Algoritmo GRASP para solução de igualdades matemáticas simples

Lucas dos Reis Cardoso

***Abstract*— This article aims to explain the steps of the GRASP algorithm and for that it uses it to solve simple mathematical equalities. The GRASP algorithm has two main steps: the construction of an initial solution and the continuous refinement of this solution in order to find one that satisfies the initial problem. We will see how the algorithm was used in order to solve the proposed problem.**

***Index Terms*— GRASP, local search, greedy algorithm**

***Resumo*— Este artigo tem como objetivo explicar os passos do algoritmo GRASP e para isso o utiliza para solucionar igualdades matemáticas simples. O algoritmo GRASP tem duas etapas principais: a construção de uma solução inicial e o contínuo refinamento dessa solução com a finalidade de encontrar uma que satisfaça o problema inicial. Veremos como que o algoritmo foi usado a fim de resolver o problema proposto.**

***Palavras Chave*— GRASP, busca local, algoritmo guloso**

I. Introdução

GRASP *(Greedy Randomized Adaptive Search Procedure)* [1] é um procedimento guloso que funciona com duas principais fases: uma de construção de uma possível solução e a outra de busca local sobre as possíveis soluções a fim de refinar o resultado. Na fase inicial, ele gera de forma aleatória um conjunto de soluções, seleciona as mais promissoras e, em cima dessas, realiza um procedimento de busca local para obter soluções de melhor qualidade.

A inspiração para escrita deste artigo se origiou do trabalho de Denny Hermawanto [2], onde foi utilizado um algoritmo genético para solução do mesmo problema. O intuito deste artigo é seguir com a mesma ideia, sendo uma fonte de estudo para iniciantes na área.

O novo algoritmo proposto não tem intenção de ter um melhor desempenho, mas apenas uma nova forma de tentar solucionar o problema.

II. Definição do problemas

As igualdades matemáticas que o algoritmo GRASP proposto irá resolver estão no seguinte formato:

$$a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 + ... + a_nx_n = c \quad (1)$$

Os valores de a_1 até a_n são os coeficientes dos termos da igualdade e as variáveis encontradas pelo algoritmo são as de x_1 até x_n, que precisam ser valores inteiros positivos, e a soma de todos os termos precisam ser igual a constante c. Para exemplificar, neste arigo iremos encontrar os valores de x para a igualdade abaixo:

$$1x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 = 50 \quad (2)$$

Inicialmente, serão construídas várias soluções inicias, onde as melhores serão selecionadas e em cima delas será realizada uma busca local para obter melhores valores. Esses passos serão repetidos várias vezes a fim de refinar os possíveis resultados e chegar na solução da igualdade.

Por fim, será mostrado os resultados encontrados ao modificar parâmetros do algoritmo, vendo como que ao modificar, por exemplo, a quantidade de termos nas possíveis soluções iniciais pode alterar a qualidade da solução final.

III. Passos do Algoritmo

Como já dito anteriormente, o algoritmo tem duas principais fases: a construção da solução inicial e o refinamento dessa solução. Nas próximas subseções serão detalhados esses passos do algoritmo.

1. Construção da solução inicial

Para este problema, a solução inicial será formada por N possíveis candidatos, onde cada candidato será formado pelos valores de x_1 até x_n gerados de forma aleatória entre 1 e a constante c. Como temos cinco termos na nossa igualdade, nossos candidatos também terão cinco termos. Abaixo são exemplificados cinco possíveis candidados para os valores de x, onde os valores nos seus respectivos índices representam de x_1 até x_n. Para o exemplo, serão gerados 10 candidados para facilitar a visualização, porém, como veremos mais a frente, aumentando o número desses candidatos podemos reduzir o tempo total de execução do algoritmo para obter a solução desejada.

N_1 = [7, 11, 4, 12, 4]
N_2 = [2, 15, 10, 4, 22]
N_3 = [6, 1, 11, 43, 23]
N_4 = [11, 33, 11, 21, 37]
N_5 = [27, 4, 21, 44, 22]
N_6 = [9, 30, 33, 21, 27]
N_7 = [40, 27, 39, 16, 24]
N_8 = [35, 3, 31, 48, 17]
N_9 = [28, 24, 32, 20, 33]
N_{10} = [41, 29, 27, 37, 38]

Após a geração dos primeiros possíveis candidados, é necessário selecionar um número M de melhores candidados para

ser realizado a busca local. Para o nosso exemplo, serão selecionados os dois melhores candidados que, ao aplicar na igualdade do exemplo inicial, obtiveram o valor mais próximo de 50. Novamente, a escolha de 2 candidados facilita a visualização, porém é um número que pode ser aumentado quando o algoritmo não for executado manualmente. Neste caso, os candidados N_1 e N_2 são selecionados por estarem mais próximos da constante c.

$$SN_1 = 1*7 + 2*11 + 3*4 + 4*12 + 5*4 = \mathbf{38}$$
$$SN_2 = 1*2 + 2*15 + 3*10 + 4*4 + 5*22 = \mathbf{53}$$
$$SN_3 = 1*6 + 2*1 + 3*11 + 4*43 + 5*23 = 84$$
$$SN_4 = 1*11 + 2*33 + 3*11 + 4*21 + 5*37 = 113$$
$$SN_5 = 1*27 + 2*4 + 3*21 + 4*44 + 5*22 = 118$$
$$SN_6 = 1*9 + 2*30 + 3*33 + 4*21 + 5*27 = 120$$
$$SN_7 = 1*40 + 2*27 + 3*39 + 4*16 + 5*24 = 146$$
$$SN_8 = 1*35 + 2*3 + 3*31 + 4*48 + 5*17 = 134$$
$$SN_9 = 1*28 + 2*24 + 3*32 + 4*20 + 5*33 = 137$$
$$SN_{10} = 1*41 + 2*29 + 3*27 + 4*37 + 5*38 = 172$$

Esta foi a primeira iteração da criação de uma solução inicial, os candidatos que não foram selecionados serão descartados e novos candidados serão gerados na próxima subseção, que terão como base os melhores candidados selecionados nesta etapa.

2. *Busca local nos melhores candidatos*

Tendo selecionado os melhores candidados da solução inicial, o próximo passo é realizar uma busca ao redor deles para criar um conjunto de novas soluções, onde os melhores candidados serão novamente selecionados.

Para realização da busca local, serão gerados oito novos candidados para voltar a número original de dez candidados, sendo quatro originados do primeiro melhor candidado o os outros quatro do segundo melhor. Caso o número de candidatos originais seja maior que 10, este número de novos candidatos gerados deve ser ajustado para acomodar este maior número.

Para a geração desses novos candidados, para cada termo do atual, é gerado um número na faixa $[0.5, 2]$ para ser multiplicado por ele, assim, os termos dos novos candidados podem assumir valores entre metade do valor anterior ou o dobro. A seguir, está a nova população de candidados baseados nos termos de $N1$ e $N2$. Vale destacar que os valores gerados na faixa $[0.5, 2]$ estão distribuídos de maneira uniforme e o valor resultante ao ser multiplicado pelo elemento do candidato será arredondado para cima para não ter como resultado o valor 0.

N_1 **= [7, 11, 4, 12, 4]**
N_2 **= [2, 15, 10, 4, 22]**
N_3 = [8, 7, 8, 10, 4]
N_4 = [4, 7, 8, 15, 3]
N_5 = [13, 19, 5, 16, 6]
N_6 = [9, 16, 7, 15, 8]
N_7 = [2, 25, 12, 3, 14]
N_8 = [2, 29, 11, 7, 40]
N_9 = [3, 26, 7, 4, 29]
N_{10} = [2, 23, 18, 5, 21]

Agora, esses valores são aplicados novamente na igualdade inicial, os dois melhores candidados são selecionados e essas duas etapas se repetem diversas vezes.

$$SN_1 = 1*7 + 2*11 + 3*4 + 4*12 + 5*4 = 38$$
$$SN_2 = 1*2 + 2*15 + 3*10 + 4*4 + 5*22 = \mathbf{53}$$
$$SN_3 = 1*8 + 2*7 + 3*8 + 4*10 + 5*4 = 37$$
$$SN_4 = 1*4 + 2*7 + 3*8 + 4*15 + 5*3 = 37$$
$$SN_5 = 1*13 + 2*19 + 3*5 + 4*16 + 5*6 = 59$$
$$SN_6 = 1*9 + 2*16 + 3*7 + 4*15 + 5*8 = \mathbf{55}$$
$$SN_7 = 1*2 + 2*25 + 3*12 + 4*3 + 5*14 = 56$$
$$SN_8 = 1*2 + 2*29 + 3*11 + 4*7 + 5*47 = 89$$
$$SN_9 = 1*3 + 2*26 + 3*7 + 4*4 + 5*29 = 69$$
$$SN_{10} = 1*2 + 2*23 + 3*18 + 4*5 + 5*21 = 69$$

Dos novos candidados os de melhor resultado foram os N_2 e N_6. Além disso, é fácil de visualizar que os novos valores estão muito mais próximos do valor desejado de 50, indicando que os candidados estão convergindo para a solução desejada. Estes passos serão repetidos diversas vezes, para no fim ser obtido valores que satisfaçam a igualdade. O primeiro elemento da população resultante será o valor mais próximo do objetivo.

Após execução desses passos por diversar vezes, os resultados finais tendem a convergir para um valor único, ou seja, a variedade de soluções geradas é pequena. Para obter maior variedade, o algoritmo deve ser executado quantas vezes for necessário.

IV. Resultados obtidos

Para obter os resultados foi definido cinco casos de teste e, para cada caso de teste, o algoritmo será executado 10 vezes para medir a qualidade dos resultados obtidos. Nas próximas subseções os casos de teste serão detalhados e os resultados serão exibidos.

1. *Caso de teste 1*

No primeiro caso de teste serão gerados 10 soluções para os parâmetros abaixo. As soluções serão listadas e a distância do valor solução informado, sendo que o valor de 0 seria uma solução exata.

	população	melhores selecionados	qtd incógnitas	iterações
valor	50	10	5	100

Tabela I
Primeiro caso de teste

Nº	x1	x2	x3	x4	x5	d
1	15	4	3	2	2	**0**
2	3	5	2	4	3	**0**
3	8	5	2	4	2	**0**
4	4	3	3	4	3	**0**
5	8	5	3	2	3	**0**
6	7	2	4	3	3	**0**
7	5	9	3	2	2	**0**
8	5	5	4	2	3	**0**
9	2	5	2	3	4	**0**
10	8	2	5	2	3	**0**

Tabela II
Resultados de 10 execuções do caso de teste 1.

Para 5 incógnitas o caso de teste 1 conseguiu obter soluções exatas.

2. Caso de teste 2

Neste caso de teste, vamos reduzir o número da população, melhores selecionados e de iterações para observar o comportamento.

	população	melhores selecionados	qtd incógnitas	iterações
valor	25	5	5	50

TABELA III
SEGUNDO CASO DE TESTE

Nº	x1	x2	x3	x4	x5	d
1	11	4	2	4	2	**1**
2	8	3	4	2	3	**1**
3	6	4	3	3	3	**0**
4	8	6	4	2	2	**0**
5	9	5	3	3	2	**0**
6	5	5	3	3	3	**1**
7	8	2	2	3	4	**0**
8	3	3	6	2	3	**0**
9	7	3	5	3	2	**0**
10	4	8	2	2	3	**1**

TABELA IV
RESULTADOS DE 10 EXECUÇÕES DO CASO DE TESTE 2.

Ao reduzir a população, melhores selecionados e iterações começamos a obter resultados não exatos, com distância de 1 do valor desejado.

3. Caso de teste 3

Neste caso de teste, vamos manter os parâmetros do teste anterior, mas vamos voltar a quantidade de iterações para 100. Dobrando as iterações, os resultados devem melhorar.

	população	melhores selecionados	qtd incógnitas	iterações
valor	25	5	5	100

TABELA V
TERCEIRO CASO DE TESTE

Nº	x1	x2	x3	x4	x5	d
1	11	2	4	2	3	**0**
2	2	7	4	3	2	**0**
3	5	6	2	3	3	**0**
4	7	5	5	2	2	**0**
5	4	4	5	2	3	**0**
6	6	4	2	5	2	**0**
7	19	2	3	2	2	**0**
8	8	4	4	3	2	**0**
9	8	4	4	3	2	**0**
10	5	7	3	3	2	**0**

TABELA VI
RESULTADOS DE 10 EXECUÇÕES DO CASO DE TESTE 3.

Assim como o esperado, dobrando o número de iterações obtivemos resultados melhores em comparação ao anterior, mesmo mantendo a população e a quantidade de melhores selecionados.

4. Caso de teste 4

Neste caso de teste, vamos manter todos os parâmetros iguais ao anterior, porém vamos aumentar o número de incógnitas para 8, definidas pela nova igualdade abaixo:

$$1x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + 6x_6 + 7x_7 + 8x_8 = 100 \quad (3)$$

	população	melhores selecionados	qtd incógnitas	iterações
valor	25	5	8	100

TABELA VII
QUARTO CASO DE TESTE

Nº	x1	x2	x3	x4	x5	x6	x7	x8	d
1	16	12	6	4	4	3	3	3	**57**
2	25	4	7	3	2	3	3	3	**39**
3	11	6	3	7	3	3	3	3	**38**
4	12	9	6	6	5	3	4	3	**67**
5	16	6	4	8	3	8	2	2	**65**
6	18	7	5	8	2	5	4	4	**79**
7	8	7	8	7	4	3	2	3	**50**
8	15	3	5	3	3	2	6	2	**33**
9	19	7	8	9	3	2	2	3	**58**
10	4	6	3	9	3	3	3	4	**47**

TABELA VIII
RESULTADOS DE 10 EXECUÇÕES DO CASO DE TESTE 4.

Aumentar em 3 o número e incógnitas fez o algoritmo retornar resultados muito distantes do objetivo.

5. Caso de teste 5

Neste caso de teste, vamos manter o número de incógnitas do anterior, porém vamos aumentar em cinco vezes o número de iterações.

	população	melhores selecionados	qtd incógnitas	iterações
valor	25	5	8	500

TABELA IX
QUINTO CASO DE TESTE

Nº	x1	x2	x3	x4	x5	x6	x7	x8	d
1	4	6	3	4	2	2	3	2	**0**
2	7	3	4	6	2	2	2	2	**1**
3	11	6	3	4	2	2	2	2	**0**
4	3	3	3	4	2	3	2	3	**0**
5	6	3	3	4	3	3	2	2	**0**
6	6	2	3	3	4	2	3	2	**0**
7	6	10	3	2	3	2	2	2	**0**
8	3	2	4	6	3	2	2	2	**0**
9	4	4	4	2	2	2	3	3	**1**
10	9	2	5	2	3	2	3	2	**0**

TABELA X
RESULTADOS DE 10 EXECUÇÕES DO CASO DE TESTE 5.

Aumentando o número de iterações, obtivemos resultados muito superiores, porém leva um tempo consideravelmente maior para ser executado (tempo de execução não contemplado nos testes).

V. Conclusão

Vimos uma forma bem simples de aplicar o algoritmo GRASP para a solução de igualdades matemáticas. Como o algoritmo tende a convergir para um valor único, uma melhoria a ser pensada seria a de aumentar a variedade das soluções obtidas. Outro ponto não realizado foi a comparação de performance deste algoritmo com outros que tentam solucionar o mesmo problema.

Referências

[1] Thomas A Feo e Mauricio GC Resende. "Greedy randomized adaptive search procedures". Em: *Journal of global optimization* 6.2 (1995), pp. 109–133.

[2] Denny Hermawanto. "Genetic algorithm for solving simple mathematical equality problem". Em: *arXiv preprint arXiv:1308.4675* (2013).