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.*

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:

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

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

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.

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.

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:

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.

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.

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.

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.

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.

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.

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.

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.

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.

Desenvolvimento

Para o desenvolvimento do trabalho, adotamos o método GRASP para implementação e obtenção das soluções das instâncias. Conforme já falado anteriormente, o GRASP é executado e cria uma solução inicial a partir de um algoritmo construtivo e, depois desta solução inicial, ele aplica um método de melhoria até que não seja mais possível melhorar a solução.

Para isso, criamos uma classe chamada Grasp que, na sua criação, recebe por parâmetro a instância a ser analisada, uma estratégia de construção, uma estratégia de busca local e um número inteiro para ser definido como o máximo de iterações.

Portanto, tornou-se necessário a implementação de uma classe chamada Greedy e outra chamada LocalSearch. Cada uma destas classes receberá por parâmetro a sua estratégia para realização da solução. Estas estratégias serão explicadas nos tópicos posteriores.

Para a execução do GRASP, sua respectiva classe possui um método chamado execute, que ao final da execução retorna a melhor solução encontrada. O algoritmo implementado no método execute é simples, ele realiza um laço iniciando em 0 e indo até o número máximo de repetições que foi recebido como parâmetro na construção da

classe. Nisso, em cada iteração, o GRASP cria uma nova solução através da classe Greedy e depois aplica uma busca local nesta solução criada, utilizando a classe LocalSearch. Após, é verificado se esta nova solução é a melhor encontrada até o momento e, caso for, é salva em uma variável para retornar no final da execução do método.

Algoritmos gulosos de construção

Abaixo, será descrito quais técnicas e métodos foram utlizados para construir a solução inicial do GRASP. Para o feito, foram implementados 7 algoritmos que montam uma solução inicial cada um da sua maneira.

Algoritmo Aleatório

O Algoritmo Aleatório, desenvolvido e utilizado em várias técnicas de construção de solução, apenas seleciona posições aleatórias para ser construída uma solução. É o algoritmo mais simples dentre os listados nesta sessão. Para cada posição escolhida, ele atribuirá o valor inteiro 1 no vetor de solução.

Algoritmo Guloso

O Algoritmo Guloso é um dos princípais métodos para para resolver problemas de otimização com o objetivo de encontrar um ótimo local. Ele se torna muito atrativo ao trabalhar com problemas NP-completo ou NP-difícil, pois ele obtêm uma solução aproximada da melhor em tempo polinomial. A cada iteração, o algoritmo guloso escolhe a solução que lhe parece mais "apetitosa"que lhe vem pela frente (neste caso, a solução apetitosa seria a que mais atende a F.O). Vale ressaltar, que o Algoritmo Guloso toma decisões com base nas informações que ele possui sem se importar com as consequências em que estas decisões podem acarretar. Ele também nunca volta atrás, suas decisões tomadas são definitivas em cada iteração. A grande vantagem de sua utilização é que ele é muito rápido e eficiente. No projeto do MDP, o Algoritmo Guloso foi apenas uma das estratégias utilizadas para construção de uma solução. O Algoritmo Guloso seleciona o próximo elemento onde ele possui o maior de diferença para selecionar. Diferença, neste contexto, refere-se a soma da diferença do elemento i com relação a todos os elementos que já foram selecionados.

Guloso Ponderado

O Algoritmo Guloso Ponderado funciona de forma semelhante ao próprio Algoritmo Guloso, porém sua diferença está no momento de obter o maior valor da soma das linhas. Para calcular esta soma, o Guloso Ponderado irá percorrer a matriz e na iteração i ele irá calcular um coeficiente, onde é uma soma ponderada aonde os elementos já selecionados possuem um peso x e os não selecionados possuem um peso y.

K Guloso

O Algoritmo K Guloso recebe um valor "m"de elementos das quais ele precisa selecionar "k", onde o mesmo realiza a soma de cada linha da matriz calculada e coloca em uma lista, após finalizar, ele pega os "k"elementos maiores dessa lista e aleatóriamente escolhe um único elemento e atribui o valor booleano 'true' na lista de soluções.

K Guloso Probabilístico

O Algoritmo K Guloso Probabilístico realiza as primeiras operações de um algoritmo K Guloso comum, onde sua diferença está na hora de escolher, na lista de valores resultantes da soma das linhas, um elemento. No Algoritmo K Guloso comum, a escolha é feita aleatoriamente e grava no vetor de solução. Já neste caso, o K Guloso probabilístico soma todos os valores desta lista de somas e atribui-se 100(por cento) como todo o conjunto da lista. A probabilidade ocorre quando, cada valor desta lista é dividido pela soma total e atribuído a porcentagem deste valor em ser escolhido para a solução final. Nota-se que, quanto maior for a soma, maior será sua porcentagem de ser escolhido, e, consequentemente, quanto menor o valor da soma do conjuntos de elementos K que foram escolhidos, menor será a chance de se selecionado para a solução final.

Alfa Guloso

O Algoritmo Alfa Guloso é extremamente semelhante ao K Guloso, toda via ele não irá selecionar k elementos maiores da lista de somas e sim, uma porcentagem de elementos desta lista, ou seja, se o valor de alfa for 0,2, o algoritmo irá selecionar 20(por cento) de elementos desta lista de somas, e dentre este valor, ele irá escolher um elemento aleatório para adicionar ao vetor solução.

Alfa Guloso Probabilístico

Semelhante ao Algoritmo Alfa Guloso e K Guloso Probabilístico, o Algoritmo Alfa Guloso Probabilístico irá selecionar os elementos da lista de somas da mesma forma que o Alfa Guloso faz, toda via, desta lista, ele irá utilizar a mesma técnica que o K Guloso utiliza: soma todos os valores desta lista de soma e atribui uma porcentagem para cada valor com base na soma total realizada anteriormente. Logo, é visível notar que quanto maior o valor desta lista obtida, maior será sua probabilidade de ser selecionado.

Algoritmos de busca local

Como adotamos o GRASP como algoritmo para resolução das instâncias, precisamos de algoritmos de busca local para executarmos a segunda parte do GRASP. Após a construção da solução por meio do algoritmo guloso, a classe GRASP solicita a classe LocalSearch para que aplique uma estratégia de busca local para melhorar a solução inicial criada.

Portanto, a classe LocalSearch inicia sua execução com uma solução inicial e, a partir desta solução, ela aplica uma estratégia de busca de uma nova solução. Durante a

busca, é verificado o par de posições do set da solução que pode ser trocado, realizando um swap dessas posições para verificar se há um incremento no valor total da solução. Para realizar este swap, é necessário que uma posição do set que esteja com o valor 1 vire 0 e a outra posição do set escolhida esteja com valor 0, para que esta possa ser definida com o valor 1. Foram implementadas duas estratégias de busca para incrementar a solução. Sendo elas a busca da primeira melhora e a busca da melhor melhora.

Para as estratégias implementadas, criou-se uma interface chamada SearchStrategy. Esta interface possui um método movement, que recebe uma solução e retorna verdadeiro caso seja possível realizar um swap que incremente a função objetivo da solução recebida. Portanto, como foram implementadas duas estratégias de busca, criou-se as classes FirstImprovementSearch e BestImprovementSearch. Estas classes implementam a interface SearchStrategy e possuem comportamentos diferentes no método movement, que são explicados nas partes 7.2.1 e 7.2.2.

Como tornou-se necessário verificar a possibilidade de swap em uma solução, implementou-se dentro da classe Solution uma função chamada Delta que, passando duas posições do set, a função calcula o valor Delta e retorna ao usuário este valor. O valor delta é a diferença do valor da função objetivo com o swap e o valor da função objetivo atual. Baseando-se nisso, criou-se outro método dentro da classe Solution chamado Swap, este método recebe um posição a ser setada para 0 e outra posição a ser setada para 1 e calcula o valor da função objetivo tendo como base este swap.

Busca da primeira melhora (FirstImprovementSearch)

O algoritmo que realiza a busca da primeira melhora é executado até que ele encontre um par de posições do set para realizar o swap que o valor delta deste swap seja maior do que 0. Caso não tenha um par de posições possíveis para realizar o swap, a busca da primeira melhora retorna o valor false, fazendo com que termine a execução da busca local e passe para a próxima iteração do GRASP.

A execução do algoritmo é bem simples, são dois laços encadeados no partindo de 0 até o tamanho do set – 1. O primeiro laço verifica as posições do set que estão definidas com o valor 1 e, caso a posição atual esteja com este valor, ele passa para o laço 2. O laço 2 verifica as posições que estão com o valor 0 e, caso esteja, ele chama o método Delta da classe Solution para retornar o delta do swap destas posições. Caso o delta retornado for maior do que 0, o método movement retorna true.

Busca da melhor melhora (BestImprovementSearch)

O algoritmo que realiza a busca da melhor melhora é executado de maneira parecida ao algoritmo do tópico anterior. A diferença dele é que ele verifica todas as possibilidades de swap presentes na solução.

Portanto, a execução dele se torna um pouco mais demorada em relação ao algoritmo anterior. Esta demora se da por que o algoritmo não para quando o delta retornado é maior do que 0, o algoritmo guarda o maior delta encontrado e as posições do swap para que, no final da execução desta estratégia, ela tenha guardado o melhor delta e o melhor

par de posições para swap. Portanto, no fim de todas as iterações dos laços, a solução retorna true caso o maior delta encontrado for maior do que zero.

Parametrizações

Para obtermos flexibilidade na tentativa de encontrar a solução para cada instância, o sistema foi parametrizado por um todo. Existindo parâmetros para definição do algoritmo guloso que será escolhido, dos parâmetros específicos para cada algoritmo guloso, do total de repetições do GRASP, do algoritmo de busca local, entre outros. Para isso, também se tornou necessário utilizar uma ferramenta chamada IRACE. O IRACE é uma ferramenta que automatiza a configuração dos parâmetros e define a melhor configuração possível baseado em seus cenários.

Parâmetros

Iterated Race (IRACE)

Resultados

Instancia	Estado da Arte	Melhor Solucao	Desvio melhor	Tempo do melhor	Solucao Media	Desvio medio	Tempo Medio
GKD-c_14_n500_m50.txt	19458,57	19458,56	0,00	124691,00	19458,56	0,00	126301,00
GKD-c_13_n500_m50.txt	19366,70	19366,70	0,00	93902,00	19366,70	0,00	94942,00
GKD-c_16_n500_m50.txt	19680,21	19680,20	0,00	70668,00	19680,20	0,00	70680,00
GKD-c_20_n500_m50.txt	19604,84	19604,84	0,00	70339,00	19604,84	0,00	70116,67
GKD-c_11_n500_m50.txt	19587,13	19587,12	0,01	66079,00	19587,12	0,01	68165,33
GKD-c_10_n500_m50.txt	19703,35	19703,34	0,01	73767,00	19703,34	0,01	76296,00
GKD-c_15_n500_m50.txt	19422,15	19422,15	0,00	96716,00	19422,15	0,00	97645,00
GKD-c_9_n500_m50.txt	19221,63	19221,63	0,01	59323,00	19221,63	0,01	58875,33
GKD-c_7_n500_m50.txt	19534,31	19534,30	0,00	148178,00	19534,30	0,00	149318,00
GKD-c_12_n500_m50.txt	19360,24	19360,22	0,01	75766,00	19360,22	0,01	76073,00
GKD-c_8_n500_m50.txt	19487,32	19487,32	-0,00	107690,00	19487,32	-0,00	107324,33
GKD-c_3_n500_m50.txt	19547,21	19547,21	0,00	77452,00	19547,21	0,00	77146,67
GKD-c_2_n500_m50.txt	19701,54	19701,53	0,00	96632,00	19701,53	0,00	97236,67
GKD-c_4_n500_m50.txt	19596,47	19596,47	0,00	100841,00	19596,47	0,00	101543,00
GKD-c_1_n500_m50.txt	19485,19	19485,19	0,00	97306,00	19485,19	0,00	96128,67
GKD-c_6_n500_m50.txt	19421,55	19421,54	0,01	101596,00	19421,54	0,01	104029,00
GKD-c_5_n500_m50.txt	19602,63	19602,62	0,00	82683,00	19602,62	0,00	79958,00
GKD-c_19_n500_m50.txt	19477,33	19477,33	0,00	127843,00	19477,33	0,00	127346,33
GKD-c_17_n500_m50.txt	19331,39	19331,39	0,00	159932,00	19331,39	0,00	161716,67
GKD-c_18_n500_m50.txt	19461,39	19461,39	-0,00	78869,00	19461,39	-0,00	78671,00

Tabela 1. Legenda da tabela

Instancia	Estado da Arte	Melhor Solucao	Desvio melhor	Tempo do melhor	Solucao Media	Desvio medio	Tempo Medio
MDG-a_19_n500_m50.txt	7755,41	7750,58	4,83	223237,00	7734,89	20,52	233847,33
MDG-a_17_n500_m50.txt	7787,20	7772,46	14,74	199563,00	7761,93	25,27	207970,67
MDG-a_16_n500_m50.txt	7792,77	7792,77	0,00	242437,00	7786,91	5,86	258341,00
MDG-a_15_n500_m50.txt	7736,84	7704,25	32,59	221484,00	7693,99	42,85	232471,00
MDG-a_20_n500_m50.txt	7733,86	7718,49	15,37	213885,00	7705,20	28,66	228737,33
MDG-a_14_n500_m50.txt	7795,63	7783,23	12,40	226886,00	7782,96	12,67	228254,33
MDG-a_12_n500_m50.txt	7757,65	7739,20	18,45	224454,00	7735,13	22,52	234351,00
MDG-a_1_n500_m50.txt	7833,83	7784,04	49,79	195916,00	7781,99	51,84	214042,33
MDG-a_9_n500_m50.txt	7770,07	7748,18	21,89	202436,00	7741,53	28,54	204402,00
MDG-a_13_n500_m50.txt	7798,43	7798,43	0,00	236371,00	7796,80	1,63	247366,33
MDG-a_5_n500_m50.txt	7755,23	7698,13	57,10	235734,00	7696,26	58,97	248355,67
MDG-a_4_n500_m50.txt	7770,24	7753,46	16,78	218740,00	7750,00	20,24	228734,33
MDG-a_6_n500_m50.txt	7773,71	7751,74	21,97	237794,00	7751,59	22,12	244845,67
MDG-a_2_n500_m50.txt	7771,66	7741,73	29,93	264664,00	7726,29	45,37	240304,67
MDG-a_10_n500_m50.txt	7780,35	7747,87	32,48	233158,00	7739,47	40,88	228990,33
MDG-a_3_n500_m50.txt	7759,36	7745,07	14,29	203485,00	7743,67	15,69	210809,00
MDG-a_8_n500_m50.txt	7750,88	7734,39	16,49	261054,00	7730,39	20,49	259201,67
MDG-a_11_n500_m50.txt	7770,95	7754,50	16,45	237291,00	7743,27	27,68	248323,33
MDG-a_7_n500_m50.txt	7771,73	7738,06	33,67	244324,00	7730,33	41,40	247846,00
MDG-a_18_n500_m50.txt	7756,26	7744,88	11,38	237221,00	7743,95	12,32	245582,67

Tabela 2. Legenda da tabela

Instancia	Estado da Arte	Melhor Solucao	Desvio melhor	Tempo do melhor	Solucao Media	Desvio medio	Tempo Medio
MDG-b_7_n500_m50.txt	777232,88	775266,73	1966,14	232250,00	775165,77	2067,10	247532,00
MDG-b_8_n500_m50.txt	779168,75	776186,22	2982,53	274838,00	774494,86	4673,89	251756,00
MDG-b_6_n500_m50.txt	775153,69	773475,09	1678,60	195159,00	773475,09	1678,60	211269,33
MDG-b_9_n500_m50.txt	774802,19	772877,29	1924,90	242018,00	772844,18	1958,01	244637,67
MDG-b_3_n500_m50.txt	776768,44	776768,17	0,27	229535,00	776743,69	24,75	237679,33
MDG-b_18_n500_m50.txt	775850,75	771524,45	4326,30	209790,00	771499,25	4351,50	218579,33
MDG-b_4_n500_m50.txt	775394,63	773559,81	1834,81	235716,00	772863,38	2531,24	241772,67
MDG-b_16_n500_m50.txt	775436,25	773973,15	1463,10	249245,00	772580,75	2855,50	235057,00
MDG-b_17_n500_m50.txt	776619,13	771819,59	4799,54	208223,00	771361,33	5257,80	218236,67
MDG-b_5_n500_m50.txt	775611,06	775610,96	0,10	228910,00	775610,96	0,10	243599,00
MDG-b_13_n500_m50.txt	780191,88	773554,50	6637,37	233661,00	772354,16	7837,72	214877,33
MDG-b_12_n500_m50.txt	775492,94	771632,10	3860,84	229546,00	771189,82	4303,12	240882,67
MDG-b_15_n500_m50.txt	780300,38	776507,02	3793,35	230577,00	775853,00	4447,37	235695,00
MDG-b_10_n500_m50.txt	774961,31	773632,04	1329,27	231119,00	773509,75	1451,56	235880,00
MDG-b_14_n500_m50.txt	782232,75	781603,28	629,47	240119,00	781485,22	747,53	249845,67
MDG-b_19_n500_m50.txt	778802,94	774470,13	4332,81	215584,00	774470,13	4332,81	222088,33
MDG-b_1_n500_m50.txt	778030,63	776272,68	1757,94	260177,00	774992,07	3038,55	265891,67
MDG-b_2_n500_m50.txt	779963,69	779963,54	0,15	201191,00	779963,54	0,15	219621,67
MDG-b_11_n500_m50.txt	777468,88	774672,88	2795,99	236698,00	774354,88	3113,99	239171,33
MDG-b_20_n500_m50.txt	778644,81	778644,65	0,16	210624,00	778644,65	0,16	218290,67

Tabela 3. Legenda da tabela

Instancia	Estado da Arte	Melhor Solucao	Desvio melhor	Tempo do melhor	Solucao Media	Desvio medio	Tempo Medio
SOM-b_18_n500_m100.txt	26258,00	26246,00	12,00	865377,00	26242,33	15,67	952339,00
SOM-b_2_n100_m20.txt	1195,00	1195,00	0,00	1334,00	1195,00	0,00	1518,33
SOM-b_5_n200_m20.txt	1247,00	1243,00	4,00	4976,00	1241,00	6,00	5270,33
SOM-b_13_n400_m40.txt	4658,00	4651,00	7,00	68625,00	4649,33	8,67	70753,00
SOM-b_8_n200_m80.txt	16225,00	16225,00	0,00	72915,00	16225,00	0,00	75195,00
SOM-b_20_n500_m200.txt	97344,00	97328,00	16,00	2197360,00	97311,67	32,33	2197525,00
SOM-b_7_n200_m60.txt	9437,00	9418,00	19,00	35301,00	9418,00	19,00	36221,00
SOM-b_16_n400_m160.txt	62487,00	62426,00	61,00	1212954,00	62414,67	72,33	1316271,33
SOM-b_11_n300_m90.txt	20743,00	20734,00	9,00	206296,00	20716,00	27,00	201126,00
SOM-b_6_n200_m40.txt	4450,00	4448,00	2,00	14045,00	4446,67	3,33	14826,67
SOM-b_4_n100_m40.txt	4142,00	4139,00	3,00	3289,00	4139,00	3,00	3485,33
SOM-b_14_n400_m80.txt	16956,00	16941,00	15,00	282806,00	16940,67	15,33	301870,33
SOM-b_10_n300_m60.txt	9689,00	9684,00	5,00	89795,00	9684,00	5,00	90765,00
SOM-b_17_n500_m50.txt	7141,00	7112,00	29,00	251809,00	7098,33	42,67	232045,00
SOM-b_9_n300_m30.txt	2694,00	2676,00	18,00	25748,00	2674,67	19,33	24603,00
SOM-b_12_n300_m120.txt	35881,00	35878,00	3,00	520383,00	35878,00	3,00	519035,67
SOM-b_19_n500_m150.txt	56572,00	56572,00	0,00	2191423,00	56572,00	0,00	2195482,67
SOM-b_1_n100_m10.txt	333,00	333,00	0,00	554,00	333,00	0,00	775,67
SOM-b_15_n400_m120.txt	36317,00	36285,00	32,00	847166,00	36283,33	33,67	885019,00
SOM-b_3_n100_m30.txt	2457,00	2457,00	0,00	2737,00	2457,00	0,00	2837,33

Tabela 4. Legenda da tabela

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.