Algoritmo genético de chaves aleatórias viciadas para problemas de otimização global com restrições de caixa sujeitas a restrições não-lineares

Ricardo Martins de Abreu Silva
Centro de Informática – Universidade Federal de Pernambuco
Caixa Postal 50.740-560 – Recife – PE – Brazil
rmas@cin.ufpe.br

Mauricio G. C. Resende
Algorithms and Optimization Research Department
AT&T Labs Research, Florham Park, NJ 07932 USA
mgcr@research.att.com

RESUMO

A área de otimização global busca um mínimo ou máximo de uma função multimodal sobre um domínio discreto ou contínuo. Neste artigo, um algoritmo genético para busca de soluções aproximadas para problemas de otimização global contínua sujeitos à restrições não-lineares é proposto. Resultados experimentais ilustram sua eficiência em algumas funções do *benchmark* CEC2006 (Liang et al., 2006).

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

ABSTRACT

Global optimization seeks a minimum or maximum of a multimodal function over a discrete or continuous domain. In this paper, we propose a biased random key genetic algorithm for finding approximate solutions for bound-constrained continuous global optimization problems subject to nonlinear constraints. Experimental results illustrate its effectiveness on some functions from CEC2006 benchmark (Liang et al., 2006).

KEYWORDS. Genetic algorithm, Random keys, Global optimization, Metaheuristics.

1. Introdução

A otimização global contínua, em sua versão de minimização, tem por objetivo encontrar uma solução $x^* \in S \subseteq R^n$ tal que $f(x^*) \leq f(x),\ \forall\ x \in S$, onde S é alguma região de R^n e a função objetivo f é $f : S \to R$.

Neste artigo, consideramos o domínio S como sendo a intersecção entre um conjunto de restrições não lineares e um hiper-retângulo $\Omega = \{x = (x_1, \ldots, x_n) \in R^n : \ell \leq x \leq u\}$, onde $\ell \in R^n$ e $u \in R^n$ tal que $u_i \geq l_i$, para $i = 1, \ldots, n$, com o intuito de apresentar um algoritmo genético de chaves aleatórias viciadas para problemas de otimização global contínua com restrições de caixa sujeita a restrições não-lineares:

$$min f(x),\ \ x = (x_1, x_2, \ldots, x_n) \tag{1}$$

sujeito a:

$$g_i(x) \leq 0,\ \ i = 1, \ldots, q \tag{2}$$

$$h_j(x) = 0,\ \ j = q+1, \ldots, m \tag{3}$$

$$l_i \leq x_i \leq u_i,\ \ i = 1, \ldots, n \tag{4}$$

Dado que as restrições (2) podem ser escritas como equações com a introdução de q variáveis auxiliares $x_{n+1}, \ldots, x_{n+q}$:

$$g_i(x_1, \ldots, x_n) + x_{n+i} = 0,\ \ i = 1, \ldots, q, \tag{5}$$

com $x_k \geq 0$, $k = n+1, \ldots, n+q$, o problema original pode ser reduzido à minimização da função $F(x_1, \ldots, x_{n+q})$ definida como segue:

$$[f(x_1, \ldots, x_n) - f^*]^2 + \sum_{i=1}^{q} [g_i(x_1, \ldots, x_n) + x_{n+i}]^2 + \sum_{j=q+1}^{m} [h_j(x_1, \ldots, x_n)]^2 \tag{6}$$

sujeito a:

$$l_i \leq x_i \leq u_i,\ \ i = 1, \ldots, n \tag{7}$$

$$x_k \geq 0,\ \ k = n+1, \ldots, n+q, \tag{8}$$

onde f^* é o valor ótimo conhecido do problema (1-4), ou o melhor valor conhecido da literatura. No caso de desconhecermos o valor da solução global, se possível, nós podemos considerar f^* como um limite inferior (lower bound) apropriado. Desde que $F(x_1, \ldots, x_{n+q}) \geq 0$ para todo $l_i \leq x_i \leq u_i$, $i = 1, \ldots, n$ e $x_k \geq 0, k = n+1, \ldots, n+q$, pode-se observar que $F(x_1, \ldots, x_{n+q}) = 0 \iff f(x_1, \ldots, x_n) = f^*$; $g_i(x_1, \ldots, x_n) = x_{n+i}$, $i = 1, \ldots, q$; e $h_j(x_1, \ldots, x_n) = 0$, $j = q+1, \ldots, m$. Sendo assim, $\exists\ z^* = (x_1, \ldots, x_{n+q})^*$ viável $\ni F(z^*) = 0 \Longrightarrow z^*$ é um minimizador global do problema (1-4).

Este trabalho encontra-se organizado como segue. A heurística AGCAV para problemas de otimização global é descrita nas seções 2 e 3. Na seção 4, os resultados experimentais ilustram a eficácia e eficiência da heurística AGCAV para otimização global com restrições de caixa sujeitas a restrições não lineares. Por fim, as conclusões encontram-se descritas na secção 5.

2. Algoritmo Genético com Chaves Aleatórias Viciadas

Algoritmos Genéticos com Chaves Aleatórias (AGCA) foram introduzidos primeiramente por Bean (1994) para solucionar problemas de otimização combinatória envolvendo sequenciamento. Em um AGCA, cromossomos são representados como vetores de números reais gerados aleatoriamente no intervalo $[0, 1]$. Um algoritmo determinístico, denominado *decodificador*, toma como entrada um vetor solução e associa a ele uma solução do problema de otimização combinatória, cujo valor da função objetivo pode ser calculado.

Um AGCA evolui uma população de vetores de chaves aleatórias mediante iterações denominadas *gerações*. A população inicial é composta por p vetores de chaves aleatórias. Cada componente do vetor solução é gerado aleatoriamente de forma independente a partir do intervalo real $[0, 1]$. Após o cálculo realizado pelo decodificador da função objetivo associada a cada indivíduo na geração k, a população é particionada em dois grupos de indivíduos: um pequeno grupo de p_e indivíduos *elite*, ou seja, aqueles que apresentaram os melhores valores para a função objetivo; e o conjunto restante dos $p - p_e$ indivíduos *não-elite*. Para evoluir a população, uma nova geração de indivíduos deve ser produzida. Todos os indivíduos elite da população na geração k são copiados sem modificações para a população da geração $k + 1$. O AGCA implementa a operação de mutação via introdução de *mutantes* dentro da população. Um mutante é simplesmente um vetor de chaves aleatórias gerado da mesma maneira que um elemento da população inicial é gerado. A cada geração um pequeno número (p_m) de mutantes é introduzido na população. Com p_e indivíduos elite e p_m mutantes na população $k+1$, $p-p_e-p_m$ indivíduos adicionais devem ser produzidos para completar os p indivíduos que comporão a nova população. Isto é feito através da produção de uma prole com $p - p_e - p_m$ indivíduos mediante operações de cruzamento.

A figura 1 ilustra a dinâmica da evolução. No lado esquerdo da figura está a população atual. Após a ordenação de todos indivíduos segundo os valores da função objetivo, os melhores encontram-se na partição denominada ELITE, enquanto o restante na partição denominada NON-ELITE. Os vetores de chaves aleatórias elite são copiados sem mudança para a partição denominada TOP na próxima população (ver lado direito da figura). Uma quantidade de indivíduos mutantes são aleatoriamente gerados e em seguida inseridos na partição denominada BOT da nova população. O restante da população da próxima geração é completada através de operações de cruzamento. Em um AGCA, Bean (1994) seleciona os pais aleatoriamente a partir da população inteira. Um algoritmo genético de chaves aleatórias viciadas (AGCAV) – *biased random-key genetic algorithm* (BRKGA) (Gonçalves and Almeida, 2002; Ericsson et al., 2002; Gonçalves and Resende, 2004) – difere do AGCA no modo como os pais são selecionados para cruzamento. Em um AGCAV, cada elemento é gerado combinando um elemento selecionado aleatoriamente da partição denominada ELITE na população atual e outro da partição denominada NON-ELITE. Em alguns casos, o segundo pai é selecionado considerando-se a população inteira. Repetições na seleção de um companheiro são permitidos e portanto um indivíduo pode produzir mais que uma prole. Como $p_e < p - p_e$ é desejado, a probabilidade de um indivíduo elite ser selecionado para cruzamento é maior do que de um indivíduo não elite e portanto o indivíduo elite tem uma maior probabilidade de passar suas características para gerações futuras.

Outro fator que contribui para este fim é o cruzamento uniforme parametrizado –

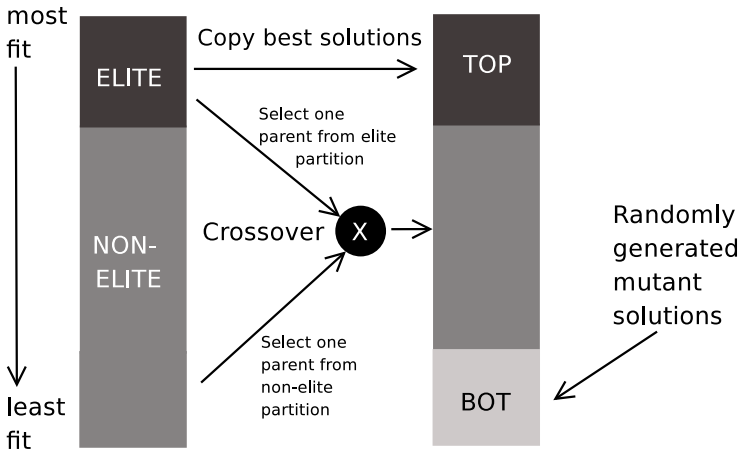

Figura 1. Transição da geração k para geração $k+1$ em um AGCAV.

parameterized uniform crossover (Spears and DeJong, 1991), o mecanismo usado para implementar os cruzamentos em AGCAV. Seja $\rho_e > 0.5$ a probabilidade de uma prole herdar um componente de vetor de seu pai elite. Seja n o número de componentes no vetor solução correspondente a um indivíduo. Para $i = 1, \ldots, n$, o i-ésimo componente $c(i)$ da prole c recebe o valor do i-ésimo componente $e(i)$ do pai elite e com probabilidade ρ_e e o valor do i-ésimo componente $\bar{e}(i)$ do pai não elite $\bar{e}$ com probabilidade $1 - \rho_e$.

A figura 2 ilustra o processo de cruzamento para dois vetores de chaves aleatórias com quatro componentes cada. O cromossomo 1 representa o indivíduo elite e o cromossomo 2 o não elite. Neste exemplo o valor de $\rho_e = 0.7$, ou seja, a prole herda o componente do pai elite com probabilidade 0.7 e do outro pai com probabilidade 0.3. O número real aleatoriamente gerado no intervalo $[0, 1]$ simula o lançamento de uma moeda viesada. Se o resultado é menor ou igual a 0.7, então o filho herda o componente do pai elite. Caso contrário, ele herda o componente do outro pai. Quando a próxima população fica completa, ou seja, quando a mesma possui p indivíduos, os valores da função objetivo são computados para todos os novos vetores de chaves aleatórias e a população é particionada em indivíduos elite e não elite a fim de iniciar uma nova geração.

As heurísticas AGCAV são baseadas no *framework* meta-heurístico de propósito

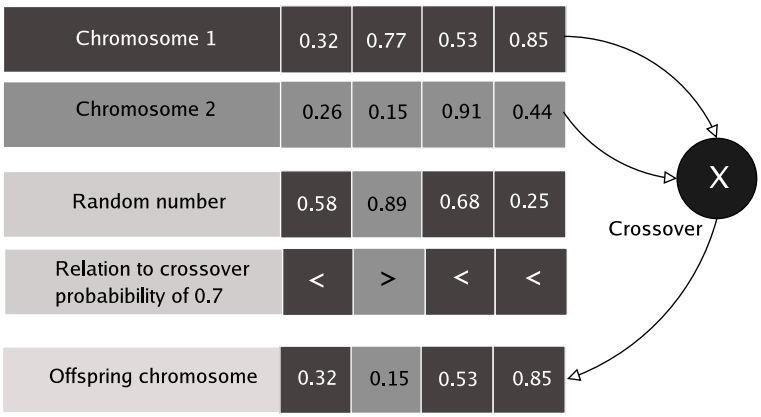

Figura 2. Cruzamento uniforme parametrizado: acasalamento em AGCAV.

geral. Neste framework, ilustrado na figura 3, há uma clara divisão entre a porção *independente do problema* e a parte dependente do algoritmo. A porção independente não tem nenhum conhecimento do problema a ser solucionado. A única conexão com o problema de otimização é a porção do algoritmo dependente do problema, onde o decodificador produz soluções a partir dos vetores de chaves aleatórias e computa o *fitness* dessas soluções. Portanto, para especificar uma heurística AGCAV precisamos apenas definir sua representação cromossômica e o decodificador.

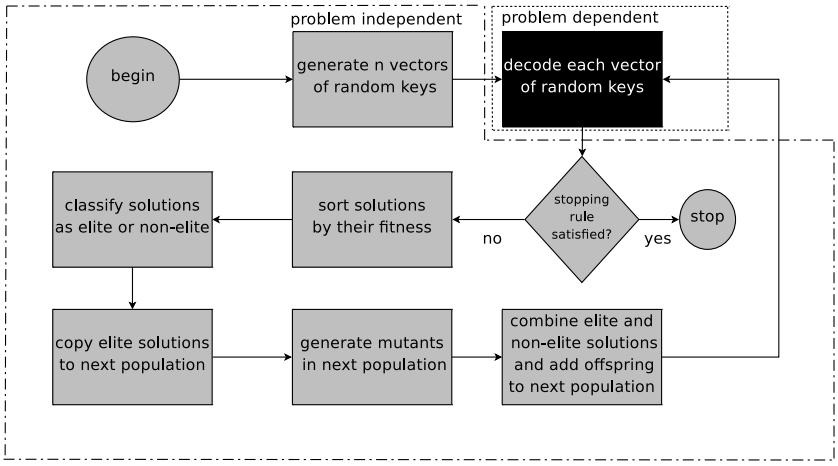

Figura 3. Fluxograma do AGCAV.

Para descrever um AGCAV para problemas de otimização global com restrições de caixa sujeitos a restrições não lineares, precisamos apenas mostrar como as soluções são codificadas como vetores de chaves aleatórias e como esses vetores são decodificados em soluções viáveis do problema:

- *Codificando uma solução em um vetor de chaves aleatórias*. Uma solução é codificada como um vetor $\chi = (\chi_1, ..., \chi_n)$ de tamanho n, onde χ_i é um número aleatório no intervalo $[0, 1]$, para $i = 1, \ldots, n$. O i-ésimo componente de χ corresponde à i-ésima dimensão do hiper-retângulo Ω.
- *Decodificando uma solução a partir de um vetor de chaves aleatórias*. Um decodificador toma como entrada um vetor de chaves aleatórias χ e retorna uma solução $x \in \Omega$ com $x_i = l_i + \chi_i \cdot (u_i - l_i)$, para $i = 1, \ldots, n$. Após obter a solução $x \in \Omega$, nós procedemos tentando melhorá-la usando a busca local descrita na próxima seção. As soluções produzidas pela busca local geralmente não estão de acordo com os genes inicialmente fornecidos pelos vetores de chaves aleatórias ao decodificador. Nesses casos, com o intuito de refletir as mudanças realizadas pela fase de busca local do decodificador, a heurística substitui o cromossomo inicial com o retornado pelo procedimento da busca local, onde $\chi_i = (x_i - l_i)/(u_i - l_i)$, para $i = 1, \ldots, n$. Durante todo processo de decodificação, o *fitness* das soluções são calculados pela função objetivo $f : S \to R$ do problema de otimização global.

3. Procedimento de melhoria local

AGCAV não faz uso de derivadas. Embora derivadas possam ser facilmente computadas para diversas funções, elas não estão sempre disponíveis ou nem sempre são eficientemente computáveis para todas as funções. A fase de melhoria local (cujo pseudocódigo encontra-se descrito na figura 4) pode ser vista como aproximando a função de

gradiente da função objetivo $f(\cdot)$. A partir de um dado ponto de entrada $x \in R^n$, o algoritmo de busca local gera uma vizinhança, e determina em quais pontos da vizinhança, caso haja algum, a função objetivo melhora. Se um ponto de melhoria é conhecido, o mesmo torna-se o ponto atual e a busca local continua.

Seja $\bar{x} \in R^n$ a solução atual e h o parâmetro atual de discretização da malha. Defina $S_h(\bar{x}) = \{x \in \Omega \mid \ell \leq x \leq u,\ x = \bar{x} + \tau \cdot h,\ \tau \in Z^n\}$ para ser o conjunto de pontos em Ω que são passos inteiros (de tamanho h) distantes de $\bar{x}$. Seja $B_h(\bar{x}) = \{x \in \Omega \mid x = \bar{x} + h \cdot (x' - \bar{x})/\|x' - \bar{x}\|,\ x' \in S_h(\bar{x}) \setminus \{\bar{x}\}\}$ a projeção dos pontos em $S_h(\bar{x}) \setminus \{\bar{x}\}$ sobre a hiper-esfera centrada em $\bar{x}$ de raio h. A vizinhança-h (*h-neighborhood*) do ponto $\bar{x}$ é definida como o conjunto de pontos em $B_h(\bar{x})$.

```
procedure LocalImprovement(x, f(·), h_s, h_e, ℓ, u, MaxPointsToExamine)
1    x* ← x;
2    f* ← f(x);
3    h ← h_s;
4    Impr ← false;
5    while h ≥ h_e do
6        NumPointsExamined ← 0;
7        while NumPointsExamined ≤ MaxPointsToExamine do
8            x ← RandomlySelectElement(B_h(x*));
9            if ℓ ≤ x ≤ u and f(x) < f* then
10               x* ← x;
11               f* ← f(x);
12               NumPointsExamined ← 0;
13               Impr ← true;
14           end if
15           NumPointsExamined ← NumPointsExamined + 1;
16       end while
17       if Impr = true then
18           return x*;
19       else
20           h ← h/2;
21       end if
22   end while
23   return x*;
end LocalImprovement;
```

Figura 4. Pseudo-código para fase de melhoria local.

O procedimento toma como entrada uma solução inicial $x \in \Omega \subseteq R^n$, a função objetivo $f(\cdot)$, vetores de limites inferiores e superiores ℓ e u, assim como os parâmetros h_s e h_e, as densidades iniciais e finais de discretização da malha, respectivamente. O número máximo de pontos $\texttt{MaxPointsToExamine} \leq \prod_{i=1}^{n} \lceil (u_i - \ell_i)/h \rceil$ em $B_h(x^*)$ a serem examinados também será considerado como um parâmetro de entrada. Caso todos estes pontos sejam examinados e nenhum ponto de melhoria for encontrado, a solução atual x^* é considerada um mínimo local-h (*h-local minimum*).

A melhor solução de melhoria atual x^* é inicializada para x na linha 1. Na linha 2, o valor da função objetivo f^* da melhor solução encontrada é inicializada para $f(x)$. Depois, o parâmetro h, o qual controla a densidade da discretização do espaço de busca, é inicializado para h_s na linha 3, e na linha 4 a variável `Impr` é definida como **false**. A partir do ponto x^*, no loop das linhas 7–16, o algoritmo seleciona aleatoriamente pontos em $B_h(x^*)$ (linha 8), um a cada vez. Na linha 9, se o ponto atual x selecionado a partir de $B_h(x^*)$ for viável e melhor do que x^*, então x^* é atualizado para x (linha 10), f^* é atualizado para $f(x)$ (linha 11), `NumPointsExamined` para zero (linha 12), `Impr` para **true** (line 13), e o loop entre as linhas 7–16 é reinicializado tendo x^* como solução inicial. Na linha 17,

se a variável `Impr` continua falsa, então na linha 20 a densidade da malha é aumentada, usando $h/2$, e o loop nas linhas 7–16 é reinicializado se $h \geq h_e$. A busca local termina se a solução mínima local-h x^* for encontrada. Neste momento, x^* é retornado pela busca local nas linhas 18 ou 23.

4. Resultados experimentais

Os experimentos foram realizados mediante um processador quad core Intel Core i7 (1.60 GHz) com Turbo Boost up to (2.80 GHz) e 16 Gb de memória, rodando Ubuntu 10.04 LTS. A implementação de AGCAV foi feita com a linguagem de programação C++ e compilada com gcc versão 4.4.3. O algoritmo usado para geração de números aleatórios é uma implementação do algoritmo Mersenne Twister descrito em Matsumoto and Nishimura (1998). Por fim, para a realização dos experimentos, os seguintes problemas teste foram utilizados: `g01` (Floudas and Pardalos, 1990), `g02` (Koziel and Michalewicz, 1999), `g03` (Michalewicz et al., 1996), `g04` (Himmelblau, 1972) e `g05` (Hock and Schittkowski, 1981), devido às suas propriedades heterogêneas descritas na Tabela 1.

Tabela 1. Para cada um dos cinco problemas (`g01`-`g05`), oriundos do *benchmark* CEC2006 (Liang et al., 2006), temos: n como sendo o número de variáveis de decisão; $\rho = |F|/|S|$ como a razão estimada entre a região viável e o espaço de busca; LI, NI, LE e NE como sendo respectivamente o no. de inequações lineares, inequações não-lineares, equações lineares e equações não lineares; e *a* como sendo o número de restrições ativas em x.

Prob.	n	type	$f(x^*)$	ρ	LI	NI	LE	NE	a
g01	13	quadrático	-15.0000000000	0.0111	9	0	0	0	6
g02	20	não-linear	-0.8036191042	99.9971	0	2	0	0	1
g03	10	polinomial	-1.0005001000	0.0000	0	0	0	1	1
g04	5	quadrático	-30665.5386717834	52.1230	0	6	0	0	2
g05	5	cúbico	5126.4967140071	0.0000	2	0	0	3	3

Em todos os cinco problemas, executamos AGCAV 200 vezes (cada vez com uma semente distinta, variando de 270001 a 270200, para o gerador de números aleatórios) com $p = 100$, $p_e = 0.2p$, $p_m = 0.1p$, $\rho_e = 0.7$, $h_s = 0.05$, $h_e = 0.00001$, $rho_{lo} = 0.15$, `MaxPointsToExamine` $= 1000$, e $\epsilon = 0.00001$. Durante uma execução, o *gap* de otimalidade é definido da seguinte forma: $GAP = F(x_1, \ldots, x_{n+q}) - F(z^*)$, onde $(x_1, \ldots, x_{n+q})$ é a melhor solução até então encontrada pela heurística e $F(z^*) = 0$. Nós então dizemos que a heurística resolveu o problema se $GAP \leq \epsilon$ com $\epsilon = 0.0000001$. Portanto, uma solução para o problema é encontrada quando o valor da respectiva função objetivo é menor do que 10^{-7}. Em cada problema, a heurística conseguiu encontrar a solução ótima (ou a melhor solução conhecida) em todas as 200 execuções.

A tabela 2 fornece os tempos mínimo, 1o. quartil, mediana, média, 3o. quartil e máximo (em segundos), assim como o desvio padrão dos tempos de execução gastos para encontrar a solução ótima (ou a melhor solução conhecida) em cada problema. De acordo com a tabela 2, em média o AGCAV encontra a solução do problema em menos de 21.62, 120.20, 0.60, 134.20, 4.00 segundos para os problemas `g01`, `g02`, `g03`, `g04` e `g05`, respectivamente. No pior caso, o AGCAV encontra as soluções em menos de 253.60, $1,076.00$, 0.88, 913.40 e 15.15 segundos, respectivamente.

Os tempos gastos para encontrar a solução ótima (ou a melhor solução conhecida) em cada problema foram calculados a fim de apresentarmos também os gráficos das distribuições dos tempos de execução (também conhecidos como gráficos tempo-para-o-alvo

Tabela 2. Tempos (em segundos) para AGCAV alcançar as soluções alvo dos problemas g01-g05.

prob.	min	1o. Qu.	mediana	média	3o. Qu.	max	desv. padrão
g01	2.76	13.24	18.44	21.62	25.36	253.60	20.324
g02	1.12	54.53	78.24	120.20	123.90	1076.00	145.316
g03	0.33	0.54	0.61	0.60	0.67	0.88	0.095
g04	5.37	51.38	83.25	134.20	125.00	913.40	157.995
g05	0.44	1.82	3.28	4.00	5.60	15.15	2.784

– *time-to-target plots* (Aiex et al., 2002, 2006)) ilustrados na figura 5. Como pode ser visto em mais detalhes a partir da figura 5(b), uma versão ampliada da figura 5(a), 95% das execuções se encerram em menos de 46.00, 377.00, 0.77, 495.00 e 9.40 segundos para os problemas g01, g02, g03, g04 e g05, respectivamente.

Também comparamos a heurística AGCAV com o GRASP contínuo – *continuous GRASP (C-GRASP) algorithm* (Hirsch et al., 2007, 2010; Silva et al., 2012) sobre o mesmo conjunto de problemas teste: g01–g05 (Facó et al., 2013). A implementação do C-GRASP utilizada neste trabalho encontra-se descrita em Silva et al. (2012). Em todos os cinco problemas, executamos o C-GRASP 200 vezes (cada vez com uma semente distinta, variando de 270001 a 270200, para o gerador de números aleatórios). Em cada problema, a heurística C-GRASP conseguiu encontrar a solução ótima (ou a melhor solução conhecida) em todas as 200 execuções. A tabela 3 fornece os tempos mínimo, 1o. quartil, mediana, média, 3o. quartil e máximo (em segundos), assim como o desvio padrão dos tempos de execução gastos para encontrar a solução ótima (ou a melhor solução conhecida) em cada problema. De acordo com a tabela 3, em média o C-GRASP encontra a solução do problema em menos de 12.08, $1,009.00$, 0.10, 604.50, 3.43 segundos para os problemas g01, g02, g03, g04 e g05, respectivamente. No pior caso, o C-GRASP encontra as soluções em menos de 19.32, $1,881.00$, 0.27, $5,472.00$ e 5.04 segundos, respectivamente.

Tabela 3. Tempos (em segundos) para o C-GRASP alcançar as soluções alvo dos problemas g01-g05.

prob.	min	1o. Qu.	mediana	média	3o. Qu.	max	desv. padrão
g01	8.72	9.91	11.22	12.08	13.20	19.32	2.793
g02	3.00	657.00	1055.00	1009.00	1321.00	1881.00	429.347
g03	0.02	0.06	0.09	0.10	0.13	0.27	0.060
g04	3.37	7.55	98.33	604.50	561.40	5472.00	1369.259
g05	1.88	2.49	3.46	3.43	4.45	5.04	1.057

Embora nas instâncias g01, g03 e g05, o desempenho da heurística C-GRASP foi um pouco melhor do que o desemepneho do algoritmo AGCAV; nos dois problemas mais difíceis, g02 e g04, o mínimo, média, mediana (50%) e 75% (3o. quartil) dos tempos de execução do AGCAV foram bem menores do aqueles descritos em (Facó et al., 2013) sobre a heurística C-GRASP.

5. Conclusões

Neste artigo, apresentamos a heurística AGCAV para busca de soluções aproximadas de problemas de otimização global sujeitos a restrições não lineares e de caixa. O enfoque é ilustrado via sua aplicação em cinco problemas oriundos do *benchmark* CEC2006 (Liang et al., 2006). Os resultados promissores ilustram o potencial do método.

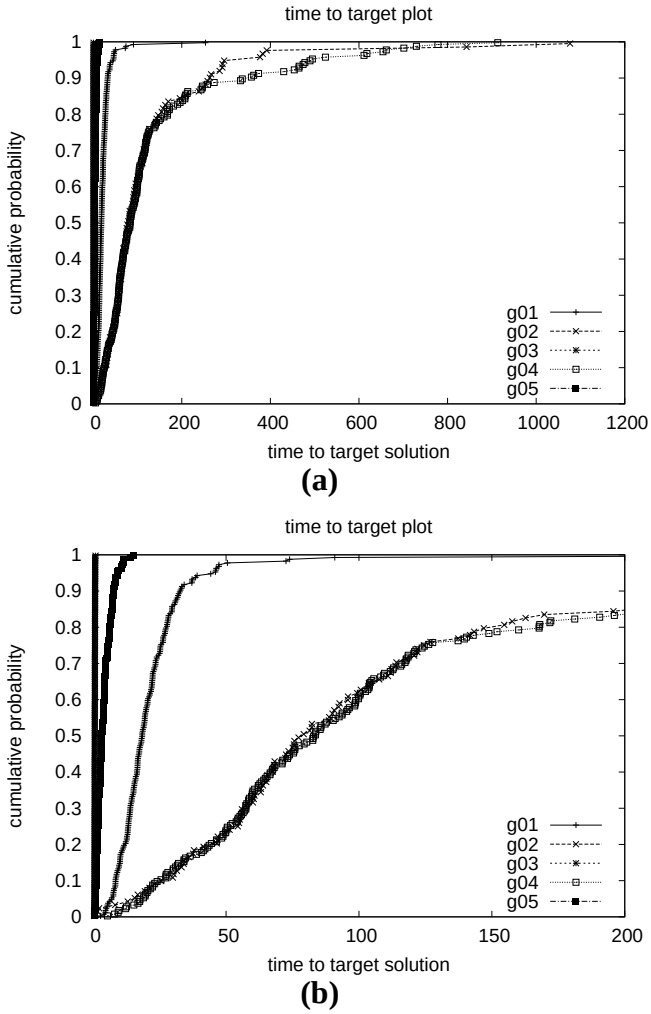

Figura 5. Gráficos da distribuição de probabilidade acumulada (gráficos tempo-para-o-alvo) dos tempos de execução do AGCAV gastos para encontrar a solução ótima (ou a melhor solução conhecida) dos problemas `g01`-`g05`**.**

Agradecimentos

A pesquisa de Ricardo M. A. Silva foi parcialmente suportada pelo CNPq, FAPEMIG, CAPES, PROPESQ e FACEPE.

Referências

Aiex, R., Resende, M., and Ribeiro, C. (2002). Probability distribution of solution time in GRASP: An experimental investigation. *J. of Heuristics*, 8:343–373.

Aiex, R., Resende, M., and Ribeiro, C. (2006). TTTPLOTS: A perl program to create time-to-target plots. *Optimization Letters*.

Bean, J. (1994). Genetic Algorithms and Random Keys for Sequencing and Optimization. *ORSA J. on Computing*, 6:154–160.

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

Facó, J. o. L., Resende, M. G., and Silva, R. M. (2013). Continuous grasp for nonlinearly-constrained global optimization. In *Congreso Latino-Iberoamericano de Investigación*

Operativa / Simpósio Brasileiro de Pesquisa Operacional, CLAIO/SBPO 2013. Sociedade Brasileira de Pesquisa Operacional.

Floudas, C. and Pardalos, P. (1990). *A collection of test problems for constrained global optimization algorithms*. Springer-Verlag New York, Inc., New York, NY, USA.

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

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

Himmelblau, D. (1972). *Applied nonlinear programming*. McGraw-Hill.

Hirsch, M., Meneses, C., Pardalos, P., and Resende, M. (2007). Global optimization by continuous GRASP. *Optimization Letters*, 1:201–212.

Hirsch, M., Pardalos, P., and Resende, M. (2010). Speeding up continuous grasp. *European Journal of Operational Research*, 205(3):507 – 521.

Hock, W. and Schittkowski, K. (1981). *Test Examples for Nonlinear Programming Codes*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Koziel, S. and Michalewicz, Z. (1999). Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7:pp.

Liang, J. J., Runarsson, T. P., Montes, E. M., Clerc, M., Suganthan, P. N., Coello, C. A., and K., D. (2006). Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization. Technical report.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30.

Michalewicz, Z., Nazhiyath, G., and Michalewicz, M. (1996). A note on usefulness of geometrical crossover for numerical optimization problems. In *Evolutionary Programming*, pages 305–312.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Silva, R., Resende, M., Pardalos, P., and Hirsch, M. (2012). A python/c library for bound-constrained global optimization with continuous grasp. *to appear in Optimization Letters*.

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