UM NOVO MÉTODO HÍBRIDO APLICADO À SOLUÇÃO DE SISTEMAS NÃO-LINEARES COM RAÍZES MÚLTIPLAS

Maurício Rodrigues Silva
Universidade Federal de Ouro Preto – Departamento Engenharia de Produção
UFOP - DEENP
João Monlevade, MG – Brasil.
dscmauricio@gmail.com

RESUMO

O presente trabalho apresenta um algoritmo híbrido que tem como objetivo a solução de problemas de otimização, originados de sistemas de equações não lineares com múltiplas soluções. O modelo é composto de dois algoritmos concatenados, onde o primeiro é estocástico, e o segundo é determinístico, executados consecutivas vezes por meio de uma estrutura de repetição, projetada de forma a forçar a busca de todas as soluções possíveis dentro dos limites do intervalo de busca. Por fim, os resultados são pós-otimizados, resultando em um grupo de diversas soluções, equivalentes às soluções alternativas do sistema não linear que representa o problema.

PALAVRAS CHAVE: Otimização, algoritmos híbridos, pesquisa operacional. metaheuristicas.

ABSTRACT

This paper presents a hybrid algorithm that aims at solving optimization problems originated from nonlinear equations systems with multiple solutions. The model is composed of two algorithms concatenated, where the first is stochastic and the second is deterministic, and executed consecutive times through an repetition structure, designed in order to force the search of all possible solutions within the limits of the range search. At last, the results are post-optimized, resulting in a group of several solutions, equivalent to the alternative of nonlinear system that represents the problem.

KEYWORDS: optimization, hybrid algorithms, operational research. metaheuristics.

1. Introdução

Este trabalho apresenta o desenvolvimento de um algoritmo híbrido (Silva, 2009), para solução de problemas de otimização não-linear com múltiplas soluções. Para isso, foi idealizada uma composição de dois algoritmos, para formação de um algoritmo híbrido, com eficiência computacional superior aos algoritmos clássicos quando aplicados isoladamente, especificamente para localização de todas as múltiplas raízes que outros. Este algoritmo apresenta eficiência computacional superior a outro localizando mais raízes que este outro algoritmo, e/ou localizando o mesmo número de raízes, mas com tempo computacional inferior. A eficiência deste algoritmo híbrido é demonstrada através de testes envolvendo problemas clássicos da literatura conhecidos por sua complexidade e pela dificuldade de localização de todas as soluções. Estes problemas são do tipo não-lineares irrestritos (ou com restrições do tipo "caixa") multimodais/multiraiz. Para composição do algoritmo híbrido serão utilizados os algoritmos Luus-Jaakola como gerador de pontos iniciais para o algoritmo Hooke-Jeeves, compondo assim o algoritmo híbrido.

O diferencial do algoritmo proposto está, primeiramente, na inserção do algoritmo de busca de Hooke e Jeeves (1961), produzindo uma estrutura híbrida em que se combina um algoritmo estocástico Luus-Jaakola (1973), com um determinístico. Com isto, o algoritmo determinístico faz uma busca na vizinhança da solução fornecida pelo algoritmo estocástico, refinando a solução e acelerando o tempo de execução em relação à abordagem estocástica isolada. Outra característica fundamental deste algoritmo híbrido está na geração de um conjunto de soluções resultante do número de execuções.

Para isso, um laço externo (N) é definido, através de um número de execuções impostas para tentar encontrar o máximo de soluções distintas possíveis. Durante a execução deste laço, uma quantidade de valores é armazenada como um conjunto de soluções. Porém, neste conjunto, diversos valores se repetem ou representam a mesma solução. Então, uma rotina de classificação define conjuntos de valores que pertencem a uma mesma solução, ou seja, um conjunto de pontos candidatos pertencentes a uma região de vizinhança de uma solução. Esta rotina forma classes de dados para cada solução. No final da execução do laço externo, um conjunto de classes é produzido. Uma última etapa, denominada pós-otimização, é ativada para filtrar somente uma solução de cada conjunto. As soluções filtradas serão as melhores encontradas pelo algoritmo, e serão exibidas no final do processo.

2. O Algoritmo Luus Jaakola

A seguir o algoritmo de Luus jaakola (1973) padrão é descrito, o qual é utilizado como núcleo do modelo apresentado neste trabalho. O procedimento que o algoritmo desenvolve se divide em praticamente duas etapas. Na primeira etapa, o algoritmo gera um vetor de números aleatórios para ajustes dos valores iniciais x^*, que dentro de um número de iterações, resultará num conjunto de soluções. Estas soluções estarão sujeitas às restrições do problema, gerando um grupo de soluções viáveis. Nesta etapa, a melhor solução para o problema, será a escolhida. Na segunda etapa, o espaço de busca é contraído de um fator de redução, e a primeira etapa é novamente executada. Por fim, depois de um número fixo de iterações externas, o resultado obtido será a melhor solução para o problema. Observe que este algoritmo finaliza com apenas uma solução. Em problemas com múltiplas raízes, o algoritmo deve ser executado diversas vezes (Silva, 2010). A Fig. 1 descreve o algoritmo original de Luus jaakola (1973).

Escolha um tamanho inicial de busca $\mathbf{r}_{(0)}$.
Escolha um número externo de iterações n_{out} *e um número interno* n_{in}.
Escolha um coeficiente de contração λ
Gere uma solução inicial $\mathbf{x}^*$
Para $i = 1$ *até* n_{out}
Para $j = 1$ *até* n_{in}
$\mathbf{x}_{(j)} = \mathbf{x}^* + R_{(j)}\mathbf{r}_{(i-1)}$, *onde* $R_{(j)}$ *é um vetor de números aleatórios entre -0.5 e 0.5.*
Se $F(\mathbf{x}_{(j)}) < F(\mathbf{x}^*)$
$\mathbf{x}^* = \mathbf{x}_{(j)}$
Fim Se
Fim Para
$\mathbf{r}_{(i)} = (1-\lambda)\mathbf{r}_{(i-1)}$
Fim Para.

Figura 1 - Algoritmo de Luus Jaakola.

3. O Algoritmo de Hooke e Jeeves

O Algoritmo proposto por Hooke-Jeeves promove dois tipos de busca: a exploratória e a padrão. A primeira etapa é a busca exploratória. Para isso, a partir de um ponto inicial, o método explora todas as direções de busca para cada variável, selecionando a melhor (onde a função-objetivo tem seu valor diminuído). Esta etapa define um novo ponto com um valor melhor para função objetivo. Depois de explorar todas as direções de busca, o método executa a próxima etapa, a busca padrão, também conhecida como de progressão ou aceleração, avançando na direção definida na última iteração até um valor $\alpha > 0$ (fator de aceleração). A partir deste ponto repete-se a primeira etapa até alcançar o próximo ponto. A Figura 2 ilustra as duas etapas do método

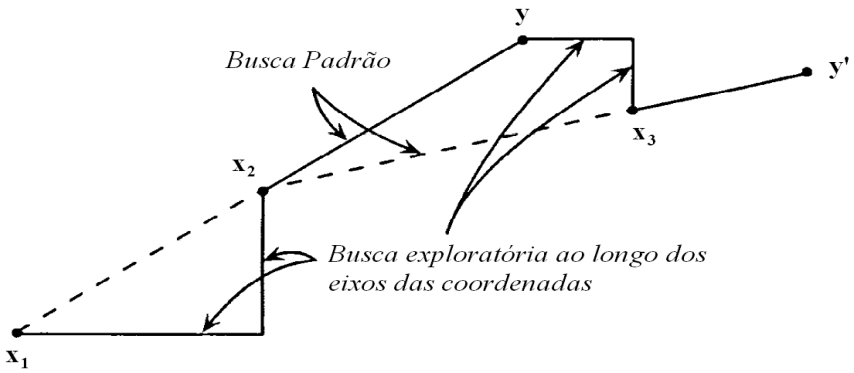

Figura 2 – Ilustração dos passos do método de Hooke-Jeeves.

Conforme a figura, a partir do ponto inicial x_1, o método executa uma busca exploratória em todas as direções, escolhendo a direção relacionada ao menor valor de $f(x_i),\ i = 1,2$, para cada eixo, chegando ao ponto x_2. Neste ponto, o método executa uma busca ao longo da direção $(x_2 - x_1)$ multiplicada de um fator de aceleração α, alcançando o ponto resultante *y*. A partir deste ponto, o método repete os passos, alcançando *y'*, e assim sucessivamente até atender ao critério de parada. A Figura 3 apresenta o método Hooke-Jeeves em forma de pseudocódigo.

Passo de Inicialização

Defina $d_1,...,d_n$ como das direções coordenadas
Escolha um escalar $\varepsilon > 0$ para determinar a parada do algoritmo
Escolha o tamanho para o passo inicial $\Delta \geq \varepsilon$, e o fator de aceleração $a > 0$
Escolha o ponto de partida x_1, faça $y_1 = x_1$ e $k = j = 1$ e vá ao passo principal,

Passo Principal

1- Se $f(y_i + \Delta d_i) < f(y_i)$, então "sucesso", faça $y_{i+1} = y_i + \Delta d_i$ e vá ao passo 2
Se $f(y_i + \Delta d_i) \geq f(y_i)$, então "falha", Neste caso:
Se $f(y_i - \Delta d_i) < f(y_i)$, faça $y_{i+1} = y_i - \Delta d_i$ e vá ao passo 2
Se $f(y_i - \Delta d_i) \geq f(y_i)$, faça $y_{i+1} = y_i$ e vá ao passo 2
2 – Se $j < n$, troque j por $j+1$ e repita o passo 1
Caso contrário:
Se $f(y_{n+1}) < f(x_k)$ vá ao passo 3
Se $f(y_{n+1}) \geq f(x_k)$ vá ao passo 4
3 – Faça $x_{k+1} = y_{n+1}$, e $y_1 = x_{k+1} + \alpha(x_{k+1} - x_k)$, Troque k por $k+1$, faça $j = 1$ e vá ao passo 1
4 – Se $\Delta \leq \varepsilon$, pare; x_k é a solução,
Caso contrário, troque Δ por $\Delta/2$, Faça $y_i = x_k$, $x_{k+1} = x_k$ troque k por $k+1$,
faça j =1, repita o passo 1
Observação:
Os passos 1 e 2 acima descrevem uma busca exploratória, No passo 3, há uma aceleração na direção $x_{k+1} - x_k$. Por fim, no passo 4, o tamanho de Δ é reduzido.

Figura 3 - Algoritmo de Hooke-Jeeves.

4. O Algoritmo Híbrido

A idéia central é explorar o espaço de busca com o Luus-Jaakola e fazer a "garimpagem" das áreas mais promissoras utilizando o Hooke-Jeeves. No processo de alternância entre os dois algoritmos, pretende-se determinar as várias soluções de problemas multimodais. O algoritmo híbrido (Silva, 2009), divide-se principalmente em duas etapas: a etapa Luus-Jaakola gera um ponto inicial, dentro dos limites da região de busca, para ser utilizado pela etapa Hooke-Jeeves. Esta recebe então este ponto, que será utilizado como ponto inicial, e é executado, gerando uma solução refinada, mais próxima do ótimo, que será uma possível solução para o problema. Sistemas não-lineares apresentam, em diversas ocasiões, mais de uma solução; portanto, para que o algoritmo localize todas as soluções existentes, estes passos devem ser executados várias vezes.

Como o algoritmo híbrido é composto das etapas denominadas LJ e HJ, é importante definir os termos relacionados aos números de *loops* utilizados pelo algoritmo. Desta formam ficam definidos três níveis de laços (*loops*). O primeiro laço é o mais interno, e está relacionado ao laço das iterações internas do algoritmo Luus-Jaakola, ao passo que o segundo nível se refere ao laço externo da etapa Luus-Jaakola. O terceiro laço é natural do algoritmo híbrido e determina o número de execuções que o algoritmo híbrido repetirá. Este laço também será utilizado para executar o algoritmo Luus-Jaakola repetidamente. A seguir são apresentadas as nomenclaturas adotadas neste trabalho para os três níveis de laços.

I-	Primeiro Nível:	Laço interno da etapa Luus-Jaakola	n_{in}
II-	Segundo Nível:	Laço externo da etapa Luus-Jaakola	n_{out}
III-	Terceiro Nível:	Laço das execuções do algoritmo híbrido	N

Com vistas a identificar todas as soluções (raízes) do problema, o algoritmo híbrido funciona com um número externo de iterações (*N*), onde cada iteração produz uma possível solução. Estes valores são armazenados em um vetor de tamanho igual ao número de iterações. No caso de uma função-objetivo com múltiplas raízes, os valores distintos encontrados nas diversas iterações são filtrados posteriormente para selecionar as diversas raízes sem repetição, e sujeitos a uma última etapa de pós-otimização.

Por fim, um conjunto de raízes é gerado, resultando um conjunto de todas as raízes distintas localizadas, que é a solução final do problema. A seguir na Figura 4, o algoritmo híbrido é apresentado.

Defina um numero de iterações n
Defina R = Vetor raízes
Defina S = Vetor Solução
Para k =1 até n faça
Execute a rotina Luus-Jaakola
Faça $x_{inicial}\ (\text{Hooke e Jeeves}) = x_{final}\ (\text{Luus - Jaakola})$
Execute a rotina Hooke e Jeeves
Faça $r_k = x_{final}\ (\text{Hooke e Jeeves})\ (r_k \in R)$
Fim para
Filtre as raízes repetidas do vetor R classificando-as por aproximação
Minimize cada classe de valores gerando um valor por classe (raízes distintas)
Copie as raízes distintas de R em S
Imprima o vetor solução S

Figura 4 – Algoritmo Híbrido.

5. Testes e Resultados

Devido ao fato do algoritmo Luus-Jaakola localizar apenas uma solução por execução, foi criado um laço externo *N* com um número suficiente de vezes para possibilitar a busca de valores distintos de soluções (possivelmente todas as soluções) (Silva, 2010). Os resultados foram analisados quanto ao tempo de execução, e ao percentual de acertos em relação ao número de execuções para cada algoritmo.

Estes percentuais foram representados através de gráficos de barras, onde as coordenadas foram dadas pelo número da raiz para eixo-*x*, e o percentual de ocorrência de acertos para o eixo-*y*. O número de amostras será de 100 execuções para cada algoritmo. Para cada função, os parâmetros foram ajustados identicamente para os dois algoritmos, para que haja uniformidade dos dados de entrada.

A Tabela 1 a seguir lista os exemplos utilizados como *benchmarks* nesta etapa de testes, que foram submetidos ao algoritmo híbrido e ao algoritmo Luus-Jaakola. Os testes nesta etapa também foram utilizados para avaliação das "taxas de acertos" dos algoritmos (isto é, o número de raízes localizadas por cada algoritmo). Para isso, uma amostragem de 100 execuções de cada algoritmo do problema em questão foi executada, e a distribuição de freqüência das ocorrências de cada raiz ilustrada em forma de histograma. Os valores de *loops* internos e externos foram variados para cada amostragem.

Tabela 1 – Problemas - testes (*benchmarks*).

Nº	*Exemplo*	*Nº de variáveis*	*Nº de raízes*
1	Função de Himmelblau (HIMM)	2	9
2	Sistema trigonométrico (ST)	2	13
3	Sistema polinomial de alto grau (SPAG)	3	12
4	Problema quase-linear de Brown (PQL)	5	3
5	Bini e Mourrain (BM)	3	8

5.1. Exemplo 1 - Função de Himmelblau (HIMM)

A função de Himmelblau é oriunda do sistema não-linear a seguir, com nove raízes (More *et al.*, 1981):

$$\begin{cases} 4x_1^3 + 4x_1x_2 + 2x_2^2 - 42x_1 - 14 = 0 \\ 4x_2^3 + 2x_1^2 + 4x_1x_2 - 26x_2 - 22 \quad = 0 \\ -5 < x_1, x_2 < 5 \end{cases} \tag{1}$$

O problema do sistema não-linear é convertido em um problema de otimização (Silva, 2009), cuja função-objetivo é representada por:

$$f(x) = (-42x_1 + 2x_2^2 + 4x_1x_2 + 4x_1^3 - 14)^2 + (2x_1^2 - 26x_2 + 4x_1x_2 + 4x_1^3 - 22)^2 \tag{2}$$

Tabela 2 – Solução do algoritmo Híbrido com 50 *loops* internos e 10 *loops* externos, tempo de execução = 0,109 s.

Raízes:	x_1	x_2	$f(x_1, x_2)$
1	-0,1279614	-1,9537150	0,0000000000
2	3,5844283	-1,8481265	0,0000000000
3	3,0000000	2,0000000	0,0000000000
4	0,0866775	2,8842547	0,0000000000
5	3,3851542	0,0738517	0,0000000000
6	-3,0730257	-0,0813530	0,0000000000
7	-0,2708446	-0,9230385	0,0000000000
8	-2,8051181	3,1313126	0,0000000001
9	3,7793102	3,2831860	0,0000000000

Para uma maior realidade dos resultados, os dois algoritmos foram executados 100 vezes e seus resultados mostrados através de um gráfico com as freqüências das ocorrências das localizações das raízes. Para isso a configuração dos números de *loops* foram aumentados para $n_{in} = 50$ *e* $n_{out} = 50$.

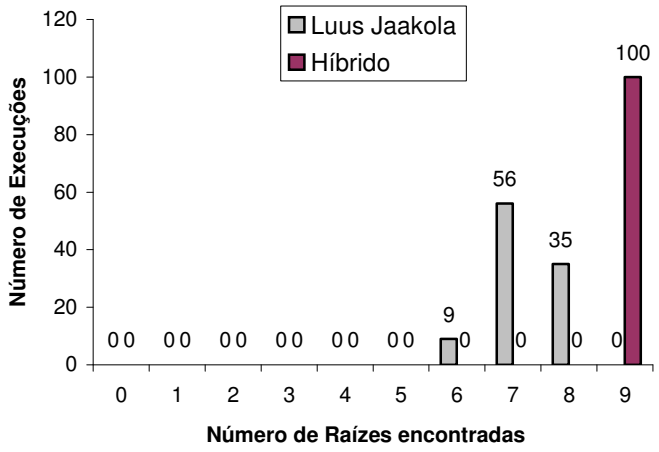

Figura 5 – Distribuição de freqüência das raízes localizadas para n_{in} = 50, n_{out} = 50 *e* N =100.

5.2. Exemplo 2 – Sistema trigonométrico (ST)

O sistema descrito pelas equações (3) foi usado por Hirsch *et al*. (2006) para testar o algoritmo C-GRASP para resolver sistemas não-lineares. Este sistema possui 13 soluções. As equações que descrevem o ST são:

$$\begin{cases} -\sin(x_1)\cos(x_2) - 2\cos(x_1)\sin(x_2) = 0 \\ \cos(x_1)\sin(x_2) - 2\sin(x_1)\cos(x_2) = 0 \\ 0 < x_1, x_2 < 2\pi \end{cases} \quad (3)$$

Tabela 3 – Solução do algoritmo Híbrido com 10 *loops* internos e 5 *loops* externos, tempo de execução = 0,234 s.

Raízes:	x_1	x_2	$f(x_1, x_2)$
1	1,5707963	4,7123890	0,0000000000
2	3,1415927	0,0000000	0,0000000000
3	1,5707964	1,5707963	0,0000000000
4	3,1415925	3,1415927	0,0000000000
5	4,7123895	4,7123890	0,0000000000
6	4,7123890	1,5707965	0,0000000000
7	6,2831855	0,0000000	0,0000000000
8	0,0000000	6,2831855	0,0000000000
9	0,0000000	3,1415927	0,0000000000
10	0,0000000	0,0000000	0,0000000000
11	6,2831855	6,2831850	0,0000000000
12	3,1415930	6,2831855	0,0000000000
13	6,2831855	3,1415925	0,0000000000

Os parâmetros foram escolhidos com uma configuração mínima para localização de todas as 13 soluções pelo algoritmo híbrido. A configuração dos *loops* externos e internos são de 5 e 10 respectivamente e N=100.

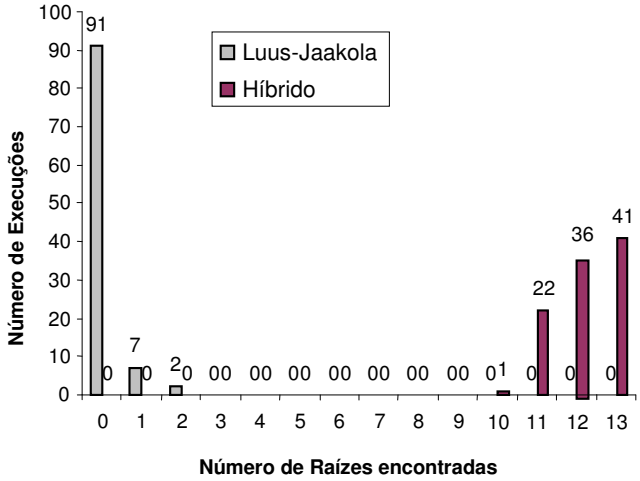

Figura 6 – Distribuição de freqüência das raízes localizadas para $n_{in} = 10$, $n_{out} = 5$ e $N = 100$.

5.3. Exemplo 3 - Sistema polinomial de alto grau (SPAG)

Este exemplo foi proposto por Kearfott (1987), e é representado pelo sistema não-linear (4), com 12 soluções:

$$\begin{cases} 5x_1^{\,9} - 6x_1^{\,5}x_2^{\,2} + x_1x_2^{\,4} + 2x_1x_3 = 0 \\ -2x_1^{\,6}x_2 - 2x_1^{\,2}x_2^{\,3} + 2x_2x_3 = 0 \\ x_1^{\,2} + x_2^{\,2} - 0.265625 = 0 \\ -0.6 < x_1 < 6 \\ -0.6 < x_2 < 0.6 \\ -5 < x_3 < 5 \end{cases} \tag{4}$$

Tabela 4 – Solução do algoritmo Híbrido com 10 *loops* internos e 5 externos, tempo de execução = 1,171 s.

Raízes:	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	-0,2797925	-0,4328302	-0,0141882	0,0000000000
2	0,0000008	0,5153883	0,0000003	0,0000000000
3	0,4670031	0,2180203	0,0000003	0,0000000000
4	-0,4669975	-0,2180341	-0,0000011	0,0000000000
5	0,2799042	-0,4327585	-0,0141888	0,0000000000
6	-0,2798981	0,4327623	-0,0141883	0,0000000000
7	-0,0000003	-0,5153880	-0,0000003	0,0000000000
8	-0,5153875	-0,0000002	-0,0124457	0,0000000000
9	0,4669540	-0,2181233	0,0000012	0,0000000000
10	0,5153885	-0,0000005	-0,0124458	0,0000000000
11	0,2799087	0,4327556	-0,0141885	0,0000000000
12	-0,4670068	0,2180156	-0,0000019	0,0000000000

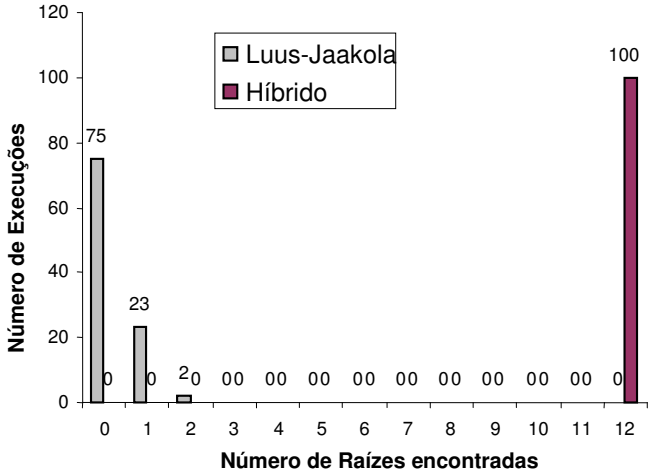

Figura 7 – Distribuição de freqüência das raízes localizadas para n_{in} = 10, n_{out} = 5 e N = 100.

5.4. Exemplo 4 – Problema quase linear de Brown (PQL)

O problema a seguir (Grosan e Abraham, 2008) possui três raízes reais e é representado por:

$$\begin{cases} f_1 = 2x_1 + x_2 + x_3 + x_4 + x_5 - 6 \\ f_2 = x_1 + 2x_2 + x_3 + x_4 + x_5 - 6 \\ f_3 = x_1 + x_2 + 2x_3 + x_4 + x_5 - 6 \\ f_4 = x_1 + x_2 + x_3 + 2x_4 + x_5 - 6 \\ f_5 = x_1 x_2 x_3 x_4 x_5 - 1 \\ -10 < x_1, x_2, x_3, x_4, x_5 < 10 \end{cases} \quad (5)$$

Primeiramente, os parâmetros para configuração do algoritmo híbrido foram de 10 *loops* internos, 5 *loops* externos e N=100. Neste caso, o algoritmo híbrido localizou todas as três raízes, enquanto que o algoritmo Luus-Jaakola, para esta mesma configuração, não localizou nenhuma. O resultado do algoritmo híbrido para este caso está mostrado na Tabela 5. Para que o algoritmo Luus-Jaakola localizasse pelo menos uma raiz, os valores dos *loops* externos e internos foram de 30 e 50 respectivamente, e N = 100. O percentual de acerto de ambos os algoritmos para comparação está apresentado no gráfico da Figura 8.

Tabela 5 – Solução do algoritmo Híbrido com 10 *loops* internos e 5 externos, tempo de execução = 0,859 s.

Raízes:	x_1	x_2	x_3	x_4	x_5	$f(x_i)$, i=1,...,5
1	1,0000011	1,0000017	1,0000005	1,0000029	0,9999930	0,0000000000
2	0,9163576	0,9163607	0,9163589	0,9163573	1,4182062	0,0000000000
3	-0,5790439	-0,5790429	-0,5790425	-0,5790423	8,8952141	0,0000000000

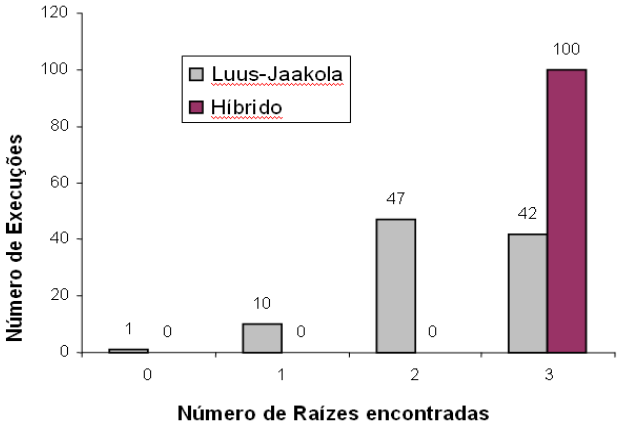

Figura 8 – Distribuição de freqüência das raízes localizadas para n_{in} = 50, n_{out} = 30 e N = 100.

5.5. Exemplo 5 - Bini e Mourrain (2004) (BM)

Neste exemplo foi utilizado um problema (Bini e Mourrain, 2004), envolvendo três equações não-lineares (6) e apresentando oito raízes.

$$\begin{cases} -x_2^{\,2}x_3^{\,2} - x_2^{\,2} + 24x_2x_3 - x_3^{\,2} - 13 = 0 \\ -x_1^{\,2}x_3^{\,2} - x_1^{\,2} + 24x_1x_3 - x_3^{\,2} - 13 = 0 \\ -x_1^{\,2}x_2^{\,2} - x_1^{\,2} + 24x_1x_2 - x_2^{\,2} - 13 = 0 \\ 0 \le x_1, x_2, x_3 \le 20 \end{cases} \quad (6)$$

Tabela 6 – Solução do algoritmo Híbrido com 50 *loops* internos e 20 *loops* externos, tempo de execução = 0,343 s.

Raízes:	X_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	0,7795480	0,7795480	0,7795480	0,0000000000
2	0,7795489	0,7795488	10,8577080	0,0000000010
3	0,3320730	4,6251769	4,6251869	0,0000000040
4	4,6251764	0,3320730	4,6251869	0,0000000006
5	4,6251817	4,6251817	4,6251822	0,0000000074
6	0,7795482	10,8577042	0,7795482	0,0000000001
7	4,6251683	4,6251950	0,3320731	0,0000000021
8	10,8577070	0,7795487	0,7795486	0,0000000006

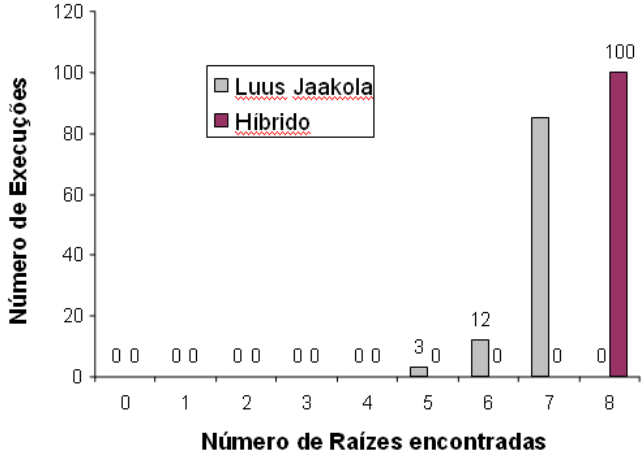

Figura 9 – Distribuição de freqüências das raízes localizadas para n_{in} = 50, n_{out} = 50 e N = 100.

A Tabela 7 mostra os resultados alcançados por ambos os algoritmos alterando-se o número de loops *N*, e os tempos consumidos por cada teste. É importante ressaltar que em todos os casos, o algoritmo Luus-Jaakola não alcança as oito raízes do problema. Outra observação relevante é que a diferença entre os tempos de execução do algoritmo híbrido em relação ao Luus-Jaakola é praticamente irrelevante para este exemplo.

Tabela 7 – Testes com os dois algoritmos variando o número de *loops N*.

Algoritmo	**Luus-Jaakola**		**Híbrido**	
Nº de *Loops N*	Nº de raízes	Tempo (segundos)	Nº de raízes	Tempo (segundos)
300	7	2,328	8	2,593
250	6	1,906	8	2,109
200	6	1,5	8	1,718
150	5	1,109	8	1,281
100	5	0,75	8	0,843
50	4	0,375	7	0,437

6. Conclusões

Conclui-se que a metodologia híbrida conserva as boas características dos métodos estocásticos de otimização global consumindo um menor número de avaliações da função-objetivo e, portanto, menor tempo de computação. Para que o Algoritmo de Luus-Jaakola tenha um desempenho semelhante ao algoritmo híbrido, ou seja, identifique o mesmo número de raízes, é necessário um número maior de iterações externas e internas demandando, por conseguinte, maior tempo computacional. O mesmo problema quando executado no algoritmo híbrido solicita um menor número de avaliações da função-objetivo, uma vez que os números de iterações externas e internas no passo Luus-Jaakola da metodologia híbrida são consideravelmente menores.

Desta forma foi demonstrada através de comparações com um método estocástico (Luus-Jaakola) e um método determinístico, a eficiência e robustez do algoritmo híbrido, através de testes exaustivos utilizando problemas conhecidos de difíceis soluções, sempre alcançando os resultados de forma precisa em todos os problemas testes avaliados, localizando todas as soluções conhecidas da literatura.

Portanto, o algoritmo híbrido atendeu às espectativas demonstrando eficiência na localização de múltiplas raízes, sempre com resultados vantajosos em relação ao algoritmo Luus-Jaakola, seja em função do tempo computacional consumido, da qualidade dos ótimos encontrados, ou mesmo no número de soluções encontradas, que é o objetivo principal do algoritmo.

Para trabalhos futuros, serão experimentados outros algoritmos clássicos da literatura, ou mesmo outro que apresente viabilidade, dentro da estrutura híbrida, buscando um melhor desempenho quando comparado com estes algoritmos isoladamente e/ou com o algoritmo híbrido apresentado neste trabalho.

7. Referências

Bini, D.A.; Mourrain, B. "Cyclo", from polynomial test suite. Disponível em: <http://wwwsop.inria.fr/saga/POL/ BASE/2. multipol/cyclohexan.html>. Acesso em: 2004.

Grosan C.; Abraham A. A new approach for solving nonlinear equations systems. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, v. 38, n. 3, p. 698-714, May 2008.

Grosan C.; Abraham A. Multiple solutions for a system of nonlinear equations. International Journal of Innovative Computing, Information and Control, v. 4, n. 9, sept. 2008.

Hirsch M. J.; Meneses C. N.; Pardalos P. M.; Resende M.G.C. Global optimization by continuous grasp. Mar. 8, 2006. Disponível em: <http://www.optimization-online.org/DB_FILE/2006/03/1344.pdf>.

Hooke, R.; Jeeves. T. A**.** (1961), "*Direct search solution of numerical and statistical problems"*. Journal of the Association for Computing Machinery, v. 8. p. 212-229,.

Kearfott, R. B. Abstract generalized bisection and a cost bound. Math. Comput. v. 49, n.179, 1987.

Luus, R.; Jaakola, T. (1973), "*Optimization by direct search and systematic reduction of the size of search region"*. AIChE Journal, vol. 19.

More J.J.; Garbow, B.S.; Hillstrom, K.E. Testing Unconstrained Optimization Software, ACM Transactions on Mathematical Software, 7, 136, 1981

Silva, M. Rodrigues. *Um Novo Método Híbrido Aplicado à Solução de Sistemas Não-Lineares com Raízes Múltiplas*, Tese de Doutorado, UERJ, Nova Friburgo, RJ, 2009.

Silva, M. Rodrigues. *Algoritmo heurístico de busca direta para solução de problemas de programação não linear irrestrita com múltiplos ótimos.* XLII SBPO – Simpósio Brasileiro de Pesquisa Operacional, Bento Gonçalves, RS, 2010.