Noname manuscript No.
(will be inserted by the editor)

A biased random-key genetic algorithm for routing and wavelength assignment under a sliding scheduled traffic model

Bruno Q. Pinto · Celso C. Ribeiro · Isabel Rosseti · Thiago F. Noronha

Received: December 2018

Abstract The problem of routing and wavelength assignment in optical networks consists in minimizing the number of wavelengths that are needed to route a set of demands, such that demands routed using lightpaths that share common links are assigned to different wavelengths. We present a biased random-key genetic algorithm for approximately solving the problem of routing and wavelength assignment of sliding scheduled lightpath demands in optical networks. In this problem variant, each demand is characterized not only by a source and a destination, but also by a duration and a time window in which it has to be met. Computational experiments show that the numerical results obtained by the proposed heuristic improved upon those obtained by a multistart constructive heuristic. In addition, the biased random-key genetic algorithm obtained much better results than an existing algorithm for the problem, finding solutions that use roughly 50% of the number of wavelengths determined by the latter.

Keywords biased random-key genetic algorithm · metaheuristics · routing and wavelength assignment · sliding scheduled lightpath demands · scheduled lightpath demands · optical networks

Bruno Q. Pinto
Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro, Uberlândia, MG 38411-104, Brazil.
E-mail: bruno.queiroz@iftm.edu.br

Celso C. Ribeiro
Universidade Federal Fluminense, Institute of Computing, Niterói, RJ 24210-240, Brazil.
E-mail: celso@ic.uff.br

Isabel Rosseti
Universidade Federal Fluminense, Institute of Computing, Niterói, RJ 24210-240, Brazil.
E-mail: rosseti@ic.uff.br

Thiago F. Noronha
Universidade Federal de Minas Gerais, Department of Computer Science, Belo Horizonte, MG 31270-901, Brazil.
E-mail: tfn@dcc.ufmg.br

1 Introduction

Given the physical topology of a WDM (Wavelength Division Multiplexing) optical network and a set of lightpath demands over this network, the problem of routing and wavelength assignment (RWA) consists in routing the lightpaths and assigning a wavelength to each of them. Lightpaths that share common links must necessarily be assigned to different wavelengths.

The demands for lightpaths are known in advance. Every lightpath must be assigned to the same wavelength along all links that form its route, since there is no wavelength conversion. There may be an arbitrary demand for lightpaths between any pair of nodes, since the traffic between them can be larger than the maximum capacity supported by a single lightpath.

Variants of this problem are characterized by different optimization criteria and traffic patterns, see e.g. [5,10,22,30,28,31,34,46,47]. The objective of the min-RWA problem variant [12,30] is to minimize the number of wavelengths that are needed to establish all lightpath requests. The problem was determined to be NP-hard in [12]. Heuristics for min-RWA appear e.g. in [5,28,30, 16,23,26,42].

Routing and wavelength assignment may not only consider static traffic models, but also dynamic and scheduled models. In the case of scheduled traffic models, each demand for lightpaths requires the use of the network over some specific time period. Two problem variants exist: (a) scheduled lightpath demands (RWA-SLD) [22,41] and (b) sliding scheduled lightpath demands (RWA-SSLD) [4,18–20,24,39,40], following its introduction by Wang et al. [45,46] in the context of service provisioning under a scheduled traffic model in reconfigurable optical networks. In the former, the demands must be assigned to the resources of the network at the specific times where they will be needed [22]. In the latter, only a time window longer than the duration of each demand is given [46].

According to Kuri et al. [22], scheduled traffic models are more realistic due to the periodic nature of data traffic, which is not homogeneous along the day. We consider the RWA-SSLD problem variant in this work, assuming that the resources of the network (capacity and wavelengths) are unbounded and minimizing the number of wavelengths that are needed to route all demands for lightpaths within the time windows assigned to each of them.

In the following, we propose and describe a biased random-key genetic algorithm (BRKGA) for routing and wavelength assignment under a sliding scheduled traffic model. The remainder of this article is organized as follows. The sliding scheduled lightpath demands problem (RWA-SSLD) and related work are presented in the next section. Section 3 gives an overview of BRKGAs and their customization to the sliding scheduled lightpath demands problem. Section 4 describes a constructive heuristic and a decoder. A test instance generator and computational experiments are reported in Section 5. The numerical results obtained by the proposed approach for approximately solving problem RWA-SSLD improved upon those obtained by a multistart constructive heuristic. We also report a comparison of the results obtained by the

biased random-key genetic algorithm with those obtained by Liu et al. [24], reproducing as closely as possible the same conditions tested in their experiments. The new heuristic outperformed the algorithm in the literature, finding solutions that use roughly 50% of the number of wavelengths determined by the latter. Concluding remarks are presented in the final section.

2 Routing and wavelength assignment under a sliding scheduled traffic model

2.1 Scheduled lightpath demands problem (RWA-SLD)

Let $G = (V, E)$ be a graph representing the network topology, where V represents the set of the nodes and E denotes the set of edges or links connecting the nodes. Let also $D = \{d_1, \ldots, d_m\}$ be a set of m demands for lighpaths, where each demand d_i is defined by a source s_i, a destination t_i, a number n_i of requested lightpaths, a starting time a_i and a finishing time b_i, for $i = 1, \ldots, m$.

Kuri et al. [22] established the problem of minimizing the number of wavelenghts needed to route all lightpath requests at their specific starting and finishing times. For each demand d_i, all n_i lightpaths should be routed along the same links using n_i different wavelenghts, for $i = 1, \ldots, m$. Skorin-Kapov [41] proposed heuristics for approximately solving RWA-SLD. Among them, DP_RWA_SLD is an adaptive greedy heuristic based on the algorithm proposed in [26] for solving problem min-RWA.

2.1.1 Heuristic DP-RWA-SLD

Algorithm 1 describes the greedy heuristic DP_RWA_SLD originally proposed by Skorin-Kapov [41] for solving RWA-SLD, based on that of Manohar et al. [26] for the min-RWA problem. Each of its iterations determines a new subset of non-conflicting demands that can be routed using different wavelengths over disjoint paths in the network.

Heuristic DP_RWA_SLD receives as inputs the graph $G = (V, E)$ representing the network and the set $D = D_{SLD}$ of demands $d_1, \ldots, d_m$ in non-increasing order of their number of requested wavelengths. Its outputs are the number λ of wavelengths needed to route all demands, as well as the set W_d of wavelengths that may be used by demand d and the set of edges $Path_d$ in the route assigned to demand d, for every $d \in D$.

Let $c_{s,t}(G)$ and $R_{s,t}(G)$ denote, respectively, the number and the set of edges in the shortest path from node s to t in $G = (V, E)$. The diameter of graph G is $diam(G) = \max\{c_{s,t}(G) : s, t \in V\}$.

The number λ of wavelengths needed to route all demands is set to zero in line 1. The loop in lines 2–5 initializes sets W_d and $Path_d$, for every demand $d \in D$. The loop in lines 6–24 is performed until a wavelength has been assigned to every demand. Line 7 initializes as empty the subset of yet unassigned demands that can be routed without conflicts. The loop in lines 8–20 investigates all

yet unassigned demands $d \in D$. A temporary copy G' of graph G is built in line 9. The loop in lines 10–14 checks in line 11 if demand d overlaps in time with each demand $p \in Partition$ already routed. In case of overlapping, the edges in path $Path_p$ cannot be used to route demand d and must be removed from the set of candidate edges $E(G')$.

If the length $c_{s_d,t_d}(G')$ of the shortest path from s_d to t_d in G' is smaller than or equal to $\max\{diam(G), \sqrt{|E|}\}$, then demand d is inserted in $Partition$ in line 16 and the arcs in path $R_{s_d,t_d}(G')$ are saved in $Path_d$ in line 17. The wavelengths indexed by $\lambda + 1, \ldots, \lambda + n_d$ will be used to route demand d in line 18. The number λ of lightpaths needed is increased by $w = \max\{n_p : p \in Partition\}$ in line 21. The demands selected in $Partition$ that will be routed with the new selected wavelengths are removed from D in line 22. Line 24 returns the number of wavelengths needed, the wavelengths assigned to the demands, and the lightpaths used to route the demands.

Algorithm 1 DP_RWA_SLD($G, D = D_{SLD}$)

```
 1: λ ← 0
 2: for all d ∈ D do
 3:    W_d ← ∅
 4:    Path_d ← ∅
 5: end for
 6: while D ≠ ∅ do
 7:    Partition ← ∅
 8:    for all d ∈ D do
 9:       G' ← G
10:       for all p ∈ Partition do
11:          if a_d ≤ a_p < b_d or a_p ≤ a_d < b_p then
12:             E(G') ← E(G') \ Path_p
13:          end if
14:       end for
15:       if c_{s_d,t_d}(G') ≤ max{diam_G, √|E|} then
16:          Partition ← Partition ∪ {d}
17:          Path_d ← R_{s_d,t_d}(G')
18:          W_d ← {λ + 1, ..., λ + n_d}
19:       end if
20:    end for
21:    λ ← λ + max{n_p : p ∈ Partition}
22:    D ← D \ Partition
23: end while
24: return λ, (W_d, d = 1, ..., m), (Path_d, d = 1, ..., m)
```

2.2 Sliding scheduled lightpath demands problem (RWA-SSLD)

This variant originally described by Wang et al. [45,46] is a generalization of problem RWA-SLD, when time windows are defined for each lightpath requested, instead of starting and finishing times. Real applications are more likely to behave as RWA-SSLD problems, e.g. whenever a company desires

to optimize the cost of a network by enforcing traffic during times with less use for routine operations such as data back-up [46]. According to Jaekel et al. [20], such strategies allow for energy optimization.

Once again, we denote by $G = (V, E)$ a graph representing the network topology, where V and E are, respectively, the set of nodes and the set of edges. We also denote by $D = \{d_1, \ldots, d_m\}$ a set of m demands for lighpaths, where each demand d_i is defined by a source s_i, a destination t_i, a number n_i of requested lightpaths, a time window starting at a_i and finishing at b_i, and a duration τ_i, for $i = 1, \ldots, m$. The starting time f_i of the transmission of lightpath request d_i is not known beforehand but should satisfy the time window constraint $a_i \leq f_i \leq b_i - \tau_i$. Table 1 illustrates an example with $m = 4$ demands for the network in Figure 1.

Table 1 Example of problem RWA-SSLD with $m = 4$ demands for lighpaths.

Demands	s_i	t_i	n_i	a_i	b_i	τ_i
d_1	A	E	2	1	9	4
d_2	B	E	1	3	10	5
d_3	A	D	3	7	12	4
d_4	C	E	4	4	11	3

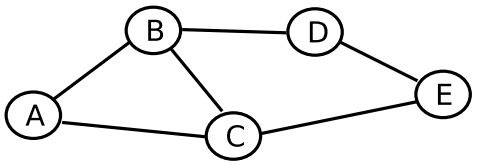

Fig. 1 Network topology example.

Figure 2 illustrates a time window starting at a_i and finishing at b_i for the scheduling of demand d_i with duration τ_i starting any time $f_i \in [a_i, b_i]$.

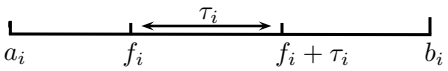

Fig. 2 Sliding time window model.

In the example of Figure 3, corresponding to task d_1 of Table 1, the starting time $f_1 = 3$ of the transmission of lightpath request d_1 can be anticipated or delayed by two time units.

If all time windows are tight, i.e., $b_i - a_i = \tau_i$, for $i = 1, \ldots, m$, then there are no time slacks and problems RWA-SSLD and RWA-SLD coincide.

For practical purposes, the length of each lightpath is bounded from above by $\max\{diam(G), \sqrt{|E|}\}$, where $diam(G)$ denotes the diameter of graph $G =$

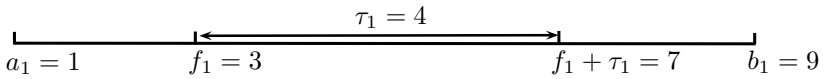

Fig. 3 Example of a sliding time window for demand d_1 of Table 1.

(V, E), i.e., the number of edges in the shortest path between the pair of nodes in V whose shortest path between them is maximum over all pairs of nodes in V [21].

2.3 Literature review

Wang et al. [45,46] introduced the problem of service provisioning on a sliding scheduled traffic model in optical networks. The approach proposed in [46] decomposes RWA-SSLD in two subproblems: (a) how to appropriately place a demand within its corresponding time window to reduce overlap in time over the set of demands; and (b) route and assign wavelengths to a set of demands under the sliding scheduled traffic model in mesh reconfigurable optical networks without wavelength conversion. Another algorithm is used to rearrange the demands that could not be satisfied by the routing and wavelength assignment algorithm by negotiating a new setup time. The primary objective of the proposed algorithms consists in scheduling the set of demands in such a way that the number of wavelengths used in the entire network is minimized when the network resources are sufficient to route all demands. If the network resources are not sufficient to route all demands, then the objective becomes to minimize the total number of wavelengths used, including those used by the demands rearranged. Network NSFNET with 14 nodes was used in the computational experiments, with the number of wavelengths that can be used in each edge limited to 30 and the number of demands $m \in [50, 400]$. The test problems considered three classes of demands: low, medium, and highly timely correlated demands [22].

Liu et al. [24] addressed the particular case of RWA-SSLD where $n_i = 1$, for $i = 1, \ldots, m$, using a partition coloring model. Network topologies USANET and NSFNET with 24 and 14 nodes, respectively, were used in the computational experiments, with the number of demands $m \in [10, 100]$. In this work, the length of any lightpath cannot be longer than the shortest path between its extremities by more than δ hops. The maximum-conflict degre-first conflict reducing algorithm (MCDF-CR) was compared with algorithm IPSR developed in [4]. The numerical results showed that on average MCDF-CR reduced the number of wavelenghts used by 9.9%.

Jaekel et al. [18] proposed integer programming formulations for both RWA-SLD and RWA-SSLD, minimizing the number of wavelengths used or the network congestion (measured in terms of the number of wavelengths on the most heavily loaded link). Networks NFSNET with 14 nodes and that in [17] with 10 nodes were used in the computational experiments, with the number of wavelengths that can be used in each edge limited to 32. Again, the

test problems considered three classes of demands: low, medium, and highly timely correlated demands [22]. The time windows had durations of 2, 4 or 6 hours.

Andrei et al. [4] proposed a new formulation for RWA-SSLD and a new heuristic named IPSR-LR (integrated provisioning of sliding requests). They assumed that all demands request the same number of wavelengths. Network USANET with 24 nodes was used in the computational experiments, with the number of demands $m \in [250, 1400]$.

Jaekel et al. [20] investigated variants of RWA-SSLD and RWA-SLD that minimize energy consumption and transmission cost over the network. They proposed integer programming formulation algorithms for problems RWA-SLD and RWA-SSLD, as well as a genetic algorithm for instances with up to 20 nodes and hundreds of demands. Networks NSFNET and ARPANET with 14 and 20 nodes, respectively, were used in the computational experiments, with the number of demands $m \in [60, 800]$. Once again following [22], the test problems included instances with three classes of demands: weakly, medium, and strongly time correlated.

In this work, we consider problem RWA-SSLD assuming that the resources of the network (capacity and wavelengths) are unbounded and minimizing the number of wavelengths that are needed to route all demands for lightpaths within the time windows assigned to each of them, using the notation introduced in Sections 2.1 and 2.2. A biased random-key genetic algorithm for RWA-SSLD is presented in the next section.

3 A biased random-key genetic algorithm for RWA-SSLD

Random-key genetic algorithms (RKGA) were introduced by Bean [6]. Solutions are associated with vectors of randomly generated real numbers called keys. A deterministic algorithm, which in this context is also called a decoder, takes a vector of real keys to produce a feasible solution of the optimization problem at hand and computes its fitness or objective value. Parents are randomly selected from the entire population for mating and crossover, with repetions allowed.

A biased random-key genetic algorithm (BRKGA) differs from an RKGA by the strategy used to select parents for crossover, see Resende and Ribeiro [36] for an advanced tutorial of methods and applications. BRKGAs have been applied very successfully to a large variety of scheduling problems and, in particular, to simpler variants of routing and wavelength assignment (see, e.g., [10, 30]).

One of the main characteristics of a BRKGA is that each new solution is generated by the combination of one solution selected at random from the subset of elite solutions of the current population with another that is always a non-elite solution. This selection is biased not only because one parent is always an elite solution, but also because it has a higher probability of passing its characteristics to the offspring.

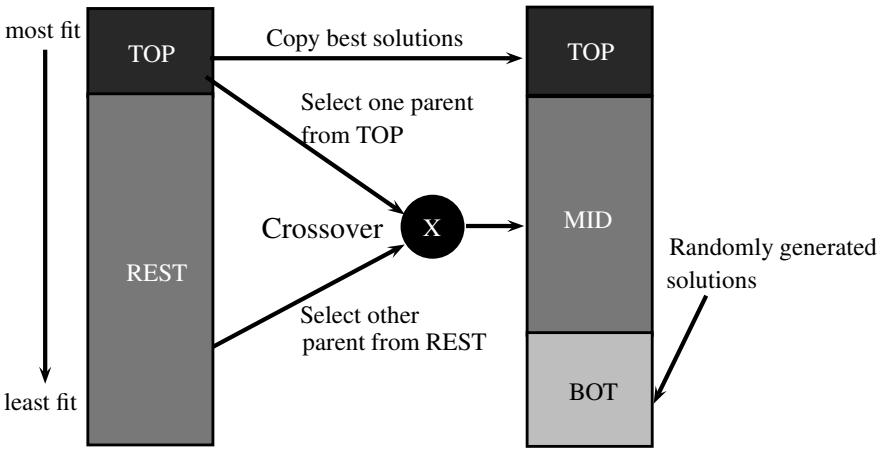

Fig. 4 Population evolution between consecutive generations of a BRKGA.

In this work, we develop a BRKGA for RWA-SSLD that evolves a population formed by vectors of real numbers. Each chromosome of this population is a vector of $|D|$ components. Each of these components, or keys, is a real number in the range $[0, 1)$, associated with one of the demands of the set D. The chromosome is decoded by an algorithm that takes the vector of real keys as input and creates a feasible solution for the problem: the decoder returns as outputs the number of wavelengths needed, the set of wavelengths that are effectively used, and the lightpaths – i.e., the routes – assigned to each demand. The decoder DECODER-SSLD will be described in Section 4.

We make use of the parametric uniform crossover strategy originally proposed in [43] for combining two parent solutions and producing a new one. According to this strategy, the solution generated by crossover inherits with a higher probability each of its keys from the best parent. BRKGAs, as the one developed here, do not make use of the standard mutation operator. Instead, the concept of mutants is used: new solutions (i.e., mutants) are introduced in the population at each generation, generated at random following the same strategy as in the initial population. They play the same role of the mutation operator in more standard genetic algorithm frameworks, diversifying the search and assisting the procedure to escape from local optima [8,9,30].

The $|D|$ real keys in each chromosome are randomly generated in the initial population. The population is divided in two subsets at each generation: TOP (formed by elite solutions) and $REST$. Subset TOP always contains the best (or elite) solutions. Subset $REST$ is also partitioned in two disjoint subsets: MID and BOT, with subset BOT containing the worst solutions. The solutions in TOP are copied to the population of the next generation, as shown in Figure 4. The solutions in BOT are replaced by new mutants, which are temporarily placed in the subset BOT of the population of the next generation. The rest of the solutions of the population of the next generation are obtained by crossover, with one parent always being randomly chosen from

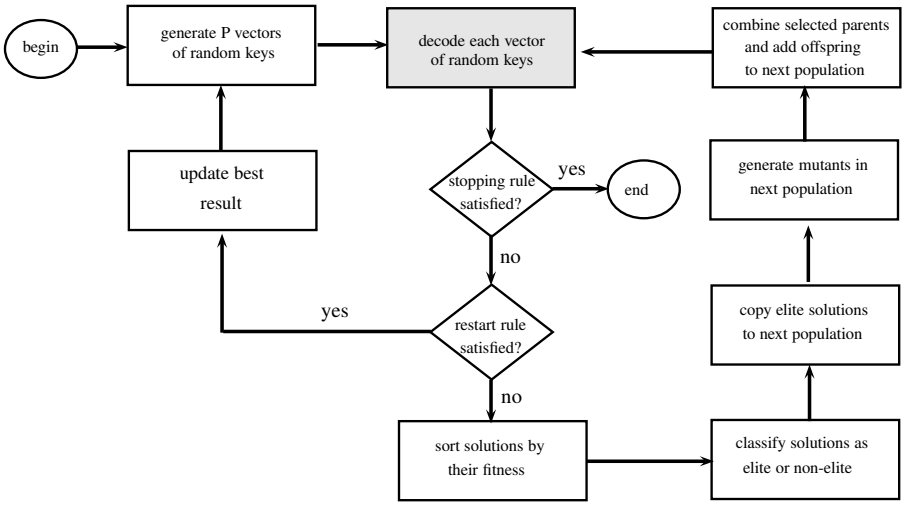

Fig. 5 BRKGA framework.

TOP and the other from *REST*. Therefore, $|MID| = |REST| - |BOT|$ solutions are created with this crossover strategy, which distinguishes a BRKGA from a simple RKGA, where both parents are randomly selected from the entire population. Since a parent solution can be chosen for crossover more than once in the same generation, elite solutions have a higher probability of passing their characteristics (i.e., their random keys) to the next generation.

4 Decoder and constructive heuristic

The implementation of a BRKGA for routing and wavelength assignment of sliding scheduled lightpath demands made use of the C++ library brkgaAPI developed by Toso and Resende [44], which is a framework for the implementation of biased random-key genetic algorithms.

The instantiation of the framework in Figure 5 requires exclusively the development of a class implementing the decoder for each specific problem, being the only problem-dependent part of the tool.

A BRKGA running in this framework requires some parameters [14]: (a) the population size $p = |TOP| + |REST|$; (b) the fraction *pe* of the population corresponding to the elite subset; (c) the fraction *pm* of the population corresponding to the mutant subset; (d) the probability *rhoe* that a solution obtained by crossover inherits its keys from the best parent; and (e) the number k of generations without improvement in the best solution until a new restart is performed.

The biased random-key genetic algorithm BRKGA-SSLD for solving RWA-SSLD makes use of the decoder DECODER-SSLD presented in Section 4.2. This decoder is built upon heuristic DP-RWA-SSLD described in the next section.

4.1 Heuristic DP_RWA_SSLD

Heuristic DP_RWA_SSLD to solve problem RWA-SSLD with sliding scheduled demands is a modification of algorithm DP_RWA_SLD to solve problem RWA-SSLD, already described in Section 2.1.1. It randomly sets the starting time of each demand d in the corresponding interval $[a_d, b_d - \tau_d]$. Algorithm DP_RWA_SLD is applied to the modified set of demands with randomly fixed starting times.

Algorithm 2 receives as inputs the graph $G = (V, E)$ representing the network and the set $D = D_{SSLD}$ of demands $d_1, \ldots, d_m$. As for the previous algorithm, its outputs are the number λ of wavelengths needed to route all demands, as well as the set W_d of wavelengths that may be used by demand d and the set of edges $Path_d$ in the route assigned to demand d, for every $d \in D$.

Heuristic DP_RWA_SSLD starts by initializing set D_{SLD} as empty in line 1. The loop in lines 2–9 progressively builds the set D_{SLD} from the demands in D_{SSLD}. For each demand $d \in D_{SSLD}$, a demand d' with the same source and destination nodes and the same number of required wavelengths is created in lines 3 to 5. The starting time $a_{d'}$ of demand d' is randomly selected in the interval $[a_d, b_d - \tau_d]$ in line 6. The finishing time of demand d' is set in line 7. Demand d' is inserted in set D_{SLD} in line 8. In line 10, demands $d_1, \ldots, d_m$ are renumbered in non-increasing order of their number of requested wavelengths. Ties are broken by first picking the demand with the largest shortest path from its source to its destination. Algorithm DP_RWA_SLD is applied in line 11 to the modified set of demands D_{SLD}. Line 12 returns the number of wavelengths needed, the wavelengths assigned to the demands, and the lightpaths used to route the demands.

Algorithm 2 DP_RWA_SSLD($G, D = D_{SSLD}$)

1: $D_{SLD} \leftarrow \emptyset$
2: **for all** $d \in D$ **do**
3: $\quad s_{d'} \leftarrow s_d$
4: $\quad t_{d'} \leftarrow t_d$
5: $\quad n_{d'} \leftarrow n_d$
6: $\quad$ Randomly select $a_{d'} \in [a_d, b_d - \tau_d]$
7: $\quad b_{d'} \leftarrow a_{d'} + \tau_d$
8: $\quad D_{SLD} \leftarrow D_{SLD} \cup \{d'\}$
9: **end for**
10: Renumber demands $d_1, \ldots, d_m$ in non-increasing order of their number of requested wavelenghts; in case of ties first pick the demand with the largest shortest path from its source to its destination.
11: $\lambda, (W_d, d = 1, \ldots, m), (Path_d, d = 1, \ldots, m) \leftarrow$ DP_RWA_SLD(G, D_{SLD})
12: **return** λ, $(W_d, d = 1, \ldots, m)$, $(Path_d, d = 1, \ldots, m)$

4.2 Decoder DECODER-SSLD

Every solution of the sliding scheduled lightpath demands problem (RWA-SSLD) is represented by a set of $m = |D_{SSLD}|$ random keys. Each key is a number in the interval $[0, 1)$, which is associated with a vertex of the graph. Each vector of random keys is decoded by an algorithm that receives the keys as inputs and constructs a feasible solution to RWA-SSLD: the decoder returns the number of wavelengths needed, the wavelengths assigned to the demands, and the lightpaths used to route the demands. The random keys are used to establish the starting times of the demands and interfere with the order in which they are renumbered by the decoder.

Decoder DECODER-SSLD to RWA-SSLD, whose pseudo-code is given by Algorithm 3, is based on the heuristic DP_RWA_SSLD. It receives the random keys $\alpha_d \in [0, 1), \forall d \in D_{SSLD}$, as additional parameters. The only adaptations with respect to the pseudo-code of the heuristic in Algorithm 2 appear in lines 6 and 10. By setting $a_{d'} = a_d + (b_d - \tau_d - a_d) \cdot \alpha_d$ in line 6, the starting time a'_d of demand d' can be set as small as a_d (in case the random key $\alpha_d = 0$) and very close to $b_d - \tau_d$ (in case α_d is close to one). In line 10, the random keys are used as a tie breaking criterion as the demands are renumbered.

Algorithm 3 DECODER-SSLD$(G, D = D_{SSLD}, \alpha)$

1: $D_{SLD} \leftarrow \emptyset$
2: **for all** $d \in D$ **do**
3: $\quad s_{d'} \leftarrow s_d$
4: $\quad t_{d'} \leftarrow t_d$
5: $\quad n_{d'} \leftarrow n_d$
6: $\quad a_{d'} \leftarrow a_d + (b_d - \tau_d - a_d) \cdot \alpha_d$
7: $\quad b_{d'} \leftarrow a_{d'} + \tau_d$
8: $\quad D_{SLD} \leftarrow D_{SLD} \cup \{d'\}$
9: **end for**
10: Renumber demands $d_1, \ldots, d_m$ in non-increasing order of their number of requested wavelenghts; in case of ties first pick the demand with the largest random key.
11: $\lambda, (W_d, d = 1, \ldots, m), (Path_d, d = 1, \ldots, m) \leftarrow$ DP_RWA_SLD(G, D_{SLD})
12: **return** λ, $(W_d, d = 1, \ldots, m)$, $(Path_d, d = 1, \ldots, m)$

5 Numerical experiments

In this section, we evaluate the performance of the biased random-key genetic algorithm for RWA-SSLD, based on the decoder DECODER-SSLD described in Section 4.2.

The algorithm was implemented in C++ with the GNU GCC compiler C/C++ version 5.4.0. The experiments were performed on a i7-6500U processor with a 2.50 GHz clock and 8 GB of RAM running the operating system Linux Ubuntu 16.04 LTS with parallel processing features disabled.

5.1 Instance generation

Data from different randomly generated test problems that have been used to evaluate algorithms previously developed for variants of RWA-SLD and RWA-SSLD [4,18,20,22,24,41,45,46] are not available from the literature.

Each problem instance is defined by a network topology (the graph G) and by a set of sliding scheduled demands (the set D_{SSLD}). The test instances considered in this work use the same graphs provided by Noronha, Resende and Ribeiro [29] and described in Table 2. The demands have been generated by the original source code developed by Kuri et al. [22], which was also used in [41]. The code made available by the authors has a parameter $C \in [0, 1]$ that establishes the time correlation between the demands. A time correlation close to 0 means that the demands are weakly time correlated, while a time correlation close to 1 means that the demands are strongly time correlated. Smaller time correlations produce sets of demands with less overlapping.

Table 2 Network topologies used in the instances generated for the numerical experiments.

Topology	Nodes	Edges
ATT	90	274
Brazil	27	140
EON	20	78
Finland	31	102
NSF	14	42

Each demand d generated for RWA-SLD by the code in [22] is transformed into a SSLD demand d' by the creation of a sliding window from its starting and finishing times a_d and b_d, respectively. The sliding window for the corresponding SSLD demand d' is $[a_{d'}, b_{d'}]$, with $a_{d'} = a_d - \gamma_1(b_d - a_d)$ and $b_{d'} = b_d + \gamma_2(b_d - a_d)$, with γ_1 and γ_2 randomly selected in the interval $[0, B]$.

The computational experiments reported in the next section have been performed on 60 randomly generated test instances: for each of the five network topologies described in Table 2, 12 instances have been generated with the number of demands $m \in [100, 10000]$, the time correlation $C \in [0.600, 0.994]$, and $B \in \{2, 4, 10\}$.

5.2 Parameter tuning

The best parameter settings for algorithm BRKGA-SSLD have been determined using the automatic tuning tool IRACE [3,25,32]. We investigated the best settings for the following parameters: p (population size), pe (fraction of the population corresponding to the elite subset), pm (fraction of the population corresponding to the mutant subset), and $rhoe$ (probability that a solution generated by crossover inherits its keys from the best parent).

Table 3 Test instances (15) used in the tuning experiment with IRACE.

Instance	Demands
ATT100.C990.B2	100
ATT1000.C900.B2	1000
ATT8000.C950.B10	8000
Brazil100.C980.B2	100
Brazil2000.C900.B10	2000
Brazil4000.C990.B4	4000
EON500.C900.B2	500
EON1000.C950.B4	1000
EON10000.C990.B10	10000
Finland100.C900.B2	100
Finland1000.C600.B2	1000
Finland3000.C994.B10	3000
NSF200.C900.B4	200
NSF2000.C950.B2	2000
NSF8000.C990.B10	8000

The IRACE tuning experiment automatically performed 1000 runs [7,27] of BRKGA-SSLD on 15 additional problem instances generated as described in Section 5.1 and listed in Table 3. Each of these runs was limited to one hour and we tested the value ranges suggested by Gonçalves and Resende [13]. The value ranges used by IRACE and the best values obtained for each of the parameters are reported in Table 4.

Table 4 Value ranges used by IRACE and best parameter values.

Parameter	Value ranges	BRKGA-SSLD
p	50, 51, ..., 100	83
pe	0.10, 0.11, ..., 0.25	0.17
pm	0.10, 0.11, ..., 0.30	0.20
$rhoe$	0.50, 0.51, ..., 0.80	0.61

5.3 Numerical results

We first compared the proposed algorithm BRKGA-SSLD with a straightforward multistart heuristic for RWA-SSLD implemented by Algorithm 4, which receives as inputs the graph $G = (V, E)$ representing the network and the set $D = D_{SSLD}$ of demands $d_1, \ldots, d_m$. Once again, its outputs are the number λ of wavelengths needed to route all demands, as well as the set W_d of wavelengths that may be used by demand d and the set of edges $Path_d$ in the route assigned to demand d, for every $d \in D$. The multistart heuristic MDP_RWA_SSLD repeatedly applies the randomized heuristic DP_RWA_SSLD, until some stopping criterion is reached. The number of consecutive iterations without improvement and the number of wavelengths needed to route all demands are initialized in lines 1 and 2, respectively. The

loop in lines 3–15 implements the stopping criterion. Line 4 performs a call to the randomized heuristic DP_RWA_SSLD. If the number λ' of wavelengths in the new solution does not improve the number λ of wavelengths in the incumbent, then the number of iterations without improvement is updated in line 6. Otherwise, the number of iterations without improvement is reset to zero in line 8 and the incumbent is updated in lines 9–13. Line 16 returns the number of wavelengths needed, the wavelengths assigned to the demands, and the lightpaths used to route the demands in the best solution found.

Algorithm 4 MDP_RWA_SSLD($G, D = D_{SSLD}$)

```
1: niter ← 0
2: λ ← ∞
3: while stopping criterion is not reached do
4:    λ', (W'_d, d = 1,...,m), (Path'_d, d = 1,...,m) ← DP_RWA_SSLD(G, D)
5:    if λ' ≥ λ then
6:       niter ← niter + 1
7:    else
8:       niter ← 0
9:       λ ← λ'
10:      for all d ∈ D do
11:         W_d ← W'_d
12:         Path_d ← Path'_d
13:      end for
14:   end if
15: end while
16: return λ, (W_d, d = 1,...,m), (Path_d, d = 1,...,m)
```

We considered the 60 test instances generated in Section 5.1 for the comparative evaluation of heuristic BRKGA-SSLD proposed in this work with the MDP_RWA_SSLD multistart heuristic. We observe that these test instances do not contain any of the instances considered in the tuning experiment. Ten independent runs using different seeds were performed for each algorithm. Algorithm MDP_RWA_SSLD was made to stop after 1000 iterations without improvement in the best known solution. For each test instance, we used the average time taken by algorithm MDP_RWA_SSLD as the stopping criterion for BRKGA-SSLD. Therefore, the same stopping criterion is applied to both heuristics.

For each instance, Tables 5 and 6 present the best and average solution values over ten runs of each algorithm. The last column in each of these tables shows the computation time for MDP_RWA_SSLD, which was also taken as the stopping criterion for BRKGA-SSLD. Cells in boldface show the algorithms that obtained the best results for each instance. These results show that BRKGA-SSLD obtained systematically better solutions than MDP_RWA_SSLD. While MDP_RWA_SSLD found a solution for only one instance (EON8000.C950.B10) that was not matched by BRKGA-SSLD, the latter found unmatched solutions by MDP_RWA_SSLD for 29 out of the 60 test problems. Similarly, MDP_RWA_SSLD found the best average solution

Table 5 Results for algorithms BRKGA-SSLD and MDP_RWA_SSLD - Part I.

		MDP_RWA_SSLD		BRKGA-SSLD		Running
Instance	Demands	Best	Average	Best	Average	time (s)
ATT100.C800.B10	100	**12**	**12.00**	**12**	**12.00**	1.507
ATT100.C900.B2	100	**14**	**14.00**	**14**	**14.00**	1.507
ATT500.C800.B2	500	13	13.10	**12**	**12.80**	12.296
ATT500.C900.B4	500	14	14.00	**13**	**13.90**	14.248
ATT500.C950.B10	500	**15**	15.80	**15**	**15.10**	11.582
ATT1000.C800.B4	1000	14	14.10	**13**	**13.40**	27.615
ATT1000.C990.B10	1000	24	24.50	**23**	**23.30**	32.726
ATT2000.C500.B2	2000	**15**	**15.00**	**15**	**15.00**	57.273
ATT2000.C900.B4	2000	19	19.00	**18**	**18.90**	56.808
ATT4000.C900.B4	4000	**19**	**19.00**	**19**	**19.00**	177.383
ATT8000.C990.B10	8000	**30**	30.70	**30**	**30.00**	839.191
ATT10000.C994.B10	10000	**30**	30.70	**30**	**30.20**	1189.970
Brazil200.C980.B2	200	**10**	**10.00**	**10**	**10.00**	1.390
Brazil400.C980.B2	400	**13**	**13.00**	**13**	**13.00**	3.289
Brazil600.C980.B2	600	**12**	**12.10**	**12**	12.40	5.950
Brazil800.C980.B2	800	17	17.00	**14**	**16.50**	8.090
Brazil1000.C950.B10	1000	11	11.60	**10**	**11.00**	15.222
Brazil2000.C950.B10	2000	12	12.80	**11**	**11.90**	46.812
Brazil3000.C950.B10	3000	**15**	15.70	**15**	**15.00**	107.099
Brazil4000.C950.B10	4000	13	13.00	**12**	**12.50**	146.872
Brazil5000.C950.B10	5000	**18**	**18.00**	**18**	**18.00**	208.167
Brazil6000.C950.B10	6000	**15**	**15.00**	**15**	**15.00**	314.619
Brazil7000.C950.B10	7000	17	17.00	**16**	**16.90**	405.849
Brazil8000.C950.B10	8000	20	20.00	**16**	**16.10**	546.530
EON200.C900.B10	200	**10**	**10.00**	**10**	**10.00**	1.010
EON200.C990.B2	200	16	16.70	**15**	**15.40**	1.659
EON1000.C950.B10	1000	**13**	**13.00**	**13**	**13.00**	10.470
EON1000.C990.B2	1000	24	24.00	**22**	**23.30**	12.980
EON4000.C950.B10	4000	**17**	17.10	**17**	**17.00**	169.986
EON5000.C994.B4	5000	22	22.40	**21**	**21.70**	254.200
EON6000.C990.B10	6000	**24**	25.10	**24**	**24.40**	435.820
EON7000.C990.B10	7000	24	24.40	**23**	**23.50**	486.826
EON8000.C950.B10	8000	**19**	**19.20**	20	20.00	669.642
EON8000.C994.B4	8000	**24**	25.00	**24**	**24.70**	623.265
EON9000.C990.B10	9000	**25**	25.60	**25**	**25.10**	924.462
EON10000.C994.B10	10000	**25**	26.10	**25**	**25.70**	997.888
Finland100.C950.B4	100	**10**	10.30	**10**	**10.00**	0.915
Finland200.C950.B4	200	**17**	**17.00**	**17**	**17.00**	1.365
Finland300.C950.B4	300	14	14.00	**13**	**13.90**	3.278
Finland400.C950.B4	400	14	14.20	**13**	**13.00**	4.150
Finland1000.C950.B4	1000	16	16.00	**15**	**15.90**	12.678
Finland1000.C990.B2	1000	18	18.60	**17**	**17.20**	16.293

values for only two instances (Brazil600.C980.A2 and EON8000.C950.A10), while MDP_RWA_SSLD found the best averages for 43 instances. Table 7 summarizes the following results from the above tables:

– *#Best*: number of test instances where some algorithm obtained the best value (the higher this value, the better is the corresponding algorithm).

Table 6 Results for algorithms BRKGA-SSLD and MDP_RWA_SSLD - Part II.

		MDP_RWA_SSLD		BRKGA-SSLD		Running
Instance	Demands	Best	Average	Best	Average	time (s)
Finland2000.C950.B4	2000	18	18.00	**17**	**17.30**	38.067
Finland2000.C990.B2	2000	22	22.00	**21**	**21.80**	38.822
Finland3000.C990.B2	3000	**23**	23.70	**23**	**23.30**	88.459
Finland4000.C990.B2	4000	24	24.00	**23**	**23.40**	167.492
Finland8000.C994.B10	8000	25	25.00	**24**	**24.20**	542.394
Finland10000.C994.B10	10000	**25**	**25.90**	**25**	**25.90**	1041.525
NSF200.C950.B2	200	15	15.50	**14**	**14.40**	1.043
NSF200.C980.B10	200	**16**	**16.00**	**16**	**16.00**	1.116
NSF300.C900.B4	300	**17**	**17.00**	**17**	**17.00**	1.271
NSF300.C990.B2	300	24	24.10	**22**	**22.00**	1.915
NSF500.C900.B4	500	**17**	**17.00**	**17**	**17.00**	2.740
NSF800.C990.B4	800	25	25.70	**24**	**24.80**	8.138
NSF1000.C880.B10	1000	20	20.00	**16**	**16.90**	8.460
NSF2000.C920.B2	2000	**17**	17.70	**17**	**17.50**	37.105
NSF4000.C990.B2	4000	**30**	30.80	**30**	**30.00**	142.126
NSF6000.C980.B4	6000	28	28.00	**26**	**26.40**	321.744
NSF8000.C994.B4	8000	29	29.50	**28**	**28.80**	567.039
NSF10000.C994.B10	10000	**31**	31.90	**31**	**31.50**	819.027

Table 7 Comparative results for the multistart heuristic and the biased random-key genetic algorithm.

	MDP_RWA_SSLD	BRKGA-SSLD
#Best	31	**59**
#BestM	17	**58**
SumBest	202	**374**
Avgdevrun (%)	3.83	**1.49**
Avgdev (%)	4.04	**0.09**
AvgdevM (%)	3.10	**0.11**

- *#BestM*: number of test instances where some algorithm obtained the best average value (the higher this value, the better is the corresponding algorithm).
- *SumBest*: number of runs where some algorithm obtained the best value (the higher this value, the better is the corresponding algorithm).
- *Avgdevrun*: average relative deviation between the value obtained by some algorithm over all runs of some instance and the best value obtained (for this instance) considering all tested algorithms (the smaller this value, the better is the corresponding algorithm).
- *Avgdev*: average relative deviation between the best value obtained by some algorithm for some instance and the best value obtained (for this instance) considering all tested algorithms (the smaller this value, the better is the corresponding algorithm).
- *AvgdevM*: average relative deviation between the average value obtained by some heuristic for some instance and the best average value obtained (for this instance) considering all tested algorithms (the smaller this value, the better is the corresponding algorithm).

The results summarized in Table 7 make the case for BRKGA-SSLD. They show that BRKGA-SSLD consistently obtained the best results for all criteria considered, finding 59 best solution values and 58 best average values out of the 60 instances. BRKGA-SSLD was also the heuristic that found the solutions leading to the smallest values for the results reporting average relative deviations from the best values.

In the following, we compare the runtime distributions (also known as time-to-target plots or ttt-plots) of heuristics BRKGA-SSLD and MDP_RWA_SSLD. These plots display on the vertical axis the probability that some algorithm will find a solution at least as good as a given target within some given processing time, which appears in the abscissa axis. Time-to-target plots have also been claimed in [15] as a very effective tool to characterize the behavior of running times of stochastic local search algorithms. In the experiment reported in the sequel, the two heuristics are made to stop whenever a solution with cost smaller than or equal to a given target is found. The target is set as the cost of the best known solution for the instance. Each heuristic was run 200 times, with different initial seeds for the pseudo-random number generator, and the empirical probability distribution of the time taken by each heuristic to find a solution at least as good as the target is plot. We followed the methodology proposed by Aiex et al. [1,2] to build the empirical distributions. A probability $p_i = (i - \frac{1}{2})/200$ is associated with the i-th smallest running time t_i and the points (t_i, p_i) are plotted, for $i = 1, \ldots, 200$. The more to the left is a plot, the better is the corresponding algorithm.

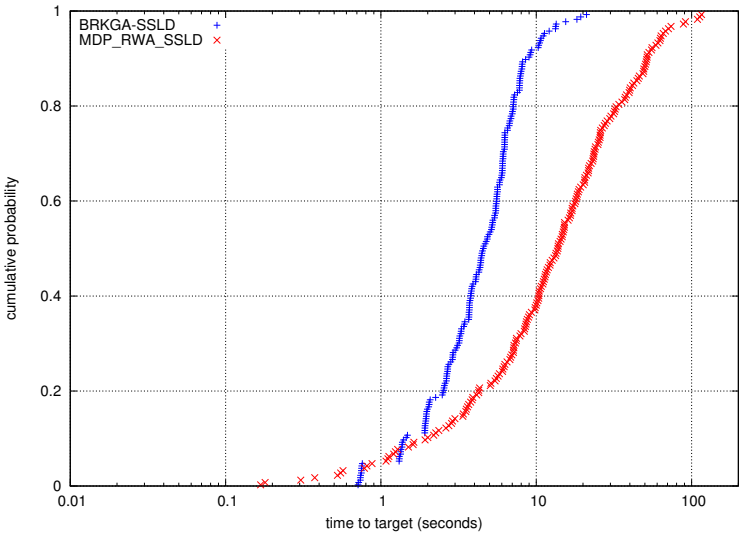

Fig. 6 Runtime distributions for instance ATT500.C950.B10 with target set at 15.

Time-to-target plots for instance ATT500.C950.B10 are displayed in Figure 6, with the target set to 15 (best solution value obtained by all heuristics

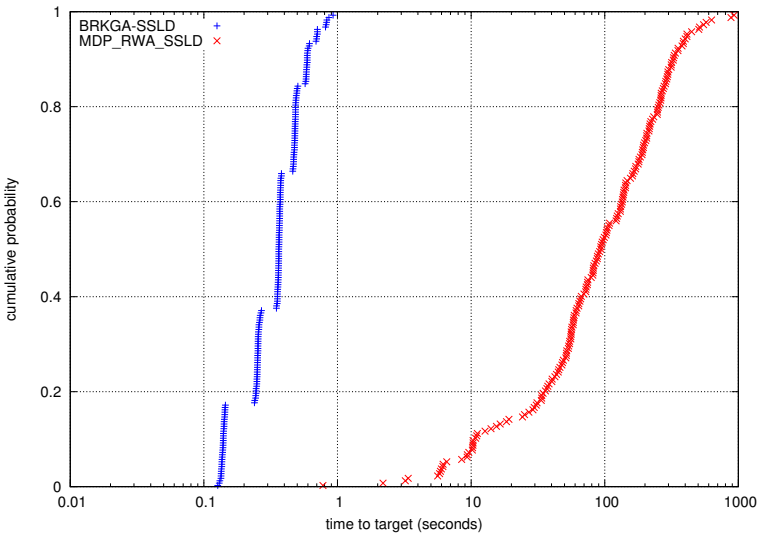

Fig. 7 Runtime distributions for instance NSF300.C990.B2 with target set at 23.

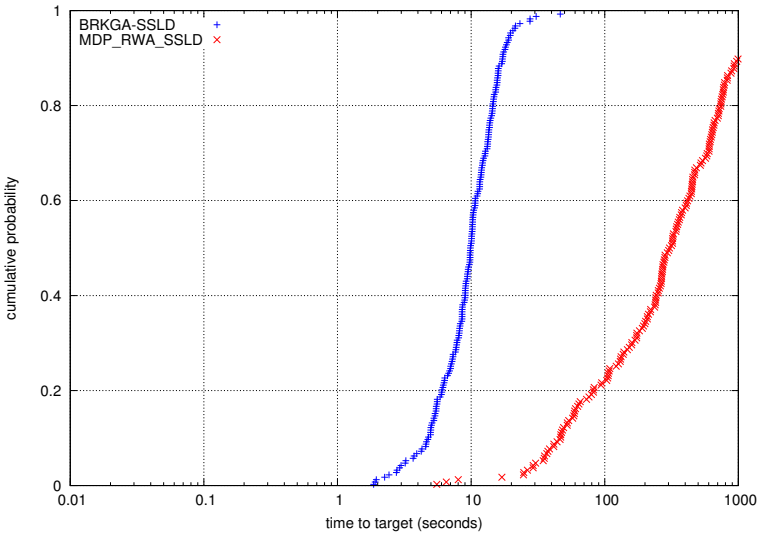

Fig. 8 Runtime distributions for instance NSF800.C990.B4 with target set at 24.

in Table 5). Plots for instance NSF300.C990.B2 appear in Figure 7, with the target set to 23 (average between the solution values obtained by heuristics BRKGA-SSLD and MDP_RWA_SSLD in Table 6). The ttt-plots for instance NSF800.C990.B4 displayed in Figure 8, with the target set to 24 (best value found by BRKGA-SSLD in Table 6) show that MDP_RWA_SSLD failed to reach the target within the time limit in 19 runs. The ttt-plots show that,

given the same running time, BRKGA-SSLD is capable to find with higher probability than MDP_RWA_SSLD solutions that are as good as the targets.

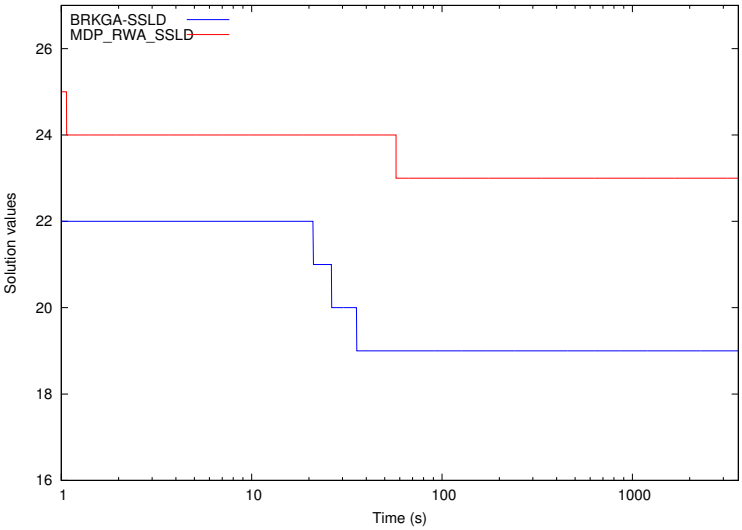

Fig. 9 Evolution of the best solution value obtained by BRKGA-SSLD and MDP_RWA_SSLD along the 3600 first seconds of running time for instance NSF300.C990.B2: best value found by BRKGA-SSLD is 19.

Figures 9 and 10 show how the best solutions encountered by BRKGA-SSLD and the multistart heuristic evolve along the first 3600 seconds of running time for each of the instances NSF300.C990.B2 and NSF800.C990.B4, respectively. They show that BRKGA-SSLD obtains better solutions faster than the multistart algorithm. It is capable to continuously evolve the solution population and to improve the best value. The best solution found by BRKGA-SSLD is better than that obtained by MDP_RWA_SSLD anytime along these runs.

Restart strategies are very effective in order to reduce the running times to reach target values. We hybridized heuristic BRKGA-SSLD with restart(κ) strategy first proposed in [35,37]. In this context, the population is completely renewed after that κ generations have been performed without improvement in the best solution obtained, with κ being a new parameter. The performance of strategy restart(κ) was evaluated for $\kappa = 100$, 200, and 500.

Figure 11 displays runtime distributions for all variants of BRKGA-SSLD, with and without restarts, on instance NSF300.C990.B2. For each strategy restart(κ), 200 runs have been performed. The running time of each run was limited to 1000 seconds and the target value was set to 20. The figure shows that BRKGA-SSLD without restarts failed to reach the target in 21 runs. Figure 12 displays runtime distributions for all variants of BRKGA-SSLD, with and without restarts, on instance NSF800.C990.B4. The running time of

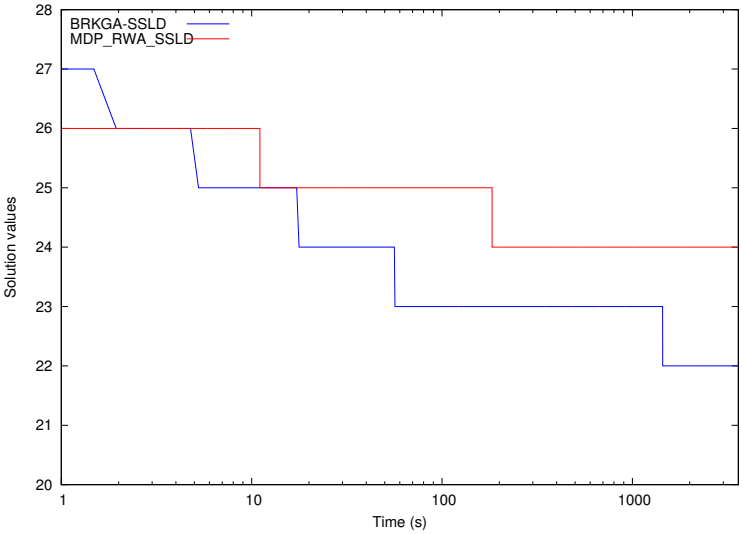

Fig. 10 Evolution of the best solution value obtained by BRKGA-SSLD and MDP_RWA_SSLD along the 3600 first seconds of running time for instance NSF800.C990.B4: best value found by BRKGA-SSLD is 22.

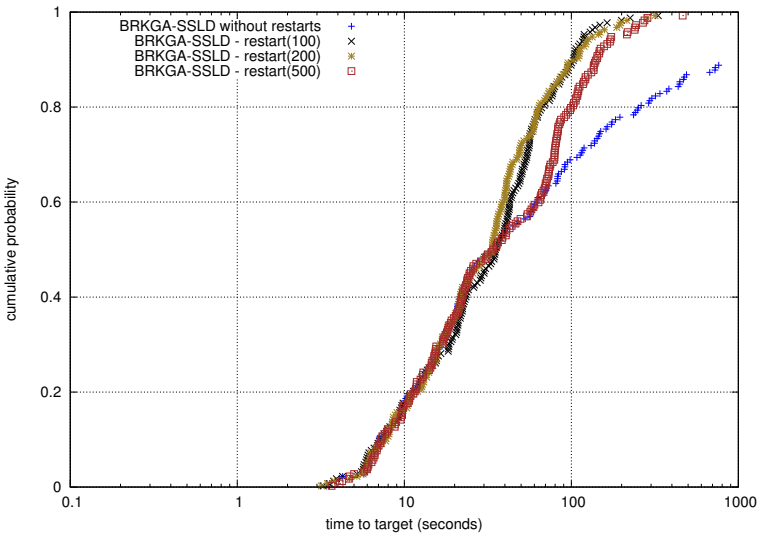

Fig. 11 Runtime distributions for heuristic BRKGA-SSLD with restart(κ) strategies on instance NSF300.C990.B2 with target set at 20.

each run was limited to 10000 seconds and the target value was set to 22. The figure shows that BRKGA-SSLD without restarts failed to reach the target in 18 runs. For these two instances, strategies restart(100) and restart(200) presented the best results, corresponding to the leftmost plots.

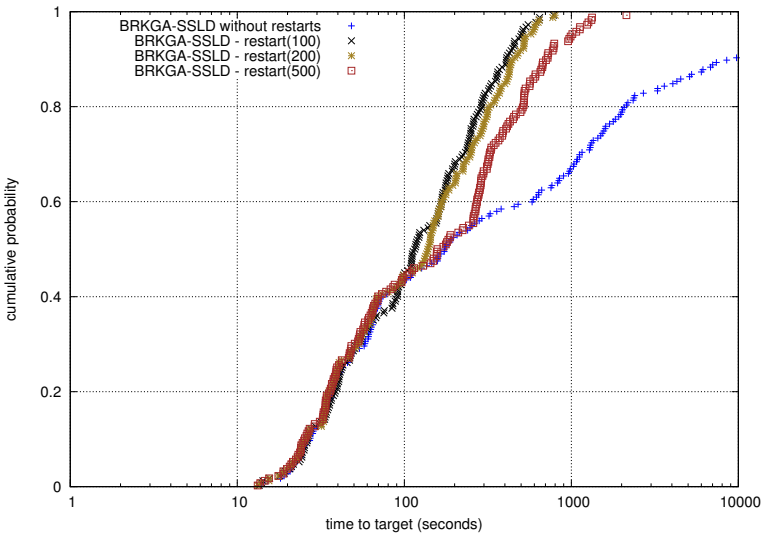

Fig. 12 Runtime distributions for heuristic BRKGA-SSLD with restart(κ) strategies on instance NSF800.C990.B4 with target set at 22.

Table 8 Restart strategies on instance NSF800.C990.B4: compared to the strategy without restarts, strategies restart(100) and restart(200) reduced the average running time to less than 11% of the value without restarts.

	Average running times in quartile (seconds)				
Strategy	1st	2nd	3rd	4th	average
BRKGA-SSLD without restarts	29.45	82.88	713.29	5984.37	1702.50
BRKGA-SSLD with restart(100)	29.63	77.29	<u>178.36</u>	<u>391.49</u>	<u>169.19</u>
BRKGA-SSLD with restart(200)	28.51	<u>76.61</u>	194.44	442.77	185.58
BRKGA-SSLD with restart(500)	<u>28.21</u>	77.90	278.58	730.27	278.74

Resende and Ribeiro [35,37] observed that the effect of restart strategies is mainly noticed in the longest runs. Table 8 illustrates the behavior of the restart strategies on instance NSF800.C990.B4, considering 200 runs of each variant, the target set at 22, and a time limit of 10000 seconds. We focus on the columns of this table corresponding to the third and fourth quartiles, whose entries correspond to running times in the heavy tails of the runtime distributions. The restart strategies strongly affect mostly these two quartiles, which is a desirable characteristic. Compared to the strategy without restarts, strategies restart(100) and restart(200) were both able to reduce the average running time to less than 11% of the original value. The best results for each quartile are underlined. Algorithm BRKGA-SSLD with strategy restart(100) outperformed the other variants tested, with the smallest average running times. In the fourth quartile, BRKGA-SSLD without restarts failed to reach the target in 18 runs.

The results of these experiments show that BRKGA-SSLD together with restart(100) or restart(200) strategies performed the best over all tested vari-

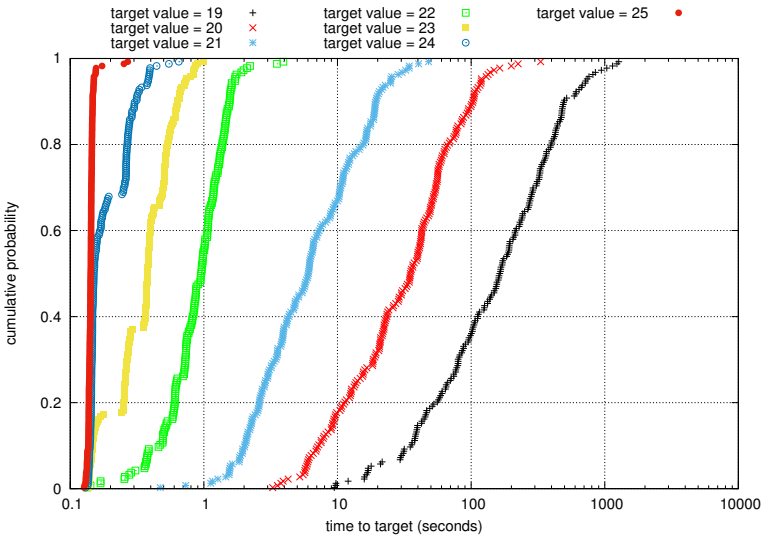

Fig. 13 Runtime distributions for heuristic BRKGA-SSLD with restart(100) strategy on instance NS300.C990.B2 as the target value ranges from 19 to 25.

ants, illustrating that restart strategies are capable to reduce the execution times of long runs of metaheuristics for optimization problems, although their use has been systematically neglected in the literature.

To conclude the experiments in this section, Figure 13 displays ttt-plots for BRKGA-SSLD with strategy restart(100) on instance NSF300.C990.B2 with the running time limited to 10000 seconds as the target ranges from 19 to 25. Figure 14 shows the average running time (in seconds) over 200 runs for each target value. We observe that, as the target decreases and the problem gets harder, the search strategy takes longer to reach the target.

5.4 Comparing BRKGA-SSLD with MCDF-CR

In the final experiment, we compare the proposed heuristic BRKGA-SSLD with the MCDF-CR algorithm from [24]. Comparisons with other algorithms are not possible, either because they handle different problem variants or because there is not enough information available about the experiments reported in the literature.

Since the source or executable codes of algorithm MCDF-CR, as well as the test problems for the numerical experiments reported in [24] were not available from the authors, we randomly generated 100 test instances, divided into 10 groups, with the number of demands m ranging from 10 to 100, using the same parameters described in [24].

Always following Liu et al. [24], and in order to reproduce as closely as possible the same conditions tested in their experiments, we used the same US

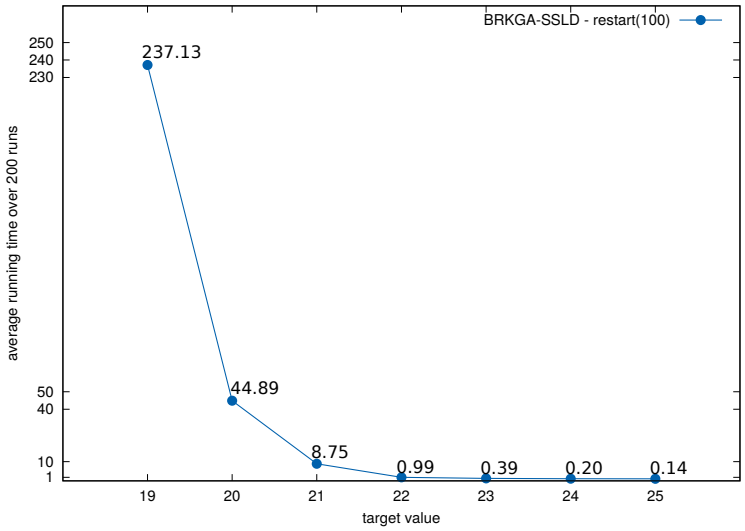

Fig. 14 Average running times (seconds) for heuristic BRKGA-SSLD with restart(100) strategy on instance NSF300.C990.B2 as the target ranges from 19 to 25.

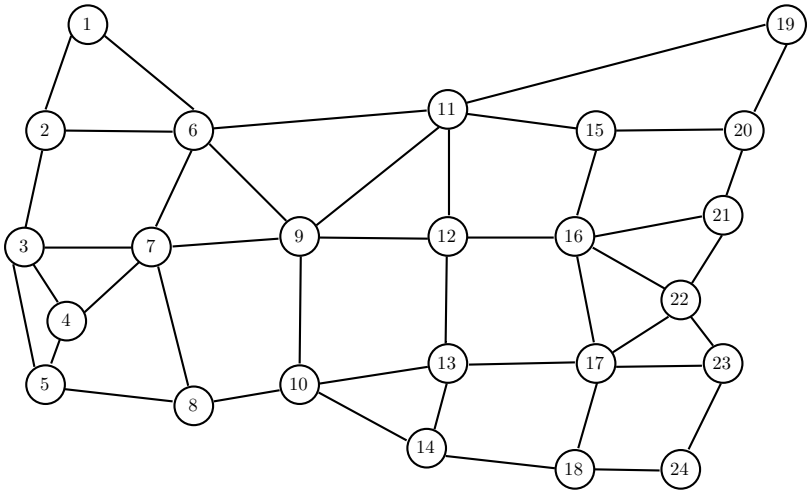

Fig. 15 US nationwide topology with 24 nodes and 43 edges.

nationwide topology with 24 nodes and 43 edges [24] depicted in Figure 15. We considered a 24-hour period sliced into 48 time slots of 30 minutes each. The source node s_d and the destination node t_d of each demand d were randomly selected. The starting and finishing times of the sliding window of this demand were defined as $a_d = \min(t_1, t_2)$ and $b_d = \max(t_1, t_2)$, respectively, with t_1 and t_2 randomly generated in the interval $[0, 48]$, while its duration τ_d was randomly generated in the interval $[0.5 \times (b_d - a_d), (b_d - a_d)]$.

Table 9 Average number of wavelengths over ten instances with the same number of demands for algorithms MCDF-CR [24] and BRKGA-SSLD (running for one, ten, and 100 generations).

		BRKGA-SSLD		
Demands	MCDF-CR	1 generation	10 generations	100 generations
10	1.71	1.20	1.20	1.20
20	2.64	1.52	1.42	1.40
30	3.43	1.92	1.71	1.63
40	4.13	2.20	2.06	2.00
50	5.04	2.31	2.25	2.09
60	5.67	2.91	2.78	2.67
70	6.40	3.16	3.06	2.91
80	7.09	3.39	3.18	3.10
90	7.34	3.87	3.63	3.39
100	8.24	4.62	4.18	4.10

In order to make the heuristic BRKGA-SSLD comparable with MCDF-CR, the right-hand side of the inequality in the test in line 15 of Algorithm 1 has to be changed from $\max\{diam_G, \sqrt{|E|}\}$ to $c_{s_d,t_d}(G) + \delta$, meaning that the length of any lightpath cannot be longer than the shortest path between its extremities by more than δ hops. We used $\delta = 1$, to reproduce the experiment in the work of Liu et al. [24].

Table 9 displays the average number of wavelengths calculated by algorithms MCDF-CR and BRKGA-SSLD. The number of wavelengths shown for MCDF-CR are average results obtained over 50 runs of the algorithm in [24]. Since the original numerical results were not available, we needed to extract data from the plot in Figure 8 of [24]. We used WebPlotDigitizer [11,38], a web-based plot digitizing tool, for extracting data from a variety of plots, including the coordinates of interrupted time-series data. The average results for BRKGA-SSLD are obtained over 100 runs of the heuristic for each problem size, corresponding to ten runs of each of the ten instances with the same number of demands. Results are given for BRKGA-SSLD running for one, ten, and 100 generations.

The results obtained by BRKGA-SSLD and MCDF-CR displayed in Table 9 are reproduced in Figure 16. This figure shows that BRKGA-SSLD clearly outperforms MCDF-CR, finding solutions that make use of roughly 50% of the number of wavelengths determined by MCDF-CR. In addition, we observe that the performance ratio between BRKGA-SSLD and MCDF-CR increases as the number of demands gets larger.

6 Concluding remarks

We developed a biased random-key genetic algorithm for approximately solving the problem of routing and wavelength assignment of sliding scheduled lightpath demands (RWA-SSLD). In this problem variant, each demand is characterized not only by a source and a destination, but also by a duration and a time window in which it has to be met.

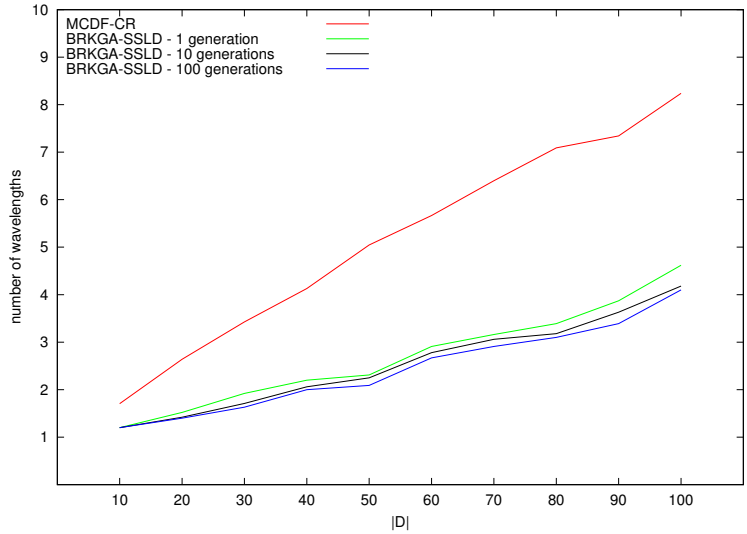

Fig. 16 Comparative results for algorithms BRKGA-SSLD and MCDF-CR [24].

The decoder for heuristic BRKGA-SSLD is based on a randomized extension of the adaptive greedy heuristic DP_RWA_SLD originally proposed by Skorin-Kapov [41] for solving a simpler variant of the problem (RWA-SLD). The latter is based on the adaptive greedy heuristic of Manohar et al. [26] for the original min-RWA problem without time constraints.

Although restart strategies are known to be capable to reduce the execution times of long runs of metaheuristics for optimization problems [35,37], their use has been systematically neglected in the literature. The numerical experiments also showed that the restart strategies gave a significant contribution to improve the robustness and the efficiency of the algorithm, since the BRKGA-SSLD algorithm combined with a restart strategy outperformed the original heuristic.

The results obtained by the biased random-key genetic algorithm are also compared with those obtained by Liu et al. [24], in a experiment reproducing as closely as possible the same conditions tested in their experiments. The new heuristic outperformed the algorithm in the literature, finding solutions that use roughly 50% of the number of wavelengths determined by the latter.

The numerical results supporting these findings illustrate the efficiency and the effectiveness of the BRKGA approach for solving the problem of routing and wavelength assignment of sliding scheduled demands, as well as other routing and scheduling problems [36]. In addition, we have made available through Mendeley [33] a complete data set of benchmark instances, with all the input data used in the numerical experiments performed in this work.

Acknowledgements The authors are grateful to J. Kuri for sending the Perl script for generating the test data. Work of Celso C. Ribeiro was partially supported by CNPq research

grant 303958/2015-4 and by FAPERJ research grant E-26/201.198/2014, and concluded during a visit to Laboratoire d'Informatique de Modélisation et d'Optimisation des Systèmes of Université Clermont Auvergne, France. This work was also partially sponsored by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Finance Code 001.

References

1. R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8:343–373, 2002.
2. R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro. TTTPLOTS: A Perl program to create time-to-target plots. *Optimization Letters*, 1:355–366, 2007.
3. P. Alfaro-Fernández, R. Ruiz, F. Pagnozzi, and T. Stützle. Exploring automatic algorithm design for the hybrid flowshop problem. In *12th Metaheuristics International Conference*, pages 201–203, Barcelona, 2017.
4. D. Andrei, H. H. Yen, M. Tornatore, C. U. Martel, and B. Mukherjee. Integrated provisioning of sliding scheduled services over WDM optical networks. *IEEE/OSA Journal of Optical Communications and Networking*, 1:94–105, 2009.
5. D. Banerjee and B. Mukherjee. A practical approach for routing and wavelength assignment in large wavelength-routed optical networks. *IEEE Journal on Selected Areas in Communications*, 14:903–908, 1996.
6. J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6:154–160, 1994.
7. S. Bouamama and C. Blum. A population-based iterated greedy algorithm for the knapsack problem with setup. In *12th Metaheuristics International Conference*, pages 558–565, Barcelona, 2017.
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. *International Transactions in Operational Research*, 22:823–839, 2015.
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. *International Transactions in Operational Research*, 24:1061–1077, 2016.
10. J. S. Brandão, T. F. Noronha, and C. C. Ribeiro. A biased random-key genetic algorithm to maximize the number of accepted lightpaths in WDM optical networks. *Journal of Global Optimization*, 65:813–835, 2016.
11. D. Drevon, S. R. Fursa, and A. L. Malcolm. Intercoder reliability and validity of WebPlotDigitizer in extracting graphed data. *Behavior Modification*, 41:323–339, 2017.
12. T. Erlebach and K. Jansen. The complexity of path coloring and call scheduling. *Theoretical Computer Science*, 255:33–50, 2001.
13. J. F. Gonçalves and M. G. C. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17:487–525, 2011.
14. J. F. Gonçalves, M. G. C. Resende, and R. F. Toso. Biased and unbiased random key genetic algorithms: An experimental analysis. In *Abstracts of the 10th Metaheuristics International Conference*, Singapore, 2013.
15. H. H. Hoos and T. Stützle. Evaluation of Las Vegas algorithms - Pitfalls and remedies. In G. Cooper and S. Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 238–245, Madison, 1998.
16. E. Hyytia and J. Virtamo. Wavelength assignment and routing in WDM networks. In *Fourteenth Nordic Teletraffic Seminar*, pages 31–40, Copenhagen, 1998.
17. R. R. Iraschko, M. H. MacGregor, and W. D. Grover. Optimal capacity placement for path restoration in STM or ATM mesh-survivable networks. *IEEE/ACM Transactions on Networking*, 6:325–336, 1998.
18. A. Jaekel and Y. Chen. Demand allocation without wavelength conversion under a sliding scheduled traffic model. In *Fourth International Conference on Broadband Communications, Networks and Systems*, pages 495–503, Raleigh, 2007.
19. A. Jaekel and Y. Chen. Resource provisioning for survivable WDM networks under a sliding scheduled traffic model. *Optical Switching and Networking*, 6:44–54, 2009.

20. A. Jaekel, J. Pare, Y. Chen, A. Shaabana, and F. Luo. Traffic grooming of scheduled demands for minimizing energy consumption. *Photonic Network Communications*, 29:151–163, 2015.
21. J. M. Kleinberg. *Approximation algorithms for disjoint paths problems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, 1996.
22. J. Kuri, N. Puech, M. Gagnaire, E. Dotaro, and R. Douville. Routing and wavelength assignment of scheduled lightpath demands. *IEEE Journal on Selected Areas in Communications*, 21:1231–1240, 2003.
23. G. Li and R. Simha. The partition coloring problem and its application to wavelength routing and assignment. In *Proceedings of the First Workshop on Optical Networks*, Dallas, 2000.
24. Z. Liu, W. Guo, Q. Shi, W. Hu, and M. Xia. Sliding scheduled lightpath provisioning by mixed partition coloring in WDM optical networks. *Optical Switching and Networking*, 10:44–53, 2013.
25. M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The IRACE package: Iterated race for automatic algorithm configuration, 2011. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium.
26. P. Manohar, D. Manjunath, and R. K. Shevgaonkar. Routing and wavelength assignment in optical networks from edge disjoint path algorithms. *IEEE Communications Letters*, 6:211–213, 2002.
27. J. Maschler, T. Hackl, M. Riedler, and G. R. Raidl. Enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem. In *12th Metaheuristics International Conference*, pages 118–127, Barcelona, 2017.
28. T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro. Efficient implementations of heuristics for routing and wavelength assignment. In C. C. McGeoch, editor, *Experimental Algorithms*, volume 5038 of *Lecture Notes in Computer Science*, pages 169–180. Springer, Berlin, 2008.
29. T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro. Instances for the routing and wavelength assignment problem. http://www2.ic.uff.br/~celso/grupo/rwa.html, 2010. Last accessed October 13, 2018.
30. T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro. A biased random-key genetic algorithm for routing and wavelength assignment. *Journal of Global Optimization*, 50:503–518, 2011.
31. T. F. Noronha and C. C. Ribeiro. Routing and wavelength assignment by partition colouring. *European Journal of Operational Research*, 171:797–810, 2006.
32. L. Pérez Cáceres, M. López-Ibáñez, and T. Stützle. An analysis of parameters of IRACE. In *Proceedings of the 14th European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 8600 of *Lecture Notes in Computer Science*, pages 37–48. Springer, Berlin, 2014.
33. B. Q. Pinto, C. C. Ribeiro, I. Rosseti, and T. F. Noronha. Input data and detailed numerical results for 'A biased random-key genetic algorithm for routing and wavelength assignment under a sliding scheduled traffic model in WDM optical networks'. https://data.mendeley.com/datasets/r76d3pjnnk/, 2018. Last accessed on October 17, 2018.
34. R. Ramaswami and K. N. Sivarajan. Routing and wavelength assignment in all-optical networks. *IEEE/ACM Transactions on Networking*, 3:489–500, 1995.
35. M. G. C. Resende and C. C. Ribeiro. Restart strategies for GRASP with path-relinking heuristics. *Optimization Letters*, 5:467–478, 2011.
36. M. G. C. Resende and C. C. Ribeiro. Biased-random key genetic algorithms: An advanced tutorial. In *Proceedings of the 2016 Genetic and Evolutionary Computation Conference - GECCO'16 Companion Volume*, pages 483–514, Denver, 2016. Association for Computing Machinery.
37. M. G. C. Resende and C. C. Ribeiro. *Optimization by GRASP*. Springer, Boston, 2016.
38. A. Rohatgi. *WebPlotDigitizer (Version 4.1)*, 2018. Last visit in October 6, 2018.
39. C. V. Saradhi and M. Gurusamy. Scheduling and routing of sliding scheduled lightpath demands in WDM optical networks. In *Conference on Optical Fiber Communication and the National Fiber Optic Engineers Conference*, pages 1–3, Anaheim, 2007.

40. C. V. Saradhi, M. Gurusamy, and R. Piesiewicz. Routing fault-tolerant sliding scheduled traffic in WDM optical mesh networks. In *5th International Conference on Broadband Communications, Networks and Systems*, pages 197–202, 2008.
41. N. Skorin-Kapov. Heuristic algorithms for the routing and wavelength assignment of scheduled lightpath demands in optical networks. *IEEE Journal on Selected Areas in Communications*, 24:2–15, 2006.
42. N. Skorin-Kapov. Routing and wavelength assigment in optical networks using bin packing based algorithms. *European Journal of Operational Research*, 177:1167–1179, 2007.
43. W. Spears and K. A. De Jong. On the virtues of parameterized uniform crossover. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, San Mateo, 1991. Morgan Kaufman.
44. R. F. Toso and M. G. C. Resende. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30:81–93, 2015.
45. B. Wang, T. Li, X. Luo, and Y. Fan. Traffic grooming under a sliding scheduled traffic model in WDM optical networks. In *IEEE Workshop on Traffic Grooming in WDM Networks*, San Jose, 2004.
46. B. Wang, T. Li, X. Luo, Y. Fan, and C. Xin. On service provisioning under a scheduled traffic model in reconfigurable WDM optical networks. In *2nd International Conference on Broadband Networks*, pages 13–22, Boston, 2005.
47. H. Zang, J. P. Jue, and B. Mukherjeey. A review of routing and wavelength assignment approaches for wavelength-routed optical wdm networks. *Optical Networks Magazine*, 1:47–60, 2000.