Introdução aos algoritmos genéticos de chaves aleatórias viciadas

Mauricio G. C. Resende
Departamento de Pesquisas em Algoritmos e Otimização – AT&T Labs Research
180 Park Avenue, Florham Park, NJ 07932 – E.U.A.
mgcr@research.att.com

RESUMO

Um algoritmo genético de chaves aleatórias viciadas (biased random-key genetic algorithm – BRKGA) é uma metaheurística evolutiva para problemas de otimização discreta e global baseada no algoritmo de chaves aleatórias de Bean (1994). Cada solução é representada por um vetor de n chaves aleatórias, onde uma chave aleatória é um número real, gerado aleatoriamente, no intervalo contínuo $[0, 1)$. Um decodificador mapeia um vetor de chaves aleatórias numa solução do problema de otimização e calcula o custo desta solução. O algoritmo começa com uma população de p vetores de chaves aleatórias. A cada iteração, os vetores são particionados em um pequeno conjunto com os melhores elementos (o conjunto elite) e o restante (o conjunto não-elite). Todos os elementos elite são copiados sem alteração para a população da próxima iteração. Um número pequeno de vetores de chaves aleatórias (os mutantes) é adicionado à população da próxima iteração. O restante da população da próxima iteração é composta de soluções geradas pela combinação uniforme parametrizada de Spears e DeJong (1991) de pares de soluções, onde uma solução é elite e a outra não.

PALAVRAS CHAVE. Algoritmo genético, Chaves aleatórias, Otimização, Metaheurísticas.

ABSTRACT

A biased random-key genetic algorithm is an evolutionary metaheuristics for discrete and global optimization, based on the genetic algorithm with random keys of Bean (1994). Each solution is encoded as a vector of n random keys, where a random key is a real number, randomly generated, in the continuous interval $[0, 1)$. A decoder maps each vector of random keys to a solution of the optimization problem being solved and computes its cost. The algorithm starts with a population of p vectors of random keys. At each iteration, the vectors are partitioned into two sets, a smaller set of high-valued elite solutions, and the remaining non-elite solutions. All elite elements are copied, without change, to the next population. A small number of random-key vectors (the mutants) is added to the population of the next iteration. The remaining elements of the population of the next iteration are generated by combining, with the parametrized uniform crossover of Spears e DeJong (1991), pairs of solutions, where one is elite and the other not.

KEYWORDS. Genetic algorithm, Random keys, Optimization, Metaheuristics.

1. Algoritmo genético de chaves aleatórias

Bean (1994) descreve uma nova classe de algoritmos genéticos para problemas de otimização combinatória onde soluções podem ser representadas como vetores de permutação. Esse algoritmos, chamados de *algoritmos genéticos de chaves aleatórias* (ou RKGA, do inglês *random-key genetic algorithms*), representam uma solução do problema de otimização com um vetor de chaves aleatórias. Uma chave aleatória é um número real, gerado aleatoriameente, no intervalo contínuo $[0, 1)$. Um *decodificador* é um procedimento que mapeia um vetor de chaves aleatórias numa solução do problema de otimização e calcula o custo desta solução. O decodificador de Bean (1994) simplesmente ordena os elementos do vetor de chaves e com isso gera uma permutação que corresponde aos índices dos elementos ordenados.

Um RKGA evolui uma população, ou conjunto, de p vetores de chaves aplicando o princípio darwinista, onde os indivíduos mais fortes de uma população têm mais chance de encontrar um parceiro e com isso perpetuar seu material genético. O algoritmo começa com uma população inicial de p vetores de n chaves aleatórias e produz uma série de populações. Na k-ésima geração, os p vetores da população são particionados em um conjunto pequeno de $p_e < p/2$ vetores que correspondem às melhores soluções (este conjunto se chama *elite*) e um outro conjunto com o restante da população (chamado de *não-elite*). Todos os vetores elite são copiados, sem mudança, para a população da $k + 1$-ésima geração. Esse elitismo caracteriza o princípio darwinista em um RKGA. Em seguida, p_m vetores de chaves aleatórias sao introduzidos na população da $k + 1$-ésima geração. Esses vetores são chamados de *mutantes* e têm o mesmo papel dos operadores de mutação nos algoritmos genéticos clássicos, i.e. de evitar que a população convirja para um ótimo local não global. Para completar os p elementos da população da $k + 1$-ésima geração, $p - p_e - p_m$ vetores são gerados combinando pares de soluções da população da k-ésima geração, ambos escolhidos aleatoriamente, com a combinação uniforme parametrizada de Spears e DeJong (1991). Sejam a e b os vetores escolhidos como pais e c o filho resultante. No esquema de Spears e DeJong (1991), $c[i]$, o i-ésimo componente do vetor filho, recebe a i-ésima chave de um dos pais. Recebe a chave $a[i]$ com probabilidade ρ_a e $b[i]$ com probabilidade $\rho_b = 1 - \rho_a$.

2. Algoritmo genético de chaves aleatórias viciadas

Como vimos na seção 1, o algoritmo de Bean simula o darwinismo apenas com o elitismo. Por sua vez, o algoritmo de chaves aleatórias viciadas (ou BRKGA (Gonçalves e Resende , 2011a), do inglês *biased random-key genetic algorithm*) vai um pouco além do RKGA no que se refere ao darwinismo. O BRKGA difere do RKGA na forma que os pais são selecionados para cruzamento e como o cruzamento é implementado.

Ambos algoritmos selecionam os pais aleatoriamente e com reposição. Sendo assim, um pai pode ter mais de um filho por geração. Porém, enquanto que no RKGA ambos os pais são escolhidos da população inteira, no BRKGA um pais é sempre escolhido do conjunto elite enquanto que o outro é escolhido do conjunto não-elite (ou, em alguns casos, da população inteira). Como $p_e < p/2$, um vetor elite no BRKGA tem probabilidade $1/p_e$ de ser escolhido a cada cruzamento, o que é maior do que $1/(p - p_e)$, a probabilidade de um vetor não-elite ser escolhido. Pela mesma razão, a probabilidade de um vetor elite específico ser escolhido no BRKGA é maior do que $1/p$, a probabilidade um dado elite ser escolhido no RKGA.

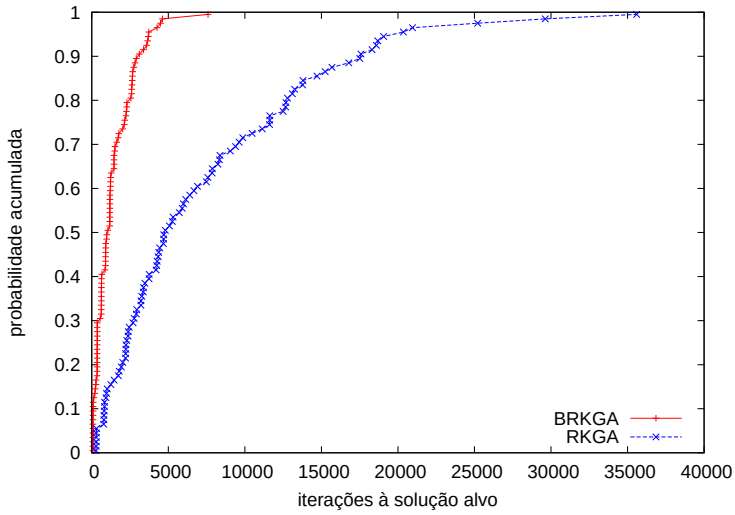

Figura 1. Comparação das distribuições de iterações a uma solução alvo dos algoritmos BRKGA e RKGA.

Ambos os algoritmos fazem a combinação dos pais a e b com o com a cruzamento uniforme parametrizado de Spears e DeJong (1991) para gerar o filho c. Enquanto que no RKGA cada pai pode ser o pai a ou o pai b, no BRKGA o pai a é sempre o pai elite enquanto que o pai b é o não-elite. Como $\rho_a > 1/2$, no BRKGA o filho c tem maior probabilidade de herdar as chaves do pai elite, enquanto que no RKGA isso não ocorre necessariamente.

Essa pequena diferença entre o BRKGA e o RKGA quase sempre faz com que o BRKGA seja superior ao RKGA (Gonçalves et al., 2012). A Figura 1 compara as distribuições de iterações a uma solução alvo de um BRKGA e um RKGA para um problema de recobrimento por pares (Breslau et al., 2011). A figura mostra com clareza a superioridade do BRKGA em relação ao RKGA para esse problema. Esse comportamento na prática independe do problema sendo resolvido.

3. Um modelo para a implementaçao do BRKGA

Algoritmo 1 descreve o pseudo-código de um BRKGA para minimização de $f(x)$, onde $x \in X$ e X é um conjunto discreto de soluções e $f : X \to \mathbb{R}$. Essa implementação é uma versão multi-partida do BRKGA onde várias populações são evoluídas em sequencia e a melhor solução dentre as melhores de todas as populações é retornada como resultado do algoritmo. Após a descrição do pseudo-código, justificaremos o método multi-partida.

Na linha 2, o valor f^* da melhor solução encontrada é inicializado com um valor grande, i.e. não menor do que $f(x^0)$, onde $x^0 \in X$ é alguma solução viável do problema. A evolução de cada população é feita nas linhas 3 a 28. O algoritmo termina quando um critério de parada na linha 3 for satisfeito. Este critério pode ser, por exemplo, número de populações evoluídas, tempo total, ou qualidade da melhor solução encontrada.

Na linha 4 a população sendo evoluída é inicializada com $p = |\mathcal{P}|$ vetores de chaves aleatórias. A evolução da população $\mathcal{P}$ ocorre nas linhas 5 a 27. Essa evolução termina quando um critério de reinicialização é satisfeito na linha 5. Em geral, esse critério é um número máximo de gerações sem que a melhor solução de $\mathcal{P}$ tenha melhorado. A cada

```
1  BRKGA(|𝒫|, |𝒫_e|, |𝒫_m|, n, ρ_a)
2  Inicialize o valor da melhor solução encontrada: f* ← ∞;
3  enquanto critério de parada não for satisfeito faça
4      Gere a população 𝒫 com vetores de n chaves aleatórias;
5      enquanto critério de reinicialização não for satisfeito faça
6          Avalie o custo de cada solução nova em 𝒫;
7          Particione 𝒫 em dois conjuntos: 𝒫_e e 𝒫_ē;
8          Inicialize a população da próxima geração: 𝒫⁺ ← 𝒫_e;
9          Gere o conjunto de mutantes 𝒫_m, cada mutante com n
           chaves aleatórias ;
10         Adicione 𝒫_m à população da próxima geração:
           𝒫⁺ ← 𝒫⁺ ∪ 𝒫_m;
11         para todo i ← 1 até |𝒫| − |𝒫_e| − |𝒫_m| faça
12             Escolha o pai a aleatoriamente de 𝒫_e;
13             Escolha o pai b aleatoriamente de 𝒫_ē;
14             para todo j ← 1 até n faça
15                 Jogue uma moeda viciada com probabilidade
                   ρ_a > 0.5 de dar cara;
16                 se jogada der cara então c[j] ← a[j] ;
17                 senão c[j] ← b[j];
18             fim
19             Adicione o filho c à população da próxima geração:
               𝒫⁺ ← 𝒫⁺ ∪ {c};
20         fim
21         Atualize a população: 𝒫 ← 𝒫⁺;
22         Ache a melhor solução χ⁺ em 𝒫:
           χ⁺ ← argmin{f(χ) | χ ∈ 𝒫};
23         se f(χ⁺) < f* então
24             χ* ← χ⁺;
25             f* ← f(χ⁺);
26         fim
27     fim
28 fim
29 retorna χ*
```

Algoritmo 1: Modelo para algoritmo genético de chaves aleatórias viciada com reinicialização.

geração, ou iteração, as seguintes operações são realizadas: Na linha 6 todas as soluções novas (filhos e mutantes) são decodificadas e seus custos avaliados. Note que cada decodificação e avaliação nesse passo pode ser feita simultaneamente, i.e. em paralelo. Na linha 7, a população $\mathcal{P}$ é particionada em duas subpopulações $\mathcal{P}_e$ (elite) e $\mathcal{P}_{\bar{e}}$ (não elite), onde $|\mathcal{P}_e| < |\mathcal{P}|/2$ e contém as $|\mathcal{P}_e|$ melhores soluções de $\mathcal{P}$ e $\mathcal{P}_{\bar{e}}$ consiste do restante de $\mathcal{P}$, ou seja $\mathcal{P} \setminus \mathcal{P}_e$. $\mathcal{P}^+$ é a população da próxima geração. Ela é inicializada na linha 8 com as soluções elite da geração atual. Na linha 9 a subpopulação $\mathcal{P}_m$ de mutantes é gerada. Cada mutante é um vetor de n chaves aleatórias. O número de mutantes gerados satisfaz a restrição $|\mathcal{P}_m| < |\mathcal{P}|/2$. Essa subpopulação é adicionada à população $\mathcal{P}^+$ da próxima geração na linha 10.

Com $|\mathcal{P}_e| + |\mathcal{P}_m|$ vetores inseridos na população $\mathcal{P}^+$, é necessário a geração de $|\mathcal{P}| - |\mathcal{P}_2| - |\mathcal{P}_m|$ filhos para se completar os $|\mathcal{P}|$ vetores da população $\mathcal{P}^+$. Isso é feito nas linhas 11 a 20. Nas linhas 12 e 13 os pais a e b são escolhidos, respectivamente, de forma aleatória das subpopulações $\mathcal{P}_e$ e $\mathcal{P}_{\bar{e}}$. A geração do filho c a partir dos pais a e b ocorre nas linhas 14 a 18. Uma moeda viciada (com probabilidade $\rho_a > 1/2$ de dar cara) é lançada n vezes. Se o i-ésimo lance der cara, o filho herda a i-ésima chave do pai a. Caso contrário, ele herda a chave do pai b. Gerado o filho, c é adicionado à população $\mathcal{P}^+$ na linha 19.

A geração de $\mathcal{P}^+$ termina quando ela tiver $|\mathcal{P}|$ elementos. Na linha 21, $\mathcal{P}^+$ é copiado em $\mathcal{P}$ para se iniciar outra geração. A melhor solução da atual população em evolução é computada na linha 22 e se o valor desta solução for melhor do que todos as soluções examinadas até agora, ela e seu custo são anotados nas linhas 24 e 25 como χ^* e f^*, respectivamente. χ^*, a melhor solução encontrada em todos as populações é retornada pelo algoritmo na linha 29.

4. Reinicialização do BRKGA

Como a maioria dos métodos estocásticos de busca, a variável aleatória *tempo à solução alvo* de um BRKGA tem uma distribuição empirica que aproxima uma exponencial com deslocamento. Por outro lado, a variável aleatória discreta *iterações à solução alvo* tem uma distribuição empírica geométrica com deslocamento.

Considere na Figura 2, a distribuição empírica do número de iterações do BRKGA para encontar uma solução ótima da instância `stn243` do problema de recobrimento de triplas de Steiner (Resende et al., 2012).

O gráfico iterações-ao-alvo (Aiex et al., 2007) é gerado excecutando um BRKGA 100 vezes, cada vez usando uma semente diferente para o gerador de números aleatórios e anotando o número de iterações que o algoritmo leva para encontrar uma solução alvo, nesta caso ótima. A figura mostra que 25% das execuções não necessitaram de mais de 55 iterações para achar uma solução ótima, 50% levaram no máximo 74 iterações e 75% no máximo 245. Entretanto, 10% precisaram de mais de 4597 iterações, 5% mais de 5532 iterações, 2% mais de 7061 e a rodada mais longa durou 9903 iterações. Este é um comportamento típico de uma variável aleatória com distribuição geométrica com deslocamento.

Seja I a variável aleatória número de iterações à uma solução alvo. Para a instância `stn243`, a Figura 2 sugere que $\Pr(I \geq 246) \approx 1/4$. Reinicializando após 246 iterações e supondo a independência das execuções, $\Pr(I \geq 492 \mid I \geq 246) \approx 1/4$. Portanto, $\Pr(I \geq 492) = \Pr(I \geq 246) \times \Pr(I \geq 492 \mid I \geq 246) \approx 1/4^2$. É fácil provar por indução que a probabilidade do algoritmo levar pelo menos k ciclos de 246 iterações é aproximada-

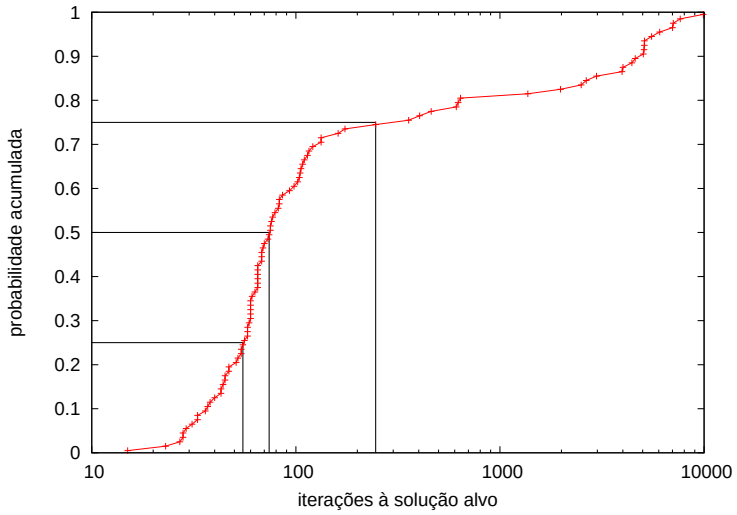

Figura 2. Distribuição de iterações a uma solução alvo do algoritmo BRKGA sem reinicialização.

mente $1/4^k$. Por exemplo, a probabilidade do algoritmo com reinicialização levar mais de 1230 iterações (cinco ciclos de 246 iterações entre reinicializações) é aproximadamente $1/4^5 = 1/1024 \approx 0.1\%$. Essa probabilidade é bem menor que os aproximadamente 20% de probabilidade que o algoritmo sem reinicialização leverá mais que 1230 iterações.

A análise acima utiliza uma estratégia de reinicialização que difere um pouco da estratégia proposta aqui para o BRKGA. Na nova estratégia, semelhante à estratégia de reinicialização do GRASP com religamento de caminhos proposta por Resende e Ribeiro (2011), ao invés de reinicializar a cada k iterações, a reinicialização é feita após k_r iterações sem que a melhor solução encontrada tenha sido melhorada.

A Figure 3 compara um BRKGA sem reinicialização com outro que reinicializa a cada 246 iterações sem melhoria do valor da melhor solução encontrada na instância `stn243` de recobrimento de triplas de Steiner. A figura claramente mostra que tanto o número médio de iterações ao ótimo como o seu desvio padrão são menores na versão com reinicialização do que na versão sem.

5. BRKGA com múltiplas populações

A descrição do BRKGA até esse ponto involveu apenas uma única população. Entretanto, é possível implementar um BRKGA com mais de uma população (Gonçalves e Resende, 2011b).

Suponha que o BRKGA tenha π populações $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_\pi$, cada uma com p vetores de chaves aleatórias. Neste caso, as π populações são inicialmente populadas, cada uma independentemente, com p vetores de chaves aleatórias na linha 4 do pseudo-código do Algoritmo 1 e o laço nas linhas 6 a 26 é aplicado a cada uma dessas π populações. As populações executam um intercâmbio de informação a cada k_p iterações do laço nas linhas 5 a 27 do Algoritmo 1. Nesta troca de informações, as k_m melhores soluções de cada população tomam os lugares das $(\pi - 1)k_m$ piores soluções de cada população.

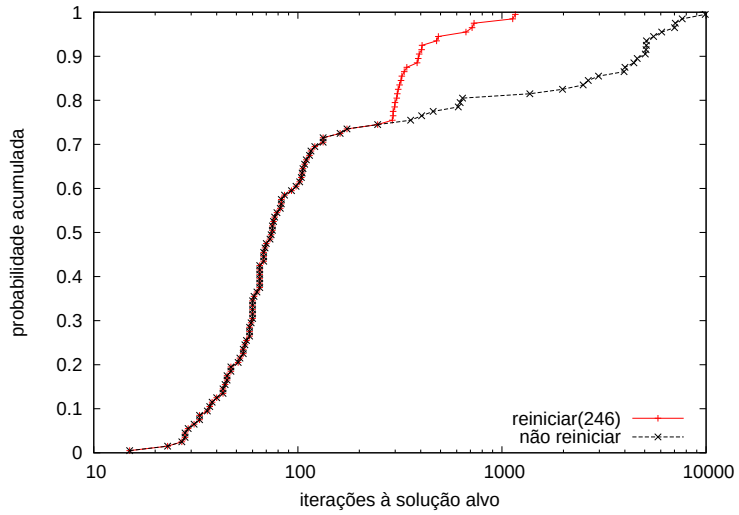

Figura 3. Distribuições de iterações a uma solução alvo (ótima) dos algoritmo BRKGA com reinicialização e sem reinicialização na instância `stn243` do problema de recobrimento de triplas de Steiner.

6. Especificação de um BRKGA

A especificação de um BRKGA requer especificação de como é representada uma solução, como é feito a decodificação e quais são os valores dos parâmetros do algoritmo.

Como toda solução é representada por um vetor de chaves aleatórias de n chaves, é necessário a especificação do valor de n.

O decodificador é um algoritmo determinístico que toma como entrada um vetor de n chaves aleatórias e produz como saída uma solução do problema de otimização assim como o custo desta solução. Um decodificador é, em geral, uma heurística. Se o decodificador possuir busca local, então será necessário também especificar como é feito o ajuste do vetor de chaves aleatórias de forma que o vetor corresponda à solução ótima local encontrada pela busca local quando a decodificação é feita sem o uso da busca local. Consulte Resende et al. (2012) para ver um exemplo de ajuste vetorial.

Vários parâmetros precisam ser especificados. A Tabela 6 lista esses parâmetros assim como valores que na prática se mostraram satisfatórios (Gonçalves e Resende , 2011a).

7. API para BRKGA

Para simplificar a implementação de BRKGAs, Toso e Resende (2012) propõem um *Application Programming Interface (API)*, ou biblioteca C++, para o BRKGA. O API é eficiente e fácil de usar. A biblioteca é portátil e trata automaticamente de vários aspectos do BRKGA, como gerenciamento das populações e a dinâmica evolutiva. O API é implementado em C++ e usa uma arquitetura orientada a objetos. Em sistemas com *OpenMP* (OpenMP, 2013) instalado, o API permite decodificações de chaves em paralelo. O usuário somente necessita implementar o decodificador e especificar os critérios de parada, reinicialização e intercâmbio de populações, assim como especificar os parâmetros do algoritmo.

O API é de acesso aberto e está disponível na página `http://www.research.att.com/~mgcr/src/brkgaAPI`.

Tabela 1. Parâmetros e valores recomendados.

Parâmetro	**Valor recomendado**
tamanho p da população	$p = \max\{3, \lfloor \kappa_p \times n \rfloor\}$, onde $\kappa_p > 0$
tamanho p_e da população elite	$p_e = \max\{1, \lfloor \kappa_a \times p \rfloor\}$, onde $\kappa_p \in \{0.10, 0.25\}$
tamanho p_m da população de mutantes	$p_m = \max\{1, \lfloor \kappa_m \times p \rfloor\}$, onde $\kappa_m \in \{0.05, 0.20\}$
probabilidade ρ_a de herança da chave elite	$\rho_a = \kappa_a > 1/2$, onde $\kappa_a \in \{0.55, 0.95\}$
iterações k_r sem melhoria para reinicialização	$k_r = \text{argmin}\{\Pr(k \text{ iterações à solução alvo}) \geq 0.75\}$
número π de populações paralelas	$\pi \in \{1, \ldots, 5\}$
frequência k_p de intercâmbio de populações	$k_p \in \{50, \ldots, 100\}$
número k_m de soluções trocadas	$k_m \in \{1, 2, 3\}$
critérios de parada (exemplos)	tempo de execução, número máximo de iterações, número máximo de reinicializações, obtenção de uma solução tão boa quanto o alvo.

8. Comentários finais

Este artigo reviu os algoritmos genéticos de chaves aleatórias viciadas (BRKGA, do inglês *biased random-key genetic algorithms*). Depois de introduzir o algoritmo de Bean (1994) no qual o BRKGA se baseia, o artigo aponta duas pequenas diferenças entre os dois algoritmos e mostra a melhoria no desempenho do BRKGA em relação ao algoritmo de Bean decorrente dessas pequenas alterações. Um modelo para a implementação de um BRKGA é descrito e aspectos como reinicialização e uso de múltiplas populações são discutidos. O artigo conclui mostrando como se especifica um BRKGA e apresenta um API em C++ para o BRKGA que permite fácil realização do algoritmo.

A metaheurística BRKGA tem sido aplicada a inúmeros problemas de otimização, dentre eles:

- *Telecomunicações:* Ericsson et al. (2002), Buriol et al. (2005), Noronha et al. (2011), Noronha et al. (2011), Reis et al. (2011), Ruiz et al. (2011), Pedrola et al. (2011), Goulart et al. (2011), Resende (2012), Morán-Mirabal et al. (2012), Pedrola et al. (2013), Duarte et al. (2013) e de Andrade et al. (2013a).
- *Transportes:* Buriol et al. (2010), Grasas et al. (2013) e Stefanello et al. (2013).
- *Escalonamento:* Gonçalves et al. (2005), Valente et al. (2006), Valente e Gonçalves (2008), Gonçalves et al. (2008), Mendes et al. (2009), Gonçalves et al. (2011), Tangpattanakul et al. (2012) e Gonçalves e Resende (2012a).
- *Empacotamento:* Gonçalves (2007), Gonçalves e Resende (2011b), Gonçalves e Resende (2012b) e Gonçalves e Resende (2013).
- *Recobrimento:* Breslau et al. (2011) e Resende et al. (2012).
- *Otimização em redes:* Andrade et al. (2006), Buriol et al. (2007), Fontes e Gonçalves (2007), Coco et al. (2012) e Fontes e Gonçalves (2012).
- *Engenharia de produção:* Gonçalves e Beirão (1999), Gonçalves e Almeida (2002), Gonçalves e Resende (2004), Moreira et al. (2012) e Morán-Mirabal et al. (2013b).
- *Ajuste automático de parâmetros em heurísticas:* Festa et al. (2010) e Morán-Mirabal et al. (2013a).
- *Leilões combinatórios:* de Andrade et al. (2013b).
- *Otimização global contínua:* Silva et al. (2012), Silva et al. (2013a) e Silva et al. (2013b).

Referências

Aiex, R. M., Resende, M. G. C. e Ribeiro, C. C. (2007). TTTPLOTS: A perl program to create time-to-target plots. *Optimization Letters*, 1:355–366.

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

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

Breslau, L., Diakonikolas, I., Duffield, N., Gu, Y., Hajiaghayi, M., Johnson, D. S., Karloff, H., Resende, M. G. C. e Sen, S. (2011). Disjoint-path facility location: Theory and practice. Em *Proceedings of the Thirteenth Workshop of Algorithm Engineering and Experiments (ALENEX11)*, páginas 68–74.

Buriol, L. S., Hirsch, M. J., Querido, T., Pardalos, P. M., Resende, M. G. C. e Ritt, M. (2010). A biased random-key genetic algorithm for road congestion minimization. *Optimization Letters*, 4:619–633.

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

Buriol, L. S., Resende, M. G. C. e Thorup, M. (2007). Survivable IP network design with OSPF routing. *Networks*, 49:51–64.

Coco, A. A., Noronha, T. F. e Santos, A. C. (2012). A biased random-key genetic algorithm for the robust shortest path problem. Em *Proceedings of Global Optimization Workshop (GO2012)*, páginas 53–56.

de Andrade, C. E., Miyazawa, F. K. e Resende, M. G. C. (2013a). Evolutionary algorithm for the k-interconnected multi-depot multi-traveling salesmen problem. Em *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*. ACM.

de Andrade, C. E., Miyazawa, F. K., Resende, M. G. C. e Toso, R. F. (2013b). Biased random-key genetic algorithms for the winner determination problem in combinatorial auctions. Relatório técnico, AT&T Labs Research, Florham Park, New Jersey.

Duarte, A., Martí, R., Resende, M. G. C. e Silva, R. (2013). Heuristics for the regenerator location problem. Relatório técnico, AT&T Labs Research, Florham Park, New Jersey.

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

Festa, P., Gonçalves, J. F., Resende, M. G. C. e Silva, R. M. A. (2010). Automatic tuning of GRASP with path-relinking heuristics with a biased random-key genetic algorithm. Em Festa, P., editora, *Experimental Algorithms*, volume 6049 do *Lecture Notes in Computer Science*, páginas 338–349. Springer.

Fontes, D. B. M. M. e Gonçalves, J. F. (2012). A multi-population hybrid biased random key genetic algorithm for hop-constrained trees in nonlinear cost flow networks. *Optimization Letters*. Publicado online 13 de junho.

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

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

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

Gonçalves, J. F. e Beirão, N. C. (1999). Um algoritmo genético baseado em chaves aleatórias para sequenciamento de operações. *Revista Associação Portuguesa de Desenvolvimento e Investigação Operacional*, 19:123–137.

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

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

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

Gonçalves, J. F. e Resende, M. G. C. (2012a). A biased random-key genetic algorithm for job-shop scheduling. Relatório técnico, AT&T Labs Research, Florham Park, New Jersey.

Gonçalves, J. F. e Resende, M. G. C. (2012b). A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research*, 29:179–190.

Gonçalves, J. F. e Resende, M. G. C. (2013). A biased random-key genetic algorithm for a 2D and 3D bin packing problem. *International J. of Production Economics*. Publicado online 18 de abril.

Gonçalves, J. F., Resende, M. G. C. e 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.

Gonçalves, J. F., Resende, M. G. C. e Toso, R. F. (2012). Biased and unibiased random key genetic algorithms: An experimental analysis. Relatório técnico, AT&T Labs Research, Florham Park, New Jersey.

Goulart, N., de Souza, S. R., Dias, L. G. S. e Noronha, T. F. (2011). Biased random-key genetic algorithm for fiber installation in optical network optimization. Em *IEEE Congress on Evolutionary Computation (CEC 2011)*, páginas 2267–2271. IEEE.

Grasas, A., Ramalhinho, H., Pessoa, L. S., Resende, M. G. C., Caballé, I. e Barba, N. (2013). On the improvement of blood sample collection at clinical laboratories. Relatório técnico, AT&T Labs Research, Florham Park, New Jersey.

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

Morán-Mirabal, L. F., González-Velarde, J. L. e Resende, M. G. C. (2013a). Automatic tuning of GRASP with evolutionary path-relinking. Em *Proceedings of Hybrid Metaheuristics 2013 (HM 2013)*. Springer, Ischia, Italy. No prelo, *Lecture Notes in Computer Science*.

Morán-Mirabal, L. F., González-Velarde, J. L. e Resende, M. G. C. (2013b). Randomized heuristics for the family traveling salesperson problem. Relatório técnico, AT&T Labs Research, Florham Park, New Jersey.

Morán-Mirabal, L. F., González-Velarde, J. L., Resende, M. G. C. e Silva, R. M. A. (2012). Randomized heuristics for handover minimization in mobility networks. Relatório técnico, AT&T Labs Research, Florham Park, New Jersey.

Moreira, M. C. O., Ritt, M., Costa, A. M. e Chaves, A. A. (2012). Simple heuristics for the assembly line worker assignment and balancing problem. *J. of heuristics*, 18:505–524.

Noronha, T. F., Resende, M. G. C. e Ribeiro, C. C. (2010). A biased random-key genetic algorithm for routing and wavelength assignment. *J. of Global Optimization*, 50:503–518.

OpenMP (2013). `http://openmp.org/wp/`, acessado em 11 de maio de 2013.

Pedrola, O., Careglio, D., Klinkowski, M., Velasco, L., Bergman, K. e Solé-Pareta, J. (2013). Metaheuristic hybridizations for the regenerator placement and dimensioning problem in sub-wavelength switching optical networks. *European J. of Operational Research*, 224:614–624.

Pedrola, O., Ruiz, M., Velasco, L., Careglio, D., González de Dios, O. e Comellas, J. (2011). A GRASP with path-relinking heuristic for the survivable IP/MPLS-over-

WSON multi-layer network optimization problem. *Computers & Operations Research*. Publicado online 7 de novembro de.

Reis, R., Ritt, M., Buriol, L. S. e Resende, M. G. C. (2011). A biased random-key genetic algorithm for OSPF and DEFT routing to minimize network congestion. *International Transactions in Operational Research*, 18:401–423.

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

Resende, M. G. C. e Ribeiro, C. C. (2011). Restart strategies for GRASP with path-relinking heuristics. *Optimization Letters*, 5:467–478.

Resende, M. G. C., Toso, R. F., Gonçalves, J. F. e Silva, R. M. A. (2012). A biased random-key genetic algorithm for the Steiner triple covering problem. *Optimization Letters*, 6:605–619.

Ruiz, M., Pedrola, O., Velasco, L., Careglio, D., Fernández-Palacios, J. e Junyent, G. (2011). Survivable IP/MPLS-over-WSON multilayer network optimization. *J. of Optical Communications and Networking*, 3:629–640.

Silva, R. M. A., Resende, M. G. C. e Pardalos, P. (2013a). A Python/C++ library for bound-constrained global optimization using biased random-key genetic algorithm. Relatório técnico, AT&T Labs Research, Florham Park, New Jersey.

Silva, R. M. A., Resende, M. G. C. e Pardalos, P. M. (2013b). Finding multiple roots of box-constrained system of nonlinear equations with a biased random-key genetic algorithm. No prelo, *J. of Global Optimization*.

Silva, R. M. A., Resende, M. G. C., Pardalos, P. M., e Gonçalves, J. F. (2012). Biased random-key genetic algorithm for bound-constrained global optimization. Em *Proceedings of Global Optimization Workshop (GO2012)*, páginas 133–136.

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

Stefanello, F., Buriol, L. S., Hirsch, M. J., Pardalos, P. M., Querido, T., Resende, M. e Ritt, M. (2013). On the minimization of traffic congestion in road networks with tolls. Relatório técnico, AT&T Labs Research, Florham Park, New Jersey.

Tangpattanakul, P., Jozefowiez, N. e Lopez, P. (2012). Multi-objective optimization for selecting and scheduling observations by agile earth observing satellites. Em *Parallel Problem Solving from Nature – PPSN XII*, volume 7492 do *Lecture Notes in Computer Science*, páginas 112–121. Springer.

Toso, R. F. e Resende, M. G. C. (2012). A C++ application programming interface for biased random-key genetic algorithms. Relatório técnico, AT&T Labs Research, Florham Park, New Jersey.

Valente, J. M. S. e Gonçalves, J. F. (2008). A genetic algorithm approach for the single machine scheduling problem with linear earliness and quadratic tardiness penalties. *Computers & Operations Research*, 35:3696–3713.

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