BIASED RANDOM-KEY GENETIC ALGORITHMS FOR COMBINATORIAL OPTIMIZATION

JOSÉ FERNANDO GONÇALVES AND MAURICIO G. C. RESENDE

ABSTRACT. Random-key genetic algorithms were introduced by Bean (1994) for solving sequencing problems in combinatorial optimization. Since then, they have been extended to handle a wide class of combinatorial optimization problems. This paper presents a tutorial on the implementation and use of biased random-key genetic algorithms for solving combinatorial optimization problems. Biased random-key genetic algorithms are a variant of random-key genetic algorithms, where one of the parents used for mating is biased to be of higher fitness than the other parent. After introducing the basics of biased random-key genetic algorithms, the paper discusses in some detail implementation issues, illustrating the ease in which sequential and parallel heuristics based on biased random-key genetic algorithms can be developed. A survey of applications that have recently appeared in the literature is also given.

1. INTRODUCTION

Combinatorial optimization can be defined by a finite ground set $E = \{1, \ldots, n\}$, a set of feasible solutions $F \subseteq 2^E$, and an objective function $f : 2^E \to \mathbb{R}$. Throughout this paper, we consider the minimization version of the problem, where we search for an optimal solution $S^* \in F$ such that $f(S)^* \leq f(S)$, $\forall S \in F$. Given a specific combinatorial optimization problem, one can define the ground set E, the cost function f, and the set of feasible solutions F. For instance, in the case of the traveling salesman problem on a graph, the ground set E is that of all edges in the graph, $f(S)$ is the sum of the costs of all edges $e \in S$, and F is formed by all edge subsets that determine a Hamiltonian cycle.

Combinatorial optimization finds applications in many settings, including routing, scheduling, inventory control, production planning, and location problems. These problems arise in real-world situations (Pardalos and Resende, 2002) such as in transportation (air, rail, trucking, shipping), energy (electrical power, petroleum, natural gas), and telecommunications (design, location, operation).

While much progress has been made in finding provably optimal solutions to combinatorial optimization problems employing techniques such as branch and bound, cutting planes, and dynamic programming, as well as provably near-optimal solutions using approximation algorithms, many combinatorial optimization problems arising in practice benefit from heuristic methods that quickly produce good-quality

Date: October 8, 2009.

Key words and phrases. Genetic algorithms, biased random-key genetic algorithms, random-key genetic algorithms, combinatorial optimization, metaheuristics.

This research was partially supported by Fundação para a Ciência e Tecnologia (FCT) project PTDC/GES/72244/2006. AT&T Labs Research Technical Report.

solutions. Many modern heuristics for combinatorial optimization are based on guidelines provided by metaheuristics.

Metaheuristics are high level procedures that coordinate simple heuristics, such as local search, to find solutions that are of better quality than those found by the simple heuristics alone. Many metaheuristics have been introduced in the last thirty years (Glover and Kochenberger, 2003). Among these, we find greedy randomized adaptive search procedures (GRASP), simulated annealing, tabu search, variable neighborhood search, scatter search, path-relinking, iterated local search, ant colony optimization, swarm optimization, and genetic algorithms.

In this paper, we introduce a metaheuristic called *biased random-key genetic algorithms*. This framework for building heuristics for combinatorial optimization is general and can be applied to a wide range of problems. An important characteristic of the framework is the clear divide between the problem-independent component of the architecture and the problem-specific part. This allows for reuse of software and permits the algorithm designer to concentrate on building the problem specific decoder.

The paper is organized as follows. In Section 2 we introduce biased random-key genetic algorithms. Issues related to the efficient implementation of sequential and parallel versions of these heuristics are discussed in Section 3. In Section 4 examples of biased random-key genetic algorithms on a wide range of combinatorial optimization problems are given. Concluding remarks are made in Section 5.

2. Biased random-key genetic algorithms

Genetic algorithms, or GAs, (Goldberg, 1989; Holland, 1975) apply the concept of *survival of the fittest* to find optimal or near-optimal solutions to combinatorial optimization problems. An analogy is made between a solution and an *individual* in a *population*. Each individual has a corresponding *chromosome* that encodes the solution. A chromosomes consists of a string of *genes*. Each gene can take on a value, called an *allele*, from some alphabet. Chromosomes have associated with them a *fitness* level which is correlated to the corresponding objective function value of the solution it encodes. Genetic algorithms evolve a set of individuals that make up a population over a number of *generations*. At each generation, a new population is created by combining elements of the current population to produce offspring that make up the next generation. Random mutation also takes place in genetic algorithms as a means to escape entrapment in local minima. The concept of survival of the fittest plays into genetic algorithms when individuals are selected to mate and produce offspring. Individuals are selected at random but those with better fitness are preferred over those that are less fit.

Genetic algorithms with random keys were first introduced by Bean (1994) for solving combinatorial optimization problems involving sequencing. In this paper we refer to this class of genetic algorithms as *random-key genetic algorithms* (RKGA). In a RKGA, chromosomes are represented as a string, or vector, of randomly generated real numbers in the interval $[0, 1]$. A deterministic algorithm, called a *decoder*, takes as input any chromosome and associates with it a solution of the combinatorial optimization problem for which an objective value or fitness can be computed. In the case of Bean (1994), the decoder sorts the vector of random keys and uses the indices of the sorted keys to represent a sequence. As we will see shortly, decoders play an important role in RKGAs.

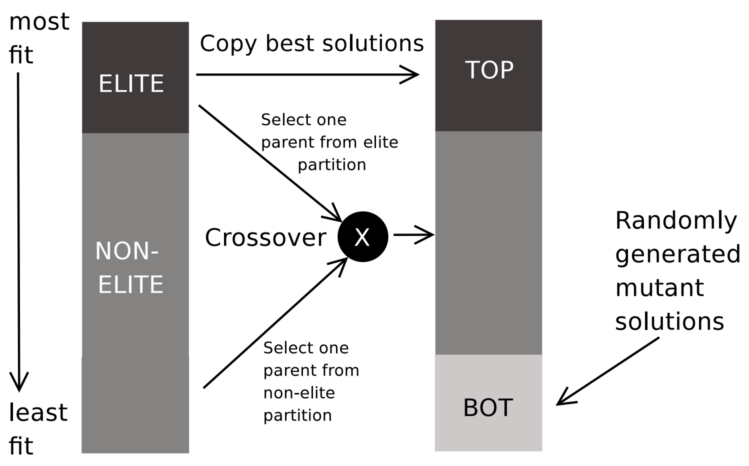
most fit
least fit
ELITE
NON-ELITE
Copy best solutions
Select one parent from elite partition
Crossover
X
Select one parent from non-elite partition
TOP
BOT
Randomly generated mutant solutions

FIGURE 2.1. Transition from generation k to generation $k+1$ in a BRKGA.

A RKGA evolves a population of random-key vectors over a number of iterations, called *generations*. The initial population is made up of p vectors of random-keys. Each allele is generated independently at random in the real interval $[0,1]$. After the fitness of each individual is computed by the decoder, the population is partitioned into two groups of individuals: a small group of p_e *elite* individuals, i.e. those with the best fitness values, and the remaining set of $p-p_e$ *non-elite* individuals, where $p_e < p-p_e$. To evolve the population, a new generation of individuals must be produced. An RKGA uses an *elitist strategy* since all of the elite individuals of generation k are copied unchanged to generation $k+1$. This strategy keeps track of good solutions found during the iterations of the algorithm resulting in a monotonically improving heuristic. Mutation is an essential ingredient of genetic algorithms, used to enable GAs to escape from entrapment in local minima. RKGAs implement mutation by introducing *mutants* into the population. A mutant is simply a vector of random keys generated in the same way that an element of the initial population is generated. At each generation a small number of p_m mutants are introduced into the population. Mutant solutions are random-key vectors and consequently can de decoded into valid solutions of the combinatorial optimization problem. With the p_e elite individuals and the p_m mutants accounted for in population $k+1$, $p-p_e-p_m$ additional individuals need to be produced to complete the p individuals that make up the population of generation $k+1$. This is done by producing $p-p_e-p_m$ offspring through the process of mating.

Figure 2.1 illustrates the evolution dynamics. On the left of the figure is the current population. After all individuals are sorted by their fitness values, the best fit are placed in the elite partition labeled ELITE and the remaining individuals are placed in the partition labeled NON-ELITE. The elite random-key vectors are copied without change to the partition labeled TOP in the next population (on the right side of the figure). A number of mutant individuals are randomly generated and placed in the new population in the partition labeled BOT. The remainder of the population of the next generation is completed by crossover. In a RKGA, Bean

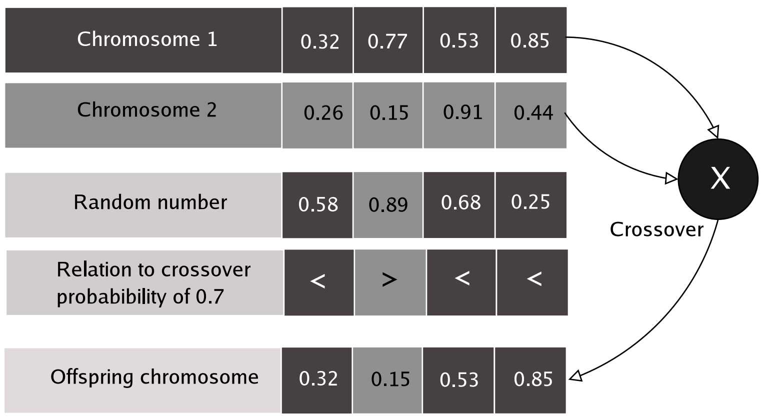

FIGURE 2.2. parameterized uniform crossover: mating in BRKGAs.

(1994) selects two parents at random from the entire population. A *biased random-key genetic algorithm,* or BRKGA (Gonçalves and Almeida, 2002; Ericsson et al., 2002; Gonçalves and Resende, 2004), differs from a RKGA in the way parents are selected for mating. In a BRKGA, each element is generated combining one element selected at random from the partition labeled ELITE in the current population and one from the partition labeled NON-ELITE. In some cases, the second parent is selected from the entire population. Repetition in the selection of a mate is allowed and therefore an individual can produce more than one offspring. Since we require that $p_e < p - p_e$, the probability that an elite individual is selected for mating is greater than that of a non-elite individual and therefore the elite individual has a higher likelihood to pass on its characteristics to future generations. Another factor contributing to this end is *parameterized uniform crossover* (Spears and DeJong, 1991), the mechanism used to implement mating in BRKGAs. Let $\rho_e > 0.5$ be the probability that an offspring inherits the allele of its elite parent. Let n denote the number of genes in the chromosome of an individual. For $i = 1, \ldots, n$, the i-th allele $c(i)$ of the offspring c takes on the value of the i-th allele $e(i)$ of the elite parent e with probability ρ_e and the value of the i-th allele $\bar{e}(i)$ of the non-elite parent $\bar{e}$ with probability $1 - \rho_e$. In this way, the offsrping is more likely to inherit characteristics of the elite parent than those of the non-elite parent. Since we assume that any random key vector can be decoded into a solution, then the offspring resulting from mating is always valid, i.e. can be decoded into a solution of the combinatorial optimization problem.

Figure 2.2 illustrates the crossover process for two random-key vectors with four genes each. Chromosome 1 refers to the elite individual and Chromosome 2 to the non-elite one. In this example the value of $\rho_e = 0.7$, i.e. the offspring inherits the allele of the elite parent with probability 0.7 and of the other parent with probability 0.3. A randomly generated real in the interval $[0, 1]$ simulates the toss of a biased coin. If the outcome is less than or equal to 0.7, then the child inherits the allele of the elite parent. Otherwise, it inherits the allele of the other parent. In this example, the offspring inherits the allele of the elite parent in its first, third, and fourth genes. It resembles the elite parent more than it does the other parent.

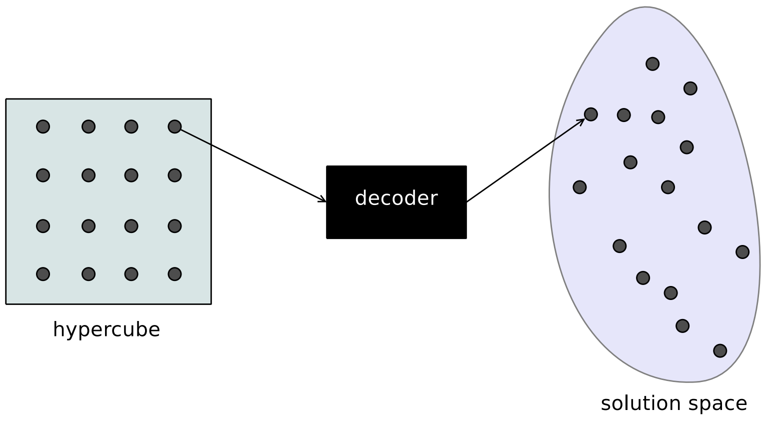

FIGURE 2.3. Decoder used to map solutions in the hypercube to solutions in the solution space where fitness is computed.

When the next population is complete, i.e. when it has p individuals, fitness values are computed for all of the newly created random-key vectors and the population is partitioned into elite and non-elite individuals to start a new generation.

A BRKGA searches the solution space of the combinatorial optimization problem indirectly by searching the continuous n-dimensional hypercube, using the decoder to map solutions in the hypercube to solutions in the solution space of the combinatorial optimization problem where the fitness is evaluated. Figure 2.3 illustrates the role of the decoder.

BRKGA heuristics are based on a general-purpose metaheuristic framework. In this framework, depicted in Figure 2.4, there is a clear divide between the *problem-independent* portion of the algorithm and the *problem-dependent* part. The problem-independent portion has no knowledge of the problem being solved. It is limited to searching the hypercube. The only connection to the combinatorial optimization problem being solved is the problem-dependent portion of the algorithm, where the decoder produces solutions from the vectors of random-keys and computes the fitness of these solutions. Therefore, to specify a BRKGA heuristic one need only define its chromosome representation and the decoder.

Consider, for example, a set covering problem where one is given an $m \times n$ binary matrix $A = [a_{i,j}]$ and wants to select the smallest cover, i.e. the smallest subset of columns $J^* \subseteq \{1, 2, \ldots, n\}$ such that, for each row $i = 1, \ldots, m$, there is at least one $j \in J^*$ such that $a_{i,j} = 1$. One possible BRKGA heuristic for this problem defines the vector of random keys x to have n random keys in the real interval $[0, 1]$. The j-th key corresponds to the j-th column of A. The decoder selects column j to be in J^* only if $x_j \geq 0.5$. If the resulting set J^* is a valid cover, then the fitness of the cover is $|J^*|$. Otherwise, start with set J^* and apply the standard greedy algorithm for set covering: while there are uncovered rows, find the unselected column that if added to J^* covers the largest number of yet-uncovered rows, breaking ties by the index of the column. Add this column to set J^*. When the resulting set J^* is a valid cover, scan the columns in the cover from first to last to check if each column $j \in J^*$ is redundant, i.e. if $J^* \setminus \{j\}$ is a cover. If so, then remove j from J^*. When no column can be removed, stop. The fitness of the cover is $|J^*|$. Note that, as

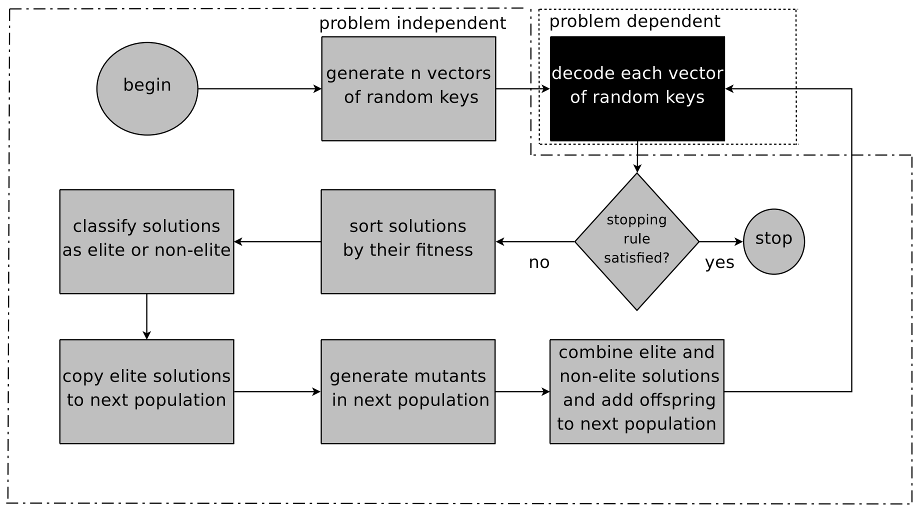
problem independent
problem dependent
begin
generate n vectors of random keys
decode each vector of random keys
stopping rule satisfied?
no
yes
stop
classify solutions as elite or non-elite
sort solutions by their fitness
copy elite solutions to next population
generate mutants in next population
combine elite and non-elite solutions and add offspring to next population

FIGURE 2.4. Flowchart of a BRKGA

required, this decoder is a deterministic algorithm. For a given vector of random keys, applying the decoder will always result in the same cover.

Though BRKGAs only use randomly generated keys, they are much better at finding optimal or near-optimal solutions that a purely random algorithm. Figure 2.5 provides strong evidence that there is *learning* taking place in a BRKGA. The figure shows the distributions of objective function values of the 100-element population of a BRKGA and the repeated generation of sets of 100 random solutions for a set covering by pairs problem (Breslau et al., 2009). Let $i, j, k \in \{1, 2, \ldots 100\} \times \{1, 2, \ldots 100\} \times \{1, 2, \ldots 100\}$. The covering-by-pairs problem considered here has 76,916 triplets, where a triplet $\{i, j, k\}$ indicates that the pair $\{i, j\}$ covers element k. The objective is to find the smallest cardinality subset $S^* \subseteq \{1, 2, \ldots 100\}$ such that the union of all pairs $\{i, j\}$ with $i, j \in S^* \times S^*$ covers all the 100 elements indicated by the k values. The optimal solution, which we plot as a reference, is 21 and was computed by solving an integer programming model with the commercial integer programming solver CPLEX. As one can observe, while the BRKGA quickly finds an optimal solution is less than 2 seconds, the random multistart heuristic is still quite far from the optimal after 600 seconds having only found a best solution of size 38.

As discussed earlier in this paper, a biased random-key genetic algorithm and an (unbiased) random-key genetic algorithm differ slightly in the way they select parents for mating. The biased variant always selects one parent from the set of elite solutions whereas the unbiased variant selects both parents from the entire population. This way, offspring produced by the biased version are more likely to inherit characteristics of elite solutions. This likelihood is further emphasized through the parameterized uniform crossover used by both variants to combine the parents and produce the offspring. Though this is apparently only a very slight difference, it almost always leads to a big difference in how these variants perform. BRKGAs tend to find better solutions than RKGAs if given the same running time and have a much higher probability of finding a solution with a specified target

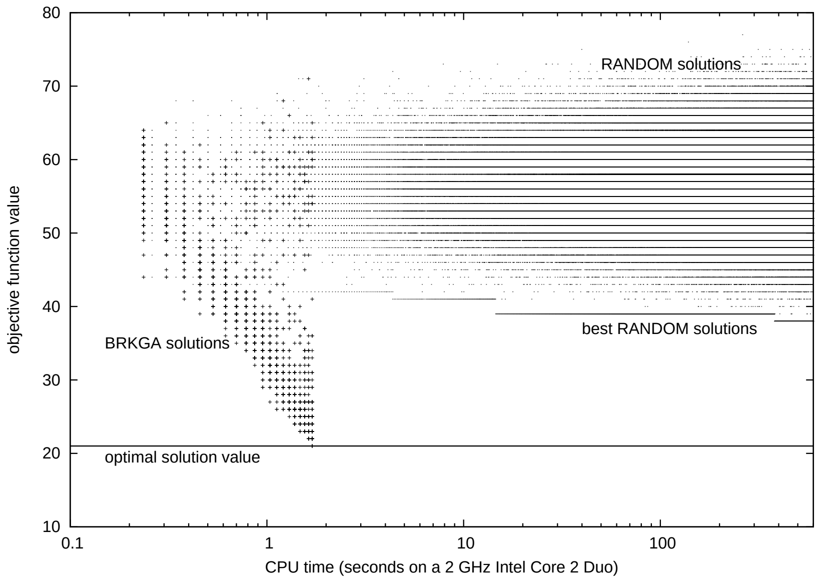

FIGURE 2.5. Comparing a BRKGA with a random multistart heuristic on an instance of a covering by pairs problem.

solution value in less time. To illustrate this, consider Figure 2.6 which shows time-to-optimal plots for a covering-by-pairs problem with 220 elements and 456,156 triplets. The plots compare running times to find an optimal solution for 200 independent runs of each of three variants: a BRKGA, a RKGA[1], and a heuristic (RKGA-ord) that is similar to a RKGA except that the offspring inherit the allele of the better fit of the two parents with probability ρ_e. The figure clearly shows that the BRKGA finds optimal solutions in less time than its unbiased counterparts. For example, by 325s, the time that the RKGA takes to solve any one of its 84 attempts, the BRKGA solves 184 of its 200 attempts. Ordering the parents, as is done in RKGA-ord, improves the RKGA, but not enough to do better than the BRKGA. For example, by 216s, the time that RKGA-ord takes to solve any one of its 200 attempts, the BRKGA solves 176 of its 200 attempts. Though we illustrate this on only a single instance of a single problem type, we have observed that this behavior is typical for a wide range of problems (Gonçalves et al., 2009b).

3. Implementation issues

In this section, we discuss some issues related to the implementation of BRKGA heuristics. We focus on the separation of the problem independent and dependent portions of the heuristic, types of decoders, initial population, population partitioning, parallel implementations, and post-optimization based on pairwise path-relinking between elite set solutions.

3.1. **Components of a BRKGA heuristic.** As discussed earlier in this paper, RKGAs have problem-independent and problem-dependent modules. This makes

[1]Due to excessively long running time, we only carried out 84 independent runs with the RKGA variant.

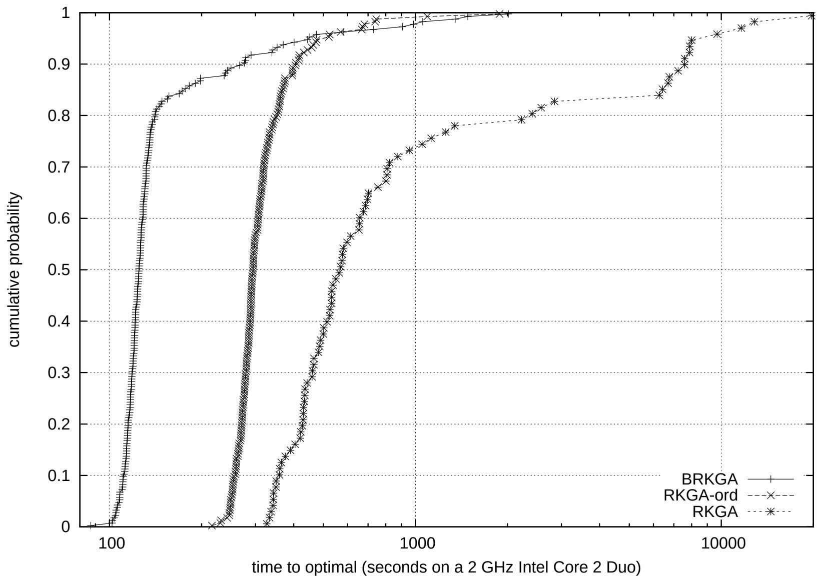

FIGURE 2.6. Time to target plots compare running times needed to find the optimal solution of a 220 element covering by pairs problem with a BRKGA and two variants of an RKGA.

it possible to design a general-purpose problem-independent solver that can be reused to implement different heuristics. That way, when designing a new heuristic for a specific combinatorial optimization problem, one need only implement the problem-dependent part, namely the decoder.

The problem-independent module has few basic components. These components depend on the number of genes in the chromosome of an individual (n), the number of elements in the population (p), the number of elite elements in the population (p_e), the number of mutants introduced at each generation into the population (p_m), and the probability that an offspring inherits the allele of its elite parent (ρ_e). The population is stored in the $p \times n$ real-valued matrix pop, where the i-th chromosome is stored in row i of pop. After populating matrix pop with real-valued random numbers generated uniformly in the interval $[0, 1]$, the fitness of each chromosome is evaluated by the problem-dependent decoder. The fitness value of the i-th chromosome is stored in the i-th position of the p-dimensional array fitness.

Each generation of an BRKGA heuristic consists of the following five steps:

(1) Sort array fitness in increasing order and reorder the rows of pop according to the sorted values of array fitness. The elements of pop do not actually need to be moved. Only an array with their positions is needed. For ease of description, we assume in this discussion that the rows of pop are actually moved to reflect the sorted values of fitness.
(2) Mate $p - p_e - p_m$ pairs of parents, one whose index in pop is an integer random number uniformly generated in the interval $[1, p_e]$ and the other whose index is an integer random number uniformly generated in the interval $[p_e + 1, p]$. The i-th offspring resulting from the crossover is temporarily

stored in row i of the real-valued $(p - p_e - p_m) \times n$ matrix `tmppop`. Mating is achieved by generating n uniform random numbers $\{r_1, \ldots, r_n\}$ in the interval $[0, 1]$. For $j = 1, \ldots, n$, if $r_j \leq \rho_e$, then the j-th gene of the offspring inherits the j-th allele of the elite parent. Otherwise, it inherits the allele of the other parent.

(3) Generate at random p_m mutant chromosomes of size n. These mutants are generated by the same module used to generate the initial population. The i-th mutant chromosome is stored in row $p_m + i - 1$ of matrix `pop`.

(4) Copy the $(p - p_e - p_m) \times n$ matrix `tmppop` to rows $p_e + 1, \ldots, p - p_m$ of matrix `pop`.

(5) Evaluate the fitness of the chromosomes in rows $p_e + 1, \ldots, p$ of matrix `pop` and store these values in positions $p_e + 1, \ldots, p$ of array `fitness.`

This process is applied repeatedly. Each iteration is called a generation. There are many possible stopping criteria, including stopping after a fixed number of generations from the beginning, after a fixed number of generations since the generation of the last solution improvement, after a time limit is reached, or after a solution at least as good as a given threshold is found.

3.2. **Decoders.** Decoders play an important role in BRKGA heuristics since they make the connection between the solutions in the hypercube and the fitness of their corresponding solutions in the solution space of the combinatorial optimization problem. They can range in complexity from very simple, involving a direct mapping between the random key and the solution, to intricate, such as random-key driven construction heuristics with local search, or even black box computations.

Suppose the solution space is made up of all permutations of $\Pi_n = \{1, 2, \ldots n\}$ as is the case for the quadratic assignment problem. Bean (1994) showed that simply sorting the vector of random keys results in a permutation of its indices. If one wants to select p of n elements of a set, assign a random key to each element of the set, sort the vector of random keys, and select the elements corresponding to the p smallest keys. Composite vectors of random keys are also useful. Suppose n items need to be arranged in order and that each element can be placed in one of two states, say *up* or *down*. Define a vector of random keys of size $2n$ where the first n keys are sorted to define the order in which the items are placed and the last n keys determine if the item is placed in the *up* or *down* position. In this case, a key greater than or equal to one half indicates the *up* position while a key less than half corresponds to the *down* position. In Section 4 we give more examples of simple and complex decoders.

3.3. **Parameter setting.** Random-key genetic algorithms have few parameters that need to be set. These parameters are the number of genes in a chromosome (n), the population size (p), the size of the elite solution population (p_e), the size of the mutant solution population (p_m), and the elite allele inheritance probability (ρ_e), i.e. the probability that the gene of the offspring inherits the allele of the elite parent. Though setting these parameters is sort of an art-form, our experience has led us to set the parameters as shown in Table 3.3.

Below, we illustrate the effect of population size, elite solution population size, mutant solution population size, and elite allele inheritance probability on the random variable time-to-optimal solution. We use the 100-element covering-by-pairs instance used earlier to compare the BRKGA and the random multi-start heuristic.

TABLE 1. Recommended parameter value settings

parameter	description	recommended value
p	size of population	$p = an$, where $1 \leq a \in \mathbb{R}$ is a constant and n is the length of the chromosome
p_e	size of elite population	$0.10p \leq p_e \leq 0.25p$
p_m	size of mutant population	$0.10p \leq p_m \leq 0.30p$
ρ_e	elite allele inheritance probability	$0.5 < \rho_e \leq 0.8$

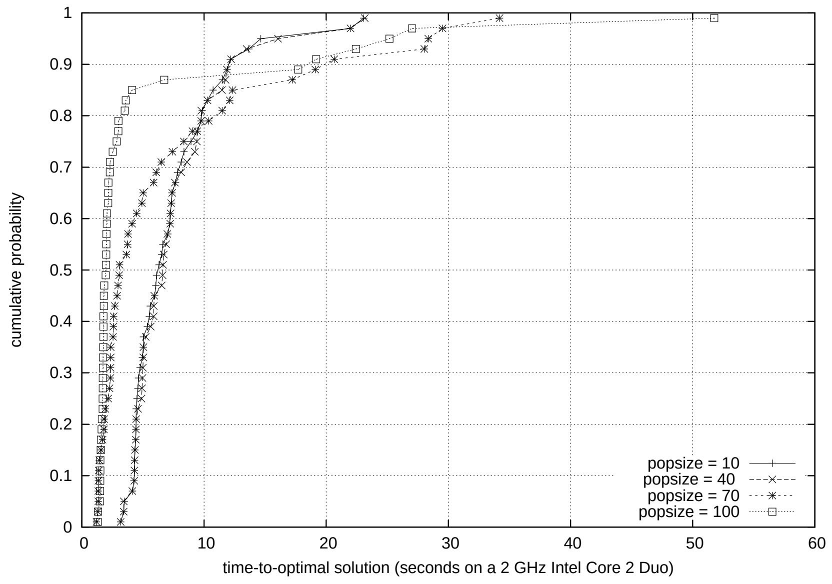

FIGURE 3.1. Effect of population size on time to find an optimal solution.

The basic parameter setting uses a population of size $p = 100$, a population of elite solutions of size $p_e = 15$, a mutant population size of $p_m = 10$, and an elite allele inheritance probability of $\rho_e = 0.7$.

Figure 3.1 compares four settings for population size: 10, 40, 70, and 100. For each setting, the BRKGA was independently run 50 times and CPU times to optimal solution were recored. While there is not much difference between the small population settings of 10 and 40, one can begin to observe speedups for the population of 70 and even more on the population of 100. Since time per generation increases with population size, in those instances that many generations are needed to find an optimal solution, the large-population BRKGAs tend to take longer than their small population counterparts. This is clearly made up for by the many more short running times of the large population variants.

Figure 3.2 shows time-to-optimal solution plots for four different elite population sizes: 5, 15, 25, and 50. The figure shows that elite sets of 15 to 25% of the full

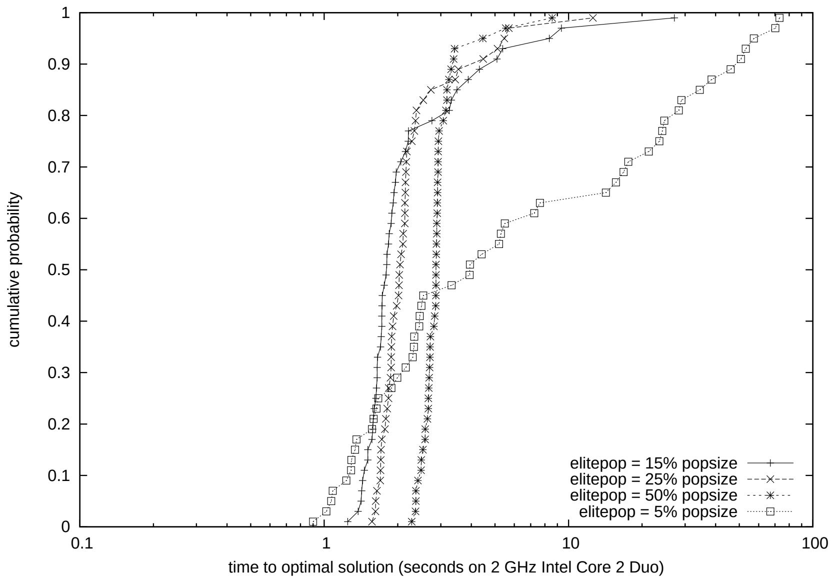

FIGURE 3.2. Effect of elite set size on time to find an optimal solution.

population tend to cause the BRKGA to perform better that a large set of 50% of the population and much better than a small set with only 5% of the population.

Figure 3.3 illustrates the effect of the size of the set of mutant solutions on the time taken by the BRKGA to find an optimal solution. Four sizes were used: 3%, 10%, 30%, and 50% of the full population. The figure shows that it does not pay off to use either a too small or too large set of mutant solutions. The runs using 10% of the full population as the mutant set appear to lead to the BRKGA with the best performance. The large mutant set of half of the population led to the BRKGA with the worse performance.

Figure 3.4 illustrates the effect of different values of inheritance probability on the time to find an optimal solution. Four values were used for ρ_e: 30%, 50%, 70%, and 90%. While $\rho_e = 30\%$ violates the requirement that $\rho_e > 50\%$ and does not lead to a BRKGA with good performance, it is not as bad a being very greedy and using $\rho_e = 90\%$. The greedy variant turned out to have the worst performance while the two implementations using probabilities in the recommended range did the best.

3.4. **Starting population.** In Section 2, we initialize the population with p vectors of random keys. An alternative, is to populate the starting population with a few solutions obtained with another heuristic for the problem being solved. This was done, for example, in Buriol et al. (2005), where a BRKGA is proposed for solving the weight setting problem in OSPF routing. The initial population is made up of one element with the solution found by the heuristic InvCap while the remaining elements are randomly generated. While for all BRKGAs it is easy to apply the decoder to find the solution corresponding to a given vector of random keys, the opposite, i.e. finding a vector of random keys from a solution of the combinatorial optimization problem, may not be straightforward. This, however, was not the case

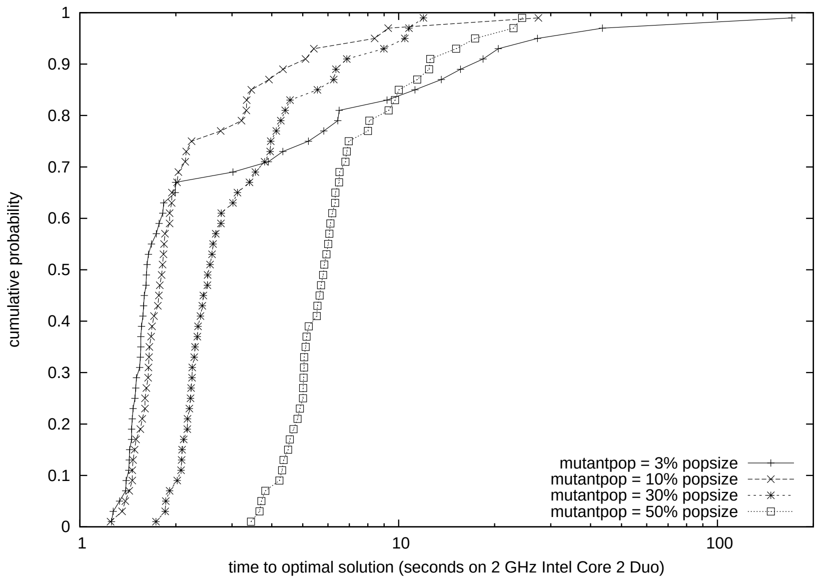

FIGURE 3.3. Effect of mutant set size on time to find an optimal solution.

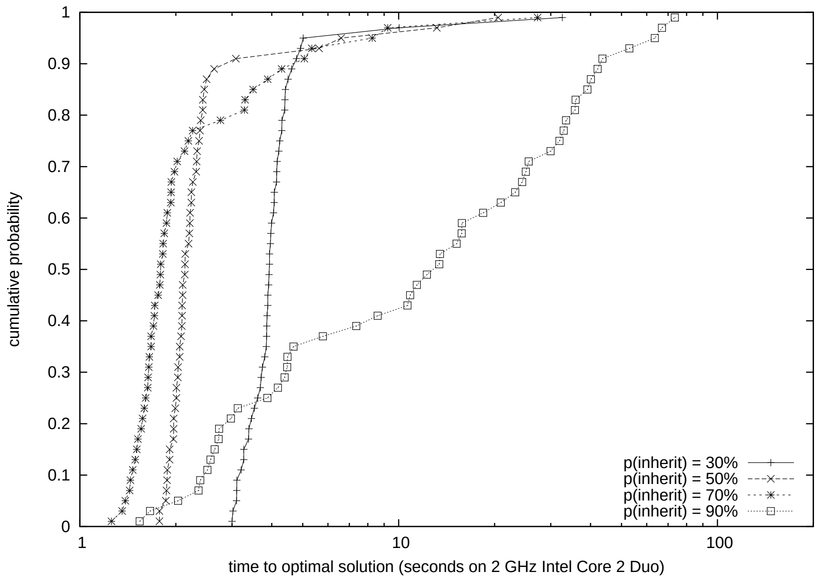

FIGURE 3.4. Effect of inheritance probability on time to find an optimal solution.

in Buriol et al. (2005) where given the solution space consists of vectors of integer weights in the range $[1, w_{max}]$ and therefore recovering vectors of random keys is trivial. Another type of solution representation that is easy to map back to a vector of random keys is a permutation array.

3.5. **Parallel implementation.** Biased random key genetic algorithms have a natural parallel implementation. Candidates for parallelization include the operations

- generate p vectors of random keys,
- generate p_m mutants in next population,
- combine elite parent with other parent to produce offspring,
- decode each vector of random keys and compute its fitness.

Since each of these four operations involves independent computations, they can each be computed in parallel. The first three of these operations are not as computationally intensive as the fourth and on those operations parallelization is not expected to contribute to significantly speedup the overall algorithm. On the other hand, the last operation (decoding and fitness evaluation) can easily account for most of the overall cycles and one should expect a significant speedup in the execution of the program by parallelizing it.

Another type of parallelization is the use of multiple populations. Multiple populations evolve independently of one another and periodically exchange solutions.

4. Applications

In this section, we give examples of biased random-key genetic algorithms. For each application, we provide a brief description of the problem and descriptions of the chromosome (solution encoding) and the decoder. We begin by considering some applications in communication networks, including OSPF routing, survivable network design, and routing and wavelength assignment. We then consider the problem of assigning tolls in a transportation network to minimize road congestion. This is followed by a number of scheduling applications, including job shop scheduling, resource constrained single- and multi-project scheduling, single machine scheduling, and assembly line balancing. We conclude with applications to manufacturing cell formation, two-dimensional packing, and concave-cost network flow optimization.

4.1. **OSPF weight setting.** Ericsson et al. (2002) and Buriol et al. (2005) describe BRKGA heuristics for a problem in Internet Protocol network routing. They address the weight-setting problem in OSPF (Open Shortest Path First) routing. A related BRKGA is described in Reis et al. (2009), where DEFT (Distributed Exponentially-Weighted Flow Splitting), a different routing protocol, is used.

4.1.1. *Problem definition.* Consider a directed network graph $G = (N, A)$ where N denotes the set of nodes (where routers are located) and A denotes the set of links connecting the routers with a capacity c_a for each $a \in A$, and a demand matrix D that, for each pair $(s, t) \in N \times N$, gives the demand $d_{s,t}$ in traffic flow from node s to node t. The *OSPF weight-setting problem* consists in assigning positive integer weights $w_a \in [1, w_{max}]$ to each arc $a \in A$, such that a measure of routing cost is minimized when the demands are routed according to the rules of the OSPF protocol. The routing cost is a function of the link capacities and the total traffic that traverses each link. In OSPF, traffic between nodes s and t is routed on a shortest-weight path connecting these nodes. The OSPF protocol allows for $w_{max} \leq 65535$.

4.1.2. *Solution encoding.* Each solution is encoded as a vector x of random keys of length $n = |A|$, where the i-th gene corresponds to the i-th link of G.

4.1.3. *Chromosome decoder.* To decode a link weight w_i from x_i (for $i = 1, \dots, n$), simply compute $w_i = \lceil x_i \times w_{max} \rceil$. Once link weights are computed, shortest weight graphs from each node to all other nodes in the graph can be derived, traffic can be routed, the total traffic on each link computed resulting in a routing cost which is the fitness of the solution. Buriol et al. (2005) apply a fast local search to the solution in an attempt to further reduce the routing cost. Let A^* be the set of five links with the highest routing cost values. For each link $i \in A^*$, a local improvement heuristic attempts to increase w_i by one unit at a time in a specified range. If the total routing cost can be reduced this way, a new set A^* is constructed and the process repeats itself. If, after scanning the five links, the cost cannot be reduced, then the procedure stops.

4.2. **Survivable network design.** Given a set of nodes in a network, a traffic matrix estimating the demand, or traffic, between pairs of these nodes, a set of arcs, each having endpoints at a pair of the given nodes, a set of possible fiber link types, each with an associated capacity and cost per unit of length, and a set of failure configurations, the survivable network design problem seeks to determine how many units of each cable type will be installed in each link such that all of the demand can be routed on the network under the no failure and all failure modes such that the total cost of the installed fiber is minimized. Buriol et al. (2007) proposed a BRKGA to design survivable networks where traffic is routed using the Open Shortest Path First (OSPF) protocol and there is only one link type. Andrade et al. (2006) extended this BRKGA to handle composite links, i.e. the case where there are several fiber types.

4.2.1. *Problem definition.* Given a directed graph $G = (V, E)$, where V is the set of routers and E is the set of potential arcs where fiber can be installed, and a demand matrix D, that for each pair $(u, v) \in V \times V$, specifies the demand $D_{u,v}$ between u and v. Arc $e \in E$ has length d_e. Link types are numbered $1, \dots, T$. Link type i has capacity c_i and cost per unit of length p_i. We wish to determine integer OSPF weights $w_e \in [1, 65535]$ as well as the number of copies of each link type to be deployed at each arc such that when traffic is routed according to the OSPF protocol in a no-failure or any single arc failure situation there is enough installed capacity to move all of the demand and the total cost of the installed capacity is minimized.

4.2.2. *Solution encoding.* Assume arcs in E are numbered $1, \dots, |E|$. A solution of the survivable network design problem is encoded as a vector x of $|E|$ random keys. The i-th key corresponds to the i-th arc.

4.2.3. *Chromosome decoder.* To produce the OSPF weight w_i of the i-th arc, scale the random key by the maximum weight, i.e. set $w_i = \lceil x_i \times 65535 \rceil$. For the no-failure mode and each failure mode, route the traffic using the OSPF protocol using the computed arc weights, compute the loads on each arc and record the maximum load over the no-failure and all failure modes. For each arc, determine an optimal allocation of link types such that the resulting capacity of the set of composite links is enough to accommodate the maximum load on the arc. Compute the cost of the required links.

4.3. **Routing and wavelength assignment.** The problem of routing and wavelength assignment (RWA) in wavelength division multiplexing (WDM) optical networks consists in routing a set of lightpaths (a lightpath is an all-optical point-to-point connection between two nodes) and assigning a wavelength to each of them, such that lightpaths whose routes share a common fiber are assigned different wavelengths. Noronha et al. (2008) propose a BRKGA for routing and wavelength assignment with the goal of minimizing the number of different wavelengths used in the assignment (this variant of the RWA is called min-RWA). This BRKGA extends the best heuristic in the literature by embedding it into an evolutionary framework.

4.3.1. *Problem definition.* Given a bidirected graph $G = (V, E)$ that represents the physical topology of the optical network, where V is the set of nodes and E is the set of fiber links, and a set T of lightpaths to be established. Each lightpath is characterized by its pair of endpoints $\{s, t\} \in V \times V,\ s \neq t$. Each lightpath is routed on a single path from s to t and is assigned the same wavelength for the entire path. If two lightpaths share an arc, they must be assigned different wavelengths. The objective is to minimize the number of wavelengths used.

4.3.2. *Solution encoding.* A solution of the routing and wavelength assignment problem is encoded in a vector x of $|\mathcal{T}|$ random keys, where $|\mathcal{T}|$ is the number of lightpaths. The key x_i corresponds to the i-th lightpath, for $i = 1, \ldots, |\mathcal{T}|$.

4.3.3. *Chromosome decoder.* Skorin-Kapov (2007) proposed the current state-of-the-art heuristics for min-RWA. Each wavelength is associated with a different copy of the graph G. Lightpaths that are arc disjointly routed on the same copy of G are assigned the same wavelength. Copies of G are associated with the bins and lightpaths with the items of an instance of the bin packing problem. Therefore, min-RWA can be reformulated as the problem of packing all the lightpath requests in a minimum number of bins. Let $\mathit{minlength}(i)$ be the number of hops in the path with the smallest number of arcs between the endnodes of ligthpath i in G. These values are only used for sorting the lightpaths in the decoding heuristics, even though the lightpaths are not necessarily routed on shortest paths. This occurs because whenever a lightpath is routed on a copy of G (or, equivalently, placed in the corresponding bin), all arcs in its route are deleted from this copy to avoid that other lightpaths use them. Therefore, the next lightpaths routed in this copy of G might be routed on a path that is not a shortest path in the original graph G. The classical best fit decreasing heuristic is used to pack the lightpaths. Since the number of lightpaths is usually much greater than the diameter of the graph, there are many lightpaths with the same *minlength* value. In the case of ties, Skorin-Kapov (2007) recommended breaking them randomly. The BRKGA uses the vector of random keys to randomly perturb the values of $\mathit{minlength}(i)$ and get rid of the ties. These values are adjusted as $\mathit{minlength}(i) \leftarrow \mathit{minlength}(i) + x(i)$.

4.4. **Tollbooth location and tariff assignment.** In transportation networks, it is desirable to direct traffic so as to minimize congestion, thus decreasing user travel times and improving network utilization. One way to persuade drivers to avoid certain routes and favor others is by charging toll for drivers to use certain segments of the network. The objective of the tollbooth location and tariff assignment problem is to locate a given number of tollbooths on links of the network and determine toll values to impose on users of those links such that the average user travel time is minimized. Buriol et al. (2009) describe a BRKGA for this problem.

4.4.1. *Problem definition.* Given a network topology and certain traffic flow demands, we levy tolls on arcs, seeking an efficient system such that the resulting set of least-cost user paths is optimal for the overall system. Consider a directed graph $G = (N, A)$, with N representing the set of nodes and A the set of arcs. Each arc $a \in A$ has an associated capacity c_a and cost Φ_a, which is a function of the load ℓ_a (or flow) on the arc, the time t_a to transverse the arc when there is no traffic on the arc, a power parameter n_a, and a parameter Γ_a. In real-world traffic networks, arc (road segment) delays are generally described by nonlinear functions associated with these network congestion parameters. We assume that Φ_a is a strictly increasing, convex function. In addition, define $K \subseteq N \times N$ to be the set of commodities, or origin-destination (OD) pairs, having $o(k)$ and $d(k)$ as origin and destination nodes, respectively, for all $k \in K = \{1, \ldots, |K|\}$. Each commodity $k \in K$ has an associated demand of traffic flow Δ_k defined, i.e. for each OD pair $\{o(k), d(k)\}$, there is an associated amount of flow Δ_k that emanates from node $o(k)$ and terminates at node $d(k)$. Furthermore, define x_a^k to be the contribution of commodity k to the flow on arc a. The traffic optimization problem can be written as

$$\min \Phi = \sum_{a \in A} \ell_a t_a \left[1 + \Gamma_a (\ell_a / c_a)^{n_a}\right] / \sum_{k \in K} \Delta_k$$

subject to

$$\ell_a = \sum_{k \in K} x_a^k, \; \forall a \in A,$$

$$\sum_{i:(j,i) \in A} x_{(j,i)}^k - \sum_{i:(i,j) \in A} x_{(i,j)}^k = \begin{cases} -\Delta_k, & \forall\, j \in N,\; k \in K : \; j = d(k), \\ \Delta_k, & \forall\, j \in N,\; k \in K : \; j = o(k), \\ 0, & \forall\, j \in N,\; k \in K : \; j \neq o(k),\; j \neq d(k), \end{cases}$$

$$x_a^k \geq 0, \; \forall\, a \in A,\; k \in K.$$

Given a number κ of tolls to place in the network, the objective is to determine a set of κ arcs in A where tolls will be placed and tariffs for each toll such that if users travel on least-cost routes, the resulting x_a^k decision variables (for all $a \in A$ and $k \in K$) will be such that the above traffic optimization problem is solved.

4.4.2. *Solution encoding.* A solution of the tollbooth location and tariff assignment problem is encoded in a vector χ of $2|A|$ random keys. The first $|A|$ random keys correspond to the tariffs on the arcs while the last $|A|$ keys are used to indicate whether a toll is to be placed on an arc.

4.4.3. *Chromosome decoder.* Define a binary variable y_a for each arc $a \in A$ which takes on value 1 if and only if a toll is levied on arc a. For each arc $a \in A$, let π_a denote the tariff levied by the toll on arc a. Finally, let T_a be the value of the maximum toll that can be levied on arc a. Given a chromosome χ with $2|A|$ random keys, let $y_a = 1$ if and only if $\chi_{|A|+a} \geq 0.5$. The corresponding tariff on arc a is $\pi_a = \lceil \chi_a \times T_a \rceil \times y_a$. To compute the decision variables x_a^k of the traffic assignment problem, all demands are routed on least-cost routes in the network. A local search procedure is applied on the tariffs to attempt to decrease the value of the value of the objective function of the traffic assignment model. The crossover operator handles the last $|A|$ random keys in a way that is slightly different from the standard parameterized uniform crossover that is applied to the first $|A|$ random keys. For all arcs on which both parent solutions agree on whether or not to place a toll, the child inherits the random key of any one of the parents. If the parents do not agree on all locations, then additional tolls will need to be assigned in the child chromosome to guarantee that κ arcs have tolls. For each additional toll, the child inherits the chromosome of a parent having $\chi_a \geq 0.5$ with probability that favors inheritance from the elite parent.

4.5. **Job-shop scheduling.** Gonçalves et al. (2005) present a BRKGA heuristic for the job-shop scheduling problem.

4.5.1. *Problem definition.* We are given n jobs, each composed of several operations that must be processed on m machines. Each operation uses one of the m machines for a fixed duration. Each machine can process at most one operation at a time and once an operation initiates processing on a given machine it must complete processing on that machine without interruption. The operations of a given job have to be processed in a given order. The problem consists in finding a schedule of the operations on the machines that minimizes the makespan C_{max} , i.e. the finish time of the last operation completed in the schedule, taking into account the precedence constraints.

4.5.2. *Solution encoding.* Let p be the number of operations. The proposed random-key vector x used to encode a solution has size $2p$. Its first p genes determine the priorities of the operations, i.e. x_i corresponds to the priority of operation i, for $i = 1, \ldots, p$. The last p genes are used to encode the delay used to schedule an operation, i.e. for $i = 1, \ldots, p$, x_{p+i} is used to compute the delay of operation i. The delay of operation i is defined to be $x_{p+i} \times D$, where D is the duration of the longest operation.

4.5.3. *Chromosome decoder.* A parameterized active schedule is constructed using the priorities and delays encoded in the chromosome. This schedule is an active schedule, i.e. it allows a machine to be idle even when there is an operation available for it to process. Among all operations i that would require a delay at most $x_{p+i} \times D$, the operation i with the highest priority x_i is scheduled on the machine.

4.6. **Resource constrained project scheduling.** In project scheduling a set of activities needs to be scheduled. Precedence relations between activities constrain the start of an activity to occur after the completion of another. The objective is to minimize the makespan, i.e. minimize the completion time of the last scheduled activity. When activities require resources with limited capacities we have resource constrained project scheduling. Mendes et al. (2009) describe a BRKGA for the resource constrained project scheduling problem.

4.6.1. *Problem definition.* A project consists of $n + 2$ activities. To complete the project, each activity has to be processed. Let $J = \{0, 1, \ldots, n, n + 1\}$ denote the set of activities to be scheduled and $K = \{1, \ldots, k\}$ the set of resources. Activities 0 and $n + 1$ are dummies, have no duration, and represent the initial and final activities. The activities are interrelated by two kinds of constraints: (1) Precedence constraints force each activity j to be scheduled after all predecessor activities P_j are completed; (2) Activities require resources with limited capacities. While being processed, activity j requires $r_{j,k}$ units of resource type $k \in K$ during every time instant of its non-preemptable duration d_j. Resource type k has a limited capacity of R_k at any point in time. The parameters d_j, $r_{j,k}$, and R_k are assumed to be integer, nonnegative, and deterministic. For the project start and end activities, we have $d_0 = d_{n+1} = 0$ and $r_{0,k} = r_{n+1,k} = 0$ for all $k \in K$. Let F_j represent the finish time of activity j. A schedule can be represented by a vector of finish times $(F_1, \ldots, F_{n+1})$ and its makespan is $C_{max} = \max\{F_1, \ldots, F_{n+1}\}$. The problem consists in finding a schedule of the activities, taking into account the resources and the precedence constraints, that minimizes the makespan.

4.6.2. *Solution encoding.* A solution is encoded with a vector x of $2n$ random keys. The first n keys correspond to the priorities of the activities while the last n are used to determine the delay when scheduling an activity.

4.6.3. *Chromosome decoder.* For each activity $j \in J$ not yet scheduled, the delay $\delta_j = x_{n+j} \times 1.5 \times \bar{\delta}$ is computed, where $\bar{\delta}$ is the maximum duration of any activity. Activities are scheduled, one at a time, at discrete points in time, starting from time $t = 0$. At time t, all activities $j \in J$ whose precedecessors have completed processing or will have completed processing by time $t + \delta_j$ are considered to be candidates to be scheduled. These activities are scheduled in the order determined by their priorities (the priority of activity j is x_j). Each is scheduled as soon as all of its predecessors complete processing and all resources it requires are available. The next schedule time is the earliest completion time among all activities being processed at and after time t. This process is repeated until all activities have been scheduled. The makespan C_{max} is the completion time of the last activity to complete processing. A new and more effective decoder for this problem is described in Gonçalves et al. (2009a)

4.7. **Resource constrained multi-project scheduling.** In the resource constrained multi-project scheduling problem, activities that make up several projects must be scheduled. These activities share one or more resources having limited capacities. Associated with each project are its release and due dates. The project cannot begin processing before the release date and should finish as close as possible to its due date. There are penalties associated with earliness, tardiness, and total processing time of the project and the objective is to schedule the activities such that the sum of the penalties of the projects is minimized. Gonçalves et al. (2008) describe a BRKGA for resource constrained multi-project scheduling.

4.7.1. *Problem definition.* The problem consists of a set $\mathcal{I}$ of projects, where each project $i \in \mathcal{I}$ is composed of activities $j = \{N_{i-1} + 1, \ldots, N_i\}$, where activities $N_{i-1} + 1$ and N_i are dummies and represent the initial and final activities of project i. $\mathcal{J}$ is the set of activities and $\mathcal{K} = \{1, \ldots, k\}$ is a set of renewable resources types. The activities are interrelated by two kinds of constraints. First, precedence constraints force each activity $j \in \mathcal{J}$ to be scheduled after all its predecessor activities $\mathcal{P}_j$ are completed. Second, processing of the activities is subject to the availability of resources with limited capacities. While being processed, activity $j \in \mathcal{J}$ requires $r_{j,k}$ units of resource type $k \in \mathcal{K}$ during every time instant of its non-preemptable duration d_j. Resource type $k \in \mathcal{K}$ has a limited availability of R_k at any point in time. Parameters d_j, $r_{j,k}$, and R_k are assumed to be non-negative and deterministic. We assume that start and end activities of each project have zero processing times and do not require any resource. Activities 0 and $N+1$ are dummy activities, have no duration, and correspond to the start and end of all projects. Activity 0 preceeds all of the dummy initial activities of the individual projects and activity $N+1$ is preceeded by all of the dummy final activities of all the jobs. Using these dummy actvities, the multi-project scheduling problem can be treated as if it were a single project. The objective is to minimize $a\sum_{i\in\mathcal{I}}(aT_i^3+bE_i^2+c(CD_i-BD_i)^2/CPD_i)$, where T_i, E_i, CD_i, BD_i, and CPD_i are, respectively, the tardiness, earliness, conclusion time, start time, and critical path duration of project i.

4.7.2. *Solution encoding.* The encoding of the solution is identical to the one used in the BRKGA for single-project scheduling described in Section 4.6, i.e. a vector x of $2n$ random keys. The first n keys correspond to the priorities of the activities while the last n are used to determine the delay when scheduling an activity.

4.7.3. *Chromosome decoder.* The decoder is identical to the one used in the BRKGA for single-project scheduling described in Section 4.6 except that instead of computing the makespan, this decoder computes the penalty $a\sum_{i\in\mathcal{I}}(aT_i^3+bE_i^2+c(CD_i-BD_i)^2/CPD_i)$ as the fitness of the chromosome.

4.8. **Early tardy scheduling.** Valente et al. (2006) describe a BRKGA for a single machine scheduling problem with earliness and tardiness costs and no unforced machine idle time. Such problems arise in just-in-time production, where goods are produced only when they are needed, since jobs are scheduled to conclude as close as possible to their due dates. The early cost can be seen, for example, as the cost of completing a project early in PERT-CPM analyses, deterioration in the production of perishable goods, or a holding cost for finished goods. The tardy cost is often associated with rush shipping costs, lost sales, or loss of goodwill. It is assumed that no unforced machine idle time is allowed, and therefore the machine is only idle when no jobs are available for processing. This assumption represents a type of production environment where the machine idleness cost is higher than the cost incurred by completing a job early, or the machine is heavily loaded, so it must be kept running in order to satisfy the demand.

4.8.1. *Problem definition.* A set of n independent jobs $\{J_{1,}\ldots,J_n\}$ must be scheduled without preemptions on a single machine that can handle at most one job at a time. The machine and the jobs are assumed to be continuously available from time zero onwards and machine idle time is not allowed. Job J_j, $j=1,\ldots,n$, requires a processing time p_j and should ideally be completed on its due date d_j. For any schedule, the earliness and tardiness of J_j can be respectively defined as $E_j=\max\{0,d_j-C_j\}$ and $T_j=\max\{0,C_j-d_j\}$, where C_j is the completion time of J_j. The objective is to find the schedule that minimizes the sum of the earliness and tardiness costs of all jobs, i.e. $\sum_{j=1}^{n}(h_jE_j+w_jT_j)$, where h_j and w_j are, respectively, the per unit earliness and tardiness costs of job J_j.

4.8.2. *Solution encoding.* A solution of the early tardy scheduling problem is encoded in a vector x of n random keys that, when sorted, corresponds to the ordering that the jobs are processed on the machine.

4.8.3. *Chromosome decoder.* Given a vector x of n random keys, a solution is produced by first sorting the vector to produce an ordering of the jobs. The jobs are scheduled on the machine and the total cost is computed. A simple local search scans the jobs, from first to last, testing if consecutive jobs can be swapped in the order of processing. If a swap decreases the cost of the schedule, the swap is done, the cost recomputed, and the scan continues from that job until the last two jobs are tested.

4.9. **Single machine scheduling with linear earliness and quadratic tardiness penalties.** Valente and Gonçalves (2008) present a BRKGA for a single machine scheduling problem with linear earliness and quadratic tardiness penalties. They consider an objective function with linear earliness and quadratic tardiness costs. A linear penalty is then used for the early jobs, since the costs of maintaining and managing this inventory tend to be proportional to the quantity held in stock. However, late deliveries can result in lost sales, loss of goodwill, and disruptions in stages further down the supply chain. A quadratic tardiness penalty is used for the tardy jobs. In many situations this is preferable to the more usual linear tardiness or maximum tardiness functions. Finally, no machine idle time is allowed.

4.9.1. *Problem definition.* A set of n independent jobs $\{J_1, \dots, J_n\}$ must be scheduled on a single machine that can handle at most a single job at a time. The machine is assumed to be continuously available from time zero onwards, and preemptions are not allowed. Job J_j, for $j = 1, \dots, n$, requires a processing time p_j and should ideally be completed on its due date d_j. For any schedule, the earliness and tardiness of J_j can be respectively defined as $E_j = \max\{0, d_j - C_j\}$ and $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of J_j. The objective is to find a schedule that minimizes the sum of linear earliness and quadratic tardiness costs $\sum_{j=1}^{n}(E_j + T_j^2)$, subject to the constraint that no machine idle time is allowed.

4.9.2. *Solution encoding.* A solution of the single machine scheduling problem with linear earliness and quadratic tardiness penalties is encoded in a vector x of n random keys that, when sorted, corresponds to the ordering that the jobs are processed on the machine.

4.9.3. *Chromosome decoder.* Given a vector x of n random keys, a solution is produced by first sorting the vector to produce an ordering of the jobs. The jobs are scheduled on the machine and the total cost is computed. Then three simple local search procedures, adjacent pairwise interchange (API), 3-swaps (3SW), and largest cost insertion (LCI) are applied. At each iteration, API considers in succession all adjacent job positions. A pair of adjacent jobs is swapped if such an interchange improves the objective function value. If necessary, the solution is updated. This process is repeated until no improvement is found in a complete iteration. Next, 3SW is applied. It is similar to API, except that it considers three consecutive job positions instead of an adjacent pair of jobs. All possible permutations of these three jobs are analysed, and the best configuration is selected. If necessary, the solution is updated. Once more, the procedure is applied repeatedly until no improvement is possible. Finally LCI is applied. At each iteration, LCI selects the job with the largest objective function value. The selected job is removed from its position i in the schedule, and inserted at position j, for all $j \neq i$. The best insertion is performed if it improves the objective function value. If necessary, the solution is updated. This process is also repeated until no improving move is found.

4.10. **Assembly line balancing.** Assembly or fabrication lines are used to manufacture large quantities of standardized products. An assembly line consists of a sequence of m workstations, connected by a conveyor belt, through which the product units flow. Each workstation performs a subset of the n operations necessary for manufacturing the products. Each product unit remains at each station for a fixed time C called the cycle time. In traditional assembly lines, workstations are consecutively arranged in a straight line. Each product unit proceeds along this line and visits each workstation once. The major decision consists in defining an assignment of operations to workstations such that the line efficiency is maximized. Gonçalves and Almeida (2002) describe a BRKGA for assembly line balancing.

4.10.1. *Problem definition.* In the assembly line problem, a single product is manufactured in large quantities in a process involving n operations, each of which takes t_j time units to process, for $j = 1, \dots, n$. Operations are partially ordered by precedence relations, i.e. when an operation j is assigned to a station k, each operation i which precedes j must be assigned to one of the workstations. Each operation must be assigned to exactly one workstation. The sets of operations S_k, $k = 1, \dots, m$, are called workstation loads. Workstations are numbered consecutively along the line. The total operation time of the operations assigned to a station k, called workstation time $t(S_k)$, must not exceed the cycle time, i.e. $t(S_k) = \sum_{\{j \in S_k\}} t_j \leq C$, for $k = 1, \dots, m$. Gonçalves and Almeida (2002) deal with the SALBP-1 variant of the problem, where we are given the cycle time C and the objective is to minimize the number m of stations.

4.10.2. *Solution encoding.* A solution of the assembly line problem is encoded in a vector x of n random keys, where n is the number of operations. The key x_i corresponds to the piority of the i-th operation.

4.10.3. *Chromosome decoder.* The decoder takes as input a vector x of n random keys and returns an assignment of operations to work stations. The random key x_i is the priority of operation i. Given a set of operation priorities, a station-oriented heuristic is used to assign operations to workstations. This procedure starts with station 1 and considers the other stations successively. In each iteration, the operation with highest priority in the candidate set is chosen and assigned to the current station. The current station is closed and the next station is opened when the candidate set is empty, i.e. when adding any operation to the station would exceed the cycle time. Subsequently, a local search procedure is used to try to improve the solution obtained by the station-oriented heuristic. The local search attempts to swap long operations scheduled in downstream workstations with shorter operations in upstream workstations with the objective of freeing up a downstream workstation.

4.11. **Manufacturing cell formation.** The fundamental problem in cellular manufacturing is the formation of product families and machine cells. Gonçalves and Resende (2004) present a BRKGA for manufacutring cell formation.

4.11.1. *Problem definition.* Given P products and M machines, we wish to assign products and machines to a number of product-machine cells such that inter-cellular movement is minimized and machine utilization within a cell is maximized. Let the binary matrix $A = [a]_{i,j}$ be such that $a_{i,j} = 1$ if and only if product i uses machine j. By reordering the rows and columns of A and moving the cells so they are located on or near the diagonal of the reordered matrix, a measure of efficacy of the solution can be defined to be $\mu = (n_1 - n_1^{out})/(n_1 + n_0^{in})$, where n_1 is the number of ones in A, n_1^{out} is the number of ones outside the diagonal blocks, and n_0^{in} is the number of zeroes inside the diagonal blocks. We seek to maximize μ.

4.11.2. *Solution encoding.* A solution to the cellular manufacturing problem is encoded as a vector x of $M+1$ random keys, where the first M random keys are used to assign the machines to cells and the last random key determines the number of cells. Assuming that the smallest cell allowed has dimension 2×2, the maximum number of cells is $\bar{C} = \lfloor M/2 \rfloor$. The number of cells in a solution is therefore $C = x_{M+1} \times \bar{C}$ and machine i is assigned to cell $x_i \times C$.

4.11.3. *Chromosome decoder.* The decoder first assigns products to the cell that maximizes the efficacy with respect to the machine-cell assignments. Once products are assigned, then machines are reassigned to the cells that maximize the efficacy. This process of reassigning products and machines is repeated until there is no further increase in the efficacy measure.

4.12. **Constrained two-dimensional orthogonal packing.** In the constrained two-dimensional (2D), non-guillotine restricted, packing problem, a fixed set of small weighted rectangles has to be placed, without overlap, into a larger stock rectangle so as to maximize the sum of the weights of the rectangles packed. Gonçalves (2007) proposed the first BRKGA for this problem. This was improved in Gonçalves and Resende (2009), where a new BRKGA, that uses a novel placement procedure and a new fitness function to drive the optimization, was proposed.

4.12.1. *Problem definition.* The two-dimensional packing problem consists in packing into a single large planar stock rectangle (W, H), of width W and height H, n smaller rectangles $(w_i, h_i), i = 1, \ldots, n$, each of width w_i and height h_i. Each rectangle i has a fixed orientation (i.e. cannot be rotated); must be packed with its edges parallel to the edges of the stock rectangle; and the number x_i of pieces of each rectangle type that are to be packed must lie between P_i and Q_i, i.e. $0 \leq P_i \leq x_i \leq Q_i$, for all $i = 1, \ldots, n$. Each rectangle $i = 1, \ldots, n$ has an associated value equal to v_i and the objective is to maximize the total value $\sum_{i=1}^{n} v_i x_i$ of the rectangles packed. Without significant loss of generality, it is usual to assume that all dimensions W,H, and $(w_i, h_i), i = 1, \ldots, n$, are integers.

4.12.2. *Solution encoding.* A solution of the two-dimensional packing problem is encoded in a vector x of $2N$ random keys, where $N = \sum_{i=1}^{n} n_i$. The first N random keys correspond to the ordering that the rectangles are packed while the last N keys indicate how the rectangles are to be placed in the stock rectangle.

4.12.3. *Chromosome decoder.* Given a vector x of random keys, the rectangles are packed by scanning x starting from the first component. For $i = 1, \ldots, N$, let $t = \lceil x_i \times n \rceil$ denote the type of rectangle to be packed next. If there are no more rectangles of type t available to be packed, the decoder proceeds to the next value of

i. Otherwise it proceeds to pack one or more rectangles of type t, up to the maximum number of available rectangles of that type using a heuristic determined by the value of x_{N+i}. If $x_{N+i} \leq 0.5$, then the left-bottom heuristic is used. Otherwise, the rectangle is placed using the bottom-left heuristic. If the left-bottom heuristic is applied, a vertical layer of rectangles is placed. Similarly, if the bottom-left heuristic is used, a horizontal layer of rectangles is placed. The fitness of the chromosome is the total weight of the packed rectangles plus a term that tries to capture the improvement potential of different packings which have the same total value.

4.13. **General concave minimum cost flow.** Fontes and Gonçalves (2007) proposed a BRKGA for the general minimum concave cost network flow problem (MCNFP). Concave cost functions in network flow problems arise in practice as a consequence of taking into account economic considerations. For example, fixed costs may arise and economies of scale often lead to a decrease in marginal costs. The genetic algorithm makes use of a local search heuristic to solve the problem. The local search algorithm tries to improve the solutions in the population by using domain-specific information. The BRKGA is used to solve instances with both concave routing costs and fixed costs.

4.13.1. *Problem definition.* Given a graph $G = (W, A)$, where W is a set of $n + 1$ nodes (node $n+1$ denotes the source node and nodes $1, \ldots, n$ denote demand nodes) and a set A of m directed arcs, $A \subseteq \{(i, j) : \ i, j \in W\}$. Each node $i \in W \setminus \{n + 1\}$ has an associated nonnegative integer demand value r_i. The supply at the source node equals the sum of the demands required by the n demand nodes. A general nondecreasing and nonnegative concave cost function g_{ij} is associated with each arc (i, j) and satisfies $g_{ij}(0) = 0$. The objective is to find a subset S of arcs to be used and the flow x_{ij} routed through these arcs, such that the demands are satisfied and at minimum cost. A concave MCNFP has the property that it has a finite solution if and only if there exists a direct path going from the source node to every demand node and if there are no negative cost cycles. Therefore, a flow solution is a tree rooted at the single source spanning all demand nodes. Thus, the objective is to find an optimal tree rooted at the source node that satisfies all customers demand at minimum cost.

4.13.2. *Solution encoding.* A solution of the MCNFP is encoded in a vector x of n random keys that corresponds to the priorities of the demand nodes used in the tree-constructor procedure of the decoder.

4.13.3. *Chromosome decoder.* The decoder builds a tree rooted at the source node. The node priorities in x are used to determine the order by which nodes are considered by the tree constructor. The algorithm repeatedly performs three steps until either a tree or an infeasible solution is produced. The first step consists in finding the highest priority node not yet supplied. In the second step, the algorithm seeks the set of nodes that can act as a parent for the node found in the first step. In the third and last step, the parent is chosen as the highest priority node that does not create an infeasibility, if one exists. A potential solution becomes infeasible if a cycle cannot be avoided. In this case, a high cost is associated with the solution. After a solution is constructed, a local search procedure is applied to it. The local search tries to improve upon a given solution by comparing it with solutions obtained by replacing an arc currently in the solution by an arc not in the solution such that the new solution is still a tree.

5. Concluding remarks

This paper addressed biased random key genetic algorithms (BRKGA), a metaheuristic for combinatorial optimization. BRKGA is contrasted with the random key genetic algorithm (RKGA) of Bean (1994). Though BRKGAs closely resemble RBGAs, they differ in the way parents are selected for mating. Though minor, this difference can lead to heuristics that are much more effective than their RBGA counterparts. The components of BRKGAs are described and their integration into a metaheuristic framework is proposed. This framework separates the problem-independent part of the procedure from the part that is problem-dependent. This way, a BRKGA can be defined by specifying how solutions are encoded and decoded, making it easy to tailor BRKGAs for solving specific combinatorial optimization problems. Implementation issues, including parallelization of the heuristic, are addressed. The paper concludes with a number of applications, where for each one, the encoding and decoding is described in detail.

References

D. V. Andrade, L. S. Buriol, M. G. C. Resende, and M. Thorup. Survivable composite-link IP network design with OSPF routing. In *Proceedings of The Eighth INFORMS Telecommunications Conference*, 2006.

J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA J. on Computing*, 6:154–160, 1994.

L. Breslau, I. Diakonikolas, N. Duffield, Y. Gu, M. Hajiaghayi, D. S. Johnson, H. Karloff, M. G. C. Resende, and S. Sen. Node placement for path disjoint monitoring. Technical report, AT&T Labs Research, Shannon Laboratory, Florham Park, NJ 07932 USA, 2009.

L. S. Buriol, M. G. C. Resende, C. C. Ribeiro, and M. Thorup. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. *Networks*, 46: 36–56, 2005.

L. S. Buriol, M. G. C. Resende, and M. Thorup. Survivable ip network design with ospf routing. *Networks*, 49:51–64, 2007.

L. S. Buriol, M. J. Hirsch, P. M. Pardalos, T. Querido, M. G. C. Resende, and M. Ritt. A hybrid genetic algorithm for road congestion minimization. In *Proceedings of the XLI Symposium of the Brazilian Operational Research Society (XLI SBPO)*, Porto Seguro, Brazil, 2009.

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

D. B. M. M. Fontes and J. F. Gonçalves. Heuristic solutions for general concave minimum cost network flow problems. *Networks*, 50:67–76, 2007.

F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

J. F. Gonçalves. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. *European J. of Operational Research*, 183:1212–1229, 2007.

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

J. F. Gonçalves and M. G. C. Resende. An evolutionary algorithm for manufacturing cell formation. *Computers and Industrial Engineering*, 47:247–273, 2004.

J. F. Gonçalves and M. G. C. Resende. A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. Technical report, AT&T Labs Research, Florham Park, NJ 07733 USA, 2009.

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

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

J. F. Gonçalves, M. G. C. Resende, and J. J. M. Mendes. A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. Technical report, AT&T Labs Research, 2009a.

J. F. Gonçalves, M. G. C. Resende, and R. M. A. Silva. Biased versus unbiased random key genetic algorithms: A experimental analysis. Technical report, AT&T Labs Research, 2009b.

J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

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

T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro. A genetic algorithm with random keys for routing and wavelength assignment. Technical report, AT&T Labs Research, Florham Park, NJ 07932, 2008.

P. M. Pardalos and M. G. C. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, 2002.

R. Reis, M. Ritt, L. S. Buriol, and M. G. C. Resende. A memetic algorithm for OSPF and DEFT routing to minimize network congestion. Technical report, AT&T Labs Research, Florham Park, NJ 07932, 2009.

N. Skorin-Kapov. Routing and wavelength assignment in optical networks using bin packing based algorithms. *European Journal of Operational Research*, 177: 1167–1179, 2007.

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

J. M. S. Valente and J. F. Gonçalves. A genetic algorithm approach for the single machine scheduling problem with linear earliness and quadratic tardiness penalties. *Computers and Operations Research*, 35:doi:10.1016/j.cor.2008.11.016, 2008.

J. M. S. Valente, J. F. Gonçalves, and R. A. F. S. Alves. A hybrid genetic algorithm for the early/tardy scheduling problem. *Asia-Pacific J. of Operational Research*, 23:393–405, 2006.

J.F. Gonçalves: LIAAD, Faculdade de Economia do Porto, Universidade do Porto, Porto, Portugal
E-mail address: `fgoncal@fep.up.pt`

M.G.C. Resende: Algorithms & Optimization Research Department, AT&T Labs Research, Florham Park, New Jersey, USA
E-mail address: `mgcr@research.att.com`