programming algorithms based on manipulating potential maximal cliques graph [7,20,21].

In [29], it was noticed that the best constructive heuristics for CCP are algorithms based on ordering the vertices by some criterion. Two constructive heuristics using this idea are described in detail in Section 2.

The competition track B of the 2017 Parameterized Algorithms and Computational Experiments Challenge (PACE 2017, see [17]) addressed exact algorithms for CCP. A benchmark set of 200 test instances was proposed for the problem in this challenge. These instances will be used in the computational experiments with the biased random-key genetic algorithm (BRKGA) proposed in this work for CCP.

1.3. Our contributions and organization of this work

In this work, we propose a BRKGA for solving CCP. The computational experiments reported in Section 6 show that our approach outperforms the constructive heuristics in the literature and achieves competitive results when compared to the exact algorithms evaluated in the PACE 2017 challenge. To the best of our knowledge, the BRKGA algorithm developed in this work is the first metaheuristic for CCP.

The remainder of this article is organized as follows. In Section 2, we present some preliminaries and describe constructive heuristics for CCP. Section 3 gives a general description of biased random-key genetic algorithms and presents their customization to CCP. Section 4 describes the decoder of the BRKGA. Section 5 presents the algorithm developed to inject good initial solutions built by the constructive solutions coded as random keys in the initial population of the BRKGA. Computational results are presented in Section 6. Concluding remarks are drawn in the last section.

2. Preliminaries and constructive heuristics

This section presents some preliminaries in Section 2.1, with notations and definitions necessary to understand the heuristics described in Section 2.2.

2.1. Preliminaries

Given a graph $G = (V, E)$, with $|V| = n$, a vertex $v \in V(G)$ is a *simplicial* if its neighborhood (*i.e.*, the subset of vertices of V adjacent to v) induces a clique in G. In 1961, Dirac [18] proved that any chordal graph is complete or has at least two non-adjacent simplicial vertices. In 1965, Fulkerson and Gross [22] used this result to develop a polynomial-time algorithm for recognizing chordal graphs. Their algorithm finds and removes simplicial vertices of the input graph until they are all removed. If the resulting graph is empty, the input graph is chordal; otherwise, it is not. Note that whenever the input graph is chordal, the algorithm finds a sequence of simplicial vertices in each intermediary graph.

A permutation $\rho = v_1, v_2, \ldots, v_n$ of the vertices of graph G is a *perfect elimination ordering* if, for every $i = 1, 2, \ldots, n$, v_i is a simplicial vertex of the subgraph $G[X]$ induced in G by X, where $X = \{v_i, \ldots, v_n\}$. Considering Dirac's characterization of chordal graphs, a graph is chordal if and only if it admits a perfect elimination ordering of its vertex set. The algorithm of Fulkerson and Gross [22] guarantees that the input graph G is chordal if and only if it finds a perfect elimination ordering of G.

According to the strategy established by Fulkerson and Gross [22], the following process based on perfect elimination orderings can transform any graph G into a chordal one by edge additions. First, create a copy of G and name it G'. Then, successively select at random a vertex v of the current graph G', add edges to G' until the neighborhood of v becomes a clique (*i.e.*, vertex v becomes simplicial), and remove x from the current graph G'. When G' becomes an empty graph, the set E' of edges added during this process is a chordal completion of G, and the permutation associated with the order of the vertex removals is a perfect elimination ordering of the chordal graph G^* obtained from G by adding the edges in E'. The following result holds:

Remark 1. Given a permutation ρ of the vertices of G, there is a unique supergraph G_ρ of G such that:

(1) $V(G) = V(G_\rho)$;
(2) $E(G) \subseteq E(G_\rho)$;
(3) ρ is a perfect elimination ordering of G_ρ (thus, G_ρ is chordal); and
(4) $E(G_\rho)$ is an inclusion-wise minimal set of edges satisfying (2) and (3) above.

2.2. Constructive heuristics

Let $E(G_\rho) \setminus E(G)$ be the minimal chordal completion of G associated with permutation ρ.

The vertex selection order described in Section 2.1 generates a perfect elimination ordering ρ, and the minimal chordal completion of G according to ρ coincides precisely with the set of edges arising from such a process.

Algorithm 1 describes a framework for implementing this process, which takes the graph G as input. Line 1 creates a copy G' of the input graph. Line 2 initializes the empty ordered list ρ that will store the permutation generated by the algorithm, *i.e.*, a perfect elimination ordering of the supergraph G_ρ of G. Line 3 initializes an empty edge set E' that will be added to G', forming a chordal completion of G. The loop from lines 4 to 14 successively removes vertices from G' until $V(G')$ becomes empty. Each loop iteration starts in line 5 by selecting a vertex $x \in V(G')$ following some specific criterion. Line 6 inserts vertex x in the queue ρ. The loop from lines 7 to 12 iterates over all pairs of nodes $a, b \in V(G')$ adjacent to x in G'. Line 8 checks if there is an edge ab in G'. If it does not exist, edge ab is added to G' in line 9. Otherwise, edge ab is added to E' in line 10. Line vertex x is removed from G' in line 13. The algorithm returns the edge set E' and the permutation ρ in line 15.

Algorithm 1. Chordal completion heuristic framework.

Require: $G = (V, E)$
1: $G' \leftarrow G$
2: $\rho \leftarrow \emptyset$
3: $E' \leftarrow \emptyset$
4: **while** $V(G') \neq \emptyset$ **do**
5: Select a vertex x from $E(G')$ using some specific criterion
6: Insert x in the end of ρ
7: **for all** pair a, b of vertices of G' which are both neighbors of x **do**
8: **if** edge ab does not exist in $E(G')$ **then**
9: $E' \leftarrow E' \cup \{ab\}$
10: **end if**
11: **end for**
12: Remove vertex x from G'
13: **end while**
14: **return** E', ρ

One can see CCP as finding a vertex permutation (*i.e.*, a perfect elimination ordering) ρ such that the chordal graph G_ρ obtained by the chordal completion of G defined ρ has the minimum number of edges.

The classical constructive heuristics in the literature [29] differ by the criterion applied in line 5 of each iteration of the Algorithm 1 to select the next vertex that will be added to the permutation ρ:

- Heuristic *min-degree*: select the vertex with minimum degree.
- Heuristic *fill-in*: let the neighborhood $N_G(v)$ of vertex v be the set of vertices adjacent to it in G. Let A_v the subset of edges of G whose extremities belong to $N_G(v)$. Denote by $\mathcal{F}_v = |N_G(v)| \cdot (|N_G(v)| - 1)/2 - |A_v|$ the number of edges that must be added between non-connected vertices of $N_G(v)$ so that this neighborhood becomes a clique. Next, select the vertex given by $\text{argmin}_{v \in V(G)} \mathcal{F}_v$, which is the one whose neighborhood is the closest to being a clique.

Figures 2(b) to 2(i) illustrate the step-by-step application of heuristic fill-in to the graph in Figure 2(a). Red vertices are those removed at each iteration of the loop in lines 4–14. Red edges indicate the neighborhood of the vertex that will be removed at each iteration. Red dotted lines indicate the edges created to complete each neighborhood.

Figure 3(a) shows that the chordal completion obtained problem by the fill-in heuristic for the graph in Figure 2(a) has three edges, although the optimal solution has only two edges, as shown in Figure 3(b).

Other constructive heuristics based on perfect elimination orderings appeared in [4, 36, 43]. The following section presents a BRKGA for CCP, whose decoder is based on the min-degree and fill-in heuristics.

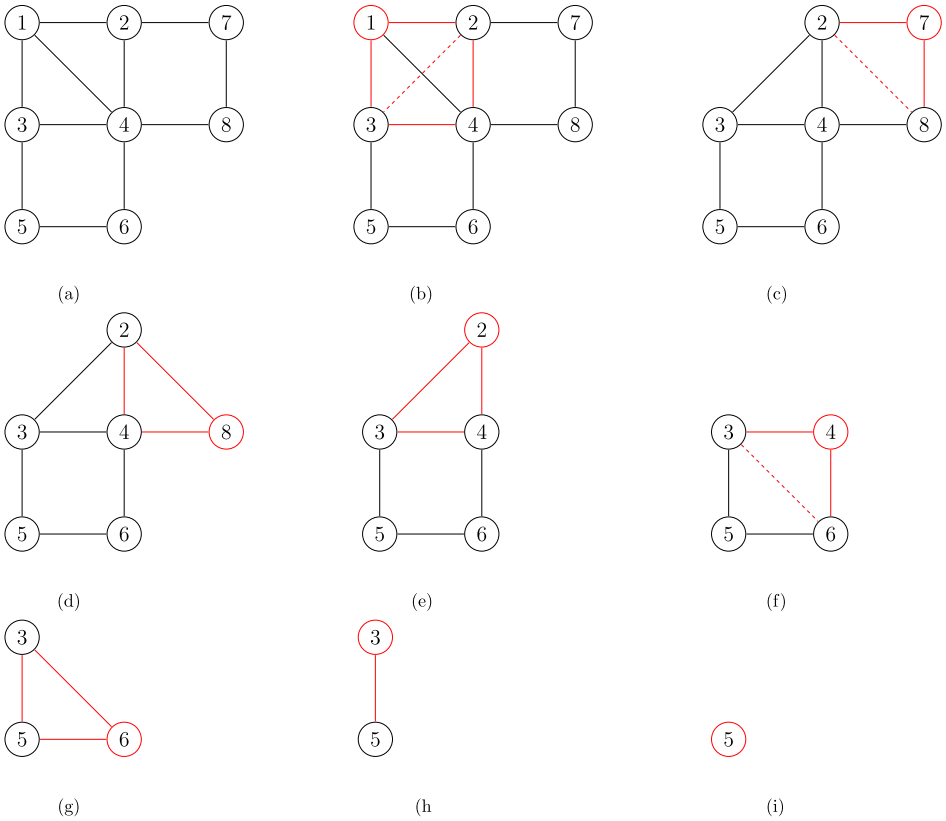

FIGURE 2. Step-by-step application of the fill-in heuristic to obtain permutation $\rho = \langle 1, 7, 8, 2, 4, 6, 3, 5 \rangle$. (a) Graph $G = (V, E)$, (b) 1st step: vertex 1, (c) 2nd step: vertex 7, (d) 3rd step: vertex 8, (e) 4th step: vertex 2, (f) 5th step: vertex 4, (g) 6th step: vertex 6, (h) 7th step: vertex 3 and (i) 8th step: vertex 5.

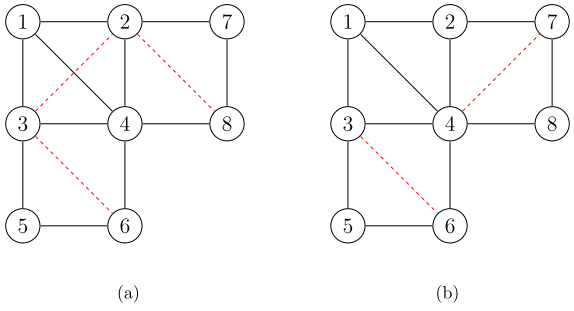

FIGURE 3. On the left, the chordal completion founded by the fill-in heuristic with $E' = \{(2, 3), (2, 8), (3, 6)\}$. On the right, the optimal solution with $E' = \{(4, 7), (3, 6)\}$. (a) Fill-in heuristic and (b) optimal solution.

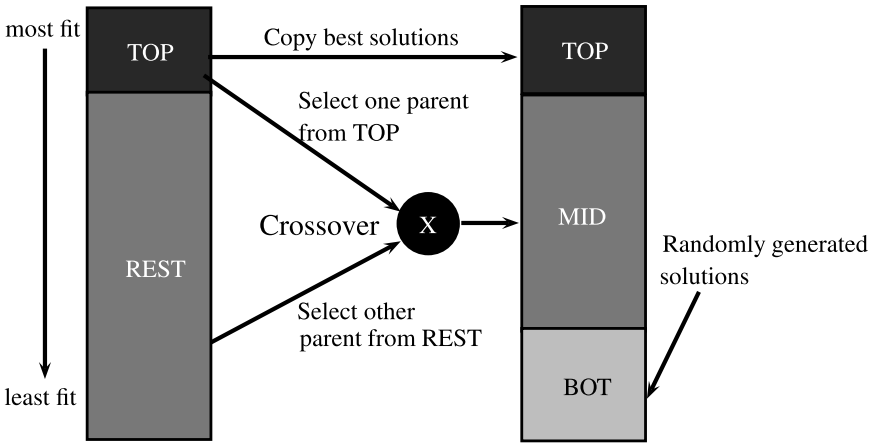

FIGURE 4. Population evolution between consecutive generations of a BRKGA.

3. Biased random-key genetic algorithms for chordal completion

Genetic algorithms with random keys, or random-key genetic algorithms (denoted by RKGA), were first introduced in [2] for combinatorial optimization problems whose solutions may be represented by permutation vectors. Solutions are represented as randomly generated vectors of real numbers in the interval $[0, 1)$ called keys. A deterministic algorithm (a decoder) takes a vector of random keys as input and associates a feasible solution to the combinatorial optimization problem at hand, for which an objective value or fitness can be computed. Two parents are selected randomly from the entire population to implement the crossover operation in an RKGA. Parents can be selected for mating more than once in the same generation.

A BRKGA differs from an RKGA in how parents are selected for a crossover; see [24] for a review. In a BRKGA, each element is generated by combining one selected randomly from the current population's elite solutions 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.

We used the parameterized uniform crossover scheme proposed in [46] to combine two parent solutions and to produce offspring. The offspring inherits each of its keys from the best fit of the two parents with a higher probability. The BRKGA developed in this work does not make use of the standard mutation operator, where parts of the chromosomes are changed with a 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 [8, 9, 41].

The keys in the chromosome are generated randomly in the initial population. Each generation's 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 formed by the worst elements in the current population. As illustrated in Figure 4, the chromosomes in TOP are copied simply to the next generation's population. The elements in BOT are replaced by newly created mutants placed in the new set BOT. The remaining elements of the new population are obtained by crossover, with one parent chosen randomly from TOP and the other from $REST$. This process distinguishes a BRKGA from the RKGA in [2], where both parents are selected randomly 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.

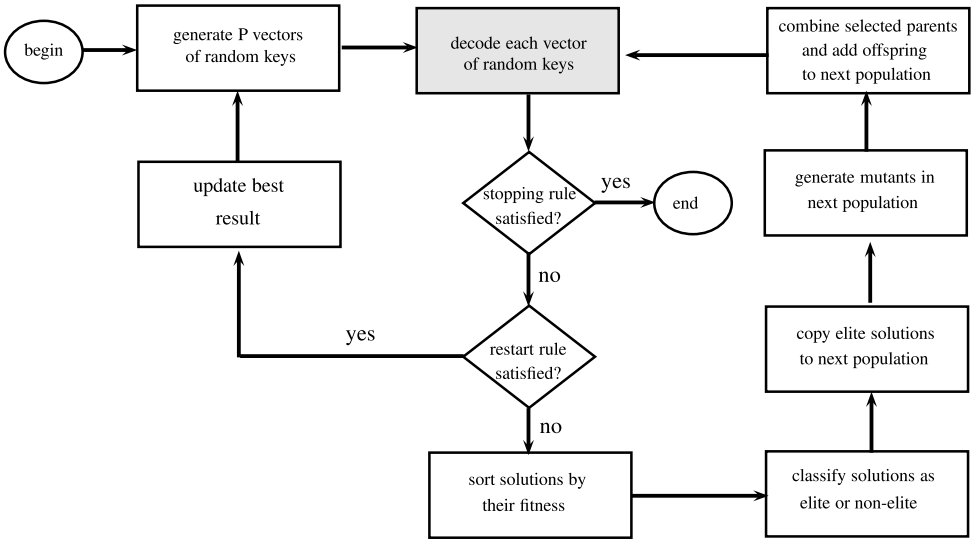

FIGURE 5. BRKGA framework.

According to [25], 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 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.

4. DECODER FOR CHORDAL COMPLETION

Implementing our BRKGA for CCP used the C++ library brkgaAPI framework developed in [47]. The instantiation of the framework shown in Figure 5 to some specific optimization problem requires exclusively the development of a class implementing the decoder for this problem. This component is the only problem-dependent part of the tool. Other applications of this framework in implementing BRKGAs can be seen, *e.g.*, in [1, 8–10, 15, 25, 40].

Algorithm 2 shows the decoder for CCP. It takes the graph G and the vector of random keys R associated with the vertices of G as input. The edges that will be added to the graph (*i.e.*, a chordal completion) are initialized with $E' = \emptyset$ in line 1. A copy G' of the graph G is created in line 2. Line 3 creates a candidate list formed by all vertices yet to be eliminated from the graph G'. The outer loop in lines 4–14 is performed until $V(G')$ is not empty. Each iteration of this loop starts in line 5. The unselected vertex in G' with the smallest random key is saved in x. The inner loop in lines 6–11 iterates over all pairs of neighbors a and b of vertex x in the current graph G'. Line 7 checks whether vertices a and b are adjacent in G'. If they are not, then edge ab is added to the current graph G' in line 8. Line 9 adds edge ab to the chordal completion under construction E'. Therefore, after completing the loop in lines 6–11, the neighborhood of x in G' is transformed into a clique. In line 12, vertex x is removed from G' and a new iteration of the outer loop starts, Line 14 returns the chordal completion corresponding to set E'.

5. INJECTING HIGH-QUALITY SOLUTIONS INTO THE INITIAL POPULATION

Computational results reported in the literature show that genetic algorithms may benefit from the quality of the solutions in the initial population, provided that enough diversity is warranted. For example, high-quality

Algorithm 2. Decoder.

Require: $G = (V, E), R$
1: $E' \leftarrow \emptyset$
2: $G' \leftarrow G$
3: $CL \leftarrow \{1, \ldots, |V|\}$
4: **while** $CL \neq \emptyset$ **do**
5: $x \leftarrow \text{argmin}\{R_i : i \in CL\}$
6: **for all** pairs a, b of neighbors of x in G' **do**
7: **if** a and b are not adjacent in G' **then**
8: Add an edge between a and b in G'
9: $E' \leftarrow E' \cup \{ab\}$
10: **end if**
11: **end for**
12: Remove vertex x and the edges adjacents to it from G'
13: $CL \leftarrow CT \setminus \{x\}$
14: **end while**
15: **return** E'

initial solutions can be explored using a greedy randomized algorithm to create the initial population instead of generating completely random solutions. Another alternative is injecting a few high-quality solutions obtained by specific construction procedures into the initial random population. These strategies have yet to be used in the context of BRKGAs.

This section shows how this idea can be implemented and explored in the context of a BRKGA for CCP. We describe how a permutation vector generated by any constructive heuristic can be coded by random keys and injected into the initial population of the BRKGA.

Algorithm 3 shows the procedure to generate a vector of random keys representing the chordal completion associated with some permutation vector ρ taken as input. The loop in lines 1–2 initializes the vector of random keys $X(1), \ldots, X(V(G))$ that will be associated with the vertices of G. In line 4, the elements of vector R are sorted in ascending order. The loop in lines 5–8 visits the vertices of the permutation vector ρ from first to last. Line 6 picks the vertex v in position i of the permutation vector ρ, and line 7 sets the ith smallest random key to vertices v of ρ. Finally, in line 9, the vector of random keys R associated with the permutation ρ is returned.

Algorithm 3. Creation of the vector of random-keys R associated with a permutation vector ρ.

Require: ρ
1: **for** $i = 1, \ldots, |V(G)|$ **do**
2: Randomly generate $X(i) \in [0, 1)$
3: **end for**
4: Sort X in ascending order.
5: **for** $i = 1, \ldots, |V(G)|$ **do**
6: Let v the vertex in position i of the permutation vector ρ
7: Set $R_v \leftarrow X(i)$
8: **end for**
9: **return** R

The following example illustrates the application of Algorithm 3. Let the input permutation vector be $\rho = \langle 5, 3, 1, 4, 2 \rangle$. Lines 1–3 generate the vector with elements $X(1) = 0.4$, $X(2) = 0.9$, $X(3) = 0.2$, $X(4) = 0.7$, and $X(5) = 0.3$, which become $X(1) = 0.2$, $X(2) = 0.3$, $X(3) = 0.4$, $X(4) = 0.7$, and $X(5) = 0.9$ in ascending order after line 4. Vertex 5 is the first in the permutation vector ρ, then R_5 is set to $X(1) = 0.2$. Vertex 3 is the second in the permutation vector ρ, then R_3 is set to $X(2) = 0.3$. Vertex 1 is the third in the permutation vector ρ, then R_1 is set to $X(3) = 0.4$ Following, we obtain $R_4 = X(4) = 0.7$, and $R_2 = X(5) = 0.9$. As illustrated

TABLE 1. Illustration of Algorithm 3 to create a vector of random-keys R associated with some permutation vector ρ.

	1	2	3	4	5
X	0.4	0.9	0.2	0.7	0.3
Sorted X	0.2	0.3	0.4	0.7	0.9
ρ	5	3	1	4	2
R	0.4	0.9	0.3	0.7	0.2

in Table 1, at the end, the vector of random-keys $R = (0.4, 0.9, 0.3, 0.7, 0.2)$ encodes the original permutation $\rho = \langle 5, 3, 1, 4, 2\rangle$.

The numerical results in Section 6.3 illustrate how the constructive heuristics min-degree and fill-in can be combined with Algorithm 3 to generate high-quality solutions that can be injected in the initial population of the BRKGA and how they contribute to improving the quality of the best solutions found.

6. Computational experiments

In this section, we evaluate the effectiveness of a BRKGA heuristic for CCP.

The computational experiments were performed on an Intel Core i7-4770 CPU with a 3.40 GHz clock and 16 GB of RAM, running under the Ubuntu 18.04 kernel 4.15.0-112 generic operating system. The BRKGA and the constructive heuristics were implemented in *C++*. The BRKGA uses the Mersenne Twister pseudo-random number generator [35]

6.1. Test instances

The benchmark set used in this work is the same as the PACE 2017 Challenge [17], with the test instances formed by graphs collected from several sources. In light of the application of optimizing Gaussian elimination, several instances were taken from the Matrix Market [6] and the Network Repository [44]. In the context of phylogenetic tree reconstruction, 39 instances come from alignments of mammalian DNA markers obtained from the OrthoMaM database [19, 42]. More challenging instances were created from sampled connected subgraphs of the real-world instances. The 200 graphs of the PACE 2017 Challenge were divided into 100 public instances (on which the participants could test their implementations) and 100 hidden instances (used to rank the submissions).

The computational experiments reported in this section were performed on the complete set of 200 benchmark instances. All the input data for the test instances used in the experiments reported in this work are available in Mendeley [45], together with the detailed numerical results.

6.2. Parameter tuning

The irace automatic tuning tool [34, 39] was used to configure the parameters of the BRKGA.

We sought the appropriate values for the 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 BRKGA, as well as the number k of consecutive generations without improvement on the best solution found until a restart is performed.

The irace tuning experiment was performed on 20 instances of different sizes of the PACE 2017 challenge listed in Table 2, choosing one of the listed instances in each run.

The ranges suggested for the parameter values in [24] are listed in Table 3, side-by-side with the values determined in the tuning experiment.

TABLE 2. Test instances used in the irace tuning experiment.

Instance	$\|V\|$	$\|E\|$	Instance	$\|V\|$	$\|E\|$
13	119	161	65	276	749
18	150	259	67	276	346
20	50	59	74	30	307
24	75	158	84	76	83
29	151	316	86	76	132
31	100	207	92	132	255
36	175	199	96	247	804
38	176	378	97	298	780
43	27	69	98	213	380
50	30	349	183	293	14 074

TABLE 3. Value ranges and parameter values selected by the irace tuning experiment.

Parameter	Value ranges	Selected values
p	100, 150, 200	200
pe	[0.10, 0.25]	0.19
pm	[0.10, 0.30]	0.10
$rhoe$	[0.50, 0.80]	0.68
k	50, 100	100

6.3. Initial population with injected individuals by constructive heuristics

We developed in Section 5 an original strategy for injecting externally generated high-quality solutions generated by constructive heuristics, encoded as random keys, into the initial population of the BRKGA.

In this section, we evaluate the effect of this strategy on the quality of the best solutions obtained by the BRKGA. We compare two versions of the BRKGA. The first version is the standard version of the algorithm, with the initial population entirely randomly generated. The second version is characterized by an initial population that contains two externally injected solutions, one generated by the constructive heuristic min-degree and the other by the fill-in heuristic.

For this experiment, we ran each version of the algorithm five times on ten instances of the PACE 2017 challenge with a time limit of 30 min each. Table 4 displays the numerical results. For each instance and each algorithm version, it displays the best and the average best solution values obtained over the five runs. The best values for each instance are underlined. The table shows that, for all instances, the BRKGA with the injection of externally constructed solutions obtains better results. The best solution values were improved by 6.38% on average with the injection of initial solutions. Therefore, this algorithm version will be used in the forthcoming sections and experiments. Other implementations of BRKGAs could further explore and benefit from this strategy.

6.4. Experiments with the BRKGA

The Parameterized Algorithms and Computational Experiments Challenge (PACE) was conceived to help deepen the relationship between parameterized algorithms and practice. The PACE 2017 challenge [17] addressed exact algorithms for CCP. The participants were challenged to solve the maximum number of test instances within the allowed time limit of 30 min. per instance. The winners were selected based on the number of solved hidden instances. The participants were prohibited from using SAT solvers, MIP solvers, or similar general solvers.

TABLE 4. Comparative results obtained by BRKGA with and without the injection of externally constructed high-quality solutions (best values for each instance are underlined).

	Without injection		With injection	
Instance	Best value	Avg. best value	Best value	Avg. best value
99	403	408.0	388	389.2
100	382	389.4	371	372.6
144	506	516.0	493	494.2
179	1970	2068.8	1669	1697.0
182	403	426.8	381	381.0
184	1131	1194.0	1070	1075.2
186	716	733.4	663	664.2
187	433	448.2	419	419.8
190	1225	1274.0	1076	1078.0
192	3282	3366.6	3096	3111.6
Avg. improvement			6.38%	9.05

TABLE 5. Comparative results between BRKGA and the exact algorithm from [32] in terms of feasibility and optimality on 200 instances of the PACE 2017 Challenge after 30 min. of running time.

(200 instances, 1000 runs)	Feasible solution	Optimal solution
BRKGA (all 5×200 runs)	1000/1000 (100%)	358/1000 (35.8%)
BRKGA (best of five runs)	200/200 (100%)	78/200 (39%)
Exact algorithm	116/200 (58%)	116/200 (58%)

To the best of our knowledge, there are no metaheuristics published in the literature for CCP. Therefore, in this section, we compare the solutions of the BRKGA proposed in Sections 3–4 using the injection strategy discussed in Section 5 with the construction heuristics fill-in and min-degree introduced in Section 2.2 and the winning algorithm of PACE 2017, see [32].

In these experiments, the solutions generated by the constructive heuristics fill-in and min-degree coded as random keys are injected into the initial population of the BRKGA. The latter was run five times, with different initial seeds for the pseudo-random number generator, for each of the 200 instances of the PACE 2017 challenge, each with a time limit of 30 min. The exact algorithm from [32] running with the same time limit of 30 min. on the 200 instances found the optimal solution for 116 instances. However, it failed to find a feasible solution for the remaining 84 instances. All algorithms were executed on the same computer.

The BRKGA found a feasible solution for every one of the five runs for all 200 instances. Concerning the 116 instances where the exact algorithm found the optimum, the BRKGA improved the initial solutions generated by the fill-in and min-degree constructive heuristics for 48 instances. The BRKGA managed to find the optimal solution in 358 of the 1000 runs and at least one of the five runs for 78 instances. Table 5 summarizes these results.

For the 38 (19%) instances where the BRKGA did not find the optimal solution in any of the five runs, but the exact method did, a comparative study was carried out regarding the percent deviation between the optimal value and the best value obtained over the five runs. Let opt be the optimal value and $best$ be the best value obtained by the BRKGA for some instance. $100 \times (best - opt)/opt$ gives the percent deviation from the optimal value. The plot in Figure 6 displays the fraction of instances with relative errors in each interval [0%, 5%], (5%, 10%], and (10%, 20%]. For all instances, the BRKGA obtained solutions with a relative error smaller than or

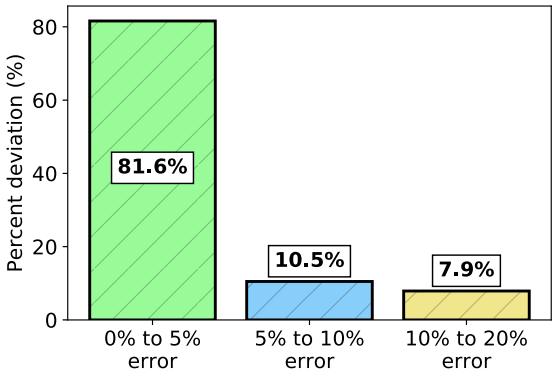

FIGURE 6. Error in percent in the instances not solved to optimality by the BRKGA.

equal to 20%. The relative error was smaller than or equal to 5% in 81.6% (*i.e.*, 31 out of 38) of the instances for which the optimum was not found.

We now compare the BRKGA with the existing constructive heuristics in the literature: min-degree and fill-in. Table 6 details the results for the 48 instances for which the exact algorithm from [32] did not find a feasible solution, and the BRKGA improved the solution obtained by the constructive heuristics. In addition to the data characterizing each instance, it gives the solution values obtained by the heuristics min-degree and fill-in, and the best solution value obtained by the BRKGA. It also gives the relative improvement in percent provided by the best solution value obtained by the BRKGA concerning the best solution among the solutions obtained by the constructive heuristics fill-in and min-degree. For each instance, let $best_{\text{BRKGA}}$ be the value of the best solution value obtained by BRKGA, and $best_{heu}$ be the minimum between the solution values obtained by the heuristics min-degree and fill-in. The relative improvement obtained by the BRKGA is given by $100 \times (best_{heu} - best_{\text{BRKGA}})/best_{heu}$. Table A.1 in A the results for all algorithms on the 200 instances used in the computational experiments.

To conclude, we illustrate the convergence of the BRKGA toward the best solution. Figure 7 shows the best solution value along the first 1000 generations for five BRKGA runs (using different seeds) on instance 179. Figure 8 shows the same experiment on instance 144. In both cases, the solution obtained by the BRKGA is compared with that obtained by the fill-in heuristic.

7. Concluding remarks

In this work, we proposed a BRKGA for finding approximate solutions to CCP.

The algorithm obtained competitive solutions with the best exact algorithm of the PACE 2017 challenge and obtained good feasible solutions for all 84 (42%) instances where the exact algorithm failed.

The computational experiments have shown that the BRKGA found optimal solutions for 78 (39%) instances. The BRKGA improved the quality of the solution in relation to the constructive heuristics that are injected into the initial population by 1.42% on average.

A very interesting conclusion from the computational experiments was that injecting high quality initial solutions coded as random keys into the initial population can make BRKGA obtain better solutions than starting with a completely random population. This strategy should be explored by other implementations of BRKGAs.

TABLE 6. Comparative results between BRKGA and the constructive heuristics on the instances where the exact algorithm did not return a feasible solution.

Instance	$\|V\|$	$\|E\|$	Min-degree	Fill-in	BRKGA	BRKGA relative improvement (%)
19	300	617	483	451	449	0.44
23	200	661	840	816	807	1.10
29	151	316	595	526	513	2.47
37	176	443	1636	1510	1506	0.26
38	176	378	879	701	690	1.57
46	201	520	2163	2012	2001	0.55
52	226	501	1357	1133	1132	0.09
59	251	2746	1747	1754	1723	1.37
65	276	749	4233	3782	3739	1.14
90	1174	1417	1396	1316	1309	0.53
91	205	421	293	278	276	0.72
93	690	1029	1089	1021	1009	1.18
95	465	1004	584	549	546	0.55
96	247	804	642	599	589	1.67
97	298	780	625	576	573	0.52
99	166	396	476	406	398	1.97
100	152	377	408	389	378	2.83
111	500	1593	4062	4057	4040	0.42
115	150	299	368	348	343	1.44
116	300	887	2493	2084	2082	0.10
140	100	297	300	289	286	1.04
143	150	345	986	922	913	0.98
144	150	433	553	532	493	7.33
145	150	790	751	682	675	1.03
146	150	1203	1341	1181	1160	1.78
156	251	672	3324	2951	2922	0.98
159	301	3612	3142	3060	3049	0.36
161	301	3416	3853	3614	3587	0.75
162	300	1192	2392	2358	2348	0.42
163	300	1964	3205	2918	2873	1.54
165	500	4960	13 557	12 807	12 793	0.11
166	500	4864	18 197	17 752	17 749	0.02
167	500	6352	11 711	11 616	11 555	0.53
174	1202	1772	1126	1009	1006	0.30
176	398	599	581	528	522	1.14
178	238	395	262	238	234	1.68
179	235	3395	2203	1828	1669	8.70
180	200	3177	2583	2505	2499	0.24
182	208	483	502	389	381	2.06
184	370	1767	1341	1095	1070	2.28
186	404	607	713	669	664	0.75
187	278	522	483	423	420	0.71
190	712	1085	1191	1087	1078	0.83
192	166	4455	3202	3183	3081	3.20
193	258	467	501	426	419	1.64
194	279	733	1444	1256	1228	2.23
195	227	1000	1640	1341	1279	4.62
197	889	2914	8847	8219	8198	0.26
Average						1.42

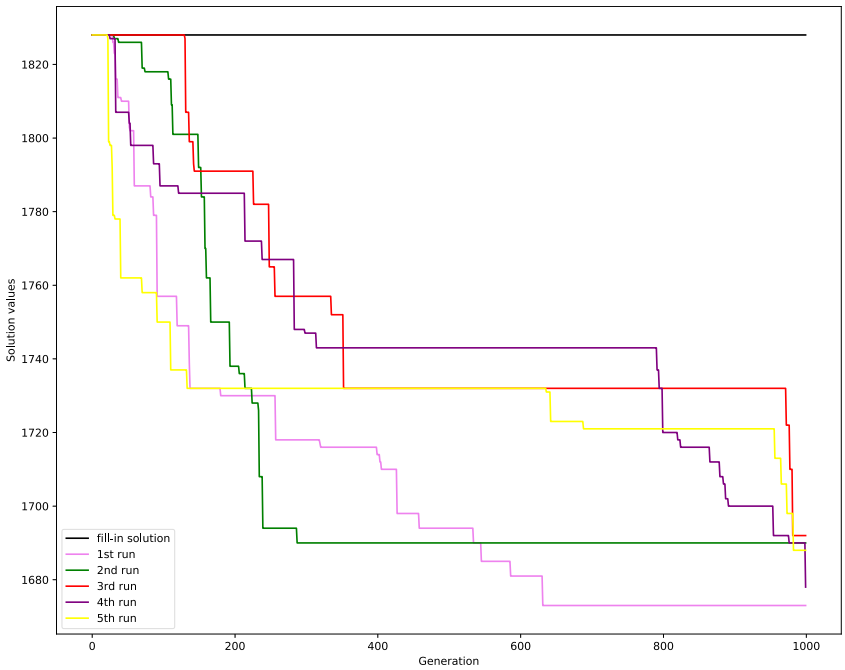

FIGURE 7. Evolution of the best value found by BRKGA along 1000 generations on instance 179.

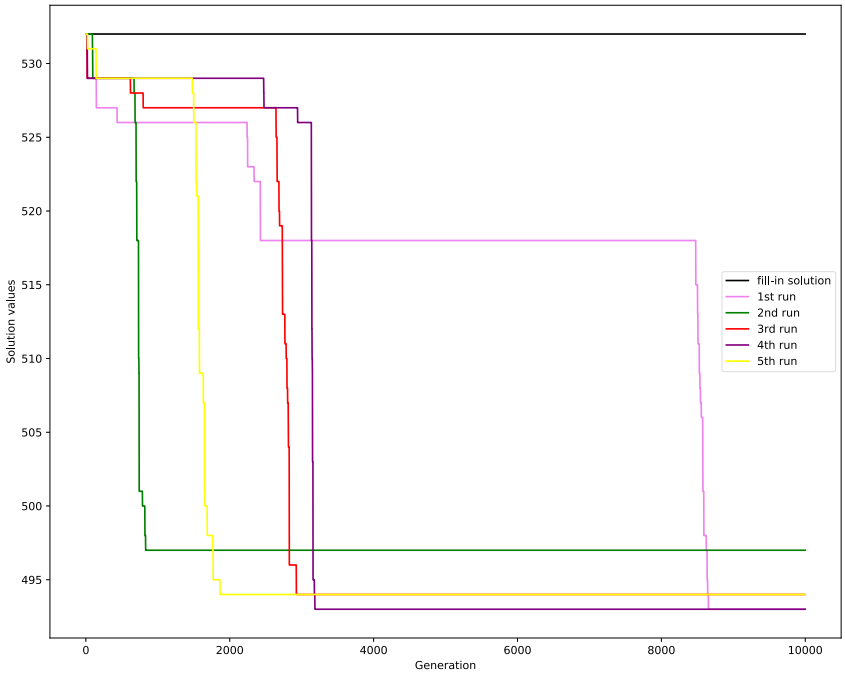

FIGURE 8. Evolution of the best value found by BRKGA along 1000 generations on instance 144.

Appendix A. Detailed computational results

Table A.1. Detailed results for min-degree, fill-in, and BRKGA on the 200 instances of the PACE 2017 challenge.

Instance	$\|V\|$	$\|E\|$	Min-degree	Fill-in	Optimal	Best BRKGA	Average BRKGA
1	2205	5964	86685.00	72850.00	–	72850.00	72850.00
2	129	4943	200.00	210.00	186.00	188.00	190.60
3	101	840	325.00	286.00	286.00	286.00	286.00
4	101	5050	0.00	0.00	0.00	0.00	0.00
5	101	110	28.00	23.00	21.00	21.00	21.00
6	101	1600	0.00	0.00	0.00	0.00	0.00
7	1030	2914	24955.00	20437.00	–	20437.00	20437.00
8	118	4880	138.00	129.00	129.00	129.00	129.00
9	27	254	9.00	9.00	9.00	9.00	9.00
10	151	7504	459.00	405.00	389.00	398.00	400.60
11	126	1095	193.00	189.00	182.00	182.00	182.80
12	126	245	1.00	0.00	0.00	0.00	0.00
13	119	161	95.00	94.00	91.00	91.00	92.40
14	126	1701	0.00	0.00	0.00	0.00	0.00
15	35 588	572 914	5407120.00	–	–	5407120.00	5407120.00
16	187	11 445	617.00	588.00	579.00	579.00	582.20
17	1000	1960	3108.00	3131.00	–	3108.00	3108.00
18	150	259	108.00	106.00	104.00	104.00	104.00
19	300	617	483.00	451.00	–	449.00	449.00
20	50	59	2.00	2.00	2.00	2.00	2.00
21	3391	4600	3540.00	3415.00	–	3415.00	3415.00
22	242	20 176	1258.00	1135.00	1133.00	1133.00	1134.60
23	200	661	840.00	816.00	–	807.00	807.00
24	75	158	72.00	70.00	63.00	63.00	63.00
25	1612	2106	1668.00	1536.00	–	1536.00	1536.00
26	120	4904	254.00	229.00	226.00	229.00	229.00
27	151	2055	0.00	0.00	0.00	0.00	0.00
28	151	1522	131.00	144.00	131.00	134.00	134.00
29	151	316	595.00	526.00	–	513.00	513.00
30	151	2376	885.00	735.00	717.00	730.00	733.00
31	100	207	159.00	160.00	149.00	150.00	150.00
32	500	1653	5711.00	5572.00	–	5572.00	5572.00
33	427	43 819	2672.00	2672.00	–	2672.00	2672.00
34	9532	35 023	289158.00	–	–	289158.00	289158.00
35	8738	291 583	2572961.00	–	–	2572961.00	2572961.00
36	175	199	59.00	53.00	46.00	46.00	47.00
37	176	443	1636.00	1510.00	–	1506.00	1506.00
38	176	378	879.00	701.00	–	690.00	690.00
39	176	15 400	0.00	0.00	0.00	0.00	0.00
40	147	7303	377.00	355.00	347.00	348.00	351.80
41	5489	107 016	452289.00	–	–	452289.00	452289.00
42	66	1460	12.00	12.00	12.00	12.00	12.00
43	27	69	3.00	0.00	0.00	0.00	0.00
44	111	4332	170.00	177.00	167.00	168.00	169.00
45	201	396	11.00	0.00	0.00	0.00	0.00
46	201	520	2163.00	2012.00	–	2001.00	2001.00
47	201	1871	1558.00	1490.00	–	1490.00	1490.00
48	201	7691	0.00	0.00	0.00	0.00	0.00
49	4960	9462	25.00	0.00	0.00	0.00	0.00
50	30	349	4.00	3.00	3.00	3.00	3.00
51	40	587	5.00	5.00	5.00	5.00	5.00

TABLE A.1. Continued.

Instance	$\|V\|$	$\|E\|$	Min-degree	Fill-in	Optimal	Best BRKGA	Average BRKGA
52	226	501	1357.00	1133.00	–	1132.00	1132.00
53	226	3777	0.00	0.00	0.00	0.00	0.00
54	226	3777	0.00	0.00	0.00	0.00	0.00
55	226	3777	0.00	0.00	0.00	0.00	0.00
56	276	20 850	4465.00	4465.00	4465.00	4465.00	4465.00
57	1624	2213	1788.00	1728.00	–	1728.00	1728.00
58	251	2020	11.00	0.00	0.00	0.00	0.00
59	251	2746	1747.00	1754.00	–	1723.00	1723.00
60	251	3481	0.00	0.00	0.00	0.00	0.00
61	251	4213	0.00	0.00	0.00	0.00	0.00
62	85	2516	78.00	78.00	78.00	78.00	78.00
63	17 758	54 196	0.00	0.00	–	0.00	0.00
64	276	7840	0.00	0.00	0.00	0.00	0.00
65	276	749	4233.00	3782.00	–	3739.00	3739.00
66	276	3840	0.00	0.00	0.00	0.00	0.00
67	276	346	226.00	213.00	207.00	210.00	210.20
68	5357	101 169	344553.00	318000.00	–	318000.00	318000.00
69	202	13 637	550.00	553.00	549.00	553.00	553.00
70	301	362	217.00	210.00	199.00	203.00	205.80
71	301	11 593	8133.00	8133.00	8133.00	8133.00	8133.00
72	301	5510	4775.00	4074.00	–	4074.00	4074.00
73	301	4206	0.00	0.00	0.00	0.00	0.00
74	30	307	0.00	0.00	0.00	0.00	0.00
75	40	539	3.00	3.00	3.00	3.00	3.00
76	72	1121	25.00	26.00	25.00	25.00	25.00
77	194	11 145	476.00	467.00	463.00	466.00	466.20
78	197	13 052	733.00	700.00	690.00	700.00	700.00
79	120	4941	157.00	157.00	157.00	157.00	157.00
80	1454	1923	1602.00	1483.00	–	1483.00	1483.00
81	303	24 393	954.00	909.00	–	909.00	909.00
82	5489	137 811	560979.00	–	–	560979.00	560979.00
83	233	16 636	723.00	637.00	552.00	637.00	637.00
84	76	83	19.00	18.00	15.00	15.00	15.00
85	76	2850	0.00	0.00	0.00	0.00	0.00
86	76	132	168.00	147.00	118.00	121.00	124.80
87	76	2228	0.00	0.00	0.00	0.00	0.00
88	5300	8271	15005.00	13715.00	–	13715.00	13715.00
89	138	6338	235.00	239.00	235.00	235.00	235.00
90	1174	1417	1396.00	1316.00	–	1309.00	1309.00
91	205	421	293.00	278.00	–	276.00	276.00
92	132	255	224.00	205.00	–	205.00	205.00
93	690	1029	1089.00	1021.00	–	1009.00	1009.00
94	357	6225	3349.00	3363.00	–	3363.00	3363.00
95	465	1004	584.00	549.00	–	546.00	546.00
96	247	804	642.00	599.00	–	589.00	589.00
97	298	780	625.00	576.00	–	573.00	573.00
98	213	380	301.00	311.00	–	302.00	302.00
99	166	396	476.00	406.00	–	398.00	398.00
100	152	377	408.00	398.00	–	378.00	378.00

TABLE A.1. Continued.

Instance	$\|V\|$	$\|E\|$	Min-degree	Fill-in	Optimal	Best BRKGA	Average BRKGA
101	225	16 819	1018.00	911.00	898.00	911.00	911.00
102	30	327	3.00	3.00	3.00	3.00	3.00
103	40	531	15.00	15.00	15.00	15.00	15.00
104	204	14 118	820.00	766.00	749.00	759.00	760.20
105	159	7429	174.00	172.00	171.00	172.00	172.00
106	30	292	1.00	1.00	1.00	1.00	1.00
107	75	110	30.00	30.00	30.00	30.00	30.00
108	200	363	189.00	186.00	183.00	183.00	183.20
109	3934	7873	27834.00	27759.00	–	27759.00	27759.00
110	100	164	161.00	155.00	145.00	151.00	151.80
111	500	1593	4062.00	4057.00	–	4040.00	4046.40
112	1000	3118	14316.00	13522.00	–	13522.00	13522.00
113	11 823	33 230	327210.00	–	–	327210.00	327210.00
114	50	51	5.00	3.00	3.00	3.00	3.00
115	150	299	368.00	348.00	–	343.00	343.20
116	300	887	2493.00	2084.00	–	2082.00	2082.40
117	180	10 098	537.00	544.00	521.00	542.00	543.20
118	194	13 060	597.00	567.00	559.00	559.00	559.00
119	89	2675	64.00	63.00	63.00	63.00	63.00
120	141	7257	344.00	311.00	307.00	307.00	307.80
121	169	9867	439.00	440.00	423.00	429.00	436.20
122	156	8022	472.00	412.00	387.00	397.00	400.20
123	183	11 540	712.00	679.00	654.00	667.00	668.00
124	50	765	8.00	8.00	8.00	8.00	8.00
125	190	12 473	586.00	584.00	581.00	582.00	582.00
126	187	12 342	632.00	636.00	611.00	632.00	632.80
127	120	4992	290.00	304.00	290.00	290.00	291.00
128	167	9504	354.00	347.00	339.00	341.00	341.00
129	27	280	3.00	3.00	3.00	3.00	3.00
130	138	6820	243.00	243.00	235.00	242.00	242.00
131	78	2014	68.00	63.00	62.00	62.00	62.40
132	258	20 422	953.00	845.00	845.00	845.00	845.00
133	35	460	2.00	2.00	2.00	2.00	2.00
134	167	8969	298.00	308.00	298.00	298.00	299.80
135	66	1463	33.00	32.00	32.00	32.00	32.00
136	76	469	62.00	48.00	47.00	47.00	47.00
137	76	606	131.00	131.00	131.00	131.00	131.00
138	100	109	13.00	13.00	13.00	13.00	13.00
139	100	239	107.00	107.00	107.00	107.00	107.00
140	100	297	300.00	289.00	–	286.00	286.00
141	126	1980	552.00	552.00	552.00	552.00	552.00
142	151	169	25.00	20.00	18.00	18.00	18.00
143	150	345	986.00	922.00	–	913.00	914.40
144	150	433	553.00	532.00	–	493.00	495.80
145	150	790	751.00	682.00	–	675.00	676.40
146	150	1203	1341.00	1181.00	–	1160.00	1163.60
147	176	203	81.00	79.00	78.00	78.00	78.00
148	176	2983	1355.00	1005.00	965.00	1005.00	1005.00
149	176	2959	1457.00	1458.00	1408.00	1458.00	1458.00
150	201	3518	1682.00	1682.00	1570.00	1682.00	1682.00
151	226	2169	3057.00	2712.00	–	2712.00	2712.00
152	226	271	135.00	127.00	122.00	123.00	123.00
153	226	2487	300.00	282.00	144.00	282.00	282.00
154	225	13 604	2278.00	2278.00	2278.00	2278.00	2278.00
155	251	308	113.00	109.00	100.00	106.00	106.40

TABLE A.1. Continued.

Instance	$\|V\|$	$\|E\|$	Min-degree	Fill-in	Optimal	Best BRKGA	Average BRKGA
156	251	672	3324.00	2951.00	–	2922.00	2926.40
157	276	3300	516.00	498.00	360.00	498.00	498.00
158	276	332	199.00	195.00	187.00	190.00	192.00
159	301	3612	3142.00	3060.00	–	3049.00	3051.00
160	301	370	202.00	195.00	194.00	194.00	194.60
161	301	3416	3853.00	3614.00	–	3587.00	3604.60
162	300	1192	2392.00	2358.00	–	2348.00	2349.00
163	300	1964	3205.00	2918.00	–	2873.00	2879.00
164	300	2740	6247.00	6330.00	–	6324.00	6326.40
165	500	4960	13557.00	12807.00	–	12793.00	12799.40
166	500	4864	18197.00	17752.00	–	17749.00	17750.00
167	500	6352	11711.00	11616.00	–	11555.00	11587.00
168	750	8116	29164.00	28927.00	–	28927.00	28927.00
169	750	8816	48806.00	46801.00	–	46801.00	46801.00
170	800	8805	33943.00	31223.00	–	31223.00	31223.00
171	62	739	232.00	221.00	220.00	220.00	220.00
172	128	1644	1190.00	1114.00	1063.00	1106.00	1112.00
173	1202	1772	1126.00	1009.00	–	1006.00	1006.40
174	142	685	788.00	717.00	654.00	692.00	696.20
175	81	893	524.00	512.00	498.00	498.00	498.00
176	398	599	581.00	528.00	–	522.00	522.80
177	126	2243	234.00	178.00	178.00	178.00	178.00
178	238	395	262.00	238.00	–	234.00	235.20
179	235	3395	2203.00	1828.00	–	1669.00	1697.00
180	200	3177	2583.00	2505.00	–	2499.00	2499.00
181	609	6586	2653.00	3323.00	–	3310.00	3319.20
182	208	483	502.00	389.00	–	381.00	381.00
183	293	14 074	184.00	184.00	184.00	184.00	184.00
184	370	1767	1341.00	1095.00	–	1070.00	1075.20
185	177	669	335.00	321.00	319.00	319.00	319.00
186	404	607	713.00	669.00	–	664.00	665.00
187	278	522	483.00	423.00	–	420.00	420.20
188	1854	21 118	9183.00	9443.00	–	9183.00	9183.00
189	281	9075	94.00	85.00	80.00	85.00	85.00
190	712	1085	1191.00	1087.00	–	1078.00	1079.40
191	188	638	185.00	162.00	161.00	161.00	161.00
192	166	4455	3202.00	3183.00	–	3081.00	3097.80
193	258	467	501.00	426.00	–	419.00	419.00
194	279	733	1444.00	1256.00	–	1228.00	1236.40
195	227	1000	1640.00	1341.00	–	1279.00	1288.80
196	34	78	12.00	12.00	12.00	12.00	12.00
197	889	2914	8847.00	8219.00	–	8198.00	8210.00
198	2426	16 630	62945.00	60130.00	–	62945.00	62945.00
199	5167	39 432	510914.00	–	–	510914.00	510914.00
200	12 645	67 053	337456.00	–	–	337456.00	337456.00

Acknowledgements. Work of Celso C. Ribeiro was supported by research grants CNPq 309869/2020-0 and FAPERJ E-26/200.926/2021, and the work of Uéverton S. Souza was supported by research grants CNPq 309832/2020-9 and FAPERJ E-26/210.400/2018 and E-26/201.344/2021.

Data availability statement. The data that support the findings of this study is available from Mendeley Data at https://data.mendeley.com/datasets/mf33kd592n, see [45].

REFERENCES

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

[2] J.C. Bean, Genetic algorithms and random keys for sequencing and optimization. *ORSA J. Comput.* **6** (1994) 154–160.

[3] D. Bergman, C.H. Cardonha, A.A. Cire and A.U. Raghunathan, On the minimum chordal completion polytope. *Oper. Res.* **67** (2019) 532–547.
[4] A. Berry, J.-P. Bordat, P. Heggernes, G. Simonet and Y. Villanger, A wide-range algorithm for minimal triangulation from an arbitrary ordering. *J. Algor.* **58** (2006) 33–66.
[5] H.L. Bodlaender, T. Kloks, D. Kratsch and H. Müller, Treewidth and minimum fill-in on d-trapezoid graphs, in *Graph Algorithms and Applications I*. Edited by R. Tamassia and I.G. Tollis. World Scientific (2002) 139–161.
[6] R. Boisvert, R. Pozo, K. Remington, B. Miller and R. Lipman, Matrix market, Online reference at http://math.nist.gov/MatrixMarket/ [last access on March 21, 2023] (2007).
[7] V. Bouchitté and I. Todinca, Treewidth and minimum fill-in: grouping the minimal separators. *SIAM J. Comput.* **31** (2001) 212–232.
[8] 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.
[9] 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.* **24** (2017) 1061–1077.
[10] J.A. Brito, G. Semaan and A. Fadel, The effective BRKGA algorithm for the k-medoids clustering problem. *RAIRO: OR* **56** (2022) 3137–3153.
[11] H.J. Broersma, E. Dahlhaus and T. Kloks, Algorithms for the treewidth and minimum fill-in of HHD-free graphs, in *Graph-Theoretic Concepts in Computer Science*. Vol. 1335 of *Lecture Notes in Computer Science*. Edited by R.H. Möhring. Springer (1997) 109–117.
[12] P. Buneman, A characterisation of rigid circuit graphs. *Discrete Math.* **9** (1974) 205–212.
[13] L. Cai, Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.* **58** (1996) 171–176.
[14] M.-S. Chang, Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs, in *Algorithms and Computation*. Volume 1178 of *Lecture Notes in Computer Science*. Edited by T. Asano, Y. Igarashi, H. Nagamochi, S. Miyano and S. Suri. Springer (1996) 146–155.
[15] A.A. Chaves, L.A.N. Lorena, E.L.F. Senne and M.G.C. Resende, Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. *Comput. Oper. Res.* **67** (2016) 174–183.
[16] F.R.K. Chung and D. Mumford, Chordal completions of planar graphs. *J. Comb. Theory Ser. B* **62** (1994) 96–106.
[17] H. Dell, C. Komusiewicz, N. Talmon and M. Weller, The PACE 2017 parameterized algorithms and computational experiments challenge: the second iteration, in *12th International Symposium on Parameterized and Exact Computation*. Vol. 89 of *Leibniz International Proceedings in Informatics*. Edited by D. Lokshtanov and N. Nishimura. Dagstuhl (2018) 30:1–30:12 (Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. Online reference at http://drops.dagstuhl.de/opus/volltexte/2018/8558 [last access on March 21, 2023]).
[18] G.A. Dirac, On rigid circuit graphs.*Abh. Math. Semin. Univ. Hambg.* **25** (1961) 71–76.
[19] E.J.P. Douzery, C. Scornavacca, J. Romiguier, K. Belkhir, N. Galtier, F. Delsuc and V. Ranwez, OrthoMaM v8: a database of orthologous exons and coding sequences for comparative genomics in mammals. *Mol. Biol. Evol.* **31** (2014) 1923–1928.
[20] F.V. Fomin and Y. Villanger, Subexponential parameterized algorithm for minimum fill-in. *SIAM J. Comput.* **42** (2013) 2197–2216.
[21] F.V. Fomin, D. Kratsch, I. Todinca and Y. Villanger, Exact algorithms for treewidth and minimum fill-in. *SIAM J. Comput.* **38** (2008) 1058–1079.
[22] D. Fulkerson and O. Gross, Incidence matrices and interval graphs. *Pac. J. Math.* **15** (1965) 835–855.
[23] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Co. (1979).
[24] J.F. Gonçalves and M.G.C. Resende, Biased random-key genetic algorithms for combinatorial optimization. *J. Heuristics* **17** (2011) 487–525.
[25] J.F. Gonçalves, M.G.C. Resende and R.F. Toso, An experimental comparison of biased and unbiased random-key genetic algorithms. *Pesqui. Operacional* **34** (2014) 143–164.
[26] R. Grone, C.R. Johnson, E.M. Sá and H. Wolkowicz, Positive definite completions of partial hermitian matrices. *Linear Algebra Appl.* **58** (1984) 109–124.
[27] R. Gysel, K. Stevens and D. Gusfield, Reducing problems in unrooted tree compatibility to restricted triangulations of intersection graphs, in *Algorithms in Bioinformatics*. Vol. 7534 of *Lecture Notes in Computer Science*. Edited by B. Raphael and J. Tang. Springer (2012) 93–105.
[28] J. Hartmanis, Computers and intractability: a guide to the theory of NP-completeness (Michael R. Garey and David D. Johnson). *SIAM Rev.* **24** (1982) 90–91.
[29] P. Heggernes, Minimal triangulations of graphs: a survey. *Discrete Math.* **306** (2006) 297–317.
[30] S. Kim, M. Kojima, M. Mevissen and M. Yamashita, Exploiting sparsity in linear and nonlinear matrix inequalities via positive semidefinite matrix completion. *Math. Prog.* **129** (2011) 33–68.
[31] T. Kloks, D. Kratsch and C.K. Wong, Minimum fill-in on circle and circular-arc graphs. *J. Algor.* **28** (1998) 272–289.
[32] Y. Kobayashi and H. Tamaki, Track B: Minimum fill-in. Online reference at https://github.com/TCS-Meiji/PACE2017-TrackB, [last access on October 30, 2022] (2017).
[33] S.L. Lauritzen and D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Stat. Soc. Ser. B* **50** (1988) 157–224.

[34] M. López-Ibáñez, J. Dubois-Lacoste, L.P. Cáceres, M. Birattari and T. Stützle, The irace package: iterated racing for automatic algorithm configuration. *Oper. Res. Perspect.* **3** (2016) 43–58.
[35] M. Matsumoto and T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.* **8** (1998) 3–30.
[36] M. Mezzini and M. Moscarini, Simple algorithms for minimal triangulation of a graph and backward selection of a decomposable Markov network. *Theor. Comput. Sci.* **411** (2010) 958–966.
[37] K. Nakata, K. Fujisawa, M. Fukuda, M. Kojima and K. Murota, Exploiting sparsity in semidefinite programming via matrix completion II: implementation and numerical results. *Math. Prog.* **95** (2003) 303–327.
[38] A. Natanzon, R. Shamir and R. Sharan, A polynomial approximation algorithm for the minimum fill-in problem. *SIAM J. Comput.* **30** (2000) 1067–1079.
[39] L. Pérez Cáceres, M. López-Ibáñez and T. Stützle, An analysis of parameters of irace, in *Evolutionary Computation in Combinatorial Optimization*. Vol. 8600 of *Lecture Notes in Computer Science*. Edited by C. Blum and D. Ochoa. Springer (2014) 37–48.
[40] B.Q. Pinto, C.C. Ribeiro, I. Rosseti and A. Plastino, A biased random-key genetic algorithm for the maximum quasi-clique problem. *Eur. J. Oper. Res.* **271** (2018) 849–865.
[41] B.Q. Pinto, C.C. Ribeiro, J.A. Riveaux and I. Rosseti, A BRKGA-based matheuristic for the maximum quasi-clique problem with an exact local search strategy. *RAIRO: OR* **55** (2021) S741–S763.
[42] V. Ranwez, F. Delsuc, S. Ranwez, K. Belkhir, M.-K. Tilak and E.J.P. Douzery, OrthoMaM: a database of orthologous genomic markers for placental mammal phylogenetics. *BMC Evol. Biol.* **7** (2007) 241.
[43] E. Rollon and J. Larrosa, On mini-buckets and the min-fill elimination ordering, in *Principles and Practice of Constraint Programming*, Vol. 6876 of *Lecture Notes in Computer Science*. Edited by J. Lee. Springer (2011) 759–773.
[44] R. Rossi and N. Ahmed, The network data repository with interactive graph analytics and visualization, in *Twenty-Ninth AAAI conference on Artificial Intelligence* (2015) 22–35.
[45] S.E. Silva, Test instances for the chordal completion problem. *Mendeley Data*, V1. Online publication at https://data.mendeley.com/datasets/mf33kd592n [last access on March 21, 2023] (2022).
[46] W. Spears and K.A. De Jong. On the virtues of parameterized uniform crossover, in *Proceedings of the Fourth International Conference on Genetic Algorithms*. Edited by R. Belew and L. Booker. San Mateo (1991) 230–236.
[47] R.F. Toso and M.G.C. Resende, A c++ application programming interface for biased random-key genetic algorithms. *Optimiz. Methods Soft.* **30** (2015) 81–93.
[48] L. Vandenberghe and M.S. Andersen, Chordal graphs and semidefinite optimization. *Found. Trends Optimiz.* **1** (2015) 241–433.
[49] M. Yannakakis, Computing the minimum fill-in is NP-complete. *SIAM J. Algebraic Discrete Methods* **2** (1981) 77–79.