European Journal of Operational Research 289 (2021) 17–30

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

The Multi-Parent Biased Random-Key Genetic Algorithm with Implicit Path-Relinking and its real-world applications

Carlos E. Andrade[a,*], Rodrigo F. Toso[b], José F. Gonçalves[c,d], Mauricio G. C. Resende[d,e]

[a] AT&T Labs Research, 200 South Laurel Avenue, Middletown, NJ 07748, USA
[b] Microsoft AI & Research, 555 110th Ave NE, Bellevue, WA 98004, USA
[c] Universidade do Porto, Rua Dr. Roberto Frias, s/n, Porto 4200-464, Portugal
[d] Amazon.com, Inc., 399 Fairview Avenue North, Seattle, WA 98109, USA
[e] University of Washington, 3900 E Stevens Way NE, Seattle, WA 98195, USA

ARTICLE INFO

Article history:
Received 3 July 2019
Accepted 18 November 2019
Available online 22 November 2019

Keywords:
Genetic algorithms
Path-relinking algorithms
Hybrid heuristics

ABSTRACT

In this paper, we present the Multi-Parent Biased Random-Key Genetic Algorithm with Implicit Path-Relinking (BRKGA-MP-IPR), a variant of the Biased Random-Key Genetic Algorithm that employs multiple (biased) parents to generate offspring instead of the usual two, and is hybridized with a novel, implicit path-relinking local search procedure. By operating over the standard unit hypercube, such path-relinking mechanism leverages the population representation of the BRKGA and thus provides complete independence between the local search procedure and the problem definition and implementation. This approach contrasts with traditional path-relinking procedures that are tied to the problem structure. Having both BRKGA and IPR operate over the same solution space not only makes the intensification/diversification paradigm more natural but also greatly simplifies the development effort from the perspective of the practitioner, as one only needs to develop a decoder to map unit random-key vectors to the solution space of the problem on hand. Apart from such key benefits, extensive computational experiments solving real-world problems, such as over-the-air software upgrade scheduling, network design problems, and combinatorial auctions, show that the BRKGA-MP-IPR offers performance benefits over the standard BRKGA as well as the BRKGA with multiple parents.

1. Introduction

Genetic algorithms (Holland, 1975) have become a popular choice to solve hard, large-scale, real-world optimization problems. This metaheuristic based on the principles of natural selection/survival of the fittest offers an efficient mechanism for searching a solution space that is exponentially large. As a result, genetic algorithms are extremely popular in several domains.

Among several variants of GAs (Kramer, 2017), the Biased Random-Key Genetic Algorithm (BRKGA) has been used with success both in several classical hard combinatorial optimization problems (see e.g. Resende, Toso, Gonçalves, & Silva, 2011; Rochman, Prasetyo, & Nugroho, 2017) as well as on real-world problems such as packing (Gonçalves & Resende, 2011b), scheduling (Andrade et al., 2017c; 2019a; Andrade, Silva, & Pessoa, 2019b; Pessoa & Andrade, 2018), combinatorial auctions (Andrade, Toso, Resende, & Miyazawa, 2015b), vehicle routing (Andrade, Miyazawa, & Resende, 2013; Lopes, Andrade, Queiroz, Resende, & Miyazawa, 2016), clustering (Andrade, Resende, Karloff, & Miyazawa, 2014), complex network design (Andrade et al., 2015a), placement of virtual machines in data centers (Stefanello, Aggarwal, Buriol, Gonçalves, & Resende, 2015), and machine learning (Caserta & Reiners, 2016). BRKGA has also been used to check feasibility for instances from mixed integer programming problems for which feasible solutions are challenging to find (Andrade, Ahmed, Nemhauser, & Shao, 2017a). Elements of BRKGA have appeared in (Beirão, 1997; Ericsson, Resende, & Pardalos, 2002; Gonçalves & Beirão, 1999; Gonçalves & de Almeida, 2002) However, the BRKGA methodology was formally introduced in Gonçalves and Resende (2011a), which defined a clear framework and presented the first results as such.

The main advantage of the BRKGA over other genetic algorithm variants is perhaps the fast convergence to high-quality solutions. This is mainly supported by the adoption of double elitism when generating offspring through mating, namely: (1) one parent is

* Corresponding author.
E-mail addresses: cea@research.att.com (C.E. Andrade), rofran@microsoft.com (R.F. Toso), jfgoncal@fep.up.pt (J.F. Gonçalves), mgcr@uw.edu (M.G.C. Resende).

https://doi.org/10.1016/j.ejor.2019.11.037
0377-2217/

drawn from an elite set that holds the best chromosomes found so far, and (2) the genes from the offspring are inherited from the elite parent with a higher probability. From a practical perspective, the BRKGA allows for fast prototyping, since the BRKGA solution space and the problem solution space are kept separate. Lucena, Andrade, Resende, and Miyazawa (2014) presented two variants of the BRKGA. In the gender-defining version (BRKGA-GD), each chromosome has a bit flag indicating its "gender," and mating is performed with an individual with the bit on and another with it off. In the multi-parent version (BRKGA-MP), more than two parents are used to generate a new offspring. Experimental results suggested that the BRKGA-MP can obtain better results than the BRKGA-GD. Similar behavior was reported in Rochman et al. (2017), when comparing results for the standard BRKGA and the BRKGA-GD applied to the capacitated vehicle routing problem with time windows.

For robustness, we typically find problem-specific intensification strategies such as local search or path-relinking being used within metaheuristic algorithms. In this paper, we focus on path-relinking, which exploits the neighborhood formed in the path between two feasible solutions and was first proposed by Glover (1997). The main idea of a path-relinking search is to construct a series of modifications on a base solution that leads to a guide solution, under the expectation that such intermediate solutions will be of high-quality due to the combination of elements from both solutions. The main issue with virtually all robust optimization solutions adopting path-relinking is that the procedure is developed around the problem, which makes code reuse and modularization very difficult.

In this paper, we extend the work of Lucena et al. (2014) by strengthening the BRKGA-MP with an Implicit Path-Relinking (IPR) procedure. We introduce and implement path-relinking strategies that operate on the random-key vector of the BRKGA, thus being generic and modular; in some sense, we have a meta-intensification strategy that can be paired with the BRKGA metaheuristic as building blocks for a robust optimization framework. Finally, we validate our work extensively and thoroughly using three complex real-world problems. In the first problem, one wants to design a complex hybrid wireless backhaul network that mixes several technologies such as 4G/LTE and Wi-Fi, backhauled using either fiber or microwave links (Andrade et al., 2015a). Several cost factors are used in this problem, such as equipment cost, maintenance/pole rental costs, fiber trenching, among others. In the second problem, one must schedule over-the-air software updates for a massive base of Internet of Things devices, especially connected cars (Andrade et al., 2017b; 2017c; 2019a). This problem has several practical constraints, with hundreds of thousands of updates that need to be scheduled. In the third problem, we tackle the winner determination problem in combinatorial auctions (Andrade et al., 2015b). These auctions are used for selling rights such as wave spectrum (bands) across different markets. The results show that the hybridization of BRKGA and IPR can significantly improve solution quality when compared to the standard versions.

The structure of this paper is as follows. Section 2 describes the standard BRKGA and the BRKGA-MP procedures. Section 3 discusses path-relinking, the IPR procedures, and their details. Section 4 addresses the hybridization options between BRKGA and IPR. Section 5 reports the computational experiments. Concluding remarks are made in Section 6.

2. A primer on BRKGA and the Multi-Parent BRKGA

The BRKGA is an elitist method that evolves a population of solutions in the unit hypercube, i.e., each solution is represented by a point in $[0, 1)^n$ for a given dimension n. This representation was first proposed in Bean (1994) and makes the method independent of the problem it solves. A decoder function $f : [0, 1)^n \to \mathcal{S}$ is necessary to map individuals from the BRKGA space to the problem space $\mathcal{S}$. Since all evolutionary operators are applied in the BRKGA space, custom operators based on the problem structure are unnecessary. This allows for fast prototyping and testing, thus reducing development costs. One crucial aspect of the decoder is that it must be a deterministic function, i.e., given a chromosome, the decoder should always generate the same solution. This prerequisite ensures the reproducibility of the results while not forbidding the use of random signals by decoders, say for tie breaking – if a pseudo-random number generator must be used in the decoder then it should be generated by a specific gene in the chromosome.

In the standard BRKGA, the population $\mathcal{P}$ is partitioned into *elite*, $\mathcal{P}_e$, and *non-elite* sets. The solutions in the elite set are the best of the entire population according to some performance metric (*fitness*). The non-elite set contains the remaining individuals ($\mathcal{P} \setminus \mathcal{P}_e$). In each BRKGA iteration, the elite set is copied to the next generation population $\mathcal{P}'$ and a small percentage of random solutions (called *mutants*, $\mathcal{P}'_m$) is also added to this new population. Considering that $|\mathcal{P}_e \cup \mathcal{P}'_m| \leq |\mathcal{P}|$, the population is completed with offspring, each generated from the combination (*mating*) of a randomly chosen individual from the elite set and a randomly chosen individual from the non-elite set. Mating is done using biased uniform crossover, where a gene is taken from the elite individual with probability ρ, or otherwise it is taken from the non-elite individual. For more details, the reader can refer to Gonçalves and Resende (2011a).

Since the just introduced "double elitist" evolutionary dynamics is a key enabler behind the success of standard BRKGAs, a natural extension is to improve it even further by drawing genes from a combination of several individuals. This variant is called Multi-Parent Biased Random-Key Genetic Algorithm (BRKGA-MP). A BRKGA-MP selects as many parents as the parameter π_t determines. Among these, π_e are elite parents and $\pi_t - \pi_e$ are non-elite parents. Each parent has a probability of passing its alleles to the offspring, which is calculated by taking the bias of the parent into account. Parent *bias* is defined by a pre-determined, non-increasing weighting bias function $\Phi : \mathbb{N} \to \mathbb{R}^+$ over its rank r. We used the bias functions proposed by Bresina (1996):

- Logarithmic: $\Phi(r) = 1/\log(r + 1)$;
- Linear: $\Phi(r) = 1/r$;
- Polynomial (in d): $\Phi(r) = r^{-d}$;
- Exponential: $\Phi(r) = e^{-r}$;
- Constant: $\Phi(r) = 1/\pi_t$.

Note that the constant bias give us a uniform distribution between the chosen parents.

Algorithm 1 shows the complete crossover procedure for BRKGA-MP. In lines 1 and 2, the parents are drawn and sorted according with their fitness. Lines 3–8 calculate the probabilities of drawing a gene from each parent. The function $\texttt{rank} : \mathbb{N} \to \mathbb{N}$ returns the position of the given chromosome in the sorted list of parents Q; this in turn is passed to the bias function Φ. Note that a normalization step is required to compute the proper probabilities for each parent (given by the vector *weight*). Lines 9–12 carry out the crossover. For each gene, a parent is chosen using the well-known "roulette" method that uses the probabilities calculated in the previous steps, and its allele is assigned to the gene of the new chromosome. Note that the procedure is carried out for each new offspring that is generated.

To conclude, for robustness we also employ other techniques used in several variants of genetic algorithms and metaheuristic

Algorithm 1: BRKGA-MP crossover scheme.

Input: Population $\mathcal{P}$; non-increasing function $\Phi : \mathbb{N} \to \mathbb{R}^+$; values $\pi_t \leq |\mathcal{P}|$,and $\pi_e \leq \min(\pi_t, |\mathcal{P}_e|)$.
Output: A new chromosome c.

```
Uniformly sample π_e individuals from P_e and π_t − π_e individuals from P \ P_e. Let Q be the set of these individuals;
Sort Q in non-increasing order of fitness;
total_weight ← 0;
foreach q ∈ Q in the given order do
    weight(q) ← Φ(rank(q));
    total_weight ← total_weight + weight(q);
foreach q ∈ Q do
    weight(q) ← weight(q)/total_weight;
Let c be an empty new chromosome;
for i = 1 to n do
    Select an individual q ∈ Q at random, with probabilities defined by weight;
    c[i] ← q[i];
return c;
```

algorithms in general. One of them is to evolve a set $\mathcal{I}_p$ of p populations independently and exchange their best i individuals after every g generations. This strategy is known as the island model (Whitley, Rana, & Heckendorn, 1998), and usually improves the variability of individuals, speeding up convergence and avoiding local optima. The second is to restart the algorithm if it is trapped in local optima and cannot improve the best solution after r generations, keeping only the best overall solution. However, this strategy can be too drastic because it destroys well-conserved parts of the genome that usually appear in good solutions during the evolution process. To avoid such drastic action, Andrade et al. (2019b) introduced the *shaking procedure* which instead of fully resetting the whole population, applies random modifications to the individuals in the elite set, and resets the non-elite individuals. Therefore, we guarantee diversity in the non-elite set and preserve the convergence structure in the shaken elite set.

Table 1 lists the parameters of each BRKGA variant, plus additional parameters for robustness. One disadvantage of BRKGAs in general is the large number of parameters to be set; often, automatic parameter tuning tools are necessary to effectively and efficiently find a good set of values.

Table 1
BRKGA parameters. The first column describes the parameter name and the second its definition. In the third column, parameters can be applicable to either "both" variants, only the standard ("std."), or only the multi-parent ("MP"). "Extra" indicates a parameter that hardens the metaheuristic.

Parameter	Description	Version
$\|\mathcal{P}\|$	Population size	Both
$\|\mathcal{P}_e\|$	Elite set size	Both
$\|\mathcal{P}_m\|$	Mutant set size	Both
ρ	Probability that an offspring inherits the allele of its elite parent	Only std.
π_t	Number of parents in the crossover	Only MP
π_e	Number of elite parents in the crossover	Only MP
Φ	Bias function	Only MP
p	Number of independent populations	Extra
i	Number of individuals in the exchange	Extra
g	Number of generations for exchange	Extra
r	Number of non-improving iterations until reset	Extra

3. The Implicit Path-Relinking

Path-Relinking (PR) is an intensification strategy that exploits the neighborhood formed in the path between two feasible solutions (Glover, 1997). Path-relinking has been successfully employed in several contexts such as routing (Resende & Ribeiro, 2003), covering (Pessoa, Resende, & Ribeiro, 2013), assignment (Mateus, Resende, & Silva, 2011), logic (Festa, Pardalos, Pitsoulis, & Resende, 2007), scheduling (Aiex, Binato, & Resende, 2003), etc. For a comprehensive survey, we point the reader to Ribeiro and Resende (2011). The main idea of a path-relinking search is to construct a series of modifications on a base solution that leads to a guide solution, under the expectation that such intermediate solutions will be of high-quality due to the combination of elements from both solutions. Standard path-relinking procedures start by choosing a base and a guide solution that are sufficiently diverse. Then, iteratively, a component of the base solution is chosen, removed, substituted with another component from the guide solution, and evaluated. When all components of the base solution are replaced with ones from the guide solution, the search is concluded and the best intermediate solution is returned. Components can be edges and nodes of a graph (for graph problems such as covering and routing), slots in a schedule or timetable, etc.

The one thing in common with most robust optimization solutions adopting path-relinking is that the procedure is developed around the problem, which makes code reuse and modularization very difficult. We depart from the standard approach and henceforth leverage the separation between problem space and solution space of the BRKGA to propose an *Implicit Path-Relinking* (IPR) strategy. This novel algorithm builds the path entirely in the unit hypercube of the BRKGA, making use of the (already existing) decoder function to map the intermediate vectors $v \in [0, 1]^n$ into the solution space for evaluation. The advantages of this approach are twofold, as it offers (1) considerable reduction in research and development time; and (2) robust hybridization for any random-key genetic algorithm such as the BRKGA.

While traditional PR implementations "understand" the problem structure by operating explicitly in the solution neighborhood (nodes or edges of the graph, timetable slots or items, etc), the IPR can only rely on the implicit structure of the random-key vector v. In general, random keys v_i are used in the following ways:

1. Thresholds: these keys are tied to decision variables such as nodes, edges, etc and usually indicate their inclusion into or exclusion from the solution. For instance, Resende et al. (2011) propose and analyze a BRKGA for the Steiner Triple Covering problem where key $v_i \geq 0.5$ indicates that the set P_i is part of the covering; this implementation found (thus far) the best coverings for the two largest instances in a standard problem set;
2. Permutations: the random-key vector v can induce a permutation of decision variables by first assigning each variable with an index of the vector and then sorting the vector for a permutation. As an example, let $v = \{0.3, 0.1, 0.6, 0.4\}$; and associate key 0.3 with index 0, key 0.1 with index 1, etc. Finally, sort v to obtain the permutation {1, 0, 3, 2}. The traveling salesman problem and its variants can be easily encoded in such way; nodes are assigned sequentially to keys, which are then sorted to obtain a sequence of cities to be visited.

In some hybrid cases, v is split into sub-vectors such that each sub-vector is interpreted in a particular way. As an example, a representation can have v divided into multiple non-overlapping sub-vectors or segments, each either based on permutations or thresholds.

In light of these options, we propose different routines to accommodate each one specifically. We describe these in the upcoming sections.

3.1. Direct implicit path-relinking

In the direct path-relinking strategy, the algorithm assumes that no particular structure exists in vectors v, thus the keys are exchanged directly between the incumbent and the guide solutions. Although such a strategy is similar to standard path-relinking procedures, it is too slow if only a single key is replaced at a time — it requires $O(n^2)$ calls to the decoder since for each key in the base solution, the procedure has to find the best replacement key from the guide solution. Such a route leads to a metaheuristic that focuses too much on intensification, leaving little time for diversification.

We also note that a hidden structure should be expected to appear in the vectors v throughout the evolutionary process, even though in this section we assume a unit vector without any explicit structure. This can be visualized for example in the context of a traveling salesman problem, where nearby cities form clusters of similar keys in the random key vector. Generalizing, certain portions of v can become essential building blocks for high-quality solutions; consequently, tearing apart such blocks when combining with another solution can be ineffective, since the operation may destroy the blocks.

Therefore, we propose a direct path-relinking approach that exchanges a block of keys (rather than a single key) in each iteration. Algorithm 2 depicts such a procedure for a minimization problem. Lines 1–4 set up the algorithm. First, it computes the number of blocks based on the given block size and the size of the path to pursue. Later this attribute serves as an additional stopping criterion, allowing the user to control the extent of the search. Next, we define *RB* as the set from which the block indices are sampled. We set up the base and guide solution to the input vectors and set the current best solution.

The main loop consists of lines 5–27. In each iteration, the algorithm scans all possible solutions assigning blocks from the guide solution to the incumbent solution. This is done in the inner loop 7–18. For each block, we compute the start position and offset, and extract the corresponding blocks from the incumbent and guide solution.

In line 11, we compare blocks b_1 and b_2 using the function `affectSolution` : $[0,1]^m \times [0,1]^m \to \{true, false\}$ which returns true if the swap between b_{old} and b_{new} can produce a new solution. If this assertion fails, we remove the block from consideration. We comment about this function later. In lines 14–15, the algorithm copies the block from the guide solution to the incumbent solution, and calls the decoder function to evaluate its fitness. Note that we can use the same decoder function we use in BRKGA. Then, in lines 17–18 we hold the block that has obtained the best solution in the current iteration.

Once the block search is ended, the algorithm commits the best change into the incumbent solution and remembers the best solution value found so far (lines 19–22). The block is removed from further considerations and we swap the roles of the base and the guide solutions (lines 23–24); this implements the bidirectional relinking, which produces better solutions than just forward or backward path-relinking strategies (Resende & Ribeiro, 2016). Lines 25–27 adds two stopping criteria besides the number of blocks:

Algorithm 2: Direct Implicit Path-Relinking.

Input: Vectors $base, guide \in [0,1]^n$; block size bs; path percentage $P\% \in (0,1]$; maximum time.
Output: The best solution found as a tuple (vector, value).

```
1  numblocks ← ⌈n/bs⌉;
2  pathsize ← ⌈numblocks × P%⌉;
3  Let RB = {1, ..., numblocks} be the set of remaining
   blocks to be exchanged;
4  bestsolution ← (base, ∞);
5  while RB is not empty do
6      bestblock ← (−1, ∞);
7      foreach block ∈ RB do
8          offset ← block × bs;
9          b_old ← base[offset : offset + bs];
10         b_new ← guide[offset : offset + bs];
11         if not affectSolution(b_old, b_new) then
12             RB ← RB \ {block};
13             Go to line 7;
14         base[offset : offset + bs] ← b_new;
15         solutionvalue ← decode(base);
16         base[offset : offset + bs] ← b_old;
17         if solutionvalue < bestblock.value then
18             bestblock ← (block, solutionvalue);
19     offset ← bestblock.block × bs;
20     base[offset : offset + bs] ← guide[offset : offset + bs];
21     if bestblock.value < bestsolution.value then
22         bestsolution ← (base, bestblock.value);
23     RB ← RB \ {bestblock.block};
24     swap(base, guide);
25     pathsize ← pathsize − 1;
26     if pathsize = 0 or maximum time is reached then
27         Go to line 28;
28 return bestsolution;
```

the maximum size of the path to be explored and the maximum time for the search. Finally, the algorithm returns the best solution found in line 28.

We make some observations about Algorithm 2. First, function `affectSolution` plays an important role at keeping the PR away from exploring regions already visited. For instance, suppose the vector v represents thresholds for binarization where key i is active if $v_i > 0.5$. Then, note that block $b' = [0.1, 0.6, 0.3]$ generates the same solution as block $b'' = [0.4, 0.9, 0.5]$, and thus there is no need to explore b''. Function `affectSolution` can be a generic function or a user customized function tailored to the problem to be solved. This function can be used to model many types of relationships between the subvectors such as discretizations and small/local key permutations, for example. Also, note that `affectSolution` has an important role in providing knowledge to the path-relinking procedure so it is not totally unaware of the problem structure.

The decoding procedure is also a sensitive piece of the direct IPR. Several BRKGA implementations have included local search procedures as part of the decoder for better solution quality and also faster convergence. Since such procedures can be

computationally expensive, they would severely impact the performance of the IPR. Another technique commonly found in BRKGAs with local search is to make the decoder update the chromosome (adjust the chromosome keys) to reflect the solution from the local search, since such a strategy also speeds up the convergence. However, chromosome adjustment will invalidate the IPR and should thus be disabled, since the calls to the decoder could potentially modify the incumbent solution and interfere with the path-relinking.

3.2. Permutation-based implicit path-relinking

As previously mentioned, it is common to find random-key algorithms that use random keys to induce permutations on the structures of a solution, that is, the order of the keys is used to build a solution. In such scenarios, the direct implicit path-relinking described in the previous section may be inefficient due to the nature of the solution representation in which there exists an infinite number of vectors representing the same solution. Therefore, simple key exchange frequently does not result in a new solution, even if we use key blocks and implement a proper function `affectSolution`. To address the aforementioned shortcomings, we propose the permutation-based implicit path-relinking. In this method, instead of exchanging keys between the base and guide solutions, we use the permutation from the guide solution to switch the keys of the base solution in such a way that they induce the same "sub-permutation" on the base solution.

Let us assume that we have two permutations p and p', representing sequences of elements from two distinct solutions for the problem (for instance, p and p' can be sequences of vertices representing solution for a traveling salesman problem). For a given position i in these sequences, the permutation-based implicit path-relinking checks whether such position i contains the same element in both sequences p and p'. If this is the case, i is skipped. Otherwise, the algorithm switches the keys of the base solution to induce the same position of i in the guide solution. Algorithm 3 depicts such procedure. Lines 1–4 perform the initial setup computing the path size, setting the base and guide solutions, and initializing the best solution found so far in this procedure. Line 5 initializes the set of indices to be checked. In lines 6 and 7, we create the tuples I_b and I_g to represent the permutations induced by vectors v_1 and v_2. This is why they are sorted in non-increasing order of the keys of the base and guide solution, respectively.

In the main loop (lines 8–28), the algorithm builds the path using swap moves between the positions of the elements in the permutations. In each iteration, the inner loop (lines 11–20) checks each position i: if i has the same elements in permutations I_b and I_g, we remove it from consideration and analyze the next position. Otherwise, we swap the keys of the base solution so that the element in position i matches where it is encountered in the guide solution (line 15). To clarify, consider the following example. Suppose that the base solution vector is $b = [0.1, 0.5, 0.9, 0.3, 0.7]$ and the guide solution vector is $g = [0.8, 0.4, 0.2, 0.6, 0.1]$, inducing the permutations $I_b = (1, 4, 2, 5, 3)$ and $I_g = (5, 3, 2, 4, 1)$, respectively. Note that position $i = 3$ has the same item (element 2) in both permutations and nothing must be done. On the other hand, position $i = 1$ has different elements in both permutations, i.e., $I_b[1] = 1$ and $I_g[1] = 5$. Since our goal is to have the same element in position i in both sequences, we swap the key in position $I_b[1] = 1$, which is $b[I_b[1]] = 0.1$, for the key in position $I_g[1] = 5$, which is $b[I_g[1]] = 0.7$. Such change results in the vector [0.7, 0.5, 0.9, 0.3, 0.1] and, consequently, the new permutation (5, 4, 2, 1, 3). Once we have the new base solution, the algorithm decodes it and switches the keys back to the former base solution. This procedure is carried out for each position, and we hold the one that generates the best solution (lines 18–20). After finding the best move, in lines 21–25, the algorithm commits the change, updates the best solution, removes position i_{best} from further consideration, and swaps the roles of the base and the guide solutions (forward/backward path-relinking). Lines 26–28 implement additional stopping criteria besides the number of checked elements, and in line 29, the best solution found is returned.

Algorithm 3 is very sensitive to the size of the input, since it makes $O(n^2)$ calls to the decoder. Unfortunately, we cannot use the block strategy of Algorithm 2 off-the-shelf, since permutations between blocks cannot be directly translated between the base and guide solutions. Thus, the user must control this process using the additional stopping criteria, i.e., maximum time or maximum path size, for otherwise the BRKGA will spend too much time on intensification. As with the direct implicit path-relinking, the decoding function is a sensitive piece and all guidelines made

Algorithm 3: Permutation-based Implicit Path Relinking.

Input: Vectors $v_1, v_2 \in [0, 1]^n$; path percentage $P\% \in [0, 1]$; maximum time.
Output: The best solution found as a tuple (vector, value).

1 $pathsize \leftarrow \lceil n \times P\% \rceil$;
2 $base \leftarrow v_1$;
3 $guide \leftarrow v_2$;
4 $bestsolution \leftarrow (v_1, \infty)$;
5 Let $RI = \{1, \ldots, n\}$ be the set of remaining indices to be checked;
6 Let $I_b = (1, \ldots, n)$ and $I_g = (1, \ldots, n)$ to be the set of indices of vectors v_1 and v_2 respectively;
7 Sort I_b and I_g according to non-increasing order of the corresponding keys in v_1 and v_2;
8 **while** *RI is not empty* **do**
9 $\quad i_{best} \leftarrow -1$;
10 $\quad val_{best} \leftarrow \infty$;
11 $\quad$ **foreach** $i \in RI$ **do**
12 $\quad\quad$ **if** $I_b[i] = I_g[i]$ **then**
13 $\quad\quad\quad RI \leftarrow RI \setminus \{i\}$;
14 $\quad\quad\quad$ Go to line 11;
15 $\quad\quad$ `swap`$(base[I_b[i]], base[I_g[i]])$;
16 $\quad\quad solutionvalue \leftarrow$ `decode`$(base)$;
17 $\quad\quad$ `swap`$(base[I_b[i]], base[I_g[i]])$;
18 $\quad\quad$ **if** $solutionvalue < val_{best}$ **then**
19 $\quad\quad\quad i_{best} \leftarrow i$;
20 $\quad\quad\quad val_{best} \leftarrow solutionvalue$;
21 $\quad$ `swap`$(base[I_b[i_{best}]], base[I_g[i_{best}]])$;
22 $\quad$ **if** $val_{best} < bestsolution.value$ **then**
23 $\quad\quad bestsolution \leftarrow (base, val_{best})$;
24 $\quad RI \leftarrow RI \setminus \{i_{best}\}$;
25 $\quad$ `swap`$(base, guide)$;
26 $\quad pathsize \leftarrow pathsize - 1$;
27 $\quad$ **if** $pathsize = 0$ *or maximum time is reached* **then**
28 $\quad\quad$ Go to line 29;
29 **return** *bestsolution*;

in Section 3.1 must be observed in the permutation-based implicit path-relinking.

3.3. Selecting solutions for path-relinking

One of the requirements for a successful path-relinking strategy is to choose two high-quality solutions that are different enough so that, when combining their substructures, we may produce a better solution with reasonable likelihood.

Since IPR is built on the same platform as the BRKGA-MP, we leverage the island model of multiple populations, performing path-relinking between elite chromosomes from different populations in a circular fashion. For example, suppose we have three populations. The framework performs three path-relinking operations: the first between elite individuals from populations 1 and 2, the second between populations 2 and 3, and the third between populations 3 and 1. When just one population is evolved, both base and guide individuals must be sampled from the elite set of that population.

When selecting individuals, we propose two sampling strategies. In the first, we simply pick the best individual from each population. This strategy has the advantage of always using the highest-quality individuals, with the drawback that such chromosomes may be too similar and thus constrain the path-relinking neighborhood. In the second strategy, we sample individuals the from the elite sets to improve the likelihood of having chromosomes that are different enough to result in a successful path-relinking. Since neither of the two strategies guarantees that the individuals are sufficiently different, we also use a distance function between individuals in the unit hypercube as a pre-requisite for running the IPR. In contrast, traditional path-relinking procedures implement a distance function between individuals using the particular structure of the problem.

When using threshold-based representations, we can compute the Hamming distance between the two vectors, i.e., the number of positions where the vectors differ. To compute such distance, we first apply the thresholds in each key of the random-key vectors to discretize them. After such categorical transformation, we can then compute the edit distance between the two vectors. For instance, let $v_1 = [0.1, 0.3, 0.7]$ and $v_2 = [0.5, 0.1, 0.9]$. Using the threshold of 0.4, we have a distance of one. Now using a threshold of 0.6, the distance is zero. Hamming distance computation runs in $O(n)$.

When using permutation-based representation, we can use the Kendall tau rank distance (Kendall, 1938), which is a metric that computes the number of pairwise disagreements between two ranking lists and, therefore, can be used to measure distance between two permutations. Given two vectors v' and v'', let τ' and τ'' be the order (ranking) induced by sorting v' and v'' non-increasingly, respectively. The Kendall tau distance between τ' and τ'' is

$$K(\tau', \tau'') = \Big|\{(i,j) : i < j,\ (\tau'[i] < \tau'[j] \ \wedge\ \tau''[i] > \tau''[j]) \ \vee \ (\tau'[i] > \tau'[j] \ \wedge\ \tau''[i] < \tau''[j])\}\Big|.$$

Using v_1 and v_2 from the previous example, we have $\tau' = (3, 2, 1)$ and $\tau'' = (3, 1, 2)$, which yields a distance of one. The Kendall tau distance can be computed with a simple $O(n \log n)$ algorithm, but it can be reduced to $O(n\sqrt{\log n})$ using the method presented in Chan and Pătraşcu (2010).

In population methods such as the BRKGA, it is common that the population converges to similar individuals over time. Therefore, IPR may have to test many, perhaps all pairs of elite chromosomes before proceeding to the path-relinking algorithm. This operation can take too long and impair the optimization process. Therefore, we use an additional parameter that limits the number (or percentage) of chromosome pairs tested before the path-relinking. If the IPR reaches this threshold without finding a chromosome pair suitable for the operation, the algorithm declares failure, indicating that the population is too homogeneous.

We conclude this section by listing the IPR parameters in Table 2.

Table 2
IPR parameters. The first column describes the parameter name and the second has its definition.

Parameter	Description
sel	Individual selection (random elite or best solutions)
cp%	Number/percentage of chromosome pairs to be tested
md	Minimum distance between chromosomes
typ	Path-relinking type (direct or permutation-based)
bs	Block size
ps%	Percentage/path size
max.time	Maximum time

4. Hybridizations of BRKGA-MP and IPR

Since both BRKGA-MP and IPR methods operate over the same space of solutions (the unit hypercube), the hybridization between the two methods is straightforward, as partially described in Section 3.3. However, there are two items yet to be discussed: how to mix the calls between the evolutionary steps of BRKGA and path-relinking; and how to deal with new solutions originated from the path-relinking.

In some methods, such as GRASP (Resende & Ribeiro, 2016), path-relinking procedures are used as an intensification method after the main algorithm is stopped. In this paper, we propose two different approaches. In the first, we call the IPR periodically, after a given number of generations of the BRKGA. The rationale for this approach is that the IPR may intensify the search offering alternative high-quality solutions, potentially speeding up convergence. In the second approach, the IPR is called only when the BRKGA converges and stalls for a given number of generations. Once we obtain a new solution from IPR, this solution is inserted into the population (see criterion below), and the BRKGA is resumed; hopefully, this new individual will lead to better solutions in the BRKGA. In Section 5 we describe experiments with both approaches.

The second and most critical issue in hybridization is when to insert solutions from IPR back into the BRKGA population. If the IPR solution is the best solution found so far, we can add it to the population directly. However, it is possible that the IPR solution is not the best found so far. If such solution is worse than the worst solution in the elite set, we discard it; otherwise, we compute the distance between the IPR solution and all other solutions in the elite set, using distance functions as described in Section 3.3. If the new solution respects a minimum distance between all elite solutions, we discard the worst solution of the elite set and add this IPR solution. This strategy tries to maintain a diverse population. Another common-used strategy is to replace the most similar solution having worse cost. In this case, we keep the diversity of the population but improve the overall quality of the solutions.

5. Experimental results and discussion

To assess the performance of BRKGA-MP-IPR, we make use of three real-life scenarios where the standard BRKGA was previously applied (Andrade et al., 2017b; 2017c; 2015a; Andrade et al., 2015b). In each case, the chromosomes are used in a different way to represent the solution, namely threshold-based, permutation-based, and hybrid representation. Moreover, the decoders have no local-search procedures, allowing us to evaluate the performance

of the evolutionary algorithm alone, without interference from local search procedures.

We also tested the implicit path-relinking mechanism by itself, without the evolutionary dynamics of BRKGA. For that, we sampled the individuals for path-relinking from the entire population.

The results are represented by labels as following: STD for the original/standard BRKGA; MP for BRKGA-MP (no IPR hybridization); IPR for standalone IPR; STALL for the hybrid BRKGA-MP-IPR where the IPR procedure is called when the BRKGA evolution does not improve the best solution for a given number of generations; FIXED for the BRKGA-MP-IPR where the IPR procedure is always called after a given number of generations. When needed, we use the suffix "DIR" (respectively, "PER") to explicitly indicate the use of direct (resp., permutation-based) path-relinking.

For each scenario, we carefully tuned each algorithm individually using iterated F-race (Birattari, Yuan, Balaprakash, & Stützle, 2010; López-Ibáñez, Dubois-Lacoste, Cáceres, Birattari, & Stützle, 2016) so that we can extract the best results from each algorithm. Having tuned each algorithm, we performed ten independent runs on each instance for each problem. The stopping criterion used was a maximum wall-clock time or a maximum number of iterations without improvement, both independently set for each problem.

The computational experiments were performed on a cluster of identical machines, each having an Intel Xeon E5-2650 CPU at 2.0 gigahertz (12 cores/24 threads) and 128 gigabytes of RAM running CentOS Linux 6.9. The algorithms tested in this paper are implemented in C++ and built with GNU GCC 7.2. The BRKGA-MP-IPR framework is licensed under a BSD-like license and is available at:

- https://github.com/ceandrade/brkga_mp_ipr_cpp (C++ version);
- https://github.com/ceandrade/brkga_mp_ipr_julia (Julia version);
- https://github.com/ceandrade/brkga_mp_ipr_python (Python version).

5.1. Hybrid representation: wireless backhaul network design problem

We start off testing a complex problem that uses a hybrid chromosome representation to solve the wireless backhaul network design problem (WBNDP) first described in Andrade et al. (2015a). WBNDP is a real-world, very complex problem that aims to build a routing forest for LTE small cells, mixing wireless and fiber hops. The problem consists of a set of demand points, a set of locations where small cells (radio base stations) can be installed (usually utility poles on street intersections), and a set of root points to where the small cells can connect either by fiber or wireless links. Each demand point generates traffic that can be converted into revenue. Such traffic must be served by small cells (either LTE, WiFi, or both, each one with a service radius) with limited capacity in handling traffic and number of connections. The small cells must route over wireless links with limited capacity or a fiber if it is in the last hop (close to a root point). We have equipment costs (depending of the type of the equipment), fiber trenching costs, and pole leasing costs. The objetive is to build a routing forest such net revenue (gross revenue minus deployment and operations cost) is maximized over a given time window. There are 30 instances with sizes varying from 454 poles, 24 root nodes, 2500 demand blocks, over an area of 62 kilometer square to 8740 poles, 420 root nodes, 35,750 blocks over an area of 410 square kilometer.

In Andrade et al. (2015a), the chromosome is partitioned into five sections using a mix of threshold and permutation representations. Although the decoder is somewhat convoluted, it does not contain local search routines. Since it is not straightforward to use direct or permutation-based path-relinking in a hybrid chromosome representation, we tested both versions. For the same reason, we implemented a custom routine to compute the distance between two chromosomes: first, we compute the sum of the Kendall Tau distance for each permutation-inducing block; next, we add the Hamming distance between the corresponding threshold blocks (activation parameters genes, see Section 5.1 in Andrade et al. (2015a)).

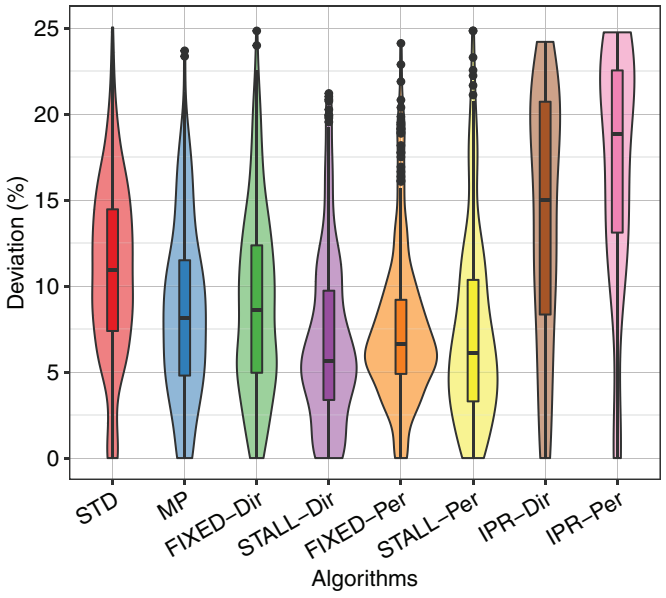

Fig. 1. Deviation from the best solutions found for the WBNDP.

The tuned parameters for this problem are described in Table 3. The first section describes the BRKGA parameters (according to Table 1), followed by the IPR parameters (description in Table 2), and the extra parameters (also described in Table 1). Each row shows the parameters for each algorithm. Note that "–" indicates "not applicable." The minimum distance between chromosomes is proportional to the chromosome size multiplied by column "*md*." For FIXED-DIR and FIXED-PER, parameter $\mathcal{I}_g$ indicates the interval, in number of generations, that the path-relinking procedure is called. For STALL-DIR and STALL-PER, parameter $\mathcal{I}_g$ indicates the number of generations without improvement in the best solution before path-relinking should be called. Also note that, for a single population, there is no individual exchange. We used one wall-clock hour or 1000 iterations without improvement as the stopping criterion, and four threads for decoding, over 30 instances presented in Andrade et al. (2015a).

Fig. 1 shows the distribution of the average percentage deviation over the best solution found for each instance. Note that STALL-DIR and STALL-PER present slightly better deviations, though FIXED-PER follows closely. The average percentages and standard deviations are 7.20 ± 5.76, 6.68 ± 5.88, and 7.63 ± 4.72, respectively. MP and FIXED-DIR presented similar results with average percentage of 8.73 ± 5.62 and 9.14 ± 5.71, respectively. The STD, IPR-DIR, and IPR-PER algorithms fell short (11.19 ± 5.62, 45.85 ± 22.10, and 47.52 ± 19.19, respectively).

To confirm our conclusions, we tested the normality of these distributions using the Shapiro–Wilk test and applied the Wilcoxon rank sum test, considered more effective than the *t*-test for distributions sufficiently far from normal and for sufficiently large sample sizes (Conover, 1980; Fay & Proschan, 2010). For all tests,

Table 3
Tuned algorithm parameters for the WBNDP.

	BRKGA							IPR						Extra			
	$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	ρ	π_t	π_e	Φ	*sel*	*cp*%	*md*	*typ*	*bs*	*ps*%	$\mathcal{I}_p$	$\mathcal{I}_i$	$\mathcal{I}_g$	$\mathcal{R}$
STD	500	30	15	0.70	–	–	–	–	–	–	–	–	–	3	2	100	300
MP	1800	24	10	–	3	10	e^{-r}	–	–	–	–	–	–	1	–	–	215
FIXED-DIR	720	32	12	–	7	10	$1/r$	Best	12	0.26	Dir.	33	3	1	–	400	410
STALL-DIR	360	22	11	–	4	6	$1/r$	Best	–	0.12	Dir.	32	31	1	–	150	500
FIXED-PER	1400	43	14	–	5	10	e^{-r}	Best	11	0.23	Perm.	–	70	1	–	275	330
STALL-PER	950	13	10	–	7	10	$1/r$	Best	–	0.08	Perm.	–	15	1	–	260	440
IPR-DIR	60	–	–	–	–	–	–	Rand.	100	0.12	Dir.	15	92	–	–	–	300
IPR-PER	100	–	–	–	–	–	–	Rand.	100	0.15	Perm.	–	85	–	–	–	290

Table 4
p-values for Wilcoxon rank sum test among the algorithms for the WBNDP. We use a confidence interval of 95% and omit *p*-values are less than 0.05.

	STD	MP	FIXED-DIR	STALL-DIR	FIXED-PER	STALL-PER	IPR-DIR
MP	–						
FIXED-DIR	–	1.00					
STALL-DIR	–	–	–				
FIXED-PER	–	0.20	–	0.76			
STALL-PER	–	–	–	1.00	0.05		
IPR-DIR	–	–	–	–	–	–	
IPR-PER	–	–	–	–	–	–	1.00

Table 5
Algorithm performance compared to the best solutions found the WBNDP.

Alg.	% Sol.	% Run	Prop. diff.	
			%	σ
STD	20.00	4.03	11.80	5.24
MP	26.67	5.67	9.39	5.65
FIXED-DIR	3.33	3.33	9.59	5.62
STALL-DIR	43.33	7.00	7.89	5.66
FIXED-PER	3.33	2.67	7.98	4.64
STALL-PER	40.00	7.00	7.93	5.14
IPR-DIR	3.33	1.33	46.56	21.52
IPR-PER	3.33	0.67	47.92	18.79

we assume a confidence interval of 95%, and used the Bonferroni method for *p*-value adjustment. Table 4 shows the *p*-values from the Wilcoxon test. We also provide, in Appendix A, the analysis of variance (ANOVA) test for each pair of algorithms, even though the distributions are not normal. As expected, ANOVA and Wilcoxon results differ slightly, but they agree in the majority of cases. Note that all BRKGA-MP versions are significantly better than STD and IPR. We cannot confirm a statistical difference between STALL-DIR, FIXED-PER, and STALL-PER. We also cannot confirm a significant difference between MP and FIXED-DIR, and MP and FIXED-DIR.

Table 5 reports the performance of the algorithms with respect to the best solutions found. Column "% Sol" shows the percentage of the number of best solutions found, and column "% Run" shows the percentage of the number of runs on which the algorithm found a best solution. The two columns under label "Prop. diff." show, respectively, the average of the proportional difference between the best solution value and the achieved value (%), and its corresponding standard deviation (σ). One can notice the superiority of the STALL versions, with more best solutions found, and a smaller deviation from the other solutions.

Fig. 2 shows performance profiles for all algorithms. The *x*-axis shows the time needed to reach a target solution value, while the *y*-axis shows the cumulative probability to reach a target solution value for the given time in the *x*-axis. The target values are the best solution for each instance. It is interesting to note that STD is faster in finding solutions within 50 minutes. However, towards the end of the optimization, all variants exhibit similar profile.

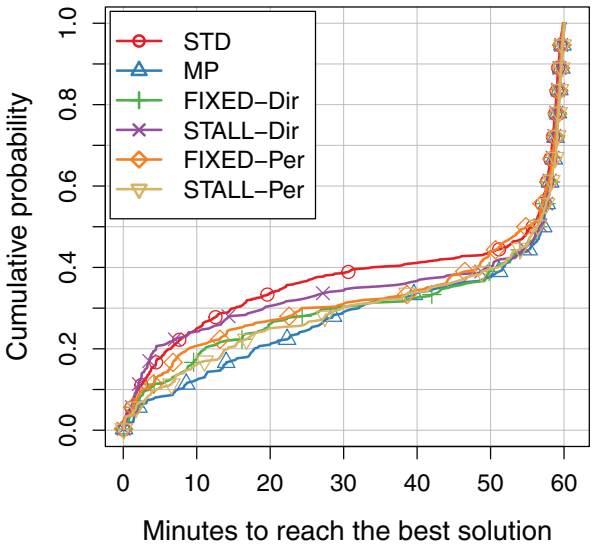

Fig. 2. Running time distributions to best solutions found for the WBNDP. The identification marks correspond to 4% of the points plotted for each algorithm.

Table 6
Statistics for IPR routine calls for WBNDP.

Alg.	Calls		Improv.		Homog.	
	Avg	Max	Avg	Max	Avg	Max
FIXED-DIR	2 ± 3	10	0 ± 1	1	0	0
STALL-DIR	4 ± 5	21	0	0	4 ± 4	21
FIXED-PER	3 ± 3	15	0	0	2 ± 3	15
STALL-PER	1 ± 1	5	0 ± 1	1	0	0

Table 6 brings the number of calls of the IPR method according to each algorithm variation. The second and third columns show the average number of calls (and standard deviation) per run, and the maximum number of calls among all runs, respectively. The fourth and fifth columns show the average and the maximum number of improved solutions obtained by IPR. The last two columns show the average and maximum number of times that the IPR procedure found the elite population to be homogeneous and bailed out. In general, the IPR methods were called a few times in this scenario. Note that STALL-DIR and FIXED-PER could not find improved solutions. Indeed, in almost all its calls, IPR declared the population too homogeneous for path-relinking. FIXED-DIR and

Table 7
Algorithm parameters for the FOTA problem. Note that column *bs* is empty since FOTA decoder is permutation-based, and therefore, it does not use blocks of keys.

	BRKGA							IPR						Extra			
	$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	ρ	π_t	π_e	Φ	*sel*	*cp*%	*md*	*typ*	*bs*	*ps*%	$\mathcal{I}_p$	$\mathcal{I}_i$	$\mathcal{I}_g$	$\mathcal{R}$
STD	500	30	15	0.70	–	–	–	–	–	–	–	–	–	3	2	100	300
MP	300	26	10	–	2	3	$\frac{1}{\log(r+1)}$	–	90	–	–	–	–	1	–	75	500
FIXED	320	24	12	–	4	6	$1/r$	Rand.	85	0.29	Perm.	–	25	1	–	75	380
STALL	360	22	11	–	4	6	$1/r$	Best	93	0.12	Perm.	–	31	1	–	65	340
IPR	100	–	–	–	–	–	–	Rand.	100	0.10	Perm.	–	17	–	–	–	240

STALL-PER could execute IPR a few times, but only one improvement was found in a given run.

5.2. Permutation representation: firmware-over-the-air scheduling problem

To test a straightforward representation, we use the Firmware-over-the-air (FOTA) scheduling problem, defined in Andrade et al. (2017b,c). In this problem, one must create a schedule for connected cars to initiate a download/update session over LTE networks. In FOTA, historical data of the car connections and LTE network utilization and capacity are given. With that, we build a download capacity map over a given time window. Cars and network capacity restrictions are also given. The objective is to find a schedule that: (1) shifts the FOTA download to quieter periods on the network to avoid rush hours, and therefore, minimizing the impact of FOTA in the network; (2) schedule as many cars as possible such that they finish the download within the time window; and (3) minimize the total completion time for the FOTA download. This problem was modeled as a machine scheduling problem. The 153 instances vary from 107 jobs and 1676 machines to 33,132 jobs and 9,916 machines over a time window of 672 periods. Due to several real-world constraints and the size of the instances, the FOTA scheduling problem is extremely challenging and hard to solve. Andrade et al. (2019a) presented a BRKGA with permutation-based representation to schedule the cars using a simple constructive decoder.

In Table 7, we list the tuned parameters for this permutation-based BRKGA. The distance between two chromosomes was computed using the Kendall tau method, and it is proportional to the size of the chromosome as described before. For this scenario, we used six wall-clock hours or 1,000 iterations without improvement as stopping criterion and four threads for decoding. There are 154 instances, as described in Andrade et al. (2019a).

In Fig. 3, one can see the distribution of the average percentage deviation with respect to the best solution found for each instance. Note that MP and MP-IPR variations presented the smallest deviations from the best solutions. On average, MP shows 1.33 ± 1.34% of deviation, FIXED shows 1.16 ± 0.79%, and STALL presents 1.20 ± 0.88%. The original version, STD, resulted on a deviation of 4.19 ± 1.75%, and IPR alone resulted on 17.56 ± 8.40%. Applying the Wilcoxon rank sum test, we confirm that all these differences (pairwise) are statistically significant at a confidence interval of 95%, except for FIXED and STALL, for which the p-value is larger than 0.05. Appendix B brings the analysis of variance (ANOVA) test for each pair of algorithms, of which results agree with Wilcoxon results. Therefore, the MP-IPR procedures are able to find much better results than the multi-parent and standard BRKGA variants alone. This can be confirmed in Table 8, as the number of best solutions found by the MP-IPR versions is significantly larger than by the other algorithms.

Fig. 4 depicts the performance profiles, similar to Fig. 2. The standalone IPR is absent from the figure since it did not result in a

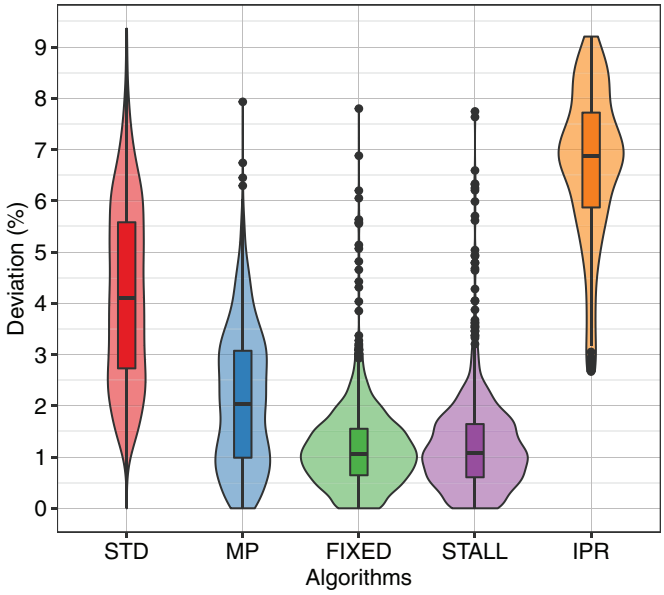

Fig. 3. Deviation from the best solutions found for the FOTA.

Table 8
Algorithm performance compared to the best solutions found the FOTA.

Alg.	% Sol.	% Run	Prop. diff.	
			%	σ
STD	0.65	0.07	4.19	1.75
MP	22.88	2.29	2.16	1.31
FIXED	37.91	3.79	1.21	0.77
STALL	38.56	3.86	1.24	0.86
IPR	0.00	0.00	17.57	8.40

Table 9
Statistics for IPR routine calls for FOTA.

Alg.	Calls		Improv.	
	Avg	Max	Avg	Max
FIXED	7 ± 10	64	3 ± 3	16
STALL	4 ± 6	44	1 ± 1	4

curve. Note that all algorithms take a long time to converge due to the difficulty of this problem. STD and MP are slower than the other algorithms. Note that FIXED presented better cumulative probability than STALL, although they converge to similar results at the end of their runs.

Table 9 exposes the number of calls of the IPR method according for each algorithm. The description is similar to Table 6, but we omitted the last two columns since no homogeneity was found in this scenario. As expected, in FIXED the IPR is called more times on

Table 10
Algorithm parameters for the WDP.

	BRKGA							IPR						Extra			
	$\lvert\mathcal{P}\rvert$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	ρ	π_t	π_e	Φ	*sel*	*cp*%	*md*	*typ*	*bs*	*ps*%	$\mathcal{I}_p$	$\mathcal{I}_i$	$\mathcal{I}_g$	$\mathcal{R}$
STD	2000	20	15	0.70	–	–	–	–	–	–	–	–	–	3	2	100	500
MP	4600	13	23	–	2	3	r^{-2}	–	92	–	–	–	–	3	1	60	300
FIXED	4670	10	29	–	2	3	e^{-r}	Best	97	0.20	Dir.	132	48	3	2	450	470
STALL	4500	11	35	–	2	3	e^{-r}	Rand.	94	0.15	Dir.	74	22	3	2	200	470
IPR	70	–	–	–	–	–	–	Rand.	100	0.02	Dir.	5	70	–	–	–	400

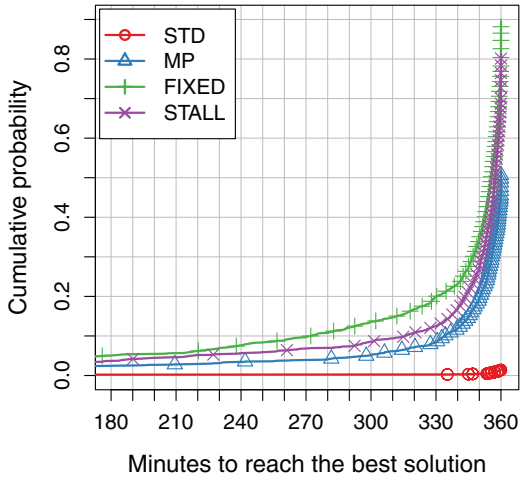

Fig. 4. Running time distributions to best solutions found the FOTA. The identification marks correspond to 4% of the points plotted for each algorithm.

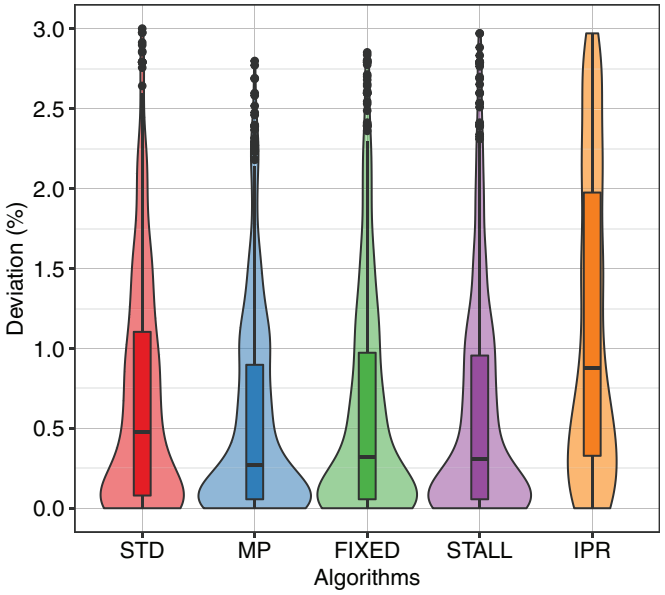

Fig. 5. Deviation from the best solutions found for the WDP.

average due to its cyclical nature, consequently resulting in more improvements. However, even if we wait to call IPR only when the BRKGA mechanism stalls, we can improve the incumbent solution; indeed, STALL calls IPR in only 20% of the runs, but those calls are still valuable in the final results.

5.3. Threshold representation: winner determination problem

For some problems, the hybridization between BRKGA-MP and IPR methods may be not as effective as expected. To show this, we experimented with the Winner Determination Problem (WDP) in combinatorial auctions, where a seller should pick a set of non-overlapping bids to maximize the total selling value. Such bids are done on different and possibly overlapping sets of items. More formally, let I be a set of items, i_1, $i_2 \in I$ be two items, and let $v : 2^I \rightarrow \mathbb{R}$ be a valuation function for sets of these items. Items i_1 and i_2 are said to be *complementary* if and only if $v(\{i_1\}) + v(\{i_2\}) \leq v(\{i_1, i_2\})$, where $\{i_1, i_2\}$ denotes a bundle of items i_1 and i_2. They are said to be *substitutes* if and only if $v(\{i_1\}) + v(\{i_2\}) \geq v(\{i_1, i_2\})$. Each bidder sets their hoffers for several subsets of sale items, i.e., each bidder b submits a function v_b to the seller. Using these functions, the seller chooses the non-overlapping best offers, maximizing their revenue. Instances for WDP varies from 40 bids to 10 goods to 1500 bids and 1500 goods.

Andrade et al. (2015b) presented six variants of decoders to solve this problem, both threshold and permutation-based. Here, we used the GA_{RA} threshold-based variation where the bids are sorted by cost/benefit and the decoder picks the non-overlapping bids such that the corresponding key in the chromosome is larger than or equal to 0.5.

The tuned parameters employed throughout this section are listed in Table 10. We used one wall-clock hour or 1,000 iterations without improvement as stopping criterion, and four threads for decoding. There are 105 instances available for this problem, as presented in Andrade et al. (2015b).

Table 11
Algorithm performance compared to the best solutions found for the WDP.

Alg.	% Sol.	% Run	Prop. diff.	
			%	σ
STD	43.27	16.44	1.06	1.06
MP	52.88	20.00	0.74	0.78
FIXED	50.00	19.90	0.85	0.88
STALL	45.19	18.85	0.87	0.92
IPR	4.81	0.67	2.74	2.31

Fig. 5 depicts the distribution of the average percentage deviation from the best solution found for each instance. Notice that MP, FIXED, and STALL all have small deviations from the best solution (mean and standard deviation of 0.59 ± 0.75, 0.68 ± 0.75, 0.70 ± 0.90, respectively). Also notice that, for the MP-IPR variants, the majority of the solutions found are quite close to the best solutions. By applying the Wilcoxon rank test, we can assert that MP, FIXED and STALL are not significantly different. However, they are significantly better than STD and IPR. The analysis of variance (ANOVA) in Appendix C shows slightly different results, where most of the algorithms have significantly different results pairwisely. However, since these distributions are not normal, these results may be skewed. Also, Table 11 shows that MP found the most substantial portion of best solutions when compared with

Table 12
Statistics for IPR routine calls for WDP.

Alg.	Calls		Improv.		Homog.	
	Avg	Max	Avg	Max	Avg	Max
FIXED	3 ± 1	6	0	0	2 ± 1	6
STALL	5 ± 2	11	0	0	5 ± 2	11

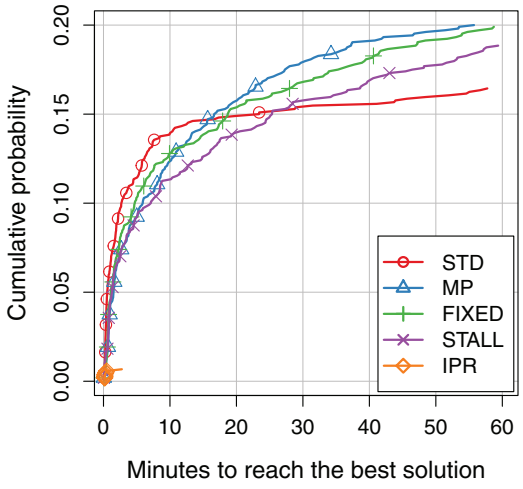

Fig. 6. Running time distributions to best solutions found for the WDP. The identification marks correspond to 4% of the points plotted for each algorithm.

the other variations, also offering the smallest proportional difference.

We investigate the use of IPR in this scenario in Table 12. Note that neither IPR variant could find better solutions within their calls. In fact, for most cases, the elite population was too homogeneous and prevented the IPR from running. In Fig. 6, we can see that MP has a higher probability of finding a good solution faster than the MP-IPR variations, since MP does not waste time with unsuccessful IPR calls. One can argue that is also a reason for MP to find more best solutions than the IPR counterparts.

Note that if the practitioner had implemented a custom path-relinking procedure for this scenario, the outcome would be very disappointing, especially after investing development time and expectations only to find out that a path-relinking procedure is not adequate for this scenario. Even though IPR could not improve the results, it would still save development time and provide a quick answer as to the feasibility or infeasibility of its adoption.

6. Final considerations

In this paper, we presented the Multi-Parent Biased Random-Key Genetic Algorithm hybridized with a novel Implicit Path-Relinking (BRKGA-MP-IPR) procedure. The BRKGA-MP uses multiple parents for mating, chosen using a biased function for faster convergence. The IPR leverages the generic unit hypercube representation of the BRKGA and applies the path-relinking procedures over that space, instead of the problem space as does the standard version of path-relinking does.

By testing the variations of the BRKGA-MP-IPR extensively on real-world problems, we conclude that both the multi-parent evolutionary dynamics and the IPR mechanism contribute to finding solutions that are much better than those found by the standard BRKGA. It must be said that the implicit path-relinking procedure is more sensitive to the application, to the point where it may fail when the BRKGA population is too homogeneous.

At the very least, the BRKGA-MP-IPR can save considerable development time. Using the same infrastructure, the user can experiment with variants of BRKGA and IPR incurring minimal development overhead. This fact not only offers savings for an optimization project using these tools but also reduces the time-to-market for a good solution to the underlying problem.

Disclaimer

The work of Rodrigo F. Toso and Mauricio G. C. Resende was done while they were employed by AT&T Labs Research in Middletown, NJ, and of José F. Gonçalves when he was employed at University of Porto, Portugal.

Acknowledgments

We thank the anonymous reviewers for providing comments that improved this paper. José F. Gonçalves thanks the support of the North Portugal Regional Operational Programme (NORTE 2020) [grant number NORTE-01-0145-FEDER-000020] under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

Appendix A. Analysis of variance for hybrid representation: wireless backhaul network design problem

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
>> STD / MP
             Df Sum Sq Mean Sq F value   Pr(>F)
Algorithm     1   1292  1292.2   41.01 2.31e-10 ***
Residuals  1018  32080    31.5
```

```
>> STD / FIXED-DIR
             Df Sum Sq Mean Sq F value   Pr(>F)
Algorithm     1    874   874.4   28.17 1.36e-07 ***
Residuals  1018  31599    31.0
```

```
>> STD / STALL-DIR
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1   2361  2360.9    75.3 <2e-16 ***
Residuals  1018  31917    31.4
```

```
>> STD / FIXED-PER
             Df Sum Sq Mean Sq F value   Pr(>F)
Algorithm     1   1939  1939.3   65.37 1.75e-15 ***
Residuals  1018  30200    29.7
```

```
>> STD / STALL-PER
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1   4260    4260   132.3 <2e-16 ***
Residuals  1018  32769      32
```

```
>> STD / IPR-DIR
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1 256104  256104    1559 <2e-16 ***
Residuals  1018 167183     164
```

```
>> STD / IPR-PER
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1 281351  281351    2184 <2e-16 ***
Residuals  1018 131151     129
```

```
>> MP / FIXED-DIR
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1     29   28.81   0.869  0.352
Residuals  598  19824   33.15
```

```
>> MP / STALL-DIR
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1    113  113.20   3.361 0.0673 .
Residuals  598  20142   33.68
```

```
>> MP / FIXED-PER
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1     46   46.36   1.505   0.22
Residuals  598  18425   30.81
```

```
>> MP / STALL-PER
            Df Sum Sq Mean Sq F value   Pr(>F)
Algorithm    1    609   609.0   17.35 3.57e-05 ***
Residuals  598  20994    35.1
```

```
>> MP / IPR-DIR
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1 208094  208094   800.7 <2e-16 ***
Residuals  598 155408     260
```

```
>> MP / IPR-PER
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1 227218  227218    1138 <2e-16 ***
Residuals  598 119376     200
```

```
>> FIXED-DIR / STALL-DIR
            Df Sum Sq Mean Sq F value  Pr(>F)
Algorithm    1    256  256.24   7.793 0.00541 **
Residuals  598  19661   32.88
```

```
>> FIXED-DIR / FIXED-PER
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1    148  148.28   4.941 0.0266 *
Residuals  598  17944   30.01
```

```
>> FIXED-DIR / STALL-PER
            Df Sum Sq Mean Sq F value   Pr(>F)
Algorithm    1    903   902.8   26.32 3.92e-07 ***
Residuals  598  20514    34.3
```

```
>> FIXED-DIR / IPR-DIR
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1 203226  203226   784.4 <2e-16 ***
Residuals  598 154928     259
```

```
>> FIXED-DIR / IPR-PER
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1 222129  222129    1117 <2e-16 ***
Residuals  598 118896     199
```

```
>> STALL-DIR / FIXED-PER
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1     15   14.67    0.48  0.488
Residuals  598  18262   30.54
```

```
>> STALL-DIR / STALL-PER
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1    197  197.11   5.658 0.0177 *
Residuals  598  20831   34.83
```

```
>> STALL-DIR / IPR-DIR
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1 217914  217914   839.4 <2e-16 ***
Residuals  598 155245     260
```

```
>> STALL-DIR / IPR-PER
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1 237474  237474    1191 <2e-16 ***
Residuals  598 119213     199
```

```
>> FIXED-PER / STALL-PER
            Df Sum Sq Mean Sq F value  Pr(>F)
Algorithm    1    319   319.3   9.991 0.00165 **
Residuals  598  19114    32.0
```

```
>> FIXED-PER / IPR-DIR
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1 214353  214353   834.9 <2e-16 ***
Residuals  598 153528     257
```

```
>> FIXED-PER / IPR-PER
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1 233756  233756    1190 <2e-16 ***
Residuals  598 117496     196
```

```
>> STALL-PER / IPR-DIR
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1 231219  231219   885.8 <2e-16 ***
Residuals  598 156098     261
```

```
>> STALL-PER / IPR-PER
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1 251355  251355    1252 <2e-16 ***
Residuals  598 120065     201
```

```
>> IPR-DIR / IPR-PER
            Df Sum Sq Mean Sq F value Pr(>F)
Algorithm    1    420   420.3   0.988  0.321
Residuals  598 254480   425.6
```

Appendix B. Analysis of variance for permutation representation: firmware-over-the-air scheduling problem

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


>> STD / MP
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1 180948  180948    5051 <2e-16 ***
Residuals  3028 108477      36


>> STD / FIXED
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1 203913  203913    5787 <2e-16 ***
Residuals  3028 106699      35


>> STD / STALL
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1 203008  203008    5748 <2e-16 ***
Residuals  3028 106934      35


>> STD / IPR
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1 134753  134753    3675 <2e-16 ***
Residuals  3010 110383      37


>> MP / FIXED
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1    693   692.7   576.5 <2e-16 ***
Residuals  3058   3674     1.2


>> MP / STALL
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1    641   640.6   501.2 <2e-16 ***
Residuals  3058   3909     1.3


>> MP / IPR
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1   3286    3286    1358 <2e-16 ***
Residuals  3040   7358       2


>> FIXED / STALL
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1      1  1.0166   1.459  0.227
Residuals  3058   2131  0.6967


>> FIXED / IPR
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1   6984    6984    3805 <2e-16 ***
Residuals  3040   5580       2


>> STALL / IPR
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1   6817    6817    3564 <2e-16 ***
Residuals  3040   5814       2
```

Appendix C. Analysis of variance for threshold representation: winner determination problem

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


>> STD / MP
             Df Sum Sq Mean Sq F value  Pr(>F)
Algorithm     1   44.9   44.93   53.98 2.9e-13 ***
Residuals  2078 1729.6    0.83


>> STD / FIXED
             Df Sum Sq Mean Sq F value   Pr(>F)
Algorithm     1   21.3  21.296   23.21 1.56e-06 ***
Residuals  2078 1907.0   0.918


>> STD / STALL
             Df Sum Sq Mean Sq F value   Pr(>F)
Algorithm     1   16.6  16.620   17.46 3.06e-05 ***
Residuals  2078 1978.4   0.952


>> STD / IPR
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1   1759  1759.4   544.2 <2e-16 ***
Residuals  2078   6719     3.2


>> MP / FIXED
             Df Sum Sq Mean Sq F value  Pr(>F)
Algorithm     1    4.4   4.360   6.667 0.00989 **
Residuals  2078 1359.0   0.654


>> MP / STALL
             Df Sum Sq Mean Sq F value  Pr(>F)
Algorithm     1    6.9   6.896   10.02 0.00157 **
Residuals  2078 1430.4   0.688


>> MP / IPR
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1   2367    2367     797 <2e-16 ***
Residuals  2078   6171       3


>> FIXED / STALL
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1    0.3  0.2894   0.374  0.541
Residuals  2078 1607.8  0.7737


>> FIXED / IPR
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1   2168  2167.8   709.6 <2e-16 ***
Residuals  2078   6348     3.1


>> STALL / IPR
             Df Sum Sq Mean Sq F value Pr(>F)
Algorithm     1   2118  2118.0   685.6 <2e-16 ***
Residuals  2078   6419     3.1
```

References

Aiex, R. M., Binato, S., & Resende, M. G. C. (2003). Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing, 29*(4), 393–430. doi:10.1016/S0167-8191(03)00014-0.

Andrade, C. E., Ahmed, S., Nemhauser, G. L., & Shao, Y. (2017a). A hybrid primal heuristic for finding feasible solutions to mixed integer programs. *European Journal of Operational Research, 263*(1), 62–71. doi:10.1016/j.ejor.2017.05.003.

Andrade, C. E., Byers, S. D., Gopalakrishnan, V., Halepovic, E., Poole, D. J., Tran, L. K., & Volinsky, C. T. (2017b). Connected cars in a cellular network: A measurement study. In *Proceedings of the 2017 internet measurement conference*. In *IMC '17* (pp. 235–241). New York, NY, USA: ACM. doi:10.1145/3131365.3131403.

Andrade, C. E., Byers, S. D., Gopalakrishnan, V., Halepovic, E., Poole, D. J., Tran, L. K., & Volinsky, C. T. (2017c). Managing massive firmware-over-the-air updates for connected cars in cellular networks. In *Proceedings of the second ACM international workshop on smart, autonomous, and connected vehicular systems and services*. In *CarSys '17* (pp. 65–72). ACM. doi:10.1145/3131944.3131953.

Andrade, C. E., Byers, S. D., Gopalakrishnan, V., Halepovic, E., Poole, D. J., Tran, L. K., & Volinsky, C. T. (2019a). Scheduling software updates for connected cars with limited availability. *Applied Soft Computing, 82*, 105575. doi:10.1016/j.asoc.2019.105575.

Andrade, C. E., Miyazawa, F. K., & Resende, M. G. C. (2013). Evolutionary algorithm for the *k*-interconnected multi-depot multi-traveling salesmen problem. In *Proceedings of the fifteenth annual conference on genetic and evolutionary computation*. In *GECCO'13* (pp. 463–470). New York, NY, USA: ACM. doi:10.1145/2463372.2463434.

Andrade, C. E., Resende, M. G. C., Karloff, H. J., & Miyazawa, F. K. (2014). Evolutionary algorithms for overlapping correlation clustering. In *Proceedings of the sixteenth conference on genetic and evolutionary computation*. In *GECCO'14* (pp. 405–412). New York, NY, USA: ACM. doi:10.1145/2576768.2598284.

Andrade, C. E., Resende, M. G. C., Zhang, W., Sinha, R. K., Reichmann, K. C., Doverspike, R. D., & Miyazawa, F. K. (2015a). A biased random-key genetic algorithm for wireless backhaul network design. *Applied Soft Computing, 33*, 150–169. doi:10.1016/j.asoc.2015.04.016.

Andrade, C. E., Silva, T., & Pessoa, L. S. (2019b). Minimizing flowtime in a flowshop scheduling problem with a biased random-key genetic algorithm. *Expert Systems with Applications, 128*, 67–80. doi:10.1016/j.eswa.2019.03.007.

Andrade, C. E., Toso, R. F., Resende, M. G. C., & Miyazawa, F. K. (2015b). Biased random-key genetic algorithms for the winner determination problem in combinatorial auctions. *Evolutionary Computation, 23*, 279–307. doi:10.1162/EVCO_a_00138.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal On Computing, 2*(6), 154–160. doi:10.1287/ijoc.6.2.154.

Beirão, N. C. L. F. (1997). Sistema de apoio à decisão para sequenciamento de operações em ambientes job shop. https://repositorio-aberto.up.pt/handle/10216/12242.

Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-Race and iterated F-race: an overview. In *Experimental methods for the analysis of optimization algorithms* (pp. 311–336). Springer Berlin Heidelberg. doi:10.1007/978-3-642-02538-9_13.

Bresina, J. L. (1996). Heuristic-biased stochastic sampling. In *Proceedings of the thirteenth national conference on artificial intelligence*. In *AAAI'96: 1* (pp. 271–278). AAAI Press.

Caserta, M., & Reiners, T. (2016). A pool-based pattern generation algorithm for logical analysis of data with automatic fine-tuning. *European Journal of Operational Research, 248*(2), 593–606. doi:10.1016/j.ejor.2015.05.078.

Chan, T. M., & Pătraşcu, M. (2010). Counting inversions, offline orthogonal range counting, and related problems. In *Proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms* (pp. 161–173). doi:10.1137/1.9781611973075.15.

Conover, W. J. (1980). *Practical nonparametric statistics* (2nd). John Wiley & Sons. doi:10.1086/407092.

Ericsson, M., Resende, M. G. C., & Pardalos, P. M. (2002). A genetic algorithm for the weight setting problem in OSPF routing. *Journal of Combinatorial Optimization, 6*(3), 299–333. doi:10.1023/A:1014852026591.

Fay, M. P., & Proschan, M. A. (2010). Wilcoxon–Mann–Whitney or *t*-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics Surveys, 4*, 1–39. doi:10.1214/09-SS051.

Festa, P., Pardalos, P. M., Pitsoulis, L. S., & Resende, M. G. C. (2007). GRASP with path relinking for the weighted MAXSAT problem. *Journal of Experimental Algorithmics, 11*. doi:10.1145/1187436.1216581.

Glover, F. (1997). *Tabu search and adaptive memory programming — advances, applications and challenges* (pp. 1–75)). Boston, MA: Springer US.

Gonçalves, J. F., & Beirão, N. C. L. F. (1999). Um algoritmo genético baseado em chave aleatórias para sequenciamento de opearções. *Investigação Operacional, 19*, 123–137.

Gonçalves, J. F., & de Almeida, J. R. (2002). A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics, 8*(6), 629–642. doi:10.1023/A:1020377910258. ISSN 1381-1231

Gonçalves, J. F., & Resende, M. G. C. (2011a). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics, 17*, 487–525. doi:10.1007/s10732-010-9143-1.

Gonçalves, J. F., & Resende, M. G. C. (2011b). A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. *Journal of Combinatorial Optimization, 22*(2), 180–201. doi:10.1007/s10878-009-9282-1.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan press.

Kendall, M. (1938). A new measure of rank correlation. *Biometrika, 30*, 81–89. doi:10.2307/2332226.

Kramer, O. (2017). *Genetic algorithm essentials*. Springer International Publishing AG. doi:10.1007/978-3-319-52156-5.

Lopes, M. C., Andrade, C. E., Queiroz, T. A., Resende, M. G. C., & Miyazawa, F. K. (2016). Heuristics for a hub location-routing problem. *Networks, 68*(1), 54–90. doi:10.1002/net.21685.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The IRACE package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives, 3*, 43–58. doi:10.1016/j.orp.2016.09.002.

Lucena, M. L., Andrade, C. E., Resende, M. G. C., & Miyazawa, F. K. (2014). Some extensions of biased random-key genetic algorithms. In *Proceedings of the forty-sixth brazilian symposium of operational research*. In *XLVI SBPO* (pp. 2469–2480).

Mateus, G. R., Resende, M. G. C., & Silva, R. M. A. (2011). GRASP with path-relinking for the generalized quadratic assignment problem. *Journal of Heuristics, 17*, 527–567. doi:10.1007/s10732-010-9144-0.

Pessoa, L. S., & Andrade, C. E. (2018). Heuristics for a flowshop scheduling problem with stepwise job objective function. *European Journal of Operational Research, 266*(3), 950–962. doi:10.1016/j.ejor.2017.10.045.

Pessoa, L. S., Resende, M. G., & Ribeiro, C. C. (2013). A hybrid Lagrangean heuristic with GRASP and path-relinking for set *k*-covering. *Computers & Operations Research, 40*(12), 3132–3146. doi:10.1016/j.cor.2011.11.018.

Resende, M. G. C., & Ribeiro, C. C. (2003). A GRASP with path-relinking for private virtual circuit routing. *Networks, 41*(2), 104–114. doi:10.1002/net.10065.

Resende, M. G. C., & Ribeiro, C. C. (2016). *Optimization by GRASP*. Springer-Verlag New York. doi:10.1007/978-1-4939-6530-4.

Resende, M. G. C., Toso, R. F., Gonçalves, J. F., & Silva, R. M. A. (2011). A biased random-key genetic algorithm for the Steiner triple covering problem. *Optimization Letters*, 1–15. doi:10.1007/s11590-011-0285-3.

Ribeiro, C. C., & Resende, M. G. C. (2011). Path-relinking intensification methods for stochastic local search algorithms. *Journal of Heuristics, 18*, 192–214. doi:10.1007/s10732-011-9167-1.

Rochman, A. N., Prasetyo, H., & Nugroho, M. T. (2017). Biased random key genetic algorithm with insertion and gender selection for capacitated vehicle routing problem with time windows. *AIP Conference Proceedings, 1855*(1), 020025. doi:10.1063/1.4985470.

Stefanello, F., Aggarwal, V., Buriol, L. S., Gonçalves, J. F., & Resende, M. G. C. (2015). A biased random-key genetic algorithm for placement of virtual machines across geo-separated data centers. In *Proceedings of the 2015 on genetic and evolutionary computation conference*. In *GECCO '15* (pp. 919–926). New York, NY, USA: ACM. doi:10.1145/2739480.2754768.

Whitley, D., Rana, S., & Heckendorn, R. B. (1998). The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology, 7*, 33–47.