Desenvolvimento de uma solução para o problema Maximum Diversity Problem

Douglas de Souza Martins[1], **Ivens Diego Müller**[1], **Tiago Funk**[1]

[1] Departamento de Engenharia de Software
Universidade do Estado de Santa Catarina – Ibirama, SC – Brazil

ivens.muller@edu.udesc.br, martins.douglas.souza@gmail.br, tiagoff.tf@gmail.com

Abstract. *In this document, the problem Maximum Diversity Problem is presented where it consists of the representation of the following situation: A choice of N elements of a set of M elements where the sum of the distances between the elements is maximized. As the reading progresses, a more detailed explanation about the problem, its most frequently used solutions and the solution developed will be the main topic addressed in this text.*

Resumo. *Neste documento, será apresentado o problema Maximum Diversity Problem onde consiste na representação da seguinte situação: Uma escolha de N elementos de um conjunto de M elementos onde a soma das distâncias entre os elementos seja maximizada. Ao andamento da leitura, nota-se uma explicação mais detalhada sobre o problema, suas soluções mais utilizadas e a solução desenvolvida que será o principal tópico abordado neste texto.*

1. Problema

O problema, denominado Maximum Diversity Problem, é um desafio computacional que estuda a obtenção de N elementos dentre um conjunto de M elementos, onde, para cada elemento selecionado, possua a maior soma de suas distâncias. A distância, neste caso, é definida pelos atributos dos indivíduos onde o problema está sendo implantado. Considerando dois elementos pertencentes ao conjunto N, torna-se a distância entre eles calculada através da seguinte fórmula:

$$d_{ij} = \sqrt{\sum_{k=1}^{k} (s_{ik} - s_{jk})^2} \tag{1}$$

Sendo i e j os dois elementos selecionados. Neste caso, a formulação dij representa a distância euclidiana entre estes elementos. Com as distâncias então estabelecidas, pegam-se os valores obtidos e transforma-se o MDP em um problema binário quadrático, onde a variável xi recebe como valor 1 se o elemento em questão foi selecionado ou 0 se caso não for. Então o problema fica definido matematicamente como:

$$\text{Maximize} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j \tag{2}$$

$$\text{Sujeito a } \sum_{i=1}^{n} x_i = m \tag{3}$$
$$x_i = \{0, 1\}, 1 \leq i \leq n$$

2. Heurísticas e Meta-Heurísticas

Heurísticas possuem como objetivo resolver problemas com dificuldade estimada em difícil. Nota-se que, em sua solução estabelecida, não necessariamente é a mais indicada ou mais utilizada que satisfaça uma melhor performance em sua resolução. Heurísticas são conhecidas por chegar a uma solução de forma rápida e simples, por conta disso, não é considerado um método mais utilizado, pois restrições como tempo e processamento, por exemplo, não são levados em conta.

Já as meta-heurísticas são meios utilizados usualmente em problemas envolvendo maximização de uma função cujas as variáveis deste problema possuem certas restrições. Meta-heurísticas são muito usadas na resolução de problemas NP - Difíceis por entregarem uma melhor solução levando em conta fatores como tempo e processamento, diferentemente de heurísticas.

3. Instâncias

As instâncias são os possíveis casos do problema, por exemplo, se o nosso problema for somar dois números, cada par de números vai ser uma nova instância do problema para ser resolvida.

As instâncias que mais são utilizadas são as que estão disponiveis no site da optsicom (http://www.optsicom.es/mdp) e elas serão descritas rápidamente:

- SOM: são 70 matrizes que estão preenchidas com números entre e 9 gerados por um distribuição inteira uniforme.
- GKD: 145 matrizes preenchidas com valores que foram calculados com as distâncias euclidianas de pontos randomicamente gerados com suas coordenadas no intervalo de 0 a 10.
- MDG: Consiste em 100 matrizes com números selecionados randomicamente entre 0 e 10 em uma distribuição uniforme.

4. Estado-da-arte

Existem muitos algoritmos computacionais para resolver este problema, mas há aqueles em que se destacam, onde aqui são denominados de métodos de estado-da-arte. Dentre os existentes, destacam-se:

4.1. Métodos GRASP

O método GRASP é multi-start, onde a cada iteração ele chama um método de construção e após isso, aplica um método de melhoria para encontrar um local ótimo, que seria a solução final para a iteração. [Silva et al. 2004], implementou o método GRASP utilizando os métodos de construção KLD, KLDv2, e MDI, e o BLS como método de melhoria. Abaixo, segue a descrição dos métodos mencionados anteriormente:

Métodos de construção:

- KLD: O método de construção KLD trabalha com estimativas de contribuições entre um elemento e uma solução. Estas estimativas são construídas através da soma das maiores distâncias entre os elementos selecionados e não selecionados. Ao decorrer do algoritmo, forma-se uma lista com estes elementos. Após a lista ser concluída, é escolhida aleatoriamente 1 único elemento que será parte da solução desta construção. O algoritmo só termina quando todos os N elementos forem selecionados.
- KLDv2: O método de construção KLDv2 é apenas uma versão um pouco melhorada do método KLD, onde sua diferença é notada na lista de elementos de maiores distâncias, que, para formá-la, é utilizado um método adaptativo para isso.
- MDI: o método MDI também é um aperfeiçoamento de outro método, só que este vem do método KLDv2. O método KLDv2 calcula sua estimativa através das distâncias de elementos selecionados e não selecionados, já o método MDI obtêm sua estimativa através da soma das distâncias de um elemento com todos os outros elementos não selecionados e os elementos não selecionados com os elementos selecionados.

Método de melhoria:

- BLS (Best improvement local search): Este método de melhoria seria, em curtas palavras, um método de busca local. Esta busca local é realizada por uma heurística que visa a melhor substituição de um elemento selecionado por um não selecionado. O método varre os elementos selecionados e verifica a melhor troca que trará a maior diversidade entre os elementos verificados, ou seja, maior aumento da função objetivo. O algoritmo só finaliza quando todas as iterações forem concluídas e não for detectado uma diversidade maior do que a já realizada.

4.2. Métodos baseados em pesquisa local

- ITS – Iterated Tabu Search: A Busca Tabu é uma meta-heurística que seu princípio básico é realizar uma busca local sempre que o algoritmo encontra um ótimo local. Este método permite realizar movimentos não aprimorados mas não permite voltar atrás para soluções já visitadas, onde este último é barrado pelo uso de memórias denominadas listas tabu, que realiza um histórico das buscas recentes.
- A_VNS – Variable Neighborhood Search: O método VNS é, além de uma metaheurística, um método de otimização que realiza também uma busca local, muito similar ao método ITS descrito anteriormente. Este método explora uma vizinhança iterativa cada vez maior até encontrar um local ótimo para que um algoritmo de melhoria seja implementado ali. Ao localizar, todo o procedimento de expansão é repetido.

4.3. Métodos baseados em população

- G_SS: O método Scatter Search (SS) é um método baseado em população que é muito aplicado a problemas de otimização. Este método também se assemelha a Busca Tabu, já mencionada neste documento. Inclusive, este método pertence ao mesmo autor do algoritmo da Busca Tabu. De modo geral, o SS busca soluções o mais diversificadas possíveis com alta qualidade. De acordo com Jason Brownlee: "A estratégia envolve um processo iterativo, em que uma população de soluções

candidatas diversas e de alta qualidade é particionada em subconjuntos e linearmente recombinada para criar centróides ponderados de vizinhanças baseadas em amostras. Os resultados da recombinação são refinados usando uma heurística embutida e avaliados no contexto do conjunto de referência quanto a serem ou não retidos."

- MA: O algoritmo Memético tem como objetivo hibridar diferentes metaheuristicas para a resolução de um mesmo problema. De modo geral, utilizam uma abordagem baseada em população, onde um conjunto de agentes cooperantes e concorrentes, utilizando busca local, são melhorados individualmente simultaneamente que interagem ocasionalmente.

5. Implementação inicial

A implementação inicial baseia-se, basicamente, em um leitor de instâncias, um calculador da solução, uma instância e um algoritmo que cria uma solução. A implementação inicial foi desenvolvida utilizando a linguagem de programação Java, onde foram criadas as classes InstanceReader (Leitor das instâncias), Solution (Calculador da solução), Instance (Instância) e RandomicAlg (Algoritmo que realiza a solução).

Na classe principal do projeto, é lido uma instância pelo leitor de instâncias e solicitado ao algoritmo que realiza a solução para começar sua execução. Nas subseções seguintes são detalhadas as funcionalidades de cada uma das classes criadas.

5.1. InstanceReader

O leitor de instâncias tem um método chamado getInstance que recebe como parâmetro o caminho onde se encontra o arquivo com a instância e retorna um objeto da classe Instance. Esse leitor, baseado no padrão de formatação do arquivo da instância, cria uma nova instância contendo uma matriz n x n, já preenchida com os valores presentes no arquivo.

5.2. Instance

A classe Instance possui três atributos públicos, para aumentar a performance na hora de acessá-los, sendo que eles foram nomeados de n, m e matrix. Os atributos n e m são dois inteiros, n define a quantidade de elementos e m define a quantidade de elementos que devem ser escolhidos. O atributo matrix é uma matriz de double com tamanho n x n.

5.3. Solution

A classe Solution possui um atributo privado do tipo Instance e dois atributos públicos, sendo que estes atributos são denominados de value e set. O atributo value é utilizado para armazenar o valor total da solução, já o atributo set é um vetor de tamanho n, onde m elementos são preenchidos com 1 e n – m elementos preenchidos com 0. Quem preenche esses valores é o algoritmo que cria uma solução.

Para definir o valor total da solução, é utilizado o método evaluate. Este método percorre a matriz da instância e, verifica se a posição x e a posição y da matriz estão preenchidas com 1 no set e, caso estejam, o método soma no atributo value o valor do elemento na matriz.

5.4. RandomicAlg

A classe RandomicAlg é composta, basicamente, por um método chamado execute que recebe uma instância por parâmetro. Este método é executado por um tempo x e contém um algoritmo que fica criando, randomicamente, soluções da instância e guarda em uma variável qual foi o melhor resultado obtido.

Para ser criada a solução da instância, este método preenche uma lista com tamanho n e adiciona nesta lista números inteiros de 0 até n – 1. Após, é utilizado o método shuffle presente na classe Collections do próprio Java. Este método pega a lista criada e embaralha os dados. Após, é selecionado os números presentes entre as posições 0 e m – 1 desta lista e, no set da Solution, é definido as posições respectivas a estes números como 1.

Ao final do tempo de execução do algoritmo, ele imprime no console o valor da melhor solução obtida e set definido desta solução.

6. Proposta do trabalho

Para proposta do documento será desenvolvido um novo algoritmo que visa retornar a melhor solução do problema em um menor tempo. A forma como será abordado esse método consiste em estudar algoritmos já consagrados na literatura e mudar alguns componentes do algoritmo buscando formas de resolver o problema, mesclando técnicas ainda não utilizadas juntas e avaliar o seu desempenho, levando em conta buscar a melhor solução.

Para fazer isso será modificado o método já conhecido GRASP (nome autor) composto por métodos construtivos e seu método de melhoria da solução. Para tal modificação será subtistuido o seu principal método de melhoria, o BLS, para um tipo de busca local denominada Iterated Tabu Search, aonde espera-se obter a melhor solução utilizando uma forma diferente.

Referências

Silva, G. C., Ochi, L. S., and Martins, S. L. (2004). Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem. In *International Workshop on Experimental and Efficient Algorithms*, pages 498–512. Springer.