UNIVERSIDADE ESTADUAL DE CAMPINAS

PO120 – PROGRAMAÇÃO INTEIRA

UTILIZAÇÃO DO ALGORITMO GRASP PARA RESOLUÇÃO DE PROBLEMAS DE EMPACOTAMENTO DE CÍRCULO

Pedro Esch de Campos Gomes Maced 186787, José Alberto Jara Quintanilla 258729 Victor Crepaldi Ramos 271692, Mateus de Assis Chacon Lima 241715

1. Introdução

Nas últimas décadas, diversos algoritmos de otimização têm sido desenvolvidos para abordar problemas de otimização, mas muitos deles enfrentam limitações ao lidar com problemas complexos de grande escala. Isso ocorre devido ao aumento exponencial do espaço de busca, tornando a busca exaustiva impraticável, e à necessidade de fazer suposições que nem sempre são válidas em métodos aproximados tradicionais. Como resposta a essas limitações, foram propostos algoritmos meta-heurísticos inspirados em processos naturais, que se destacam por sua capacidade de lidar com problemas de alta dimensão e complexidade, oferecendo soluções próximas ao ótimo em um custo computacional razoável [1].

As meta-heurísticas são estratégias inteligentes para aprimorar procedimentos heurísticos, combinando conceitos de exploração e exploração do espaço de busca de forma adaptável. Elas não garantem a otimalidade, mas visam encontrar soluções de alta qualidade eficientemente. Esses algoritmos podem ter comportamento estocástico e são classificados de diversas maneiras, sendo reconhecidos por sua capacidade de guiar e modificar heurísticas subordinadas para produzir soluções de alta qualidade em problemas complexos [1].

Para a realização deste trabalho será utilizado o GRASP (Procedimento de Busca Adaptativa Aleatória Gananciosa), que é uma técnica de otimização que segue uma abordagem iterativa, composta por duas etapas: construção e busca local. Durante a fase de construção, uma solução é gradualmente montada. Caso essa solução não seja viável, um procedimento de correção é aplicado para torná-la viável. Após obter uma solução viável, a busca local explora sua vizinhança em busca de um mínimo ou máximo local. A melhor solução encontrada ao longo das iterações é retida como resultado final [2]. A Figura 1 representa de forma simplificada como é o funcionamento da meta heurística.

```
procedure GRASP(Max_Iterations, Seed)
1     Read_Input();
2     for k = 1,...,Max_Iterations do
3           Solution ← Greedy_Randomized_Construction(Seed);
4           if Solution is not feasible then
5                 Solution ← Repair(Solution);
6           end;
7           Solution ← Local_Search(Solution);
8           Update_Solution(Solution, Best_Solution);
9     end;
10    return Best_Solution;
end GRASP.
```

Figura 1: Pseudocódigo GRASP [2].

Na fase de construção do processo GRASP, uma solução viável é construída iterativamente, um elemento de cada vez. Em cada iteração de construção, a escolha do próximo elemento a ser adicionado é determinada pela ordenação de todos os elementos em uma lista de candidatos com base em uma função gananciosa. Essa função mede o benefício (limitado à visão de curto prazo) de selecionar cada elemento. A heurística é adaptativa porque os benefícios associados a cada elemento são atualizados a cada iteração da fase de construção para refletir as mudanças causadas pela seleção do elemento anterior. O componente probabilístico de um GRASP é caracterizado pela escolha aleatória de um dos melhores candidatos na lista, mas não necessariamente o melhor candidato absoluto. A lista dos melhores candidatos é chamada de lista restrita de candidatos (RCL). Essa técnica de escolha permite a obtenção de soluções diferentes a cada iteração do GRASP, sem comprometer necessariamente o poder do componente ganancioso adaptativo do método [3].

O GRASP tem sido utilizado em diversas áreas, alguns exemplos são: programação, roteamento, lógica, particionamento, localização e layout, aplicações teóricas de grafos, problemas quadráticos e outros problemas de atribuição, transporte, telecomunicações, biologia, desenho automático, sistemas de energia elétrica, entre outas [4].

Essa técnica oferece tanto vantagens quanto desvantagens em sua aplicação. Entre as vantagens, destacam-se a simplicidade de implementação, baseada em algoritmos gananciosos e busca local. No entanto, é importante mencionar que alguns parâmetros, como as restrições na lista de candidatos e o número de iterações, exigem ajustes cuidadosos. Por outro lado, entre as desvantagens, observa-se que o desempenho do GRASP depende significativamente da qualidade das soluções iniciais, uma vez que a randomização entre iterações não incorpora informações prévias. Além disso, o GRASP não faz uso da memória das informações coletadas durante a busca, o que pode limitar sua eficácia em certos cenários [5].

1.1 Materiais e métodos

Esta seção mostra o procedimento realizado para aplicar o algoritmo GRASP aos problemas de empacotamento de círculo.

1.1.1 Definição do problema

O problema de encontrar o empacotamento mais denso de *n* objetos iguais em um espaço delimitado é um desafio clássico em diversos campos científicos e de engenharia. Farkas Bolyai, um matemático húngaro, foi provavelmente o primeiro autor na literatura matemática a estudar a densidade de uma série de círculos empacotados em uma forma delimitada. Em seu principal trabalho, "Tentamen", Bolyai apresentou um empacotamento denso e regular de círculos iguais em um triângulo equilátero, embora tais empacotamentos nem sempre sejam ótimos, mesmo sendo baseados em empacotamentos de grade hexagonal. O problema também foi explorado em problemas japoneses antigos, que continham resultados notáveis relacionados ao empacotamento de círculos [6].

O problema de empacotamento de círculos se refere à questão de como posicionar um número máximo de círculos de tamanhos iguais ou diferentes dentro de um contêiner de forma específica sem sobreposição. O contêiner pode ter várias formas, como um retângulo, um círculo, ou qualquer outra forma geométrica, para e a realização deste artigo o contêiner tem formato de um círculo. O objetivo é otimizar o arranjo dos círculos para que caibam o máximo possível dentro do contêiner, minimizando o espaço não utilizado. Este é um problema clássico de otimização combinatória e é conhecido por ser NP-difícil;

Na prática, várias abordagens heurísticas e algoritmos são empregados para encontrar soluções aproximadas para o problema de empacotamento de círculos. Essas abordagens incluem algoritmos genéticos, GRASP, pesquisa local, entre outros. Cada método tem suas próprias vantagens e desvantagens, dependendo das especificidades do problema, como o tamanho e a forma dos contêineres e dos círculos. Os resultados desses algoritmos podem ser utilizados em diversas aplicações práticas, desde o design de layouts industriais até a organização de elementos em interfaces gráficas de usuário.

1.1.2 Modelagem matemática

A seguir é apresentado o modelo matemático para resolver o problema de empacotamento de círculos:

Variáveis de decisão:

- R: o raio do contêiner circular
- (xi,yi): as coordenadas do centro do i-ésimo círculo dentro do contêiner

$$Min\ R$$

$$s.a\ \left(x_i^2 + y_i^2\right) + r^2 \leq R^2 \qquad (1)$$

$$\sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \geq 2r,\ para\ i \neq j \qquad (2)$$

A função objetivo do problema é encontrar o contêiner de forma circula com o menor raio possível. Sendo que a restrição 1 garante que cada círculo está completamente contido dentro do contêiner. A segunda restrição assegura que nenhum par de círculos se sobreponha. Sendo que as variáveis de decisão pertencem ao conjunto dos reais.

1.2.3 Instâncias

Instância	Quantidade de Círculo	Raios
1	7	1,2,. . . ,7
2	15	1, 2 e 3 (5× cada)
3	30	1, 2 e 3 (10× cada)
4	50	1, 2, 3, 4 e 5 (10× cada)
5	100	1,2,. . . ,10 (10× cada)

1.2 Experimentos Computacionais

1.2.1 Solver

O Solver utilizado para resolução deste problema foi o Gurobi. Inicialmente este estava apresentando grande dificuldade para encontrar soluções, isso se deve ao fato de que o problema de empacotamento de círculo é NP-hard, este trabalho com equações de segundo grau, o que dificulta a solução. Para tanto, utilizou-se um heurística em espiral para que o solver tivesse uma solução inicial e posteriormente alcançasse a solução ótima. Mesmo utilizando essa estratégia só se alcançou solução ótima na primeira instância. Na segunda instância e na terceira o solver conseguiu chegar em uma solução, porém não é a ótima. Para a quarta e quinta o solver não chegou em nenhum gap no tempo de 30 minutos. A Figura 2 mostra os resultados encontrados.

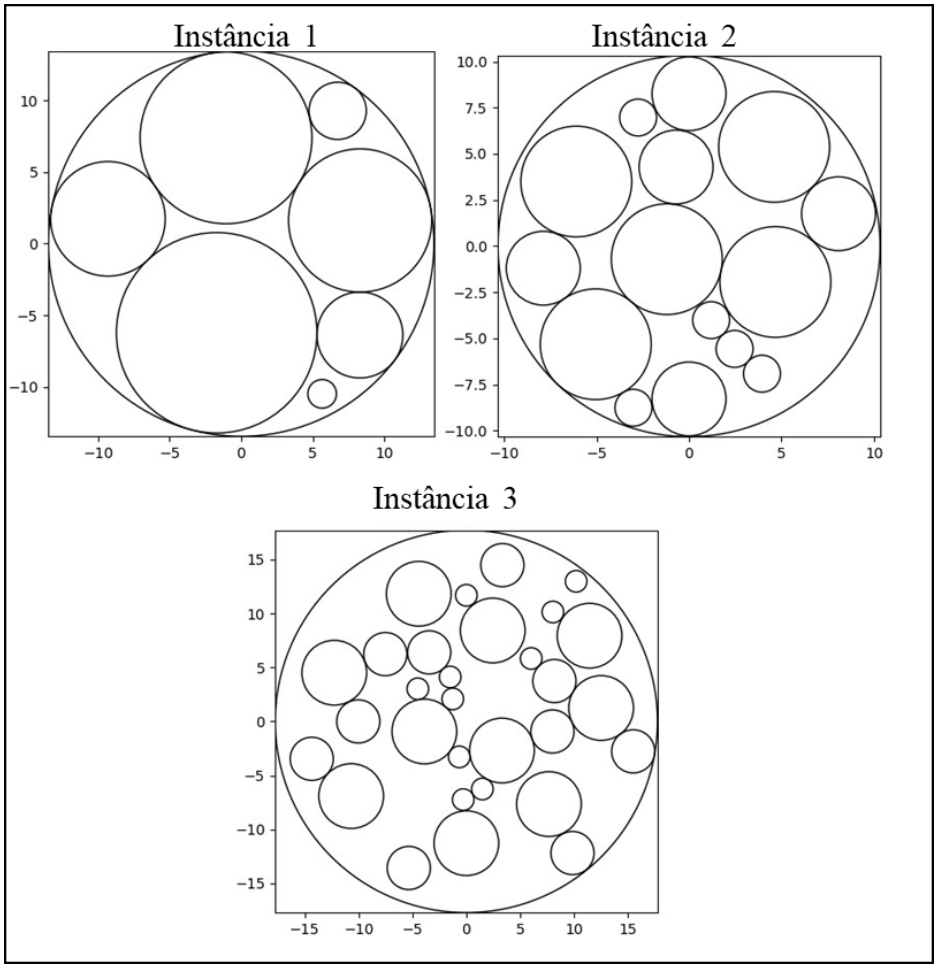

Figura 2: Soluções encontrada pelo solver

1.2.2 Heurística 1

A seguir se apresentam as características e a estruturação do algoritmo GRASP implementado.

1.2.2.1 Descrição da lista restrita de candidatos (RCL "restricted candidate list").

Foram utilizadas duas listas restritas, uma para escolha dos pacotes circulares a serem posicionados e outra para escolha da posição, desta maneira era dado preferência aos maiores pacotes primeiros e as posições mais próximas ao centro do contêiner.

1.2.2.2 Heurística construtiva.

Para a etapa construtiva o pacote era posicionado a uma distância (raio) equivalente ao raio do contêiner somado ao raio do pacote em relação ao centro do contêiner, isso era feito para uma gama de ângulos, onde era aplicado um incremento de $\frac{2\pi}{refinamento}$, onde o refinamento é a quantidade de ângulos analisados, adicionalmente era somado ao ângulo um fator aleatório entre 0 e o incremento, evitando assim o empilhamento. Naturalmente após a posição era realizada uma etapa de aproximação que reduzia a distância ao centro (raio) em 0,1 por incremento, parando apenas quando detectasse uma colisão com outro pacote. Para verificação de colisão foi utilizada a verificação de a soma dos raios era menor que a distância entre os corpos.

1.2.2.3 Operador de busca local e o método de busca (first-improving e best-improving).

A busca local implementada é relativamente simples, ela apenas pega a localização do pacote e tenta alterar um pouco os parâmetros de coordenadas de forma a aproximar o pacote do centro do contêiner, isso é feito utilizando uma mudança aleatória nas coordenadas x e y que está condida em uma redução de 0 a 0,5. Naturalmente para o caso first-improving após a primeira melhora a busca é encerrada, enquanto no caso do best-improving é utilizado o melhor resultado, em ambos os casos a busca local se encerra em no máximo 100 tentativas.

1.2.2.4 Resultados

Instancia	Alfa 1	Alfa 2	Refinamento	Busca	Tempo (s)	Raio
5	1	1	200	Best	205,8	69,5
5	0.8	1	200	Best	225,5	71,4
5	1	1	200	First	77,6	70,6
5	0.8	1	200	First	92,5	71,9
4	1	1	200	Best	49,6	27,2
4	0.8	1	200	Best	50,6	27,4
4	1	1	200	First	14,8	27,2
4	0.8	1	200	First	16,7	27,9
3	1	1	200	Best	20,4	14,4
3	0.8	1	200	Best	18,6	14
3	1	1	200	First	4,1	14,2
3	0.8	1	200	First	5,7	14,4
2	1	1	200	Best	3,3	8,2
2	0.8	1	200	Best	1,8	8,5
2	1	1	200	First	0,4	8,3
2	0.8	1	200	First	0,4	8,6
1	1	1	200	Best	1,3	14,6
1	0.8	1	200	Best	2,5	17
1	1	1	200	First	0,4	15,1
1	0.8	1	200	First	0,4	15,7

Como se pode verificar, os melhores resultados foram entregues pela fator alfa 1 = 1 e para a busca local best, a fator alfa 2 foi testado durante a construção do código é foi averiguado que a utilização de outro valor além do 1 piora significativamente o desempenho, para usá-lo de forma adequada seria necessário implementa uma lógica de controle, caso contrário nas etapas finais, onde se deseja garantir a melhor escolha, o software acabaria optando por escolhas piores aumentando assim o raio do contêiner, ademais para garantir uma melhora na resposta, deveria implementar um novo tipo de busca local que movimentasse todos os itens simultaneamente buscando reposiciona-lo de forma a reduzir o raio do contêiner. Para entender melhor essa afirmação basta verificar que há apenas 1 esfera tocando

na restrição do contêiner, dessa maneira seria possível otimizar espaço com a movimentação, pois para definir a circunferência do contêiner são necessários 3 pontos de contato.

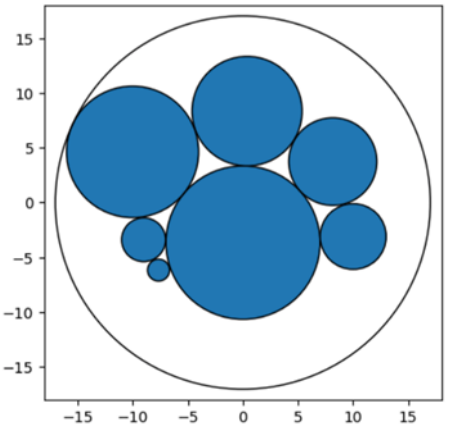

Figura 3: Resultado da instancia 1 para alfa1 = 0,8 e busca best-improving.

As instancias com mais pontos tiveram resultado de preenchimento mais interessantes, como se pode ver a seguir.

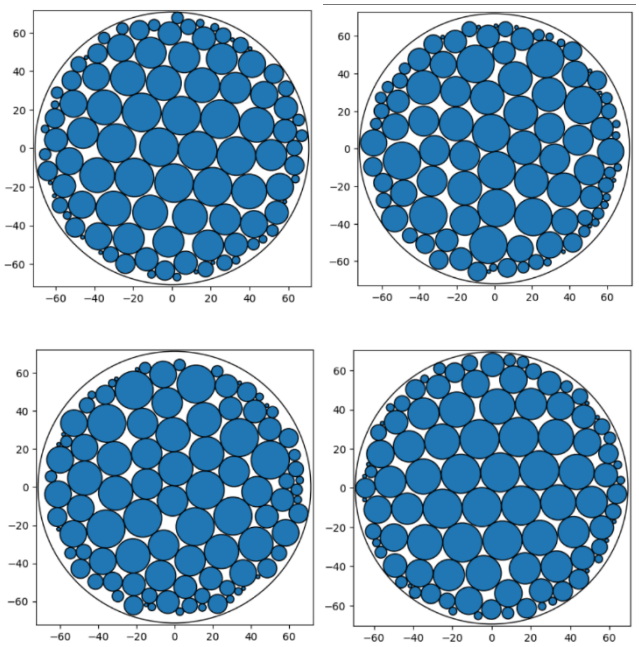

Figura 4: Resultados da instancia 5.

1.2.3 Heurística 2

A Heurística do Mosaico, ao se concentrar no design de quadrados como elemento central, oferece uma abordagem única e eficaz para a resolução de problemas e a organização de conceitos. A implementação dessa heurística, centrada no formato do quadrado, segue uma série de passos distintos:

- Seleção Cuidadosa dos Quadrados: Inicia-se o processo identificando os quadrados como estruturas fundamentais. Cada quadrado representa uma unidade independente e, ao mesmo tempo, é parte integrante de uma estrutura mais ampla.

- Organização Hierárquica: A organização hierárquica dos quadrados é fundamental. Eles podem ser arranjados de maneira a representar níveis diferentes de complexidade ou hierarquia conceitual, permitindo uma compreensão estruturada.

- Incorporação Gradual: Durante esta fase, direcionamos o foco para a incorporação gradual de elementos conceituais dentro de cada quadrado. A lógica subjacente consiste em dispor os quadrados maiores em uma sequência que se assemelha a uma espiral.

- Desenho das circunferências: Após concluir o arranjo dos quadrados, procedemos desenhando círculos em cada elemento. Em seguida, finalizamos o processo criando um círculo externo que envolve todos os círculos menores.

Os resultados obtidos ao aplicar a heurística do mosaico, culminando no desenho das circunferências, revelaram uma integração visual eficaz e uma compreensão refinada dos resultados. Para as instâncias 1 a 5, os raios das circunferências foram respectivamente de 23.00, 11.83, 16.01, 33.01 e 81.09. A estratégia de incorporar círculos dentro de quadrados, seguida pela criação de um círculo externo unificador, resultou em composições visuais que transcendem a mera representação gráfica. A Figura 5 mostra os resultados encontrados.

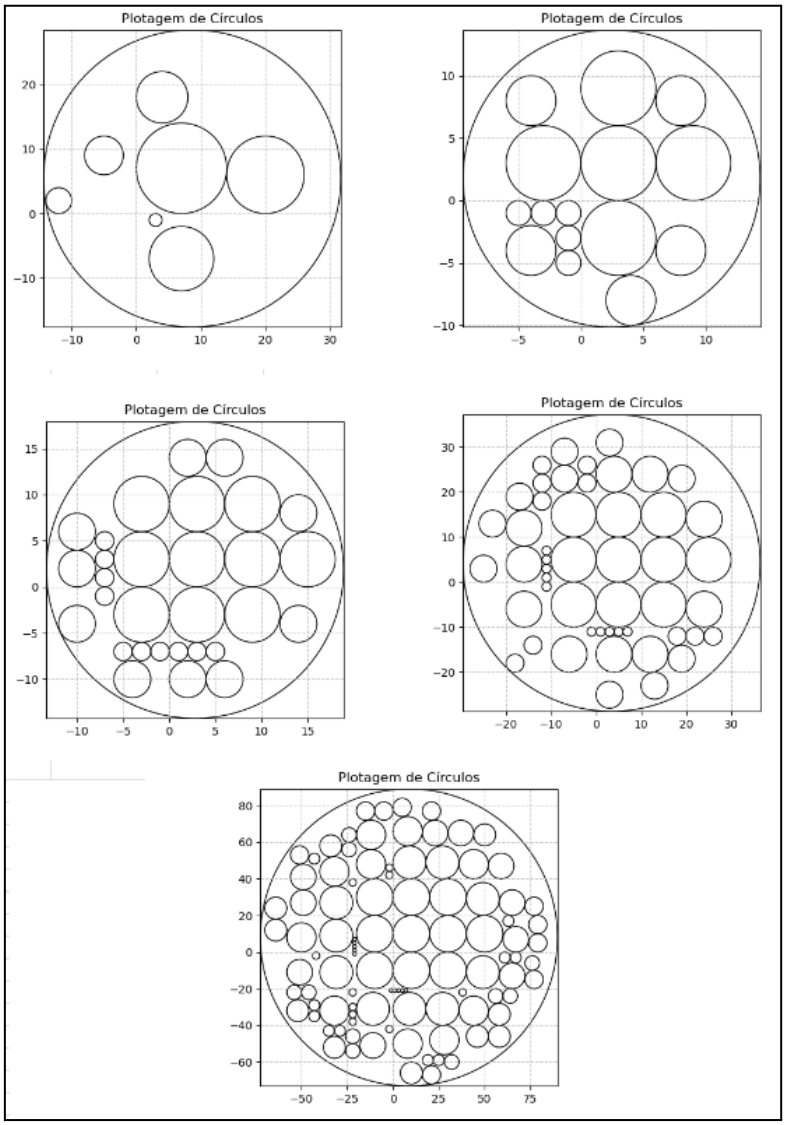

Figura 5: Soluções encontrada pela heurística do mozaico

1.4 Referências Bibliográficas

[1] Beheshti, Zahra, and Siti Mariyam Hj Shamsuddin. "A review of population-based meta-heuristic algorithms." Int. j. adv. soft comput. appl 5.1 (2013): 1-35.

[2] Resende, M. G., & Ribeiro, C. C. (2010). Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. Handbook of metaheuristics, 283-319.

[3] Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. Journal of global optimization, 6, 109-133.

[4] Festa, P., & Resende, M. G. (2009). An annotated bibliography of GRASP–Part II: Applications. International Transactions in Operational Research, 16(2), 131-172.

[5] De Lima, F. C., De Melo, J. D., & Neto, A. D. D. (2008, June). Using the Q-learning algorithm in the constructive phase of the GRASP and reactive GRASP metaheuristics. In 2008 ieee international joint conference on neural networks (IEEE world congress on computational intelligence) (pp. 4169-4176). IEEE.

[6] SZABÓ, Péter Gábor; MARKÓT, Mihály Csaba; CSENDES, Tibor. Global optimization in geometry—circle packing into the square. Essays and Surveys in Global Optimization, p. 233-265, 2005.