Some extensions of biased random-key genetic algorithms

Marina L. Lucena
Department of Informatics – Pontifical Catholic University of Rio de Janeiro
Rua Marquês de São Vicente, 225, CEP 22451-900, Rio de Janeiro, RJ, Brazil
marina.lucena@globo.com

Carlos E. Andrade*
Institute of Computing – University of Campinas
Av. Albert Einstein, 1251, CEP 13083-852, Campinas, SP, Brazil
andrade@ic.unicamp.br / ce.andrade@gmail.com

Mauricio G. C. Resende
AT&T Labs Research
200 Laurel Avenue, Middletown, NJ 07748, USA
mgcr@research.att.com

Flávio K. Miyazawa
Institute of Computing – University of Campinas
Av. Albert Einstein, 1251, CEP 13083-852, Campinas, SP, Brazil
fkm@ic.unicamp.br

ABSTRACT

In this paper, we propose two new variants of the Biased Random-Key Genetic Algorithm. In the first, the algorithm assigns a gender to each chromosome and only allows crossover between chromosomes of different gender. In the second variant, more than two parents are used to generate a new offspring. Computational experiments are conducted on instances of set covering and set packing problems. We empirically show that the multi-parent modification is able to overcome the original biased random-key genetic algorithm although it requires slightly more time and iterations to converge.

KEYWORDS. Genetic Algorithms, Crossover, Combinatorial optimization, Metaheuristics.

*Corresponding author.

1. Introduction

Biased random-key genetic algorithms (Gonçalves and Resende, 2011) are derived from the random-key genetic algorithm of Bean (1994), where each chromosome is usually a vector of randomly generated real numbers in the interval $[0, 1]$. This representation enables the algorithms to generate new individuals very easily and apply the genetic operators in a standard and well-defined way. This means that if one can write a function (decoder) to map a vector of real numbers into a solution for the problem on hand, there is no need to worry about how the genetic operators work or how the framework manages the population of chromosomes. Although some genetic parameters must be set, the overall application of the algorithm is relatively simple and straightforward. In fact, a freely-available application programming interface (API) for BRKGA has been developed (Toso and Resende, 2014).

Biased random-key genetic algorithms (BRKGAs) have been used with success both in several classical hard combinatorial optimization problems as well as on real-world problems. For example, Resende (2012) surveys applications of BRKGA in optimization problems arising in telecommunications. Gonçalves and Resende (2011, 2012, 2013) present BRKGAs for 2D and 3D packing and bin packing. BRKGAs have been applied to a number of scheduling problems, e.g. job-shop scheduling (Gonçalves et al., 2005; Gonçalves and Resende, 2014), and resource constrained project scheduling (Gonçalves et al., 2008; Mendes et al., 2009; Gonçalves et al., 2011). Andrade et al. (2013) solve the k-Interconnected Multi Depot Multi Traveling Salesman Problem with consists in creating a central cycle of nodes (main route or backbone) and, for each vertex of the central cycle, we must attach a secondary cycle (local route or network ring) that starts and ends in the vertex. In Andrade et al. (2014), the BRKGA is used to solve the Overlapping Correlation Clustering problem, where one must label objects such the error between their similarities and the labeling be minimized. In this problem, the objects can have several labels which may overlap. The problem models several problems with big data, as protein sequencing, mobile device trajectories, and set of messages and documents. The BRKGA was able to overcome the state of art algorithms for this problem.

In this paper we consider some internals of BRKGAs and propose two modifications aimed to improve the quality of solutions obtained by the algorithms. The first proposal is to define a gender for each chromosome in such way that, during the crossover, only individuals of different gender are allowed to recombine. In the second proposal, several parents are chosen from the population and ranked according to a given weighting function. Then, parameterized uniform crossover is performed such that each allele is taken from a individual chosen by the roulette method. We use instances of the set covering and set packing problem to illustrate our proposals, using the same decoding process for all variants. Computational results are presented. The structure of this paper is the following: Section 2 reviews the fundamentals of BRKGAs. Section 3 presents the BRKGA with gender definition and multi-parents. In Section 4, computational results are presented, and final comments are made in Section 5.

2. Basic BRKGA

Before we discuss the origins and main characteristics of BRKGAs, let us recall some terminology of genetic algorithms: the *chromosome* $\mathbf{c}$ is vector in a certain space $\mathcal{H}^n$ such that n is the number of components. Each component $\mathbf{c}_i$, for $i = 1, \ldots, n$, is

called *gene* and its value is called *allele*. The genetic algorithm keeps a pool of vectors called *population* and for each chromosome in this population, a *fitness function* is applied to evaluate these chromosomes. In general, the fitness function constructs a solution from the chromosome and calculates its value. These values are used as *fitness* measures of each individual. The evolutionary step consists in building a new population by combining individuals of the current population, in general, selecting alleles from them to create the offspring. An additional step, called *mutation*, is applied with low probability when an allele is chosen and modified randomly. There is a large number of variations of this process and a comprehensive list can be found in Goldberg (1989).

The first random-key genetic algorithm (RKGA) is due to Bean (1994). Bean applied this algorithm to machine scheduling problems where one seeks a sequence of machines to perform operations such that the completion time of the last operation is minimized. The RKGA is an elitist genetic algorithm based on a population of vectors $\mathbf{v} \in [0,1]^n$. This population has size p. The evolutionary process and generation of a new population consists of the following steps:

1. Copy p_e best individuals to the population of the next generation;
2. Introduce p_m randomly generated individuals in the population of the next generation (Bean calls these individuals "immigrants");
3. Until the remaining $p - p_e - p_m$ spots in the new popuation are not filled:
   (a) Select two individual uniformly at random from entire previous population;
   (b) Generate a new individual using parameterized uniform crossover (Spears and DeJong, 1991): For each allele, a coin is tossed and an allele is copied from one of the parents according to the outcome of the coin toss.

The most interesting and important characteristic of the RKGAs is the crossover between individuals. As opposed to traditional genetic algorithms, the RKGA performs the crossover without caring about the feasibility of the solutions generated for the new individuals. This is possible because of the standard chromosome encoding that is responsible to guarantee the feasibility of the decoding function. The decoding function or *decoder* $d : [0,1]^n \rightarrow \mathcal{S}$ maps the real vector with a valid solution in the solution space $\mathcal{S}$. In some problems where feasibility is hard to achieve, the decoder may generate invalid solutions, making use of a penalty factor. Also, the RKGAs have no mutation in chromosomes, but instead they introduce random individuals in the new population called immigrants or *mutants*. Mutants play the same role as mutation in traditional genetic algorithms.

The BRKGA was introduced in Gonçalves and Almeida (2002) and Ericsson et al. (2002) and follows the same structure of RKGAs with respect to the population and introduction of mutants. Algorithm 1 depicts the main loop. The main differences between them are twofold: First with respect to how individuals are chosen for crossover; and second, how this crossover is done. In a RKGA, as the individuals are chosen at random from the entire population, there is the possibility of choosing two chromosomes with bad fitness and thus generate a poor-quality offspring. In a BRKGA, the population is partitioned into two sets: an *elite set* that contains the p_e best individuals and a *non-elite set* with the remaining individuals. Thus, an individual is chosen uniformly at random from the elite set, and other is chosen from the non-elite set also uniformly at random. One can note that this biased strategy chooses a good individual trying to propagate its genes. A drawback is that this strategy leads to a faster convergence to local optima and optimization may finish too

early. To avoid this, it is very important to use a restart strategy where the entire population is reset to new random individuals.

Algorithm 1: BRKGA scheme.

```
Generate the initial population P;
while a stopping criteria is not reached do
    Decode each chromosome of P and extract their solutions and fitness;
    Sort the population P in non-increasing order of fitness. Consider the top
    p_e individuals as the elite group E;
    Copy E to the next generation Q, unaltered;
    Add p_m randomly-generated new chromosomes (mutants) to Q;
    Generate p − p_e − p_m chromosomes (offspring) by uniform parameterized
    crossover, selecting a random parent from E and another from P \ E. Add
    them to Q;
    P ← Q;
return best individual found.
```

The crossover in BRKGA is also biased. This is done using a probability ρ such that if the value of a coin toss is less than or equal to ρ, the allele is taken from the elite parent, otherwise from the non-elite parent (the coin values are drawn from an uniform distribution in the real interval $[0, 1]$). If $\rho > 0.5$, then in expectation the offspring will contain more alleles from the elite parent than the non-elite parent and, therefore, is likely to contain good structures inherited from the former. One can note that when $\rho = 0.5$ we have the standard uniform crossover, and when $\rho < 0.5$ there is more probability to inherit from the non-elite chromosome which makes little sense since the non-elite parent is worse than the elite parent. details of BRKGAs can be found in Gonçalves and Resende (2011).

3. Gender Defining and Multi-parent BRKGA

We propose major modifications to the BRKGA framework with the objective of improving the quality of the solutions found by the algorithm. In the first proposal, we try to mimic sexual reproduction that occurs in nature by assigning a gender to each chromosome. In the second proposal, more than two parents are used to create an offspring with the objective of combining the diversity of good solutions into an offspring.

The *Gender Defining Biased Random-Key Genetic Algorithm* (BRKGA-GD) assigns a bit for each chromosome such that its value defines the gender. To apply crossover, two chromosomes of different gender are chosen from the elite and non-elite set and the procedure follows the same steps of the original BRKGA. The gender of the new offspring is defined at random to be equally probable to be of any gender. It is possible that the entire population may have only one gender. In this case, the algorithm is restarted with a new random population. However, this is very rare, since it is expected that the mutants introduced in the population have a well distributed genders (recall that the mutants are generated using a uniform distribution, including for gender assignment).

The *Multi-Parent Biased Random-Key Genetic Algorithm* (BRKGA-MP) uses several parents to produce an offspring using a weighted combination of them. 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 given associated probability of passing its alleles to the offspring. This probability is calculated by taking the parent's bias into account. Parent *bias* is defined for a given weighting function. We used the bias functions proposed by Bresina (1996):

- Logarithmic: $bias(r) = \log^{-1}(r+1)$;
- Linear: $bias(r) = 1/r$;
- Polynomial (n): $bias(r) = r^{-n}$;
- Exponential: $bias(r) = e^{-r}$.

Algorithm 2 shows the complete crossover procedure. In lines 1 and 2, the parents are chosen and sorted according their fitness. Lines 3–8 calculates the probabilities of each parent. The function $rank()$ returns the position or rank of the supplied chromosome in the given order which is passed to the $bias$ function. Note that we must do a normalization step to compute the correct probabilities. 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 entire procedure is done for each new offspring that is generated.

Algorithm 2: BRKGA-MP crossover scheme.

Input: Number of genes n; Elite set E; Non-elite set R; function $bias : \mathbb{N} \to \mathbb{R}$; values $\pi_t \leq |E \cup R|$, and π_e such that $\pi_e \leq |E|$ and $\pi_e \leq \pi_t$

1 Sample uniformly random π_e individuals from E and $\pi_t - \pi_e$ individuals from R. Let Q be these individuals;
2 Sort Q in non-increasing order of fitness;
3 $total_{weight} \leftarrow 0$;
4 **foreach** $q \in Q$ *in the given order* **do**
5 $\quad weight(q) \leftarrow bias(rank(q))$;
6 $\quad total_{weight} \leftarrow total_{weight} + weight(q)$;
7 **foreach** $q \in Q$ **do**
8 $\quad weight(q) \leftarrow weight(q)/total_{weight}$;
9 Let c be an empty new chromosome;
10 **for** $i = 1$ *to* n **do**
11 $\quad$ Select an individual $q \in Q$ using the "roullete" method based on its weight;
12 $\quad c[i] \leftarrow q[i]$;
13 **return** c;

4. Experimental results

4.1. Test Problems

To evaluate the performance of the proposed algorithms, we used decoders for the set covering and set packing problems. We made no modification to these decoders and compare all results with those obtained with the original BRKGAs.

For the set covering problem, we considered the decoder of Resende et al. (2011) proposed for the Steiner triple covering problem. This problem is $\mathcal{NP}$-hard (Fulkerson et al., 1974; Garey and Johnson, 1979). Furthermore, its benchmark instances are computationally challenging (Ostrowski et al., 2010). The decoder takes a vector of size n such that each component is related to a set by its index. Thus, it builds a covering considering all sets whose components are greater than or equal to 0.5. If the result is not a valid covering, the remaining elements are fixed using a greedy procedure which selects the smallest index set that covers the maximum uncovered elements until the covering is complete. The decoder also performs a pruning phase, removing sets that do not destroy the covering. For more details, the reader can refer to Resende et al. (2011). We consider 45 set-covering instances available in OR-Library (1990).

We also applied the algorithms for the weighted set packing problem, also known to be $\mathcal{NP}$-hard (Garey and Johnson, 1979). In the weighted set packing problem, one seeks pairwise disjoint sets such the sum of their weights is maximized. We used instances for the winner determination problem generated with CATS (Leyton-Brown and Shoham, 2006) and instances from Lau and Goh (2002). The winner determination problem arises in the context of combinatorial auctions and is usually modeled as a weighted set packing problem (Cramton et al., 2006). We chose ten instances, each of size: 400; 1,000; 1,500; 2,000; and 4,000 sets. These 50 instances are considered hard to solve, as one can see in the following discussion. We use one of the decoders presented by Andrade et al. (2014). In this case, each allele corresponds to a set. The alleles are sorted in non-increasing order of values and the sets are taken in this sequence. If the current set is not disjoint from the previously selected sets, it is not considered to be in the solution. Andrade et al. (2014) presents different ways to form the initial population. In the present paper, the population is initialized exclusively with random individuals.

4.2. Computational environment and Parameters

The experiments were conducted on identical machines with two 6-core Intel Xeon 2.4 GHz CPUs (two thread per core) and 32 GBytes of RAM running GNU/Linux. Running times reported are UNIX real wall-clock times in seconds, excluding the effort to read the instance. The algorithms are implemented in the C++ language with the API of Toso and Resende (2014) and use the GNU `g++` compiler version 4.8. Random numbers were generated by an implementation of the Mersenne-Twister (Matsumoto and Nishimura, 1998).

For the set covering problem, we configure the BRKGA as in Resende et al. (2011). The population size was set to $p = 10n$ where n in the number of sets. The elite size to $p_e = \lceil 1.5n \rceil$. The number of mutants are set to $p_m = \lfloor 5.5n \rfloor$. The probability of inheriting each allele from elite parent was $\rho_e = 0.60$. We used the island model with three independent and concurrent populations. Every 100 generations, each population exchanges its two best solutions. Every 500 generations without improvement all populations are reset. We use eight simultaneous cores for decoding.

Thirty independent runs were carried out for each instance and algorithm configuration. Each run was limited to $1{,}000$ generations without improvement, or 3,600 wall-clock seconds.

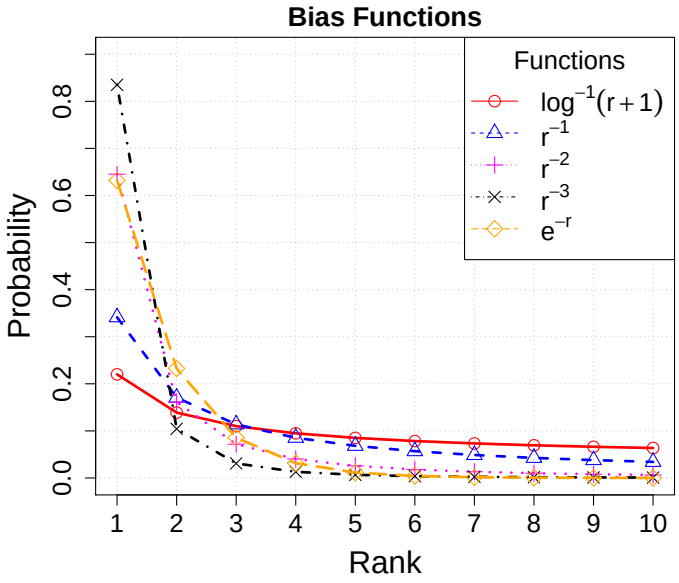


Figure 1. Probabilities induced by the bias functions.

4.3. Evaluation of Multi-parent BRKGA

We first evaluate BRKGA-MP. The first pair of parameters to be chosen is the number of parents to use in crossover and, of these, how many are elite. We tested eight parent/elite pairs: 3/1, 3/2, 6/2, 6/3, 6/4, 10/3, 10/5, and 10/7. Pairs 3/1, 6/2, and 10/3 have a small number of elite individuals (*non-elitist configuration*), while pairs 6/3 and 10/5 try to balance elite and non-elite individuals (*balanced configuration*), and pairs 3/2, 6/4, and 10/7 have more elite than non-elite chromosomes (*elitist configuration*). The next parameter is the bias function to be used. We considered the functions listed in Section 3 and chose the quadratic and cubic version of polynomial functions. We refer to each configuration as a tuple (mp, bf), where mp represents the multi-parent configuration and bf is the bias function. For example, the tuple $(3/2, r^{-1})$ indicates that we have three parents for crossover such that two are elite chromosomes, and they are weighted by the function r^{-1} applied to their rank.

We analyze the bias functions with respect to the number of parents for crossover. Figure 1 depicts the probabilities induced by each function. The X axis is the chromosome ranking and the Y axis is the allele selection probability. Functions r^{-2} (magenta dotted line with plus signal), r^{-3} (black dashed dotted line with crosses), and e^{-r} (orange dashed line with diamonds) assign high probability to the top ranked chromosome using a mechanism similar to the original BRKGA with $\rho > 0.5$. One can note that, even if one chooses a large number of non-elite individuals for crossover (cases 3/1, 6/2, and 10/3), the elite individuals still have a better chance to be chosen. Functions $\log^{-1}(r+1)$ (red solid line with circles) and r^{-1} (blue dashed line with triangles) are softer and evenly distribute the probabilities although assigning higher probabilities to the first two individuals. They enable a more interbred offspring made up of alleles from several parents. Note that, in this case, it may be better to choose a configuration such as 3/2, 6/4, and 10/7 to keep good structures from different elite chromosomes in the population.

Table 1 shows the results for each configuration in finding the optimal or the best

Table 1. BRKGA–MP performance for several settings. For set covering instances, the % run is the percentage of runs where an optimal solution was found. For set packing instances, the % run is the percentage of runs where the best known solution value was found.

Inst.	M.P.	$\log^{-1}(r+1)$		r^{-1}		r^{-2}		r^{-3}		e^{-r}	
		% Run	Gap	% Run	Gap	% Run	Gap	% Run	Gap	% Run	Gap
Set Covering	3/1	90.66	0.03	92.88	0.02	94.66	0.02	95.11	0.01	91.55	0.02
	3/2	91.11	0.03	92.44	0.02	**96.44**	0.01	91.11	0.02	93.33	0.02
	6/2	88.88	0.05	87.11	0.05	90.66	0.03	93.77	0.03	92.00	0.03
	6/3	86.22	0.06	85.77	0.06	92.44	0.02	89.33	0.05	90.66	0.03
	6/4	83.11	0.08	83.11	0.07	93.33	0.02	88.00	0.04	93.22	0.02
	10/3	86.66	0.06	84.44	0.07	88.00	0.04	90.66	0.04	88.88	0.04
	10/5	83.11	0.07	81.33	0.09	92.00	0.03	85.77	0.07	91.11	0.03
	10/7	80.00	0.09	79.55	0.09	89.77	0.04	82.66	0.08	90.22	0.05
Average		86.22	0.06	85.83	0.06	**92.17**	**0.03**	89.56	0.04	91.38	0.03
Set Packing	3/1	42.50	1.41	43.33	1.31	40.83	1.40	33.75	2.14	41.25	1.45
	3/2	42.50	1.29	42.50	1.42	35.41	1.75	29.58	2.61	35.41	1.61
	6/2	42.08	1.37	41.25	1.30	36.66	1.46	26.25	2.60	37.91	1.48
	6/3	41.66	1.28	40.41	1.34	32.91	1.74	27.91	2.79	37.91	1.72
	6/4	40.41	1.32	39.58	1.35	34.58	1.82	25.41	2.95	32.91	1.81
	10/3	**43.75**	1.31	43.33	1.29	38.33	1.54	26.25	2.92	35.00	2.17
	10/5	38.33	1.37	40.83	1.33	34.58	1.79	24.58	3.38	29.58	2.70
	10/7	40.83	1.34	41.25	1.32	33.33	1.92	24.16	3.95	29.58	2.84
Average		**41.51**	**1.34**	**41.56**	**1.33**	35.83	1.68	27.24	2.92	34.95	1.97

known solution for each instance. The first column lists the instance class; the second column shows the multi-parent configuration using the same notation introduced in the beginning of this section; for each bias function, we have a pair of columns. Column "% Run" shows the percentage of runs for which the algorithm was able to obtain either the optimal solution (for set covering) or the best known solution (for set packing). Column "Gap" shows the average distance (in percentage) between the solution found and the optimal/best known value. The table has two parts, each listing a different set of instances. For set covering problems, BRKGA-MP in general obtained very good results, producing an optimal solution in more than 79% of runs. Note also that the gaps are very small, never greater than 0.09%. The best results are found by configuration $(3/2, r^{-2})$ and the ranking function r^{-2} obtained the best results on average. The set packing instances are much harder to solve and this can be seen considering the low number of runs that obtained the value of the best known solution. Although this percentage of runs was around 40%, the gaps are small and are not greater than 3.95%. The best configuration was $(10/3, \log^{-1}(r+1))$. Note that both ranking functions $\log^{-1}(r+1)$ and r^{-1} obtained similar results.

It is interesting to note that for easier instances, a strong elitist configuration leads to the best results: $(3/2, r^{-2})$ combines two elite parents with one non-elite parent using a strong bias in favor to the elite parents due to ranking function r^{-2}. On the other hand, for harder instances, the less elitist strategy $(10/3, \log^{-1}(r+1))$ works better: it combines only three elite parents with seven non-elite parents using a "soft" ranking function.

4.4. Comparing the algorithms

Table 2 shows, for each algorithm, the percentage of runs in which the optimal solution value or the best solution value was obtained, for each type of instance. The first column indicates the instance type; The second column groups the instances by their sizes. The rest of the table is partitioned into three blocks, one for each algorithm. In each block, column "% Run" shows the percentage of runs in which the optimal solution value or the best solution value was obtained, and column "GAP" shows the average percentage gap between the solution found and the optimal/best known value. For BRKGA-MP, the additional column "Config." shows the best configurations used to obtain those results.

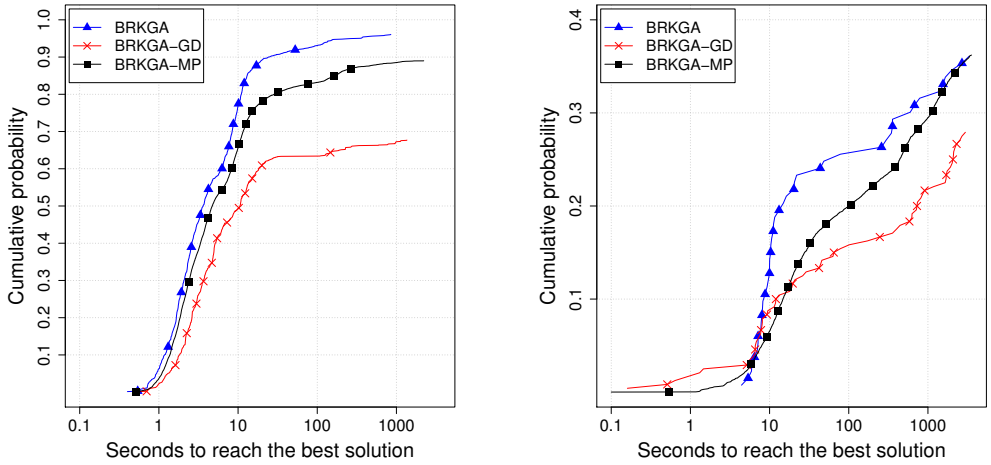
One can note that to assign gender to the chromosomes did not result in significant improvements. Note the BRKGA-GD only outperformed the others algorithms for the set packing instances of size 400. In general, BRKGA-MP outperformed the original BRKGA. The exceptions are set covering instances of size 200 and set packing instances of size 1000, where the performances of the algorithms were similar. It is important to highlight that BRKGA-MP found almost all best solution values for all "non-elitist" configurations represent by a star in the table.

Figures 2 and 3 depicts the performance profiles for the algorithms. In performance profiles, the abscissa shows the time needed to reach a target solution value, while the ordinate shows the cumulative probability to reach a target solution value for the given time (in log scale) in the abscissa. In Figure 3, instead of showing the time, the abscissa shows the number of iterations to reach a target solution value. Each algorithm is characterized by a different performance profile curve made up of (time/iterations, cumulative probability) pairs, one for each execution of the algorithm on a particular instance. Runs that took over 3,600 seconds are not shown in the figure. Therefore, the percentage of runs that concluded within the time limit can be seen as the intersection of the profile with the right hand side of the figure. One can note that the original BRKGA is able to find good solutions faster than the other algorithms and uses less iterations to do that in all cases. BRKGA-GD was not able to produce as good results as the other algorithms in any case. Note also that, although BRKGA-MP has a slower convergence than the original BRKGA, it was able to overcome BRKGA on hard set packing instances as one can observe at the very end of the curves of Figure 2b.

Table 2. Cost comparison among the algorithms. The gap is given in percentage.

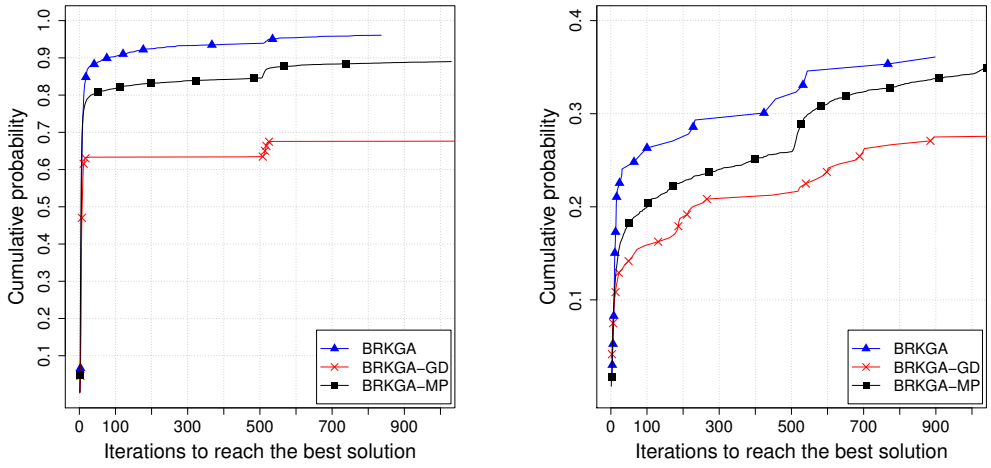
Inst.	Size	BRKGA		BRKGA-GD		BRKGA-MP		
		% Run	Gap	% Run	Gap	% Run	Gap	Config.
Set Cov.	200	**96.80**	**0.00**	65.06	0.19	96.80	0.01	$(3/2, 1/r^2)$
	300	90.66	0.03	63.33	0.20	**96.00**	**0.01**	$(3/2, 1/r^3)$
	400	99.66	0.00	78.66	0.13	**100.0**	**0.00**	$\star$
Set Pack.	400	5.26	0.46	**26.66**	**0.75**	23.33	0.55	$(3/1, 1/r)$, $(6/3, e^{-r})$
	1000	**66.66**	**1.20**	26.66	5.26	66.66	1.22	$(3/1, 1/r)$
	1500	47.36	0.95	58.33	0.83	**71.66**	**0.45**	$(10/3, 1/\log(r+1))$
	2000	0.00	2.41	0.00	15.67	**3.33**	**2.15**	$(10/3, 1/r)$
	4000	0.00	3.36	0.00	19.64	**3.33**	**2.63**	$(10/7, 1/\log(r+1))$

$\star$ All non-elitist configurations.



(a) Set covering instances. **(b)** Set packing instances.

Figure 2. Running time distributions to obtain the best solution value. The points in this plot correspond to 2% of the produced points for each algorithm.



(a) Set covering instances. **(b)** Set packing instances.

Figure 3. Distributions of the number of iterations to obtain the best solution value. The points in this plot correspond to 2% of the points produced for each algorithm.

5. Final considerations

In this paper, we presented two new new variants of biased random-key genetic algorithms. In the first approach, called BRKGA-GD, we proposed the assignment of gender to each chromosome and only allow crossover between chromosomes of different gender. In the second approach, called BRKGA-MP, we proposed to use more than two parents in crossover. We tested the algorithms for set covering and set packing instances. One can note that to differ chromosomes by gender did not result in any advantage over the original BRKGA. In fact, BRKGA-GD presented the worst results among all algorithms. The crossover between several parents was able to obtain better results than the original BRKGA, but mainly using a non-elitist strategy where more non-elite than elite individuals are chosen to generate a new offspring. On the other hand, BRKGA-MP required more time and iterations than the original BRKGA to converge.

References

Andrade, C. E., Miyazawa, F. K., and Resende, M. G. C. (2013). Evolutionary algorithm for the k-Interconnected Multi-Depot Multi-Traveling Salesmen Problem. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 463–470, New York, NY, USA. ACM.

Andrade, C. E., Resende, M. G. C., Karloff, H. J., and Miyazawa, F. K. (2014a). Evolutionary algorithms for overlapping correlation clustering. In *Proceedings of the 16th Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, New York, NY, USA. ACM. To appear.

Andrade, C. E., Toso, R. F., Resende, M. G. C., and Miyazawa, F. K. (2014b). Biased random-key genetic algorithms for the winner determination problem in combinatorial auctions. *Evolutionary Computation*. To appear.

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

Bresina, J. L. (1996). Heuristic-biased stochastic sampling. In *Proceedings of the thirteenth national conference on Artificial Intelligence*, volume 1 of *AAAI'96*, pages 271–278. AAAI Press.

Cramton, P., Shoham, Y., and Steinberg, R. (2006). *Combinatorial Auctions*. MIT Press.

Ericsson, M., Resende, M. G. C., and Pardalos, P. M. (2002). A genetic algorithm for the weight setting problem in OSPF routing. *J. of Combinatorial Optimization*, 6:299–333.

Fulkerson, D. R., Nemhauser, G. L., and Trotter, L. L. (1974). *Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of Steiner triple systems*. Springer.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Professional, Boston, MA, USA, 1st edition.

Gonçalves, J. F. and Almeida, J. (2002). A hybrid genetic algorithm for assembly line balancing. *J. of Heuristics*, 8:629–642.

Gonçalves, J. F., Mendes, J. J. M., and Resende, M. G. C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European J. of Operational Research*, 167:77–95.

Gonçalves, J. F., Mendes, J. J. M., and Resende, M. G. C. (2008). A genetic algorithm for the resource constrained multi-project scheduling problem. *European J. of Operational Research*, 189:1171–1190.

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

Gonçalves, J. F. and Resende, M. G. C. (2011b). A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. *Journal of Combinatorial Optimization*, 22:180–201.

Gonçalves, J. F. and Resende, M. G. C. (2012). A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers and Operations Research*, 12:179–190.

Gonçalves, J. F. and Resende, M. G. C. (2013). A biased random-key genetic algorithm for a 2D and 3D bin packing problem. *International J. of Production Economics*, 145:500–510.

Gonçalves, J. F. and Resende, M. G. C. (2014). An extended Akers graphical method

with a biased random-key genetic algorithm for job-shop scheduling. *International Transactions in Operational Research*, 21:215–246.

Gonçalves, J. F., Resende, M. G. C., and Mendes, J. J. M. (2011). A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *J. of Heuristics*, 17:467–486.

Lau, H. C. and Goh, Y. G. (2002). An intelligent brokering system to support multi-agent web-based 4th-party logistics. In *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '02, pages 154–, Washington, DC, USA. IEEE Computer Society.

Leyton-Brown, K. and Shoham, Y. (2006). *Combinatorial Auctions*, chapter A Test Suite for Combinatorial Auctions, pages 451–478. MIT Press.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8:3–30.

Mendes, J. J. M., Gonçalves, J. F., and Resende, M. G. C. (2009). A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers & Operations Research*, 36:92–109.

OR-Library (1990). Operations Research Library. `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`. Accessed in Oct 30, 2013.

Ostrowski, J., Linderoth, J., Rossi, F., and Smriglio, S. (2010). Solving Steiner triple covering problems. *Optima*, 83.

Resende, M. G. C. (2012). Biased random-key genetic algorithms with applications in telecommunications. *TOP*, 20:120–153.

Resende, M. G. C., Toso, R. F., Gonçalves, J. F., and Silva, R. M. A. (2011). A biased random-key genetic algorithm for the Steiner triple covering problem. *Optimization Letters*, pages 1–15.

Spears, W. M. and DeJong, K. A. (1991). On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236.

Toso, R. F. and Resende, M. G. C. (2014). A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*. DOI: 10.1080/10556788.2014.890197.