Análise de Heurísticas GRASP para o Problema da Diversidade Máxima

Geiza Cristina da Silva

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre.

Área de concentração: Otimização e Inteligência Artificial.

Orientadores: Luiz Satoru Ochi
Simone de Lima Martins

Niterói, Novembro de 2004.

Análise de Heurísticas GRASP para o Problema da Diversidade Máxima

Geiza Cristina da Silva

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre.

Área de concentração: Otimização e Inteligência Artificial.

Aprovada por:

Prof. Luiz Satoru Ochi / IC-UFF (Presidente)

Prof. Celso da Cruz Carneiro Ribeiro / IC-UFF

Profa. Nair Maria Maia de Abreu / UFRJ

Prof. Paulo Oswaldo Boaventura Netto / UFRJ

Profa. Simone de Lima Martins / IC-UFF

Niterói, Novembro de 2004.

Aos meus queridos pais e a minha sobrinha Thuane.

Agradecimentos

Aos meus orientadores Luiz Satoru Ochi e Simone de Lima Martins por direcionarem tão bem o meu trabalho. Agradeço a confiança, a paciência, o apoio nos momentos mais embaraçados, a força e incentivo na hora certa.

Aos meus pais, meu porto seguro, pela vida e pelo amor. Não há forma de agradecê-los por tudo que fizeram e fazem por mim.

À todos os meus amigos e colegas da UFF que compartilharam a construção deste trabalho, em especial, Eyder e Vinícius na implementação, Stênio, Adriana, Jacques e Adria.

Às minhas amigas pessoais, Idalmis, Cris e Leila e as meninas que compartilharam comigo o mesmo teto e momentos bons e ruins, Dani, Daniele, Lu e Melba.

À CAPES pelo financiamento dos meus estudos.

E, à Deus, Jeová Nissi, "Deus de vitória".

Resumo da Tese apresentada à UFF como parte dos requisitos necessários para a obtenção do grau de Mestre em Computação (M.Sc.)

Análise de Heurísticas GRASP para o Problema da Diversidade Máxima

Geiza Cristina da Silva

Novembro - 2004

Orientadores: Luiz Satoru Ochi
Simone de Lima Martins
Programa: Pós-Graduação em Computação

O Problema da Diversidade Máxima (PDM) corresponde à seleção de elementos a partir de uma coleção maior de forma tal que os elementos selecionados possuam a maior diversidade possível entre eles. Aplicações deste problema podem ser encontradas em várias áreas como, por exemplo, em recursos humanos, identificando indivíduos com características menos similares ou em biologia, quando se deseja obter espécies de maior diversidade. Excluindo-se casos triviais, o PDM é classificado como um problema da classe NP-difícil. Desta forma, o uso de métodos aproximativos ou heurísticos, ou seja, métodos que são capazes de obter soluções não necessariamente ótimas, mas com alguma proximidade do valor ótimo tornam-se bastante atraentes.

Neste trabalho são propostos métodos de construção e busca local que, combinados, são usados como base em diferentes propostas de heurísticas GRASP (*Greedy Randomized Adaptive Search Procedure*). Uma extensa bateria de testes é realizada e os algoritmos propostos são analisados comparando-se com outros dois algoritmos descritos na literatura. Os resultados mostram que as heurísticas propostas fornecem bons resultados na resolução de instâncias do PDM.

Abstract of Thesis presented to UFF as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

Analysis of GRASP Heuristics for the Maximum Diversity Problem

Geiza Cristina da Silva

November - 2004

Advisors: Luiz Satoru Ochi
Simone de Lima Martins
Department: Computer Science

The Maximum Diversity Problem (MDP) consists of selecting elements from some larger collection such that the selected elements have the most possible diversity among them. There are many applications that can be solved using the resolution of this problem, such as in human resources, identifying people with less similar characteristics or in biology, when it is desired to identify more diverse species. MDP belongs to the class of NP-hard problems. Thus, the use of approximation or heuristics methods which are capable to get solutions close to the optimum cost becomes quite attractive.

In this work we propose construction and local search methods which are used for the implementation of different GRASP (Greedy Randomized Adaptive Search Procedure) heuristics. An experimental study is carried out and the projected algorithms are compared with two others algorithms described in literature. Results show that good results are obtained using the proposed heuristics to solve MDP instances.

Palavras-chave

1. Problema da Diversidade Máxima
2. Metaheurística GRASP
3. Heurísticas

Glossário

GRASP	:	*Greedy Randomized Adaptive Search Procedure*;
PDM	:	Problema da Diversidade Máxima (*Maximum Diversity Problem*);
LRC	:	Lista Restrita de Candidatos (*Restricted Candidate List*).

Sumário

Resumo **iv**

Abstract **v**

Glossário **vii**

1 Introdução **2**

2 O Problema da Diversidade Máxima **6**

2.1 Descrição do Problema 6

2.2 Exemplo 7

2.3 Aplicações e Trabalhos Relacionados 9

2.4 Formulação Matemática 13

3 A Metaheurística GRASP **16**

4 Descrição dos Algoritmos de Ghosh e Andrade **21**

4.1 Algoritmo Proposto por Ghosh 21

4.2 Algoritmo Proposto por Andrade 24

5 Algoritmos Propostos **28**

5.1 Métodos de Construção 28

5.1.1 Heurística Aleatorizada das K Maiores Distâncias (HA_KMD) 29

5.1.2 Heurística Aleatorizada das K Maiores Distâncias Versão 2 (HA_KMD_v2) 33

5.1.3 Heurística Aleatorizada de Inserção mais Distante (HA_IMD) 34

5.2 Método de Busca Local 37

5.3 Algoritmos GRASP Propostos para o PDM 39

6 Testes e Resultados Computacionais 40

6.1 Geração dos Problemas Testes 40

6.2 Testes 41

6.2.1 Testes com Conjunto de Instâncias 1 42

6.2.2 Testes com Conjunto de Instâncias 2 48

Análise de Soluções 48

Análise de Tempo Computacional 54

6.2.3 Testes com Conjunto de Instâncias 3 59

Análise de Soluções 59

Análise de Tempo Computacional 61

Análise Probabilística 64

6.3 Considerações 78

7 Conclusões e Trabalhos Futuros 80

A Publicações Associadas **82**

B Tabelas **83**

B.1 Conjunto de Instâncias 2 83

B.2 Conjunto de Instâncias 3 87

Referências Bibliográficas **89**

Lista de Figuras

3.1 Pseudo-código GRASP 18

3.2 Pseudo-código para a fase de construção do GRASP 18

3.3 Pseudo-código para a fase de busca local do GRASP 19

4.1 Pseudo-código para a fase de construção do algoritmo de Ghosh . . . 23

4.2 Pseudo-código para a fase de busca local do algoritmo de Ghosh . . . 25

4.3 Pseudo-código para a fase de construção do algoritmo de Andrade . . 27

5.1 Pseudo-código de `HA_KMD` 32

5.2 Pseudo-código de `HA_IMD` 36

5.3 Pseudo-código da `BLP` 38

6.1 Gráfico de tempo de execução dos algoritmos 49

6.2 Comparação entre os algoritmos GRASP para a instância $n = 100$ e $m = 40$ com alvo igual a 4141 69

6.3 Comparação entre os algoritmos GRASP para a instância $n = 100$ e $m = 40$ com alvo igual a 4142 69

6.4 Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 20$ com alvo igual a 1240 70

6.5 Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 20$ com alvo igual a 1244 70

6.6 Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 40$ com alvo igual a 4443 71

6.7 Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 40$ com alvo igual a 4447 71

6.8 Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 60$ com alvo igual a 9425 72

6.9 Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 60$ com alvo igual a 9435 72

6.10 Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 80$ com alvo igual a 16171 73

6.11 Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 80$ com alvo igual a 16210 73

6.12 Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 30$ com alvo igual a 2666 74

6.13 Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 30$ com alvo igual a 2686 74

6.14 Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 60$ com alvo igual a 9652 75

6.15 Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 60$ com alvo igual a 9676 75

6.16 Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 90$ com alvo igual a 20640 76

6.17 Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 90$ com alvo igual a 20691 76

6.18 Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 120$ com alvo igual a 35855 77

6.19 Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 120$ com alvo igual a 35873 77

Lista de Tabelas

2.1 Tabela de classificação por sexo 8

2.2 Tabela de classificação por faixa etária 8

2.3 Tabela de classificação por grau de escolaridade 8

2.4 Tabela de características 9

2.5 Matriz de diversidade D 10

5.1 Valores de k_i em $B1$ 30

5.2 Blocos de iterações de $B2$ 30

5.3 Valores de α em $B1$ 35

5.4 Algoritmos GRASP propostos para o PDM 39

6.1 Custos das soluções obtidas pelo algoritmo exato e pelas heurísticas. O símbolo (*) significa maior valor de solução obtida pelo exato em 36.000 segundos 44

6.2 Diferença entre a solução exata e as soluções obtidas pelas heurísticas. O símbolo (**) significa a diferença entre o maior custo de solução obtida pelo exato em 36.000 segundos e os demais algoritmos 45

6.3 Tempos de processamento (em segundos) para algoritmo exato e (médio) das heurísticas. O símbolo (***) tempo de processamento algoritmo exato limitado em 36.000 segundos 46

6.4 Diferença entre média dos valores nas instâncias do Grupo A 50

6.5 Diferença entre médias dos valores nas instâncias do Grupo B 51

6.6 Diferença entre médias dos valores nas instâncias do Grupo C 52

6.7 Número de vezes em que a melhor solução é encontrada por cada algoritmo 53

6.8 Tempo de computação (em segundos) dos algoritmos para instâncias do Grupo A 56

6.9 Tempo de computação (em segundos) dos algoritmos para instâncias do Grupo B 57

6.10 Tempo de computação (em segundos) dos algoritmos para instâncias do Grupo C 58

6.11 Diferenças entre médias dos valores das soluções obtidas nas instâncias do Conjunto 3 60

6.12 Número de vezes em que a melhor solução é encontrada por cada algoritmo 61

6.13 Tempo de computação (em segundos) dos algoritmos para instâncias do Conjunto 3 63

6.14 Alvos usados no teste probabilístico 65

6.15 Melhores valores de soluções obtidas nas instâncias do Conjunto 3 . . 67

6.16 Tempo (em segundos) para que probabilidades sejam alcançadas . . . 68

B.1 Valores dos custos das soluções obtidas nas instâncias do Grupo A . . 84

B.2 Média dos valores das soluções obtidas nas instâncias do Grupo B . . 85

B.3 Média dos valores das soluções obtidas nas instâncias do Grupo C . . 86

B.4 Médias dos valores das soluções obtidas nas instâncias do Conjunto 3 88

Capítulo 1

Introdução

A área de otimização combinatória trata de caracterizar soluções e desenvolver algoritmos para resolução de problemas que envolvem um grande número (finito) de possíveis soluções. Dados um conjunto normalmente finito de soluções X e uma função $f : X \rightarrow \mathbb{R}$, no caso de maximização, o objetivo é encontrar uma solução ótima $x^* \in X$ tal que $f(x^*) \geq f(x), \forall x \in X$.

Estes problemas têm sido de grande interesse, pelas suas mais variadas aplicações de diferentes áreas. De maneira geral, na prática, os problemas de otimização são caracterizados por grandes espaços de soluções dificultando, com isso, a obtenção de uma solução de custo ótimo através de um algoritmo exato em um tempo computacional suportável.

Por este motivo, métodos que gerem soluções aproximadas são cada vez mais aplicados na área. Um algoritmo aproximativo ou heurístico é um método de obtenção de soluções para um determinado problema que não garante que as soluções obtidas sejam necessariamente ótimas, mas que tenham proximidade do ótimo em tempo de computação viável. Dentre os mais diversos métodos heurísticos estão as metaheurísticas, que têm se mostrado bastante eficientes em se tratando dos fatores qualidade da solução e tempo de computação.

Uma metaheurística é um método que guia outras heurísticas a fim de encontrar soluções factíveis para problemas particulares. Dentre as metaheurísticas mais conhecidas destacamos os Algoritmos Evolutivos [9] e o seu representante mais popular, o Algoritmo Genético(AG) [16, 31], *Simulated Annealing* [19], TS (*Tabu Search*) [11, 15, 17], GRASP (*Greedy Randomized Adaptative Search Procedure*) [7] e VNS (*Variable Neighborhood Search*) [26] entre outras.

Neste trabalho, aplicamos a metaheurística GRASP ao Problema da Diversidade Máxima (PDM). O PDM é um problema de maximização da área de otimização combinatória que tem como objetivo selecionar, a partir de um conjunto de elementos, um subconjunto com um número pré-estabelecido de elementos que tenham a maior diversidade possível entre eles. Por se tratar de um problema pertencente à classe NP-Difícil [21], a aplicação de algoritmos que gerem boas soluções em tempo computacional aceitável torna-se altamente desejável.

Este trabalho foi motivado pelo estudo realizado em [3] e em [10], pois ambos aplicaram a metaheurística GRASP para obter soluções aproximadas para o PDM. O objetivo deste trabalho é, portanto, desenvolver um novo algoritmo que seja capaz de encontrar soluções de melhor qualidade para o PDM utilizando os conceitos desta metaheurística.

A fim de comprovar a qualidade das soluções, instâncias do problema foram geradas e testes realizados sobre os algoritmos implementados em [3] e em [10] e os propostos para este trabalho.

Esta dissertação está organizada da seguinte maneira:

O Capítulo 2 apresenta o Problema da Diversidade Máxima, algumas aplicações, seus trabalhos relacionados e, por fim, as formulações matemáticas do problema.

O Capítulo 3 faz uma revisão sobre os conceitos associados à metaheurística GRASP.

No Capítulo 4 são apresentados os algoritmos desenvolvidos para o PDM por Ghosh [10] e Andrade [3].

O Capítulo 5 descreve as heurísticas de construção e busca local desenvolvidas nesta dissertação. Apresenta-se, também, os algoritmos GRASP formados pela combinação destas.

No Capítulo 6 os algoritmos da literatura e os que foram propostos são avaliados e analisados. Para isto, apresentam-se a geração dos problemas testes, os tipos de testes realizados e, por fim, os resultados alcançados.

Encerrando o trabalho, o Capítulo 7 traz conclusões e apresenta sugestões para trabalhos vindouros.

Capítulo 2

O Problema da Diversidade Máxima

Neste capítulo as características do PDM são apresentadas. Na seção 2.1 o problema é descrito. Na seção 2.2 aborda-se o problema através de um exemplo. Na seção 2.3 algumas aplicações e trabalhos relacionados ao problema são discutidos. Por fim, as formulações matemáticas existentes na literatura são apresentadas na seção 2.4.

2.1 Descrição do Problema

O Problema da Diversidade Máxima consiste em, a partir de um conjunto contendo n elementos, selecionar um subconjunto com um número pré-determinado destes elementos que possuam as *mais diversas características* possíveis entre si. Desta forma, para definir o problema, considere o conjunto de índices $N = \{1, 2, ..., n\}$, onde n é o número de elementos em N, e:

t, o número de atributos que caracterizam os elementos de N;

a_{ik}, o valor do atributo k do elemento i, $k \in \{1, ..., t\}$ e $i \in N$;

$d(i, j)$, o índice de diversidade entre dois elementos distintos i e j de N com os respectivos valores de atributos $(a_{i1}, a_{i2}, ..., a_{it})$ e $(a_{j1}, a_{j2}, ..., a_{jt})$.

O *índice de diversidade* representa o grau de diferença existente entre os atributos dos elementos i e j, que pode ser medido, por exemplo, pela distância entre eles calculada usando uma determinada métrica. De acordo com a área de aplicação, uma métrica pode refletir melhor o problema que outra [21]. Se, por exemplo, for utilizada a distância euclidiana para o cálculo do índice de diversidade, $d(i,j)$ é computado conforme descrito na equação 2.1.

$$d(i,j) = \sqrt{\sum_{k=1}^{t} (a_{ik} - a_{jk})^2} \tag{2.1}$$

Seja $D[n \times n]$ uma matriz que armazena os índices de diversidade entre cada par de elementos de N, preenchida da seguinte maneira: $d(i,j) = d(j,i)$ e $d(i,i) = 0$, $\forall (i,j) \in N$. O problema da diversidade máxima (PDM) tem como objetivo selecionar um subconjunto $M \subset N$ com m elementos que tenham a maior diversidade entre eles. O valor da diversidade de M é calculado pela equação 2.2.

$$div(M) = \sum_{i,j \in M, i<j} d(i,j) \tag{2.2}$$

2.2 Exemplo

Um exemplo para melhor compreensão do problema é mostrado a seguir.

Considere uma base de dados de uma organização fictícia que contém informações sobre oito pessoas que se ofereceram para prestar algum tipo de trabalho voluntário. A direção da organização decidiu formar um grupo de três pessoas para fazer visitas à população de uma área carente e foi consenso que esse grupo deveria conter pessoas com características mais diversas possíveis com o objetivo de que o grupo como um todo possa estar hábil a enfrentar as situações vindouras nas visitas.

Dentre as várias informações constantes desta base de dados, algumas foram consideradas importantes para avaliar a diversidade entre os voluntários desse grupo, tais como: *sexo, grau de escolaridade* e *faixa etária*.

Para determinar os elementos mais diversos entre si a partir desta base de dados, é necessário um mapeamento das características (valores qualitativos) para valores numéricos (valor quantitativo). Apresentamos esse mapeamento nas tabelas 2.1, 2.2 e 2.3.

Sexo	**Classificação**
Masculino	1
Feminino	2

Tabela 2.1: Tabela de classificação por sexo

A característica '*sexo*' (tabela 2.1) tem dois possíveis atributos, feminino e masculino e a cada um deles é, respectivamente, associado um valor inteiro 1 e 2. Já '*idade*' (tabela 2.2) é classificada por faixas etárias, com valores variando de 1 a 7. Da mesma maneira, '*escolaridade*' é dividida em graus com valores variando de 1 a 6 (tabela 2.3).

Faixa Etária	**Classificação**
18-25	1
26-32	2
33-39	3
40-47	4
47-54	5
54-60	6
+ de 60	7

Tabela 2.2: Tabela de classificação por faixa etária

Grau de Escolaridade	**Classificação**
Analfabeto	1
Básico	2
Médio	3
Secundário	4
Superior	5
Pós-Graduação	6

Tabela 2.3: Tabela de classificação por grau de escolaridade

De acordo com o mapeamento proposto, a base de dados passa a ser formada de informações quantitativas como apresentadas na tabela 2.4. Cada linha desta tabela indica o conjunto de características de cada elemento da base de dados.

Indivíduo	Nome	Sexo	Idade	Escolaridade
1	Marcelo	1	2	3
2	Gisele	2	3	6
3	Tiago	1	1	4
4	Mauro	1	4	2
5	Valéria	2	4	3
6	Célia	2	6	6
7	Maria	2	7	1
8	Nelson	1	5	5

Tabela 2.4: Tabela de características

Suponha que a organização queira determinar os três voluntários com atributos mais diversos entre si para formar o grupo de visitação. Para resolver o problema, os dados são adaptados à definição do PDM. Assim, tem-se N = conjunto de pessoas contidas na base de dados e $|N| = n = 8$. É necessário então determinar o índice de diversidade entre cada integrante da base de dados para construir a matriz de diversidades D. Neste caso, para obter o índice de diversidade, a distância euclidiana foi usada. Desta forma, os índices são calculados a partir da tabela 2.4 utilizando a equação 2.1. Por exemplo, o índice de diversidade entre Marcelo (1) e Gisele (2) é $\sqrt{5}$, pois temos: $\sqrt{(1-2)^2 + (2-3)^2 + (3-6)^2} = \sqrt{5} \approx 2.236$ que é o valor do elemento d(1,2) de D. Prosseguindo com os cálculos de diversidade de cada par de elementos é formada a matriz D apresentada na tabela 2.5.

O passo final é identificar o subconjunto de três elementos que possui a maior soma dos índices de diversidade a partir da matriz D. Observe que mesmo para um problema de tamanho reduzido como este a solução não é óbvia, e uma maneira para determiná-la seria listar todas as possibilidades de conjuntos com três elementos para verificar qual deles retorna a maior diversidade. Para este exemplo, uma solução ótima é dada pelos elementos 2, 6 e 7 com diversidade total igual a 18,487.

2.3 Aplicações e Trabalhos Relacionados

O Problema da Diversidade Máxima pode ser aplicado nas mais diversas áreas como na administração de recursos e formação de grupos de projetos [35, 6], biolo-

	1	2	3	4	5	6	7	8
1	0.000	2.236	1.414	2.236	2.236	5.099	5.547	3.605
2	2.336	0.000	3.000	4.242	3.162	5.477	6.403	2.249
3	1.414	3.000	0.000	3.605	3.316	5.477	6.782	4.123
4	2.236	4.242	3.605	0.000	1.414	4.582	3.741	3.162
5	2.236	3.162	3.316	1.414	0.000	3.605	3.605	2.249
6	5.099	5.477	5.477	4.582	3.605	0.000	5.099	1.732
7	5.547	6.403	6.782	3.741	3.605	5.099	0.000	4.582
8	3.605	2.249	4.123	3.162	2.249	1.732	4.582	0.000

Tabela 2.5: Matriz de diversidade D

gia [1], preservação ecológica [25, 29, 34] e outras áreas [36].

De acordo com a bibliografia pesquisada, o primeiro trabalho a ter a maximização da diversidade como objetivo específico data de 1977 e foi escrito por Glover, Hersh e McMillan [12]. Neste trabalho foi proposta uma heurística de aproximação e esta foi testada com dados de um programa de seleção de recursos genéticos.

Em 1993, Kuo, Glover e Dhir [21] apresentaram duas definições diferentes para o PDM. Nesta dissertação é abordada a definição MAXSUM cujo objetivo é maximizar a soma das diversidades de um conjunto de elementos selecionados a partir de uma coleção maior. Os autores apresentaram ainda formulações para o PDM e provaram que MAXSUM é NP-Difícil. A seção 2.4 apresenta as formulações propostas em [21].

Mais tarde, em 1994, os mesmos autores propuseram em [14] uma série de formulações do PDM como problema de programação inteira com o objetivo de maximizar diversidade biológica. Para ilustrar, o modelo foi aplicado à preservação de uma família de aves. Foi também apresentada uma medida de biodiversidade alternativa mostrando que diferentes medidas de diversidade podem gerar diferentes políticas de conservação ecológica.

Esta dissertação é baseada no trabalho de Ghosh [10] que propôs um algoritmo GRASP para cada o PDM. Para comprovar a qualidade das heurísticas foi implementado um algoritmo exato, criadas classes de testes com populações de no máximo 40 elementos onde deveriam ser encontradas soluções para subconjuntos

de tamanho 20% e 40% do número de elementos. Os resultados mostraram que o GRASP proposto se mostrou eficiente para os pequenos problemas representados pelas instâncias.

Em [1] Agrafiotis explorou o problema de selecionar subconjuntos de componentes apropriados para síntese química e avaliação biológica. No trabalho introduziu duas medidas para quantificar a diversidade e apresentou um algoritmo para o problema baseado em *simulated annealing*. O método foi testado usando bases de dados formadas a partir de métodos estatísticos. Os resultados foram visualizados a partir do algoritmo de mapeamento não-linear de Sammon [32]. O autor cita que o método encontra resultados satisfatórios para os testes realizados.

Em [5] foram apresentadas duas heurísticas para o PDM, uma relaxação lagrangeana e uma heurística gulosa. Para avaliar os resultados obtidos foi implementado um algoritmo *branch and bound* e, desta forma, os autores concluíram que a heurística gulosa apresentou bons resultados para os testes executados.

Em 1998, Weitz e Lakshminarayanan [35] apresentaram cinco heurísticas para o PDM. Este trabalho foi realizado para resolver uma aplicação que é criar grupos de projetos em uma classe de estudantes, de forma tal que esses estudantes sejam inseridos em um ambiente diverso, baseado em critérios determinados pela instituição, como por exemplo, nacionalidade, idade e formação. As heurísticas foram testadas em um conjunto de dados reais e foi implementado também um algoritmo exato. Os testes apontaram a heurística LCW (Método '*Lotfi-Cerveny-Weitz*') [35] como sendo a melhor para a solução do problema.

Glover e Kuo propuseram em [13] quatro heurísticas sendo duas delas construtivas e as outras destrutivas, onde a partir de uma solução heurística com e elementos ocorre uma eliminação progressiva até que se tenha uma solução viá vel com m elementos. A eficiência dessas heurísticas foram testadas através de comparação com os resultados de um método exato e para isso foram criadas diversas instâncias com até 30 elementos. Foi citado pelos autores que as heurísticas propostas obtiveram em média resultados próximos aos do algoritmo exato (a no máximo 2% do ótimo), com um tempo de computação muito menor.

Hassin, Rubinstein e Tamir [18] tratam de um problema que considera um grafo G=(V,E) completo, onde a cada aresta de E é associado um peso e $|V| = n$. Sendo $p \in \{2, ..., n\}$ e $k \in \{1, ..., n/p\}$ devem-se obter k conjuntos disjuntos, com p vértices cada um de tal forma que a soma dos pesos das arestas de cada subconjunto seja máxima. Este problema possui diversas variações e uma delas, é o PDM, no caso particular em que $k = 1$. O artigo apresenta um algoritmo aproximativo [27] com desempenho garantido de 50% do ótimo para $k \in \{1, ..., n/p\}$ e um outro com mesmo resultado para $k = 1$, porém neste caso, com um tempo computacional mais baixo.

Em [4] foi introduzido um novo modelo para tratar o problema de diversidade de grupos de trabalho que permitiu reduzí-lo ao problema de fluxo em redes e desenvolver um algoritmo exato para resolvê-lo. O modelo foi aplicado a uma base de dados real obtida em um curso de pós-graduação.

Uma heurística baseada em busca tabu foi desenvolvida para encontrar a diversidade máxima em [20]. Os autores testaram o método proposto gerando aleatoriamente instâncias de problemas com conjuntos de tamanho que variam de 100 a 1000 elementos onde desejava-se obter subconjuntos de elementos mais diversos com tamanho entre 10% a 30% do número de elementos. Os autores citam que testes foram realizados mas os resultados não foram comparados com qualquer outro método existente.

Além do trabalho de Ghosh [10], os resultados obtidos neste trabalho são comparados com os de Andrade [3] que propôs outro GRASP para solução do PDM. Nesta proposta, desenvolveu-se para a fase de construção uma heurística construtiva e incorporou-se a busca local encontrada em [10]. O autor gerou uma bateria de testes onde o tamanho da população variou de 10 a 250 elementos. Para avaliação dos resultados produzidos por seu algoritmo, foi implementado um algoritmo exaustivo para as instâncias menores (até 50 elementos) e implementou-se o algoritmo de Ghosh para a comparação dos resultados. É citado que o GRASP proposto neste trabalho mostrou-se ser mais eficiente na qualidade das soluções exigindo tempo de computação ligeiramente menor que os obtidos pelo algoritmo de Ghosh.

2.4 Formulação Matemática

Conforme exposto na seção 2.3, Kuo, Glover e Dhir apresentaram em [21] três formulações matemáticas para o PDM focando o critério MAXSUM. As duas primeiras (F1 e F2) utilizam, respectivamente, uma função objetivo quadrática e uma função objetivo linear. A terceira é também um modelo linear, com a diferença que utiliza uma transformação de variáveis em F2 para obter uma formulação com menos variáveis e menos restrições. Esta seção descreve as formulações F1 e F2.

Formulação 1 (F1)

Seja $x_i, \forall i \in N$, uma variável binária igual a 1, se e somente se, o elemento i estiver na solução e 0, caso contrário e $d(i,j)$ a diversidade entre i e j, $i,j \in N$.

O PDM pode ser descrito como um problema de programação quadrática inteira da seguinte forma:

maximizar

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d(i,j) x_i x_j, \tag{2.3}$$

sujeito a:

$$\sum_{i=1}^{n} x_i = m, \tag{2.4}$$

$$x_i \in \{0,1\}, \forall i = 1, ..., n. \tag{2.5}$$

Na formulação descrita em 2.3 a 2.5, a função objetivo quadrática (2.3) tem como meta maximizar a soma das diversidades de um subconjunto com m elementos de N. A igualdade (2.4) exige que exatamente m elementos estejam em uma solução viável e (2.5) indica que as variáveis do problema devem ser binárias.

Formulação 2 (F2)

A segunda formulação proposta em [21] baseia-se em F1 linearizada, ou seja, descreve o PDM como um problema de programação linear inteira. Considere $Q = \{(i, j) : i, j \in N, i < j\}$ e y_{ij} variáveis de decisão que tornam-se iguais a 1 se os elementos i e j fizerem parte da solução e 0, caso contrário. Desta forma, o PDM é descrito como:

maximizar

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d(i, j) y_{ij}, \tag{2.6}$$

sujeito a:

$$\sum_{i=1}^{n} x_i = m, \tag{2.7}$$

$$x_i + x_j - y_{ij} \leq 1, \quad \forall (i, j) \in Q, \tag{2.8}$$

$$-x_i + y_{ij} \leq 0, \quad \forall (i, j) \in Q, \tag{2.9}$$

$$-x_j + y_{ij} \leq 0, \quad \forall (i, j) \in Q, \tag{2.10}$$

$$y_{ij} \in \{0, 1\}, \quad \forall (i, j) \in Q, \tag{2.11}$$

$$x_i \in \{0, 1\}, \forall i \in N. \tag{2.12}$$

Nesta formulação, a função objetivo (2.6) deve maximizar a soma das diversidades de m elementos de N. A igualdade (2.7) exige que exatamente m elementos pertençam a uma solução viável. Nas restrições (2.8) pode-se observar que a variável

binária y_{ij} assume valor 1 sempre que x_i e x_j receberem tal valor e y_{ij} assume valor 0 se x_i e x_j forem ambas iguais a 0 ou se x_i ou x_j (somente uma delas) for igual a 1. As restrições (2.9) exigem que no caso em que x_i pertencer à solução então deverá existir algum j tal que $y_{ij} = 1$, $i, j \in N$. Da mesma forma que em (2.9), as restrições (2.10) exigem que se x_j pertencer à solução então deverá existir algum i tal que $y_{ij} = 1$, $i, j \in N$. As restrições (2.11) e (2.12) representam as variáveis binárias do problema, sendo $y_{ij} = 1$ se x_i e x_j estiverem na solução e 0 caso contrário e cada x_i assumindo valor 1 se pertencer à solução e 0, caso contrário.

Capítulo 3

A Metaheurística GRASP

Criada por Feo e Resende, a metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*) [7] é um processo iterativo no qual cada iteração é constituída de duas fases: uma fase de construção, e outra, de busca local.

Na fase de construção uma solução viável é criada e na etapa de busca local a solução passa por um processo de melhoria, onde a vizinhança da solução construída é investigada. As iterações se realizam enquanto um critério de parada não for satisfeito. A melhor solução encontrada ao final da execução de todas as iterações do algoritmo é a solução final.

A figura 3.1 mostra um pseudo-código para o GRASP. A linha 2 corresponde à entrada do problema que carrega os dados inerentes ao problema em questão. Nas linhas 3-7 são executadas as `MAX_ITER` iterações do GRASP. A linha 4 corresponde à fase de construção e a linha 5, a de busca local. Na linha 6, é feita a atualização da melhor solução, caso a solução encontrada nesta iteração tenha sido a melhor encontrada até o momento.

Na fase de construção, uma solução é iterativamente construída acrescentando-se a esta um elemento por vez. Cada inserção de um novo elemento é feita através da escolha aleatória em uma lista restrita de candidatos (LRC). Considere para a

construção da LRC que o problema tratado nesta dissertação é de maximização de uma função objetivo f. A LRC é composta de candidatos à solução avaliados de acordo com o benefício associado a sua inclusão na solução parcial através de uma função gulosa g. Os elementos que participarão da LRC podem ser limitados de duas formas: pelo número de elementos ou pela qualidade dos elementos.

Na primeira forma, que é utilizada nesta dissertação, os elementos candidatos à solução são ordenados em ordem decrescente do benefício segundo a função g e os p primeiros elementos são incluídos na LRC. O valor de p é definido na equação 3.1.

$$p = 1 + \alpha(a - 1) \tag{3.1}$$

Onde a é o número total de candidatos e α é um parâmetro que tem valores definidos no intervalo [0,1]. Note que se $\alpha = 0$, um algoritmo totalmente guloso estará sendo executado, já que só é possível a escolha de um único elemento a ser inserido na solução. Por outro lado, se $\alpha = 1$, a lista conterá todos os possíveis candidatos, o que resultará em um algoritmo extremamente aleatório.

Na segunda forma, considere e_{min} um elemento de menor incremento à solução parcial de acordo com g e e_{max}, um elemento de maior incremento. Os elementos candidatos à solução cujo valor retornado pela função g está no intervalo $[(1-\alpha)(e_{min}-e_{max})+e_{max}, e_{max}]$ são inseridos na LRC. Novamente α é um valor no intervalo [0,1]. Desta forma, se $\alpha = 1$, o algoritmo será absolutamente guloso, já que só elementos de valor e_{max} poderão ser inseridos na solução e se $\alpha = 0$, a lista conterá todos os possíveis candidatos, o que resultará em um algoritmo completamente aleatório.

É perceptível que a escolha do valor de α é um ponto importante na heurística GRASP. A influência desta escolha na qualidade da solução construída é estudada em [7]. O fator probabilístico do método é caracterizado pela escolha aleatória de um dos candidatos da LRC.

```
1. proc grasp(MAX_ITER)
2.    Carrega_Instancia_Entrada()
3.    para k=1 ate MAX_ITER faca
4.     Solucao = Constroi_Solucao ()
5.     Solucao = Busca_Local (Solucao)
6.     Atualiza_Solucao (Solucao, Melhor_Solucao)
7.    fim para
8.    retorna (Melhor_Solucao)
9. fim  grasp
```

Figura 3.1: Pseudo-código GRASP

Na figura 3.2 é mostrado o pseudo-código da rotina de construção de uma solução. Na linha 2 uma solução é inicializada. As linhas 3-8 mostram o laço onde uma solução é construída iterativamente, elemento a elemento. Na linha 4 é criada a lista restrita de candidatos e na linha 5, um elemento s é selecionado aleatoriamente a partir da LRC. Na linha 6 s é incorporado à solução parcial e na linha 7 a função gulosa é adaptada a fim de refletir as mudanças causadas pela inserção de s na solução parcial. Na linha 9 a solução já completamente construída é retornada.

```
1. proc Constroi_Solucao()
2.   Solucao = {}
3.   enquanto Solucao nao esta completa faca
4.    Construa LRC
5.    Selecione aleatoriamente um elemento s da LRC
6.    Solucao = Solucao ∪ {s}
7.    Adapte a funcao gulosa(s)
8.   fim enquanto
9.   retorna (Solucao)
10.fim  Constroi_Solucao
```

Figura 3.2: Pseudo-código para a fase de construção do GRASP

A busca local é uma fase de tentativa de melhoria da solução obtida na etapa de construção (*Solucao*). Nesta fase, percorre-se a vizinhança da solução corrente buscando uma solução de melhor qualidade. Por exemplo, em um problema onde o objetivo é a maximização de uma função f, entende-se que uma solução t é melhor que *Solucao* se o *f(t)* > *f(Solucao)*. Se não existir uma melhor solução na vizinhança,

a solução corrente é considerada um ótimo local. A busca termina quando um ótimo local for atingido.

O pseudo-código da busca local é apresentado na figura 3.3. As linhas 2-8 mostram o laço de melhoria da solução que continuará enquanto *Solucao* puder ser melhorada, ou seja, enquanto *Solucao* não for um ótimo local. Nas linha 3-7 ocorre a tentativa de encontrar uma melhor solução t na vizinhança de *Solucao*. Na linha 5, t passa a ser a solução corrente, caso seja melhor que *Solucao*. Na linha 9 a solução é retornada.

Ainda a respeito do procedimento de busca local, pode-se acrescentar que existem duas maneiras bastante utilizadas para implementá-lo, melhor escolha (*best improving*) e primeira melhora (*first improving*). Na primeira versão, toda a vizinhança é percorrida e a solução corrente é substituída pela melhor solução vizinha encontrada. Na segunda, a solução corrente é trocada cada vez que uma melhor solução é encontrada.

```
1.  proc Busca_Local(Solucao)
2.    enquanto Solucao nao é um ótimo local faca
3.      para toda solução t na vizinhança de Solucao faca
4.        se f(t) > f(Solucao) entao
5.          Solucao = t
6.        fim se
7.      fim para
8.    fim enquanto
9.    retorna (Solucao)
10. fim  Busca_Local
```

Figura 3.3: Pseudo-código para a fase de busca local do GRASP

Vários conceitos, alguns já utilizados em outras metaheurísticas, têm sido incorporados por diversos pesquisadores como alternativa para aumentar a eficiência do GRASP. Dentre estes podem ser citados reconexão por caminhos [22], memória de longo prazo [8], GRASP Paralelo [28], GRASP reativo [30], entre outros. O conceito de GRASP reativo é aplicado nos algoritmos propostos nesta dissertação e será descrito sucintamente a seguir.

Como citado anteriormente, a composição da lista restrita de candidatos é determinada a partir do parâmetro α. Esse parâmetro pode afetar a qualidade das soluções geradas na fase de construção. Prais e Ribeiro propuseram em [30] um método chamado GRASP reativo cuja característica principal é o auto ajuste do valor de α de acordo com a qualidade das soluções previamente encontradas.

Neste procedimento, um valor de α é selecionado aleatoriamente em um conjunto $A = \{\alpha_1, \alpha_2, ..., \alpha_m\}$ contendo m possíveis valores de α. Desta forma, já que se tem vários valores de α ao invés de um único, é possível construir diferentes listas restritas de candidatos. A cada elemento $\alpha_i \in A$ é associada uma probabilidade p_i, $i = 1, \ldots, m$.

Inicialmente, tem-se $p_i = 1/m$, o que corresponde a uma distribuição uniforme. Periodicamente, ou seja, a cada bloco de iterações, a distribuição de probabilidade é atualizada, utilizando as informações coletadas das soluções obtidas nas iterações realizadas anteriormente. Na aquisição destas informações, várias estratégias podem ser usadas. Em uma delas, o valor médio das soluções obtidas com cada valor de α é utilizado. Desta forma, no próximo bloco de iterações, o valor de α com o qual se obteve melhores soluções receberá uma probabilidade maior e, conseqüentemente, com mais chance de ser escolhido, poderá ser utilizado mais freqüentemente neste bloco de iterações.

Capítulo 4

Descrição dos Algoritmos de Ghosh e Andrade

Ghosh [10] e Andrade [3] empregaram a heurística GRASP afim de obter soluções aproximadas para o PDM. Nas seções 4.1 e 4.2, onde suas abordagens são detalhadas, considere N um conjunto de índices com n elementos, $d(i, j)$ o índice de diversidade entre dois elementos i e j e M o subconjunto de N que contém m elementos mais diversos entre si, como já definido no Capítulo 2.

4.1 Algoritmo Proposto por Ghosh

Neste algoritmo, para construir uma solução completa para o PDM, tem-se m soluções parciais. Cada solução parcial será denotada por M_{c-1} e conterá $c-1$ elementos ($1 \leq c \leq m$).

Fase de Construção

Para se gerar uma solução M_c a partir de M_{c-1} são calculados $\Delta z_L(i)$ e $\Delta z_U(i), \forall i \in N - M_{c-1}$. Δz_L é considerado um *limite inferior* de diversidade correspondente à soma das diversidades de um elemento i não pertencente à solução aos elementos

já pertencentes à solução parcial M_{c-1} acrescida da soma das distâncias de i aos $m-c$ elementos que apresentam as *menores* distâncias de i. Equivalentemente, Δz_U representa um *limite superior* de diversidade correspondendo à soma das distâncias de i aos elementos já pertencentes à solução parcial acrescida da soma das distâncias de i aos $m-c$ elementos que apresentam as *maiores* distâncias de i.

Os cálculos de $\Delta z_L(i)$ e $\Delta z_U(i)$ são detalhados nas equações 4.1 e 4.2.

$$\Delta z_L(i) = \sum_{j \in M_{c-1}} d(i,j) + \sum_{n-m+1 \leq r \leq n-c} d_i^r(Q_{ic}) \tag{4.1}$$

$$\Delta z_U(i) = \sum_{j \in M_{c-1}} d(i,j) + \sum_{1 \leq r \leq m-c} d_i^r(Q_{ic}) \tag{4.2}$$

Onde:

- $\sum_{j \in M_{c-1}} d(i,j)$ corresponde à soma das distâncias do elemento i a todos os elementos já incluídos na solução parcial;
- Q_{ic} é uma lista ordenada que contém os elementos pertencentes a $N - M_{c-1} - \{i\}$, ou seja, os elementos que ainda não estão na solução excluindo o candidato à solução i, ordenados pelas distâncias $d(i,j), j \in N - M_{c-1} - \{i\}$;
- $d_i^r(Q_{ic})$ corresponde a r-ésima maior distância em Q_{ic}.

O fator aleatório da heurística está na escolha randômica de um número u no intervalo $(0,1)$. A partir de sua determinação, calcula-se $\Delta z'(i) = (1-u)\Delta z_L(i) + u\Delta z_U(i)$. O elemento i que retorna o maior valor de $\Delta z'$ é incluído na solução parcial. Este procedimento é repetido até que seja obtida uma solução completa com m elementos.

O procedimento de construção descrito é ilustrado na figura 4.1. Na linha 2 do algoritmo uma solução vazia é criada. Nas linhas 3-18, tem-se o laço de construção

de uma solução que permanecerá em execução até que uma solução com m elementos tenha sido criada. No laço apresentado nas linhas 4-8, serão considerados todos os índices dos elementos de N que não fazem parte de *Solucao*. Para cada um deles, nas linhas 5 e 6 são computados $DeltazL$ e $DeltazU$, respectivamente. Na linha 7 é calculado $Deltaz'$. As linhas 9-15 verificam dentre todos os elementos candidatos à solução o indivíduo que tem o maior $Deltaz'$. Na linha 16, o elemento que apresenta maior valor de $Deltaz'$ é incorporado à solução. O retorno de *Solucao* já completa ocorre na linha 18.

```
1.  proc Construcao_JBG()
2.    Solucao = {}
3.    para c = 1 ate m faca
4.      para cada i ∈ N \ Solucao faca
5.        Calcule DeltazL(i)
6.        Calcule DeltazU(i)
7.        Calcule Deltaz'(i)
8.      fim para
9.      Max_Deltaz' = 0
10.     para cada i ∈ N \ Solucao faca
11.       se Deltaz'(i) > Max_Deltaz' entao
12.         Max_Deltaz' = Deltaz'(i)
13.         Max_I = i
14.       fim se
15.     fim para
16.     Solucao = Solucao ∪ {Max_I}
17.   fim para
18.   retorna Solucao
19. fim Construcao_JBG
```

Figura 4.1: Pseudo-código para a fase de construção do algoritmo de Ghosh

Fase de Busca Local

A partir do conjunto solução completo M_m, Ghosh [10] propõe uma busca local exaustiva onde a cada passo, verifica-se a melhoria associada à troca de cada elemento $i \in M$ por cada elemento $j \in N - M$. Essa melhoria é rotulada como $\Delta z(i, j)$ e é calculada da seguinte forma:

$$\Delta z(i,j) = \sum_{u \in M - \{i\}} d(j,u) - d(i,u) \tag{4.3}$$

O procedimento de busca local descrito é ilustrado na figura 4.2. Na linha 2, a variável *DeltaZ* é inicializada com a constante `INFINITO`. Nela será guardado o benefício associado à troca de um elemento i pertencente à *Solucao* por um outro elemento j não pertencente a ela. Nas linhas 3-20, temos o laço de busca na vizinhança de *Solucao* que é executado enquanto houver alguma melhoria na solução, ou seja, enquanto *DeltaZ* é maior que 0. Na linha 4, é inicializada a variável Max_DeltaZ onde será guardado o maior valor de *DeltaZ* encontrado em todas as combinações possíveis de pares (i,j) de *Solucao*. Na linha 7 é calculado *DeltaZ* para cada elemento i pertencente à solução e cada elemento j pertencente à N - *Solucao*. Nas linhas 9, 10 e 11 são guardados, respectivamente, o maior *DeltaZ*, os elementos i e j que correspondem ao maior *DeltaZ* encontrado. Nas linhas 16 e 17, a *Solucao* é alterada somente se Max_DeltaZ é maior que zero, ou seja, se foi encontrado um par (i,j) cuja inclusão em *Solucao* retorne um valor maior que aquele que já se tinha. Neste caso, na linha 18, *DeltaZ* é atualizado. Na linha 21, após o término da busca, a solução obtida é retornada.

4.2 Algoritmo Proposto por Andrade

Em seu trabalho, Andrade [3] considera dois conjuntos de valores, um chamado SD que corresponde à soma dos índices de diversidade de um determinado elemento $i \in N$ com relação aos demais elementos j, $j \in N - \{i\}$ e um outro, denominado MD que contém a média dos índices de diversidade de um elemento $i \in N$ em relação aos demais elementos j, $j \in N - \{i\}$. Os cálculos de SD e MD são detalhados nas equações 4.4 e 4.5. Estes valores são utilizados durante a fase de construção a fim de tentar identificar os melhores elementos a serem incluídos em cada solução parcial.

```
1. proc BLG (Solucao)
2.   DeltaZ = INFINITO
3.   enquanto DeltaZ > 0 faca
4.    Max_DeltaZ = 0
5.    para cada i ∈ Solucao faca
6.     para cada  j ∈ N \ Solucao faca
7.      DeltaZ = Calcula_DeltaZ(i,j)
8.      se (DeltaZ > Max_DeltaZ) entao
9.       Max_DeltaZ = DeltaZ
10.      Max_I = i
11.      Max_J = j
12.     fim se
13.    fim para
14.   fim para
15.   se (Max_DeltaZ > 0) entao
16.    Solucao = Solucao - {Max_I}
17.    Solucao = Solucao ∪ {Max_J}
18.    DeltaZ = Max_DeltaZ
19.   fim se
20.  fim enquanto
21.  retorna Solucao
22.fim Busca_JBG
```

Figura 4.2: Pseudo-código para a fase de busca local do algoritmo de Ghosh

$$SD(i) = \sum_{j \in N - \{i\}} d(i,j) \tag{4.4}$$

$$MD(i) = \frac{\sum_{j \in N - \{i\}} d(i,j)}{n} \tag{4.5}$$

Fase de Construção

O primeiro elemento da solução é selecionado aleatoriamente entre os m elementos de maior valor em SD. Para construir cada solução parcial, M_{c-1} com $2 \leq c \leq m$, primeiramente é calculado, para cada $i \in N - M_{c-1}$, o valor de $SDS(i)$ que corresponde à média da soma dos índices de diversidade entre i e j, $j \in M_{c-1}$, como pode ser visto na equação 4.6.

$$SDS(i) = \frac{\sum_{j \in M_{c-1}} d(i,j)}{c-1} \tag{4.6}$$

Em seguida, $\Delta z(i)$ é obtido a partir de $SDS(i)$ segundo o seguinte critério: se a solução está parcialmente formada e faltam apenas poucos elementos para completá-la é interessante a inclusão de elementos que possuam $SDS(i)$ alto, caso contrário, é aconselhável avaliar a contribuição que a inserção do elemento trará a solução utilizando $MD(i)$. O cálculo de $\Delta z(i)$ é detalhado na equação 4.7.

$$\Delta z(i) = \begin{cases} SDS(i) & \text{, se } SDS(i) > MD(i) \text{ e } c > \frac{m}{2} \\ \\ \frac{SDS(i)+MD(i)}{2} & \text{, caso contrário} \end{cases} \tag{4.7}$$

Após terem sido calculados todos os $\Delta z(i)'s$, eles são ordenados de forma crescente e para os m elementos de maiores $\Delta z(i)$ calcula-se o valor D que resulta da diferença entre o $\Delta z(i)_{\max}$ que corresponde ao maior $\Delta z(i)$ e $\Delta z(i)_{\min}$, o menor deles, dividido por $m-1$. O próximo passo é criar uma LRC com os Q maiores elementos cuja diferença entre seu Δz e o Δz do próximo elemento seja menor que o valor de D. Um elemento é selecionado aleatoriamente a partir da LRC. O cálculo de D pode ser visto na equação 4.8.

$$D = \frac{\Delta z(i)_{\max} - \Delta z(i)_{\min}}{m-1} \tag{4.8}$$

O procedimento de construção descrito é ilustrado na figura 4.3. A linha 2 inicializa uma solução vazia. A escolha do primeiro elemento da solução é feita na linha 3 e na linha 4, esse elemento é adicionado à *Solucao*. No trecho das linhas 5-16 uma solução completa é construída. Nas linhas 6-8 e 9-11 são calculados, respectivamente, SDS e $DeltaZ$. Na linha 12, D é calculado. A lista restrita de candidatos é criada na linha 13 e na linha 14, um elemento s é escolhido aleatoriamente desta. Na linha 15, s é incorporado à solução. Na linha 17, a solução construída é retornada.

```
1.  proc Construcao_PMA (SD, MD)
2.    Solucao = {}
3.    escolha aleatoriamente s entre os m elementos com maior SD
4.    Solucao = Solucao ∪ {s}
5.    para c = 1 ate  m  faca
6.      para cada  i ∈ N \ Solucao faca
7.        SDS(i) = Calcula_SDS(i)
8.      fim para
9.      para cada  i ∈ N \ Solucao faca
10.       DeltaZ(i) = Calcula_DeltaZ(i)
11.     fim para
12.     D = Calcula_D (DeltaZ)
13.     LRC = Cria_LRC(DeltaZ, D)
14.     escolha aleatoriamente um elemento s da LRC
15.     Solucao = Solucao ∪ {s}
16.   fim enquanto
17.   retorna Solucao
18. fim Construcao_PMA
```

Figura 4.3: Pseudo-código para a fase de construção do algoritmo de Andrade

Fase de Busca Local

Na fase de busca local de [3] foi utilizado o mesmo método de busca local proposto por Ghosh [10]. O autor cita que a intenção foi a de comparar as fases de construção dos dois algoritmos.

Capítulo 5

Algoritmos Propostos

Neste trabalho são propostos dois novos algoritmos para a etapa de construção e um para a fase de busca local de um algoritmo GRASP aplicado ao PDM. Estes algoritmos serão apresentados, respectivamente, nas seções 5.1 e 5.2 onde deve ser considerado N um conjunto de índices com n elementos, $d(i,j)$ o índice de diversidade entre dois elementos i e j e M o subconjunto de N que contém m elementos mais diversos entre si, como já definido no Capítulo 2.

5.1 Métodos de Construção

Nesta seção são apresentadas novas propostas de algoritmos de construção para a heurística GRASP. Aqui, considere que até uma solução completa seja construída, tem-se m soluções parciais e cada uma delas é a solução M_{c-1}, com $c-1$ elementos, $1 \leq c \leq m$.

Os métodos de construção apresentados incorporam estratégias que visam aumentar a qualidade das soluções. Uma dessas estratégias chama-se *GRASP Reativo* [30] que, conforme descrito no Capítulo 3, estende o conceito do GRASP original. A idéia é avaliar o parâmetro α, que determina o tamanho da LRC, durante um certo número de iterações e privilegiar os valores de α para os quais se tem encontrado as melhores soluções.

Outra estratégia conhecida é implementar uma espécie de *filtro* de soluções. Na maioria dos casos, a busca local consome uma grande parte do tempo de computação do GRASP. A estratégia aqui se resume a gerar p soluções na fase de construção de modo que somente a melhor delas seja enviada para a etapa de busca local. Desta forma, o filtro permite que seja gerada uma quantidade maior de soluções na fase de construção, o que aumenta a possibilidade de melhora na qualidade sem onerar o tempo do algoritmo com a execução de busca local em todas as soluções construídas.

5.1.1 Heurística Aleatorizada das K Maiores Distâncias (HA_KMD)

Esta heurística constrói uma solução inicial elemento a elemento pela escolha aleatória de cada elemento da solução partindo da lista restrita de candidatos que contém K elementos.

A LRC é criada da seguinte maneira: para cada elemento $i \in N$ são selecionados os K elementos j, $j \in N - \{i\}$ que apresentam os maiores índices de diversidade em relação a i, ou seja, maiores $d(i, j)$. Estes valores são somados totalizando s_i. Uma lista de índices i ordenada em ordem decrescente de valor de s_i é criada e a partir dela, são selecionados os K primeiros elementos para compor a LRC. Vale ressaltar que trata-se do mesmo K utilizado anteriormente.

Para implementar o GRASP reativo, este procedimento é dividido em dois blocos de iterações. Seja t o número total de iterações do algoritmo GRASP. O primeiro bloco $B1$ corresponde às primeiras $0.4t$ iterações e nele quatro valores distintos de K devem ser usados. Para isto, $B1$ é dividido em quatro blocos r_i, $i = 1, ..., 4$, de $0.1t$ iterações e em cada um deles é usada uma LRC de tamanho k_i correspondente ao período. Os valores de k_i estão dispostos na tabela 5.1.

Onde:

- m é o número de elementos em uma solução viável;
- $\mu = \frac{n-m}{2}$.

i	r_i	k_i
1	1 a 0.1t	$m + \mu - 0.2\mu$
2	0.1t a 0.2t	$m + \mu - 0.1\mu$
3	0.2t a 0.3t	$m + \mu + 0.1\mu$
4	0.3t a 0.4t	$m + \mu + 0.2\mu$

Tabela 5.1: Valores de k_i em $B1$

Após a execução da última iteração do bloco $B1$, a qualidade das soluções obtidas deve ser avaliada. Isto é feito calculando-se o custo médio das soluções obtidas em cada bloco, $zm_i = [\sum_{q=1}^{0.1t} z(sol_{iq})]/0.1t$, onde sol_{iq} corresponde à solução encontrada na q-ésima iteração do bloco, $q = 1, \ldots, 0.1t$ e $i = 1, \ldots, 4$ e $z(sol_{iq})$ o custo desta solução.

Em seguida, é construída uma lista LK contendo os valores k_i ordenados de forma decrescente em relação a zm_i. Desta forma, a primeira posição de LK conterá o valor de k_i que obteve maior zm_i e assim por diante.

A partir daí, inicia-se a execução do segundo bloco de iterações $B2$ composto pelas $0.6t$ iterações restantes, no qual se fará uso das informações coletadas em $B1$. Este bloco também deve ser subdividido em quatro blocos v_i, mas ao contrário do ocorre em $B1$, cada um deles conterá um número diferente de iterações conforme é mostrado na tabela 5.2. Em cada bloco v_i será usado o valor lk_i para tamanho da LRC. Desta maneira, os valores de k_i são utilizados durante um número de iterações definido de maneira proporcional à qualidade das soluções por eles gerados no bloco anterior $B1$.

Bloco	**Iterações**
v_1	0.4t a 0.64t
v_2	0.64t a 0.82t
v_3	0.82t a 0.94t
v_4	0.94t a t

Tabela 5.2: Blocos de iterações de $B2$

Neste método, aplicamos o filtro de soluções, gerando, em cada iteração, 400 soluções. Somente a melhor delas é enviada à fase de busca local. Este valor foi definido após uma bateria de testes preliminares com diferentes números de soluções

no filtro.

O pseudo-código do procedimento descrito acima é apresentado na figura 5.1.

O procedimento possui os seguintes parâmetros formais apresentados na linha 1: *iter_corrente*, a iteração GRASP atual; t, o número total de iterações GRASP; *custo_medio_sol*, um conjunto contendo o custo médio das soluções em cada bloco e *ri*, o bloco de iteração atual.

Na linha 2, *melhor_sol* que armazena a melhor solução encontrada na fase de construção é inicializada. Na linha 3, *custo_melhor_sol* também é inicializada. Esta variável guardará o custo da melhor solução obtida. Na linha 4, é calculado o valor de K a ser utilizado na iteração *iter_corrente*. Para isto, são passados como parâmetros *iter_corrente*, t e LK que guardará os valores de K ordenados pelo custo médio obtido nas iterações prévias. Deve-se notar que esse parâmetro só terá valor e será usado a partir da iteração $0.4t$, ou seja, no bloco $B2$. A LRC é construída na linha 5. Nas linhas 6-17 são geradas as *num_sol_filtro* soluções. No laço apresentado nas linhas 8-12 uma solução completa é construída. Nas linhas 13-16 é feito o teste para identificar se a solução construída é a melhor até o momento, se for, na linha 14 essa solução é atribuída à *melhor_sol* e na linha 15, o custo desta solução é atribuído à *custo_melhor_sol*. Na linha 18 é acumulada a soma dos custos das soluções obtidas em cada intervalo $r_i, i = 1, \ldots, 4$ de iterações. Nas linhas 19-22 ocorre o GRASP reativo. Caso a iteração corrente seja a de número $0.4t$ a média dos custos em cada intervalo é calculada e constrói-se LK. Na linha 23 a solução é retornada.

```
1. proc Construcao_HA_KMD (n, m, iter_corrente, t, custo_medio_sol, ri)
2.   melhor_sol = {}
3.   custo_melhor_sol = 0
4.   K = det_K(iter_corrente, t, LK)
5.   LRC = const_LRC(K)
6.   para i=1 ate num_sol_filtro faca
7.    sol = {}
8.    para k=1 ate m  faca
9.     escolha aleatoriamente um elemento s da LRC
10.    sol = sol ∪ {s}
11.    LRC = LRC - {s}
12.   fim para
13.   se (z(sol) > custo_melhor_sol) entao
14.    melhor_sol = sol
15.    custo_melhor_sol = z(sol)
16.   fim se
17.  fim para
18.  custo_medio_sol[ri] = custo_medio_sol[ri] + custo_melhor_sol
19.  se (iter_corrente == 0.4t) entao
20.   custo_medio_sol = cal_Media(custo_medio_sol)
21.   LK = const_LK(custo_medio_sol)
22.  fim se
23.  retorne melhor_sol
24. fim Construcao_HA_KMD
```

Figura 5.1: Pseudo-código de HA_KMD

5.1.2 Heurística Aleatorizada das K Maiores Distâncias Versão 2 (HA_KMD_v2)

Esta heurística é semelhante à descrita na seção anterior. A diferença entre as duas está na maneira como a LRC é alterada a cada inserção de um novo elemento na solução parcial. Tendo em vista que na estratégia `HA_KMD` a única mudança na LRC após sua construção é a retirada do elemento à medida que este é inserido na solução, `HA_KMD_v2` foi projetada para ser mais adaptativa, fazendo com que a LRC seja refeita após a inclusão de cada elemento na solução.

Para construir M_1, ou seja, na escolha do primeiro elemento, o critério usado é o mesmo de `HA_KMD`, ou seja, um elemento é selecionado aleatoriamente a partir da LRC que corresponde aos K elementos de maiores somas de diversidade.

Para se construir M_c a partir de M_{c-1}, para cada elemento $i \in N - M_c$ são selecionados os $(K - c - 1)$ elementos j, $j \in N - M_c - \{i\}$ que apresentam maiores $d(i, j)$ e a soma desses valores é realizada, resultando em sf_i. Além disso, para cada i é feita a soma dos índices de diversidade $d(i, j)$, $j \in M_c$, resultando em sd_i. Uma lista inicial de candidatos LIC contendo todos índices i em ordem decrescente de s_i, $s_i = sf_i + sd_i$, $i \in N - M_c$ é criada.

A LRC é formada pelos K primeiros elementos de LIC. Um elemento é selecionado aleatoriamente da LRC para fazer parte da solução, sendo este procedimento repetido até que uma solução completa M_m tenha sido construída.

A estratégia de GRASP Reativo é implementada da mesma maneira como em `HA_KMD`, assim como o filtro de soluções, com alteração, porém, neste método são construídas 2 soluções a cada iteração, já que este procedimento exige mais esforço computacional que o anterior. Somente a melhor solução entre as construídas passa pelo procedimento de busca local.

5.1.3 Heurística Aleatorizada de Inserção mais Distante (HA_IMD)

Este método incorpora o conceito clássico da heurística iterativa de inserção mais distante, sendo esta modificada de forma a torná-la adaptativa, gulosa e randômica.

Para construir M_1, o primeiro elemento da solução, m_1, é escolhido aleatoriamente entre todos os elementos pertencentes a N.

O segundo elemento a ser selecionado deve ser o elemento $j \in N - M_1$ que oferece o maior índice de diversidade entre ele e m_1.

A partir do terceiro elemento, na construção de M_c, deve-se calcular $Dsoma(j)$, $\forall j \in N - M_{c-1}$ apresentada na equação 5.1, onde o primeiro termo corresponde à soma dos índices de diversidade entre todos os elementos já pertencentes a solução parcial M_{c-1} e o segundo, à soma das diversidade entre o candidato j e todos os elementos da solução parcial.

$$Dsoma(j) = \sum_{1 \leq y \leq c-2} \sum_{y+1 \leq w \leq c-1} d(m_y, m_w) + \sum_{1 \leq v \leq c-1} d(m_v, j) \qquad (5.1)$$

A LRC é construída como segue. Após o cálculo de $Dsoma(j)$, uma lista inicial de candidatos (LIC) é criada. Nela são inseridos os índices dos j elementos ordenados de forma decrescente em relação a $Dsoma$. Os primeiros $\alpha \times n$ elementos são selecionados para compor a LRC.

Neste procedimento, o conceito de GRASP reativo é implementado da mesma forma que em `HA_KMD`. Seja t o número total de iterações do GRASP. O primeiro bloco de iterações $B1$ com $0.4t$ iterações é subdividido em quatro blocos r_i, contendo cada um deles $0.1t$ iterações. Tem-se o conjunto α onde para cada r_i um α_i é associado. Os valores de α_i são apresentados na tabela 5.3.

Após o término da última iteração de $B1$, deve-se avaliar quais os valores de α_i que ofereceram melhores médias de soluções. Isto é feito como em `HA_KMD`, ou seja, calculando-se o custo médio $zm_i = [\sum_{q=1}^{0.1t} z(sol_{iq})]/0.1t$ obtido em cada r_i, onde

i	r_i	α_i
1	1 a 0.1t	0.03
2	0.1t a 0.2t	0.05
3	0.2t a 0.3t	0.07
4	0.3t a 0.4t	0.1

Tabela 5.3: Valores de α em $B1$

$z(sol_{iq})$ é o custo de cada solução, $i = 1, \ldots, 4$ e $q = 1, \ldots, 0.1t$. Os valores de α_i são guardados em uma lista $L\alpha$ ordenada por zm_i.

O próximo bloco de iterações $B2$ com as últimas $0.6t$ iterações também será dividido em quatro blocos v_i, $i = 1, \ldots, 4$, como em `HA_KMD` e serão usadas as informações adquiridas em $B1$ que foram armazenadas em Lα. Em cada bloco v_i será usado o valor $l\alpha_i$ para tamanho da LRC.

A partir da LRC é selecionado, aleatoriamente, um elemento para fazer parte da solução parcial. Este processo é repetido até que uma construção completa seja obtida.

O filtro de soluções também é utilizado neste método. Para cada iteração do GRASP são criadas n soluções e somente a solução com maior custo prossegue para a fase de busca local.

O pseudo-código da `HA_IMD` é apresentado na figura 5.2.

```
1. proc Construcao_HA_IMD (n, m, iter_corrente, t, custo_medio_sol, ri)
2.   melhor_sol = {}
3.   custo_melhor_sol = 0
4.   α = det_α(iter_corrente, t, Lα)
5.   para i=1 ate num_sol_filtro do
6.     sol = {}
7.     escolha aleatoriamente o elemento m_1 de N
8.     sol = sol ∪ {m_1}
9.     para cada j ∈ N − M_1 do
10.      calcule d(m_1, j)
11.      m_2 = j|d(j, m_1) ≡ max(d(j, m1))
12.    fim para
13.    sol = sol ∪ {m_2}
14.    para c = 3 ate m do
15.      LRC = Const_LRC (α)
16.      escolha aleatoriamente um elemento s da LRC
17.      Solucao = Solucao ∪ {s}
18.    fim para
19.    se (z(sol) > custo_melhor_sol) entao
20.      melhor_sol = sol
21.      custo_melhor_sol = z(sol)
22.    fim se
23.  fim para
24.  custo_medio_sol[ri] = custo_medio_sol[ri] + custo_melhor_sol
25.  se (iter_corrente == 0.4t) entao
26.    Lα = const_Lα(custo_medio_sol)
27.  fim se
28.  retorna melhor_sol
29. fim Construcao_HA_IMD
```

Figura 5.2: Pseudo-código de HA_IMD

Na linha 2 ocorre a inicialização de $melhor\_sol$ que guardará a melhor solução obtida na fase de construção. Na linha 3 a variável $custo\_melhor\_sol$ é inicializada com valor zero. Na linha 4 a função det_α calcula o valor de α a ser utilizado na iteração $iter\_corrente$. Os parâmetros usados nesta função são: $iter\_corrente$, t e LK que guardará os valores de K ordenados pelo custo médio obtido nas iterações prévias. Como já dito anteriormente, esse parâmetro só terá valor e será usado a partir da iteração $0.4t$. Nas linhas 5-23 são construídas num_sol_filtro soluções para o PDM. Na linha 6 uma solução vazia é criada. Na linha 7 inicia-se a criação de uma solução com a escolha aleatória do primeiro elemento entre os elementos do conjunto N. Este elemento é incorporado à solução na linha 8. O cálculo para determinar o segundo elemento é feito no trecho de linhas 9-12. Na linha 10 a diversidade entre m_1, o primeiro elemento da solução, e os candidatos j é determinada. Na linha 11, o elemento j que apresentou a maior diversidade em relação a m_1 é determinado e na linha 13, ele é incluído na solução. Nas linhas 14-18 o restante da solução é construída. Na linha 15 a LRC com α elementos é construída, um elemento s da LRC é escolhido aleatoriamente na linha 16 e na linha 17, s é incluído na solução. No trecho 19-22 é determinada a melhor solução encontrada a cada solução construída. O custo médio das soluções de cada intervalo de iterações é acumulado em $custo\_medio\_sol$, na linha 24. Nas linhas 25-27 é feito o teste que determina o fim do bloco $B1$, verificando se a iteração corrente é igual a 0.4t e, caso afirmativo, é aplicado o GRASP reativo, sendo Lα construída. Na linha 28 a solução é retornada.

5.2 Método de Busca Local

Este trabalho propoẽ também um novo procedimento de busca local para o PDM, que a partir de agora será denotado por Busca Local Proposta (`BLP`). A `BLP` utiliza duas estruturas de vizinhanças: na primeira delas $V1$, é usada a busca local proposta em [10] chamada `BLG`. Ao final dessa, sendo agora a solução corrente aquela que foi gerada pela `BLG`, prossegue-se com a segunda estrutura de vizinhança denotada $V2$ que é semelhante a primeira, com a única diferença que, ao invés de a cada passo um elemento i da solução ser trocado por outro j fora da solução, são

permutados simultaneamente dois elementos da solução por dois fora da solução. A cada troca a mudança de custo com a nova solução obtida $\Delta z(i,j,l,k)$ onde $i,j \in M$ e $l,k \in N - M$ é avaliada, conforme demonstrado na equação 5.2.

$$\Delta z(i,j,l,k) = \sum_{u \in M-\{i,j\}} d(l,u) + d(k,u) - (d(i,u) + d(j,u)) \tag{5.2}$$

A BLP é encerrada quando para toda troca de elementos não houver melhoria, isto é, $\Delta z(i,j,l,k) \leq 0$.

O pseudo-código da BLP é apresentado na figura 5.3.

```
1.  proc Busca_Proposta (Solucao)
2.    Solucao = BLG (Solucao)
3.    DeltaZ = INFINITO
4.    enquanto DeltaZ > 0 faca
5.     Max_DeltaZ = 0
6.     para cada i ∈ Solucao faca
7.      j = i + 1
8.      para cada  l ∈ N \ Solucao faca
9.       k = l + 1
10.      DeltaZ = Calcula_DeltaZ(i,j,l,k)
11.      se (DeltaZ > Max_DeltaZ)entao
12.       Max_DeltaZ = DeltaZ
13.       Max_I = i
14.       Max_J = j
15.       Max_L = l
16.       Max_K = k
17.      fim se
18.     fim para
19.    fim para
20.    se (Max_DeltaZ > 0) entao
21.     Solucao = Solucao \ {Max_I, Max_J}
22.     Solucao = Solucao ∪ {Max_L, Max_K}
23.     DeltaZ = Max_DeltaZ
24.    fim se
25.   fim enquanto
26.   retorna Solucao
27. fim Busca_Proposta
```

Figura 5.3: Pseudo-código da BLP

Na linha 2 a busca local `BLG` é executada retornando *Solucao*. Na linha 3 a variável *DeltaZ* é inicializada. Nela será guardada a melhoria associada à troca dos elementos da vizinhança da solução. Nas linhas 4-25 é realizada a busca local em $V2$. Na linha 5, Max_DeltaZ é inicializada. Nas linhas 6-19 é calculado *DeltaZ*. Nas linhas 11-17 são guardados os elementos que resultaram no maior *DeltaZ*. A solução é alterada nas linhas 21 e 22 caso *DeltaZ* seja positiva. Neste caso, na linha 23 *DeltaZ* é atualizado. A solução é retornada na linha 26.

5.3 Algoritmos GRASP Propostos para o PDM

Com base nos algoritmos de construção e de busca local propostos mostrados anteriormente, foram criadas novas heurísticas GRASP para solucionar aproximadamente o PDM. A tabela 5.4 apresenta um esquema com todas as variações propostas.

As duas primeiras heurísticas apresentadas na tabela 5.4, obtém a solução inicial através da Heurística Aleatorizada das K Maiores Distâncias, `HA_KMD`, as duas seguintes, utilizam a `HA_KMD_v2` (Heurística Aleatorizada das K Maiores Distâncias Versão 2) para a fase de construção e as duas últimas, usam a Heurística Aleatorizada de Inserção mais Distante, `HA_IMD`. Quanto à busca local, são usadas a busca `BLG` proposta por Ghosh [10] e a busca local proposta neste trabalho, a `BLP`.

GRASP	Construção	Busca Local
KMD+BLG	HA_KMD	BLG
KMD+BLP	HA_KMD	BLP
KMD2+BLG	HA_KMD_v2	BLG
KMD2+BLP	HA_KMD_v2	BLP
IMD+BLG	HA_IMD	BLG
IMD+BLP	HA_IMD	BLP

Tabela 5.4: Algoritmos GRASP propostos para o PDM

Capítulo 6

Testes e Resultados Computacionais

A fim de avaliar o desempenho das heurísticas desenvolvidas para o PDM mostradas no Capítulos 4 e 5, os algoritmos implementados passaram por uma série de testes e seus resultados são discutidos neste capítulo.

Na seção 6.1 a geração dos problemas testes é apresentada. Na seção 6.2 os resultados alcançados são mostrados e na seção 6.3 eles são analisados.

6.1 Geração dos Problemas Testes

Os testes foram realizados sobre três conjuntos de instâncias que diferem, entre elas, no tamanho da população e no intervalo de valores de possível diversidade entre seus elementos. O primeiro e terceiro conjuntos de instâncias foram propostos neste trabalho enquanto o segundo foi cedido por Andrade [3]. Para todos os conjuntos, foram testadas a obtenção de subconjuntos M com número de elementos m iguais a 10%, 20%, 30% e 40% do tamanho de N. As instâncias são descritas a seguir:

Conjunto de Instâncias 1. Foram criadas 5 instâncias com índices de diversidade gerados aleatoriamente em uma distribuição uniforme de números inteiros entre 0 e 9. Essas instâncias têm conjuntos N de 10, 20, 30, 40 e 50 elementos. O

caso $n = 10$ e $m = 1$ não foi aplicado, já que não representa um problema real.

Conjunto de Instâncias 2 [3]. Esse conjunto de instâncias possui populações de 100, 150, 200 e 250 elementos. No total são 12 instâncias divididas em:

Grupo A: Os índices de diversidade foram gerados aleatoriamente em uma distribuição de números inteiros entre 1 e 9999.

Grupo B: Os índices de diversidade também foram gerados aleatoriamente, mas 50% foram gerados a partir de uma distribuição uniforme de números inteiros entre 1 e 9999 e os demais a partir de uma distribuição uniforme de números inteiros entre 1 e 4999.

Grupo C: De maneira semelhante, os índices foram obtidos aleatoriamente, com 50% deles a partir de uma distribuição uniforme de números inteiros entre 1 e 9999 e outros 50% a partir de uma distribuição uniforme de números inteiros entre 5000 e 9999.

Conjunto de Instâncias 3. Consta de um total de 20 instâncias com os índices de diversidade gerados aleatoriamente em uma distribuição uniforme de números inteiros entre 0 e 9, da mesma maneira que ocorre no primeiro conjunto, porém, neste caso, as populações possuem 100, 200, 300, 400 e 500 elementos.

6.2 Testes

Usando as instâncias descritas na seção 6.1 foram realizadas diversas baterias de testes e os resultados obtidos foram comparados entre os algoritmos propostos, `KMD+BLG`, `KMD+BLP`, `KMD2+BLG`, `KMD2+BLP`, `IMD+BLG` e `IMD+BLP`. Além disso, foi implementado o método de construção e busca local de Ghosh, sendo este GRASP denominado `CGh+BLG`. Uma variação de `CGh+BLG`, denominado `CGh+BLP`, usa a construção de Ghosh e a busca proposta neste trabalho. Houve também comparação com o algoritmo implementado por Andrade, denominado `CAn+BLG`, que cedeu seu código para testes.

Todos os algoritmos implementados foram escrito na linguagem C++ [23], usando a versão 3.2.2 do compilador GNU g++ [33]. Os experimentos foram feitos em um AMD Athlon 1.3 GHz com 256 Mbytes de RAM usando sistema operacional Mandrake Linux 9.1.

6.2.1 Testes com Conjunto de Instâncias 1

Para testar a qualidade das heurísticas utilizadas neste trabalho, foram criadas estas instâncias que correspondem a problemas de pequenas dimensões se comparadas àqueles representados pelas demais instâncias. Nesta bateria de testes, o valor da diversidade da solução obtida por cada algoritmo heurístico foi comparado ao valor da solução exata. Para encontrar soluções exatas foi utilizada a formulação de programação linear inteira desenvolvida para o problema por [21], já apresentada no Capítulo 2 e para resolver a formulação implementada utilizou-se o Pacote GLPK (GNU *Linear Programming Kit*) [24] que implementa um *branch and bound* baseado no método dual simplex.

Todas as heurísticas GRASP foram executadas três vezes para cada instância usando diferentes sementes na geração dos números aleatórios. Em cada uma das execuções foram realizadas 500 iterações GRASP.

As tabelas 6.1, 6.2 e 6.3 mostram os resultados destes experimentos. Na tabela 6.1 são ilustrados, para cada instância, as soluções exatas e as soluções encontradas por cada algoritmo GRASP. Na tabela 6.2 são mostradas as diferenças entre a solução exata e a média das soluções encontradas por cada GRASP. Esta diferença, denominada dif, foi calculada usando-se a equação 6.1. Ainda na tabela 6.2 o símbolo (-) representa diferença igual a 0, ou seja, indica que naquela instância o algoritmo obteve a melhor solução.

$$dif = (\text{solução exata} - \text{solução heurística})/\text{solução exata} * 100 \qquad (6.1)$$

Na tabela 6.3 são mostrados os tempos de processamento do GLPK para obter

a solução exata e o tempo médio das execuções das heurísticas com 500 iterações.

n	m	Exato	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	CAn+BLG
10	2	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
10	3	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**
10	4	**47**	**47**	**47**	**47**	**47**	**47**	**47**	**47**	**47**	40
20	2	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
20	4	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**
20	6	**109**	**109**	**109**	**109**	**109**	**109**	**109**	**109**	**109**	**109**
20	8	**181**	**181**	**181**	**181**	**181**	**181**	**181**	180	180	**181**
30	3	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**
30	6	**121**	**121**	**121**	**121**	**121**	**121**	**121**	**121**	**121**	**121**
30	9	**254**	**254**	**254**	**254**	**254**	**254**	**254**	**254**	**254**	**254**
30	12	**412**	**412**	**412**	**412**	**412**	**412**	**412**	**412**	**412**	**412**
40	4	**53**	**53**	**53**	**53**	**53**	**53**	**53**	49	49	**53**
40	8	**198**	**198**	**198**	**198**	**198**	**198**	**198**	192	192	**198**
40	12	**412***	**412**	**412**	**412**	**412**	**412**	**412**	**412**	**412**	**412**
40	16	681*	**688**	**688**	**688**	**688**	**688**	**688**	**688**	**688**	**688**
50	5	78*	**86**	**86**	**86**	**86**	**86**	**86**	81	81	81
50	10	268*	**311**	**311**	**311**	**311**	**311**	**311**	**311**	**311**	309
50	15	602*	**652**	**652**	**652**	**652**	**652**	**652**	**652**	**652**	627
50	20	974*	**1087**	**1087**	**1087**	**1087**	**1087**	**1087**	**1087**	**1087**	1033

Tabela 6.1: Custos das soluções obtidas pelo algoritmo exato e pelas heurísticas. O símbolo (*) significa maior valor de solução obtida pelo exato em 36.000 segundos

n	m	Exato	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	CAn+BLG
10	2	-	-	-	-	-	-	-	-	-	-
10	3	-	-	-	-	-	-	-	-	-	-
10	4	-	-	-	-	-	-	-	-	-	14,9
20	2	-	-	-	-	-	-	-	-	-	-
20	4	-	-	-	-	-	-	-	-	-	-
20	6	-	-	-	-	-	-	-	-	-	-
20	8	-	-	-	-	-	-	-	0,6	0,6	-
30	3	-	-	-	-	-	-	-	-	-	-
30	6	-	-	-	-	-	-	-	-	-	-
30	9	-	-	-	-	-	-	-	-	-	-
30	12	-	-	-	-	-	-	-	-	-	-
40	4	-	-	-	-	-	-	-	7,5	7,5	-
40	4	-	-	-	-	-	-	-	3,0	3,0	-
40	12	-**	-	-	-	-	-	-	-	-	-
40	16	1,0**	-	-	-	-	-	-	-	-	-
50	5	9,3**	-	-	-	-	-	-	-	-	-
50	10	13,8**	-	-	-	-	-	-	-	-	0,6
50	15	7,7**	-	-	-	-	-	-	-	-	3,8
50	20	10,4**	-	-	-	-	-	-	-	-	5,0
Média		2,7	-	-	-	-	-	-	0,7	0,7	1,6

Tabela 6.2: Diferença entre a solução exata e as soluções obtidas pelas heurísticas. O símbolo (**) significa a diferença entre o maior custo de solução obtida pelo exato em 36.000 segundos e os demais algoritmos

n	m	Exato	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	CAn+BLG
10	2	0,02	0,02	0,03	0,05	0,05	0,03	0,03	0,03	0,03	0,00
10	3	0,09	0,03	0,03	0,08	0,07	0,05	0,05	0,04	0,04	0,01
10	4	0,14	0,03	0,04	0,09	0,10	0,07	0,07	0,05	0,05	0,01
20	2	2,00	0,09	0,09	0,19	0,20	0,13	0,12	0,12	0,12	0,00
20	4	10,00	0,12	0,13	0,41	0,42	0,34	0,34	0,22	0,24	0,01
20	6	24,00	0,17	0,20	0,59	0,63	0,53	0,54	0,33	0,35	0,02
20	8	26,00	0,23	0,27	0,77	0,82	0,70	0,72	0,39	0,42	0,03
30	3	126,00	0,22	0,23	0,73	0,75	0,57	0,58	0,40	0,41	0,01
30	6	653,00	0,36	0,41	1,54	1,56	1,32	1,36	0,82	0,87	0,03
30	9	1413,00	0,63	0,72	2,26	2,29	2,04	2,09	1,20	1,27	0,06
30	12	2862,00	0,99	1,13	2,86	2,93	2,70	2,77	1,56	1,66	0,13
40	4	2374,00	0,44	0,46	2,18	2,13	1,58	1,55	1,03	1,24	0,02
40	8	163931,00	0,90	1,05	4,73	4,57	3,52	3,55	2,07	2,75	0,10
40	12	***	0,99	1,69	7,11	7,24	5,42	5,46	3,09	4,50	0,22
40	16	***		2,70	9,27	10,09	7,36		4,19	6,42	0,48
50	5	***	0,58	0,61	3,06	3,10	2,31	2,36	1,76	1,79	0,04
50	10	***	1,69	1,87	6,89	7,05	5,55	5,80	4,53	4,67	0,22
50	15	***	3,26	3,71	11,34	11,81	9,17	9,62	7,41	7,83	0,59
50	20	***	5,14	5,79	16,11	16,70	14,12	14,71	10,55	11,11	1,12

Tabela 6.3: Tempos de processamento (em segundos) para algoritmo exato e (médio) das heurísticas. O símbolo (***) tempo de processamento algoritmo exato limitado em 36.000 segundos

Para as instâncias $n = 10$ e $m = 2$ a $n = 40$ e $m = 8$ o GLPK pôde encontrar solução exata em tempo computacional viável. No entanto, quando o algoritmo exato foi executado para a instância $n = 40$ e $m = 12$, o algoritmo permaneceu em execução durante 25 dias e ainda assim não havia terminado o processamento. A partir desta instância, o tempo de computação foi limitado em 36.000 segundos e a melhor solução obtida foi guardada.

Nas tabelas 6.1 e 6.3, os símbolos * e ** significam, respectivamente, o valor da melhor solução obtida em 36000 segundos de processamento e tempo de processamento limitado em 36000 segundos. Na tabela 6.2, a partir da instância $n = 40$ e $m = 12$, a diferença é calculada com base no valor da melhor solução encontrada, já que não necessariamente o valor obtido pelo algoritmo exato é ótimo.

Analisando-se os resultados obtidos nesta primeira bateria de testes, pode-se concluir que todas as heurísticas propostas neste trabalho, `KMD+BLG`, `KMD+BLP`, `KMD2+BLG`, `KMD2+BLP`, `IMD+BLG` e `IMD+BLP`, foram capazes de encontrar a solução ótima em 100% das instâncias testadas onde o ótimo foi conhecido. Nas 6 instâncias em que o tempo do exato foi limitado, em uma delas as heurísticas empataram com o algoritmo exato e nas demais foi obtida soluções de valor melhores que as obtidas pelo algoritmo exato. A heurística da literatura `CGh+BLG` e o algoritmo `CGh+BLP`, mantiveram-se ambos, em média, a 0,7% do ótimo, encontrando solução ótima em 10 das 13 instâncias sem tempo limitado. Já `CAn+BLG` não pôde chegar a solução exata em uma destas 13 instâncias e em 3 das 6 onde o tempo do exato foi limitado, e esteve, em média, à 1,6% da melhor solução.

A tabela 6.3 mostra o tempo de processamento dos algoritmos e no gráfico 6.1 são plotados os tempos até a instância $n = 40$ e $m = 8$. Através da tabela pode-se ver que o tempo de processamento das heurísticas é bem menor (pelo menos 1/346 vezes menor) comparado ao tempo de computação do algoritmo exato. Nas instâncias em que o tempo foi limitado em 36.000 segundos as soluções obtidas pelo exato são de valores iguais ou piores que os das heurísticas propostas neste trabalho, devendo-se acrescentar que o maior tempo de processamento das heurísticas não ultrapassou 17 segundos. O gráfico 6.1 demonstra que o tempo das heurísticas mantêm um

crescimento muito mais lento comparadas à forma como cresce o tempo do algoritmo exato.

6.2.2 Testes com Conjunto de Instâncias 2

Para todos os grupos que compõem este conjunto de instâncias, cada algoritmo foi executado com 500 iterações e foi computado o valor médio das soluções obtidas em três execuções (com três sementes distintas para a geração dos números aleatórios). Além disso, foi computado o tempo de CPU médio das execuções. As seções a seguir apresentam uma análise dos valores das soluções obtidas pelas heurísticas e do tempo computacional exigido.

Análise de Soluções

As tabelas 6.4, 6.5 e 6.6 apresentam a diferença entre as médias dos valores de soluções obtidos em três execuções de cada algoritmo nas instâncias do Grupo A, Grupo B e Grupo C, respectivamente. O símbolo (-) representa diferença igual a 0, ou seja, indica que naquela instância o algoritmo obteve a melhor solução.

A diferença entre os valores médios, chamada $difvalmedio$, é calculada pela equacao 6.2

$$difvalmedio = (\text{maior valor médio} - \text{valor médio})/\text{valor médio} * 100 \qquad (6.2)$$

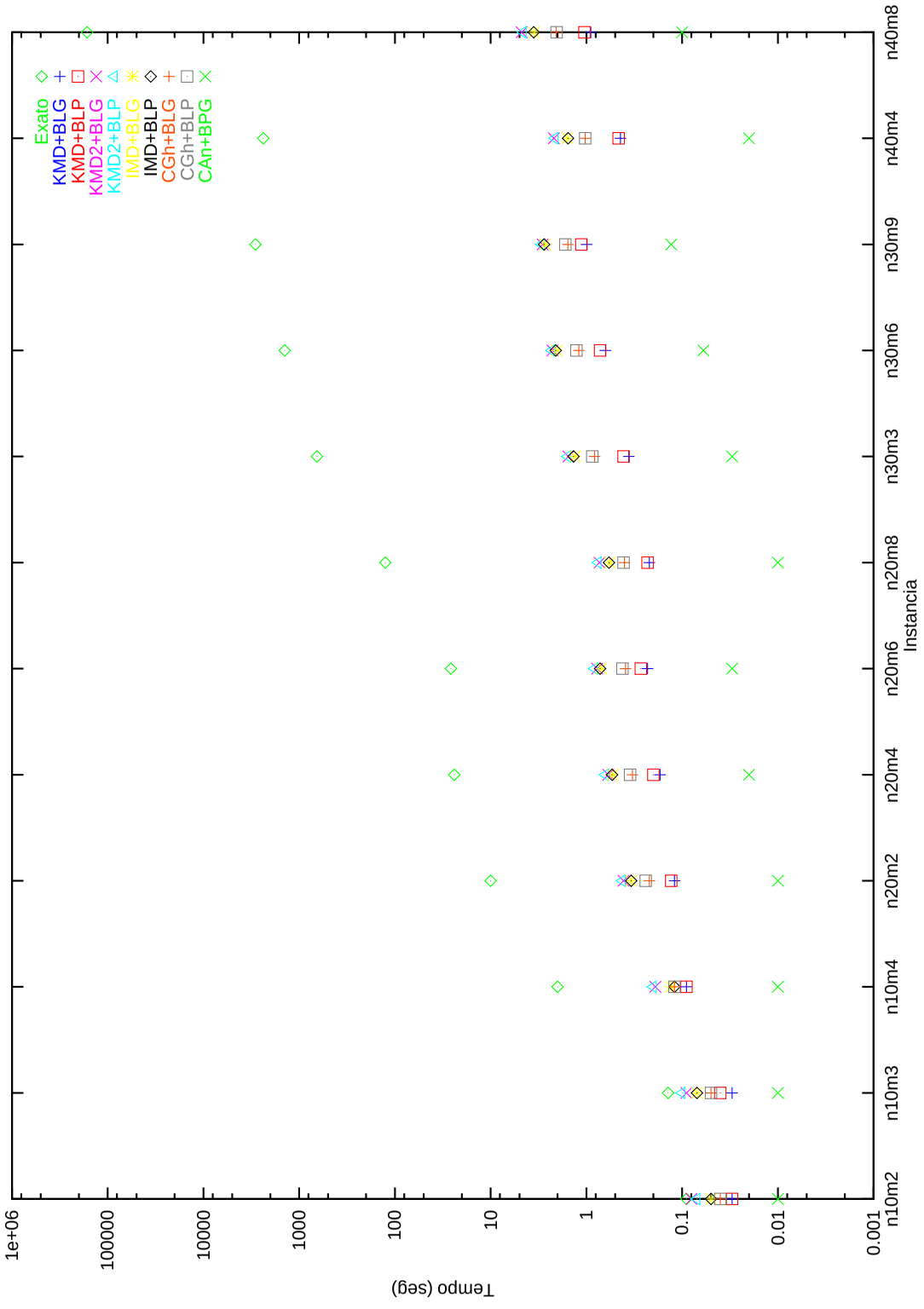

Figura 6.1: Gráfico de tempo de execução dos algoritmos

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	CAn+BLG
50	5	-	-	-	-	-	-	-	-	-
50	10	-	-	-	-	-	-	-	-	-
50	15	-	-	-	-	-	-	-	-	-
50	20	-	-	-	-	-	-	-	-	-
100	10	-	-	-	-	-	-	0,41	0,41	-
100	20	-	-	-	-	-	-	-	-	-
100	30	-	-	-	-	-	-	-	-	-
100	40	-	-	-	-	-	-	-	-	-
150	15	-	-	-	-	-	-	2,58	2,58	-
150	30	-	-	-	-	-	-	-	-	-
150	45	0,01	-	-	-	-	-	-	-	0,09
150	60	-	-	-	-	-	-	0,01	-	0,06
200	20	0,05	0,12	0,07	0,05	-	0,05	0,72	0,72	0,39
200	40	-	-	-	-	0,04	0,04	-	-	0,01
200	60	-	-	0,04	0,04	-	-	-	-	0,18
200	80	-	-	-	-	-	-	-	-	0,10
250	25	-	-	0,04	0,04	-	-	0,33	0,33	0,25
250	50	0,09	0,14	0,04	0,06	0,05	0,02	0,01	-	0,43
250	75	0,04	0,07	-	-	0,14	0,09	0,08	0,08	0,25
250	100	0,01	-	0,01	-	0,02	0,02	0,05	0,05	0,10
Média		**0,01**	0,02	**0,01**	**0,01**	**0,01**	**0,01**	0,21	0,21	0,09

Tabela 6.4: Diferença entre média dos valores nas instâncias do Grupo A

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	CAn+BLG
50	5	-	-	-	-	-	-	5,20	5,20	-
50	10	-	-	-	-	-	-	-	-	-
50	15	-	-	-	-	-	-	-	-	-
50	20	-	-	-	-	-	-	-	-	-
100	10	-	-	-	-	-	-	-	-	-
100	20	-	-	-	-	-	-	-	-	-
100	30	-	-	-	-	-	-	-	-	-
100	40	-	-	-	-	-	-	-	-	-
150	15	-	-	-	-	-	-	-	-	-
150	30	-	-	-	-	-	-	-	-	-
150	45	-	-	-	-	-	-	-	-	0,06
150	60	-	-	-	-	-	-	-	-	-
200	20	-	-	-	-	-	-	-	-	0,29
200	40	-	-	-	-	-	-	-	-	0,09
200	60	-	-	-	-	-	-	-	-	0,01
200	80	-	-	-	-	-	-	-	-	-
250	25	-	-	-	-	-	-	-	-	0,42
250	50	-	-	-	-	-	-	-	-	0,08
250	75	-	-	-	-	-	-	-	-	0,02
250	100	-	-	-	-	-	-	-	-	-
Média		-	-	-	-	-	-	0,26	0,26	0,05

Tabela 6.5: Diferença entre médias dos valores nas instâncias do Grupo B

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	And
50	5	-	-	-	-	-	-	-	-	-
50	10	-	-	-	-	-	-	0,16	-	0,16
50	15	-	-	-	-	-	-	-	-	-
50	20	-	-	-	-	-	-	-	-	-
100	10	-	-	-	-	-	-	-	-	0,26
100	20	-	-	-	-	-	-	-	-	-
100	30	-	-	-	-	-	-	-	-	-
100	40	-	-	-	-	-	-	-	-	-
150	15	-	-	-	-	-	-	0,23	-	0,23
150	30	-	-	-	-	-	-	0,12	-	0,12
150	45	-	-	-	-	0,01	0,01	-	-	-
150	60	-	-	-	-	-	-	0,01	-	0,01
200	20	-	-	-	-	-	-	0,34	-	0,34
200	40	-	-	-	-	-	-	0,12	-	0,12
200	60	-	-	-	-	-	-	0,04	-	0,04
200	80	-	-	-	-	-	-	0,04	-	0,04
250	25	-	-	-	-	0,02	0,02	0,23	0,02	0,23
250	50	-	-	-	-	-	-	0,17	-	0,17
250	75	-	-	-	-	-	-	0,11	-	0,11
250	100	-	-	-	-	-	-	0,09	-	0,09
Média		-	-	-	-	-	-	0,08	-	0,10

Tabela 6.6: Diferença entre médias dos valores nas instâncias do Grupo C

Os resultados das tabelas 6.4, 6.5 e 6.6 mostram que os algoritmos possuem um comportamento uniforme sobre instâncias que apresentam diferentes naturezas de formação.

É possível também observar que em todos os grupos de instâncias, os algoritmos propostos KMD+BLG, KMD+BLP, KMD2+BLG, KMD2+BLP, IMD+BLG e IMD+BLP obtiveram as melhores soluções na média. Isto pode ser visto na tabela 6.7 na qual é mostrado, para cada grupo, o número de vezes em que cada algoritmo obteve uma solução de melhor valor em 20 instâncias.

Algoritmo	Grupo A	Grupo B	Grupo C
KMD+BLG	15	20	20
KMD+BLP	17	20	20
KMD2+BLG	15	20	20
KMD2+BLP	16	20	20
IMD+BLG	16	20	18
IMD+BLP	15	20	18
CGh+BLG	14	19	8
CGh+BLP	14	19	19
CAn+BLG	10	17	7

Tabela 6.7: Número de vezes em que a melhor solução é encontrada por cada algoritmo

Para as instâncias do Grupo A, as heurísticas propostas estiveram em média a no máximo 0,02% da melhor solução enquanto o algoritmo de Ghosh se manteve a 0,21% e o de Andrade a 0,09%. Pela tabela 6.7, em um total de 20 instâncias, KMD+BLP obteve maior sucesso, chegando a soluções de melhores valores 17 vezes. Os algoritmos da literatura CGh+BLG e sua variação CGh+BLP obtiveram melhor soluções em 14 instâncias e CAn+BLG em 10 das 20 instâncias.

Nos dois outros grupos, B e C, os métodos propostos alcançaram resultados ainda melhores.

Pode-se observar que no Grupo B, os algoritmos propostos KMD+BLG, KMD+BLP, KMD2+BLG, KMD2+BLP, IMD+BLG e IMD+BLP obtiveram, em todos os casos, as melhores soluções. Já os algoritmos da literatura CGh+BLG e CGh+BLP ficaram, ambos, em média, a 0,26% da melhor solução e CAn+BLG a 0,05%.

No Grupo C os algoritmos propostos KMD+BLG, KMD+BLP, KMD2+BLG e KMD2+BLP

obtiveram, mais uma vez, as melhores soluções em 100% dos casos. IMD+BLG e IMD+BLP alcançaram a melhor solução em 18 das 20 instâncias, além disso, as duas heurísticas estiveram a, no máximo, 0,02% da melhor solução. Na média que considera todas as instâncias, isto representa uma diferença percentual de 0,0015. CGh+BLG alcançou as melhores soluções em somente 8 das 20 instâncias. Quanto a CGh+BLP, nota-se que a BLP pôde melhorar o desempenho apresentado pelo GRASP original de Ghosh, chegando à melhor solução em 19 do total de instâncias do grupo. O algoritmo de Andrade encontrou a melhor solução em 7 das 20 instâncias e manteve-se a 0,10% das melhores soluções apresentadas, a pior diferença observada dentre todos os algoritmos. Um fato que pode ser constatado através dos experimentos é que conforme há o aumento do tamanho das instâncias, o algoritmo de Andrade tem seu desempenho prejudicado.

Pode-se, com base nos resultados empíricos obtidos, propor uma classificação para os algoritmos em que KMD+BLG, KMD2+BLG, KMD2+BLP, IMD+BLG e IMD+BLP empatam em primeiro lugar, seguidos por KMD+BLP (por uma diferença percentual quase desprezível - 0,003% dos demais). CAn+BLG, algoritmo proposto por Andrade ocuparia o terceiro lugar. Por fim, teria-se CGh+BLP seguido por CGh+BLG, demonstrando que foi possível obter uma pequena melhora na proposta de Ghosh utilizando-se a BLP no lugar de sua busca local.

Análise de Tempo Computacional

As tabelas 6.8, 6.9 e 6.10 apresentam a média dos tempos (segundos) gastos por cada algoritmo nas instâncias do Grupo A, Grupo B e Grupo C, respectivamente.

A partir dos resultados deve-se destacar que a BLP demanda tempo de computação maior. Isto pode ser visto comparando-se os algoritmos que utilizam o mesmo método de construção mas que diferem entre si pelo uso de BLG e BLP, como ocorre, por exemplo com KMD+BLG e KMD+BLP, KMD2+BLG e KMD2+BLP, IMD+BLG e IMD+BLP, CGh+BLG e CGh+BLP. Este resultado já era esperado já que uma busca completa BLG é feita antes da mudança da estrutura da vizinhança.

Verifica-se também que a heurística proposta por Andrade demanda um tempo menor de computação. Em seguida, os algoritmos propostos, `KMD+BLG` e `KMD+BLP`. `KMD2+BLP` é o algoritmo que despende maior tempo de computação. `KMD2+BLG`, `IMD+BLG`, `IMD+BLP`, `CGh+BLP` e o algoritmo da literatura `CGh+BLG` apresentam tempo de computação parecidos. Além disso, esses algoritmos apresentam tempo com valores intermediários entre os mais rápidos e mais pesados.

Pelo tempo de computação usado pelos algoritmos em cada grupo de instâncias é possível supor que as instâncias do Grupo C correspondem àquelas mais difíceis de serem resolvidas se comparadas às demais do conjunto. Esta conclusão pode ser tirada do fato que o tempo de computação gasto por todos os algoritmos é mais elevado nas instâncias deste grupo.

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	CAn+BLG
50	5	2,6	2,6	3,9	3,8	3,0	3,0	2,4	2,4	0,0
50	10	4,5	4,6	7,2	7,4	5,8	6,0	4,6	4,8	0,1
50	15	6,9	7,2	10,7	11,0	8,6	8,9	6,7	7,1	0,4
50	20	9,8	10,2	14,0	14,5	11,4	11,8	8,7	9,0	0,7
100	10	7,9	8,2	28,7	28,7	22,6	22,8	15,5	15,7	0,3
100	20	21,4	22,6	62,1	62,9	50,7	51,7	38,7	38,1	2,3
100	30	40,7	43,0	100,9	102,5	84,1	87,3	64,4	66,5	6,7
100	40	62,6	66,4	139,4	144,0	125,2	128,9	93,5	94,7	13,0
150	15	21,9	23,0	102,7	103,5	83,0	83,9	56,2	56,3	1,9
150	30	76,5	80,4	240,8	243,7	202,6	205,8	146,4	155,3	12,4
150	45	164,1	173,0	415,4	421,2	346,1	354,2	288,1	292,2	34,6
150	60	267,7	283,4	607,0	620,1	542,3	551,2	449,3	431,6	70,1
200	20	51,6	54,3	263,1	262,6	213,4	216,2	146,6	148,1	6,7
200	40	214,8	221,8	644,6	645,8	556,3	567,0	425,4	427,4	41,1
200	60	490,0	508,4	1149,8	1175,5	1006,6	1028,9	820,5	825,4	113,7
200	80	796,4	830,4	1700,1	1744,1	1568,1	1603,0	1108,8	1116,2	228,9
250	25	108,7	114,5	536,5	542,1	445,4	451,5	322,5	317,6	17,1
250	50	505,3	525,2	1384,3	1404,2	1230,0	1238,7	982,7	974,6	101,8
250	75	1167,2	1204,8	2568,6	2647,8	2350,8	2373,3	1854,2	1889,7	278,4
250	100	1885,5	1931,0	3876,5	4023,3	3298,4	3351,0	2696,0	2744,1	579,1

Tabela 6.8: Tempo de computação (em segundos) dos algoritmos para instâncias do Grupo A

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	CAn+BLG
50	5	2,7	2,8	3,8	4,0	3,0	3,0	2,5	2,3	0,0
50	10	4,8	7,8	7,4	7,8	5,8	5,9	4,5	4,1	0,1
50	15	7,3	10,5	11,2	11,9	8,3	8,5	6,0	5,6	0,3
50	20	9,8	8,9	15,0	16,0	10,4	10,8	6,9	6,6	0,3
100	10	8,4	23,7	28,5	29,9	22,5	22,8	16,5	14,4	0,3
100	20	21,8	43,9	62,8	66,5	49,1	50,2	36,8	32,6	2,0
100	30	40,2	69,6	103,1	111,2	77,6	79,9	52,5	48,7	4,2
100	40	64,1	23,8	159,3	171,3	102,4	106,0	67,1	63,4	8,2
150	15	21,9	82,3	103,0	110,0	83,7	84,4	57,7	52,8	1,7
150	30	75,8	174,1	246,2	262,0	189,3	192,7	133,9	126,0	11,6
150	45	159,3	207,6	432,5	453,1	320,2	328,7	210,4	198,0	33,1
150	60	262,6	275,4	652,3	650,8	419,7	430,9	226,3	263,4	54,0
200	20	52,6	57,1	263,5	264,7	213,3	216,2	141,8	131,5	5,7
200	40	218,5	241,1	676,6	685,2	499,1	508,9	399,3	375,1	36,8
200	60	468,9	507,6	1220,2	1236,4	809,5	835,8	637,9	596,3	106,8
200	80	711,4	770,1	1788,4	1812,9	1090,7	1139,5	874,7	825,9	156,3
250	25	111,0	115,9	559,3	554,2	454,4	453,7	312,7	308,2	13,9
250	50	511,7	527,7	1497,2	1500,2	1091,8	1129,9	919,6	923,9	92,9
250	75	1158,9	1177,2	2860,7	2873,8	1779,2	1862,9	1701,9	1785,0	267,2
250	100	1735,4	1794,4	4111,5	4176,4	2617,3	2685,2	2472,7	2525,1	467,2

Tabela 6.9: Tempo de computação (em segundos) dos algoritmos para instâncias do Grupo B

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	CAn+BLG
50	5	2,3	2,2	3,6	3,6	2,8	2,8	2,3	2,3	0,0
50	10	4,0	4,0	6,7	6,8	5,3	5,4	4,1	4,2	0,1
50	15	6,1	6,2	9,8	10,1	7,8	8,1	5,7	6,0	0,4
50	20	8,9	9,0	12,8	13,2	10,2	10,5	7,5	7,9	0,8
100	10	7,6	7,5	26,3	26,6	20,5	20,7	14,3	14,6	0,3
100	20	20,6	20,8	58,7	59,6	47,3	48,2	34,4	35,3	2,3
100	30	39,1	40,8	96,9	98,9	79,6	81,6	56,9	58,9	6,0
100	40	58,6	61,1	133,4	136,5	114,2	117,6	84,0	87,2	11,4
150	15	19,5	20,4	94,7	95,8	76,5	77,3	51,0	51,7	1,8
150	30	69,1	72,4	223,2	227,2	180,4	184,0	124,8	129,3	11,5
150	45	149,3	155,3	388,5	396,0	314,4	321,2	235,7	243,3	30,98
150	60	241,9	250,9	564,1	575,2	463,9	474,6	329,1	339,3	53,3
200	20	47,8	50,0	244,9	246,9	201,7	204,2	132,8	134,9	6,5
200	40	201,8	209,4	623,0	631,7	538,4	547,3	372,4	382,2	38,5
200	60	468,5	482,9	1137,5	1154,0	917,6	938,8	715,4	733,4	106,5
200	80	716,5	740,4	1626,5	1652,8	1267,8	1298,2	1159,4	1178,4	199,4
250	25	143,4	148,8	570,9	577,9	468,3	475,0	301,6	308,6	16,1
250	50	605,2	624,1	1578,5	1603,4	1373,3	1390,4	979,0	1002,6	96,2
250	75	1291,3	1328,3	2925,0	2975,9	2335,7	2367,3	2000,2	2064,8	264,3
250	100	1864,0	1922,6	4273,1	4326,3	3320,0	3377,3	3000,2	3090,6	606,9

Tabela 6.10: Tempo de computação (em segundos) dos algoritmos para instâncias do Grupo C

6.2.3 Testes com Conjunto de Instâncias 3

Estas instâncias foram submetidas a dois tipos de testes. No primeiro, cada instância foi executada três vezes com cada algoritmo processando 500 iterações. O valor das soluções e o tempo de CPU foram computados. O outro tipo de teste analisa a robustez do algoritmo. Os resultados são apresentados nas subseções seguintes.

Análise de Soluções

Na tabela 6.11 são apresentadas as diferenças entre as médias dos valores das soluções e o melhor valor médio encontrado nas três execuções. As diferenças são calculadas conforme descrito na seção 6.2.2.

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	CAn+BLG
100	10	-	-	-	-	-	-	4,50	4,50	-
100	20	-	-	-	-	-	-	1,42	1,42	0,28
100	30	-	-	-	-	-	-	-	-	0,09
100	40	-	-	-	-	-	-	-	-	0,04
200	20	-	-	-	-	0,11	-	1,12	1,12	0,32
200	40	0,07	-	0,05	0,03	0,06	0,07	0,14	0,16	0,13
200	60	-	-	-	-	0,03	-	-	-	0,14
200	80	0,26	0,34	-	-	-	-	-	-	0,15
300	30	0,22	0,33	0,13	-	0,30	0,48	0,97	0,97	0,40
300	60	0,15	0,25	-	0,07	0,05	0,04	0,32	0,39	0,25
300	90	0,42	0,42	-	-	0,03	0,02	0,01	0,01	0,26
300	120	0,03	0,03	0,01	-	-	0,01	-	-	0,08
400	40	0,42	0,43	0,16	0,01	0,14	-	0,85	0,94	0,54
400	80	0,18	0,26	0,10	0,05	-	0,01	0,27	0,24	0,43
400	120	0,35	0,35	0,08	0,03	-	-	0,01	-	0,31
400	160	0,26	0,26	-	0,04	0,04	0,06	0,06	0,07	0,21
500	50	-	0,11	0,16	0,24	0,19	0,26	0,63	0,63	0,58
500	100	0,12	0,03	-	-	0,06	0,04	0,04	0,04	0,38
500	150	-	0,85	0,85	0,85	0,89	0,88	0,85	0,85	1,22
500	200	0,12	-	0,02	0,02	0,02	0,01	0,08	0,01	0,17
Média		0,14	0,16	0,08	**0,07**	0,10	0,10	0,59	0,56	0,30

Tabela 6.11: Diferenças entre médias dos valores das soluções obtidas nas instâncias do Conjunto 3

O desempenho dos algoritmos pode ser sumarizado na tabela 6.12. A cada coluna (#) mostra o número de vezes em que o algoritmo encontrou a melhor solução, em um total de 20 instâncias.

Algoritmo	#
KMD+BLG	8
KMD+BLP	8
KMD2+BLG	11
KMD2+BLP	11
IMD+BLG	6
IMD+BLP	9
CGh+BLG	5
CGh+BLP	6
CAn+BLG	1

Tabela 6.12: Número de vezes em que a melhor solução é encontrada por cada algoritmo

Vê-se que algoritmos propostos KMD2+BLG e KMD2+BLP obtiveram, ambos, a melhor solução em 11 das 20 instâncias, destacando-se dos demais. Quanto aos algoritmos da literatura, o de Andrade foi capaz de encontrar a melhor solução somente em uma instâncias e o de Ghosh, em 5 delas. Pode-se ressaltar que nas instâncias $n = 100$ e $m = 10$ e $n = 100$ e $m = 20$, todos os algoritmos propostos chegaram à melhor solução enquanto o algoritmo de Ghosh esteve a 4,5% dela. Em IMD+BLP e CGh+BLP, a BLP pôde melhorar os resultados alcançados por IMD+BLG e CGh+BLG, respectivamente.

De uma forma geral, nota-se que apesar dos algoritmos da literatura serem capazes de obter boas soluções para o problema, os algoritmos propostos se mostram mais eficientes em relação à qualidade das soluções geradas.

Análise de Tempo Computacional

A tabela 6.13 apresenta a média dos tempos de CPU (segundos). Essas instância vêm confirmando o desempenho já observado no Conjunto 2. Pode-se verificar que o algoritmo de Andrade continua sendo o mais rápido e em seguida, o algoritmo proposto KMD+BLG. Os demais, de maneira semelhante, seguem o comportamento discutido na seção 6.2.2. Acrescenta-se que, de maneira geral, os algoritmos ex-

perimentados se portam de forma similar mesmo com a variedade de instâncias utilizadas.

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	And
100	10	10,4	11,8	28,0	30,2	22,3	23,1	18,0	19,3	0,4
100	20	28,2	37,0	64,8	79,5	53,3	60,5	47,4	64,7	2,8
100	30	57,4	79,1	111,7	148,3	91,0	113,6	88,3	123,4	7,5
100	40	89,2	126,2	165,6	234,4	138,8	187,1	130,9	198,7	14,6
200	20	71,7	101,0	263,6	301,1	224,3	247,6	147,5	203,4	6,3
200	40	318,3	510,3	784,7	986,3	635,6	835,6	561,9	825,3	39,3
200	60	626,6	1006,4	1958,8	2807,0	1224,0	1715,6	1075,3	1665,9	113,5
200	80	804,3	1329,0	3051,8	4502,5	1850,0	2673,7	1647,0	2629,7	225,2
300	30	412,7	644,8	1159,4	1405,3	979,5	1189,9	712,7	908,1	32,3
300	60	1690,5	2800,1	3450,4	4774,9	2954,4	3998,7	2477,2	3557,2	215,3
300	90	3260,6	5164,7	6829,5	10514,6	5966,6	8582,0	5603,2	8151,1	605,3
300	120	4453,0	8216,8	11458,5	17517,7	10329,5	15842,3	9740,2	15672,3	1184,6
400	40	1208,7	2044,7	3202,7	4066,2	2790,0	3371,0	2469,3	3323,3	106,1
400	80	5266,5	8077,1	10464,3	15275,1	10048,6	13772,0	9651,9	14833,6	682,5
400	120	10097,5	16620,7	22038,36	33594,6	19201,3	28881,7	20210,3	32205,1	1346,0
400	160	15750,3	26943,4	32843,6	52042,6	30010,7	45161,1	32217,8	51497,7	5064,3
500	50	2934,9	4555,8	6988,6	9171,3	5625,8	7452,6	5122,4	7304,3	254,5
500	100	11921,7	19200,8	26513,3	39343,5	22781,3	32888,1	26128,9	31542,5	1611,3
500	150	29051,4	65082,3	55898,1	88158,7	49617,7	76443,5	54392,6	76068,6	7922,9
500	200	39043,3	86247,0	84678,4	137503,3	75998,8	117932,6	80252,1	115137,3	28842,6

Tabela 6.13: Tempo de computação (em segundos) dos algoritmos para instâncias do Conjunto 3

Análise Probabilística

Para participar desta análise foram selecionados os algoritmos `KMD+BLG`, `KMD+BLP`, `KMD2+BLG`, `KMD2+BLP`, `CGh+BLG` e `CAn+BLG` que obtiveram soluções médias de melhores qualidades ou menor média de tempo de processamento e um subconjunto de instâncias do Conjunto 3.

Neste tipo de experimento, para cada instância executou-se cada algoritmo 100 vezes usando sementes distintas para a geração de números aleatórios. O critério de parada neste caso é um valor alvo, ou seja, o algoritmo é encerrado quando encontra uma solução de valor igual ou maior que um alvo estabelecido.

Em cada execução, o tempo de processamento até que o valor alvo seja alcançado é armazenado. Os tempos são dispostos em ordem crescente e uma probabilidade $p_i = (i - 0,5)/100$ é associada a cada i-ésimo tempo t_i. Com isto, os pontos $z_i = (t_i, p_i)$ são plotados, estabelecendo uma distribuição de probabilidade empírica de tempo para que um algoritmo possa alcançar um valor alvo determinado, como proposto em [2].

Para cada instância foram estabelecidos dois alvos baseados nas soluções obtidas nos testes anteriores: alvo 1 (alvo fácil), que correspondente ao valor da pior solução encontrada na execução de todos os algoritmos e alvo 2 (alvo médio), que é calculado fazendo-se a média dos melhores valores de soluções encontrados. Os alvos são apresentados abaixo (tabela 6.14) e os cálculos para se chegar aos alvos foram feitos com base na tabela 6.15. Nesta tabela, os resultados em negrito são os melhores encontrados.

É necessário ressaltar que as heurísticas `HA_KMD`, `HA_KMD2` e `HA_IMD` tiveram que ser um pouco modificadas para execução deste tipo de teste, no que diz respeito ao procedimento reativo. Isto ocorre porque, já que não há um número fixo de iterações, torna-se impossível trabalhar baseado no número total de iterações. Para resolver esse impasse, o número de iterações em cada bloco foi fixado em 50 e a cada bloco, um tamanho diferente de LRC (α) é utilizado. O reativo é iniciado após a execução de 4 blocos de iterações, caso o valor alvo não tenha sido atingido até

n	m	alvo 1	alvo 2
100	20	1178	1192
100	30	2456	2457
100	40	4141	4142
200	20	1244	1247
200	40	4443	4447
200	60	9425	9435
200	80	16171	16210
300	30	2666	2686
300	60	9652	9676
300	90	20640	20691
300	120	35855	35873

Tabela 6.14: Alvos usados no teste probabilístico

então. A forma para avaliar a qualidade das soluções encontradas em cada bloco é mantida. A cada 200 iterações o reativo é aplicado, realocando-se, de acordo com os valores das soluções obtidas anteriormente, os valores de α que serão utilizados nos 4 próximos blocos de 50 iterações.

Os resultados desta bateria de testes são apresentados nas figuras 6.2 a 6.18.

A probabilidade de que um algoritmo encontre uma solução de valor maior ou igual a determinado valor alvo em um dado tempo de processamento aumenta da esquerda para a direita, ou seja, nestas figuras quanto mais à esquerda estivera curva associada a um algoritmo, melhor será seu desempenho. De acordo com os experimentos realizados, os algoritmos propostos se mostram na seguinte ordem de desempenho: `KMD+BLG`, `KMD+BLP`, `KMD2+BLG` e `KMD2+BLP`.

Tomando-se como exemplo o gráfico apresentado na figura 6.2, a probabilidade de que `KMD+BLG` encontre o valor alvo em 10 segundos é de aproximadamente 85%, caindo para aproximadamente 68% para `KMD+BLP`, 40% para `CGh+BLG`, 35% para `KMD2+BLG`, 23% para `KMD2+BLP` e 18% para `CAn+BLG`.

Ao avaliar as figuras, percebe-se que, os algoritmos `KMD+BLG`, `KMD+BLP` apresentam, na maioria dos casos, uma convergência mais rápida para os valores alvos estabelecidos. Percebe-se, ainda, que as heurísticas propostas seguem um comportamento bastante semelhante para as diferentes instâncias testadas. Este fato, no entanto, parece não ocorrer com os algoritmos da literatura `CGh+BLG` e `CAn+BLG`.

Por exemplo, na figura 6.8 e 6.9 o algoritmo `CGh+BLG` consegue alcançar probabilidades maiores de chegar à solução alvo que os demais algoritmos para maior parte dos tempos. Algo semelhante se observa na figura 6.12, mas, neste caso, `CGh+BLG` passa a apresentar probabilidades ligeiramente superior aos demais algoritmos após decorridos 10 segundos.

Por outro lado, tem-se que no alvo 2 da instância $n = 300$ e $m = 30$ (figura 6.13) a curva não pôde ser plotada pois, até um tempo de computação de 150.000 segundos, o algoritmo não conseguiu atingir o valor alvo em diversas tentativas independentes. Isto volta a ocorrer no alvo 2 da instância $n = 300$ e $m = 60$ (figura 6.15). Em algumas execuções foi possível atingir o alvo, mas o tempo médio para que isto ocorra é de 45.245 segundos. Supondo que em todas as execuções o alvo fosse atingido com este tempo médio seriam necessários mais de 45 dias para que somente este teste pudesse ser realizado.

Analisando agora o desempenho de `CAn+BLG` comparado aos demais algoritmos, observa-se que na maior parte dos experimentos o tempo para alcançar determinado valor alvo ao longo das instâncias avaliadas é maior. Isto somente não ocorre na instância $n = 300$ e $m = 30$ (figuras 6.12 e 6.13) onde `CAn+BLG` consegue alcanccar, para um determinado tempo, maiores probabilidades de obter os valores alvos usados. No alvo 2 da instância $n = 300$ e $m = 120$, o menor tempo para que o algoritmo chegasse ao alvo foi de 175.097 segundos. Como não ocorreram todas as execuções, esta curva não foi plotada.

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	CAn+BLG
100	10	**333**	**333**	**333**	**333**	**333**	**333**	318	318	**333**
100	20	**1195**	**1195**	**1195**	**1195**	**1195**	**1195**	1178	1178	**1195**
100	30	**2457**	**2457**	**2457**	**2457**	**2457**	**2457**	**2457**	**2457**	**2457**
100	40	**4142**	**4142**	**4142**	**4142**	**4142**	**4142**	**4142**	**4142**	**4142**
200	20	**1247**	**1247**	**1247**	**1247**	**1247**	**1247**	1233	1233	1245
200	40	4448	**4450**	4448	4448	4448	4448	4443	4443	4445
200	60	**9437**	**9437**	**9437**	**9437**	**9437**	**9437**	**9437**	**9437**	9425
200	80	16207	16171	**16225**	**16225**	**16225**	**16225**	**16225**	**16225**	16211
300	30	2691	2684	2691	**2694**	**2694**	2686	2666	2666	2691
300	60	**9689**	9688	9684	9679	9681	9681	9677	9677	9676
300	90	20640	20640	20727	**20734**	20728	20733	20725	20725	20685
300	120	35871	35871	**35881**	**35881**	**35881**	**35881**	**35881**	**35881**	35855
400	40	4653	4654	4649	4655	4648	**4658**	4626	4626	4635
400	80	16925	16902	16937	16940	**16956**	16948	16903	16903	16895
400	120	36175	36175	36283	36301	**36315**	36306	36304	36304	36191
400	160	62313	62313	**62483**	62454	62470	62457	62445	62445	62359
500	50	7130	7127	7115	7116	**7131**	7130	7082	7082	7085
500	100	26201	**26254**	26237	26236	26224	26222	26220	26220	26144
500	150	**58605**	56572	56572	56572	56563	56571	56572	56572	56365
500	200	97213	**97344**	97319	97320	97327	97327	97274	97274	97200

Tabela 6.15: Melhores valores de soluções obtidas nas instâncias do Conjunto 3

A tabela 6.16 apresenta o tempo de execução (segundos) necessário para que os algoritmos atinjam as probabilidades estipuladas de alcançar os valores alvo nestes experimentos. Os tempos são baseados no alvo 2.

		KMD+BLG		CGh+BLG		CAn+BLG	
n	m	50%	75%	50%	75%	50%	75%
100	40	4,23	7,37	13,51	25,09	43,66	94,50
200	20	15,83	24,95	51,70	95,30	6,75	12,45
200	40	33,52	58,20	236,62	4495,05	1276,04	2399,05
200	60	33,66	60,00	18,38	25,79	1040,85	1864,31
200	80	4,67	7,99	51,79	90,16	319,82	447,81
300	30	451,18	766,57	?	?	47,93	74,57
300	60	826,40	1292,93	?	?	1707,93	2864,00
300	90	131,05	216,71	275,29	470,82	4511,43	8495,68
300	120	13,13	13,90	18,51	19,32	?	?

Tabela 6.16: Tempo (em segundos) para que probabilidades sejam alcançadas

O símbolo ? significa que o valor alvo 2 somente foi alcançado em um tempo superior a 150 mil segundos ou não foi alcançado.

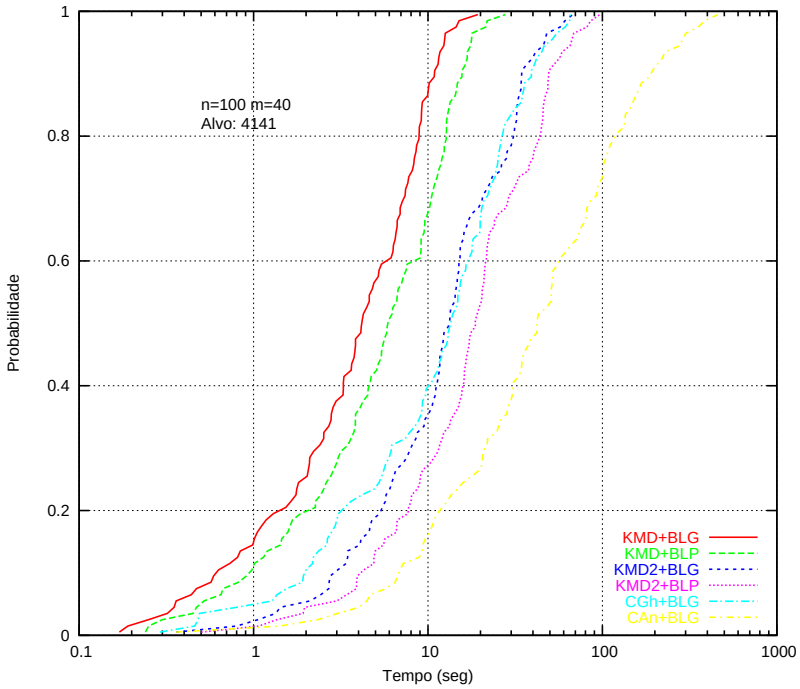

Figura 6.2: Comparação entre os algoritmos GRASP para a instância $n = 100$ e $m = 40$ com alvo igual a 4141

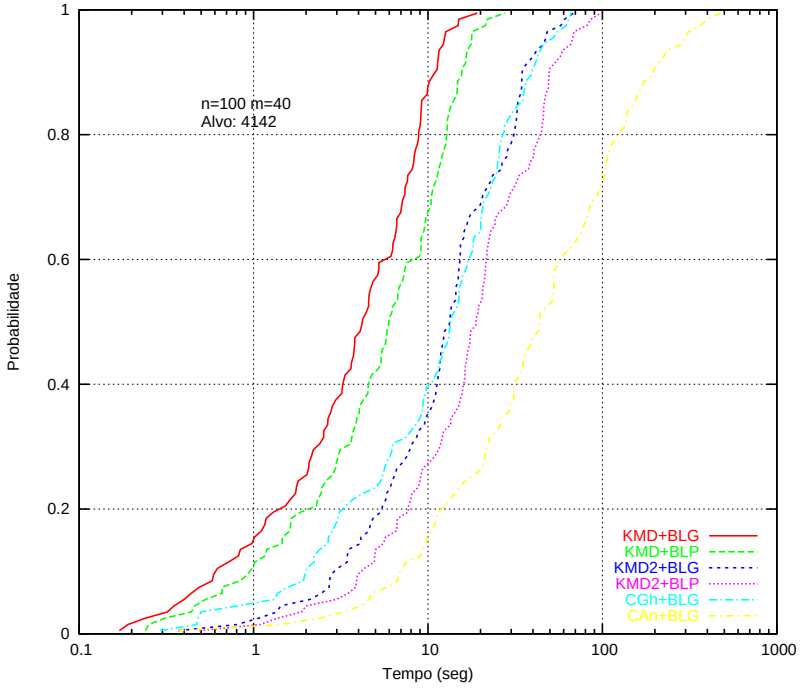

Figura 6.3: Comparação entre os algoritmos GRASP para a instância $n = 100$ e $m = 40$ com alvo igual a 4142

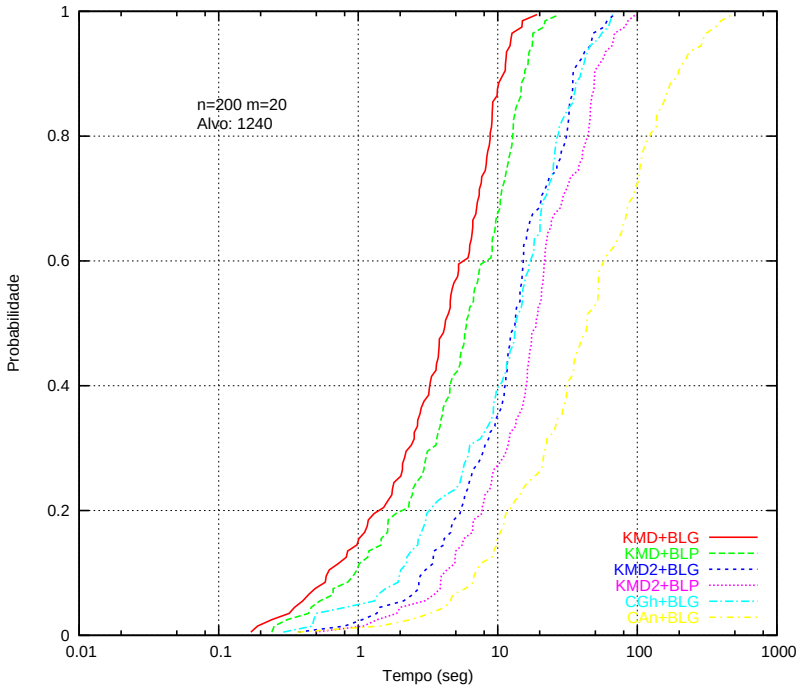

Figura 6.4: Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 20$ com alvo igual a 1240

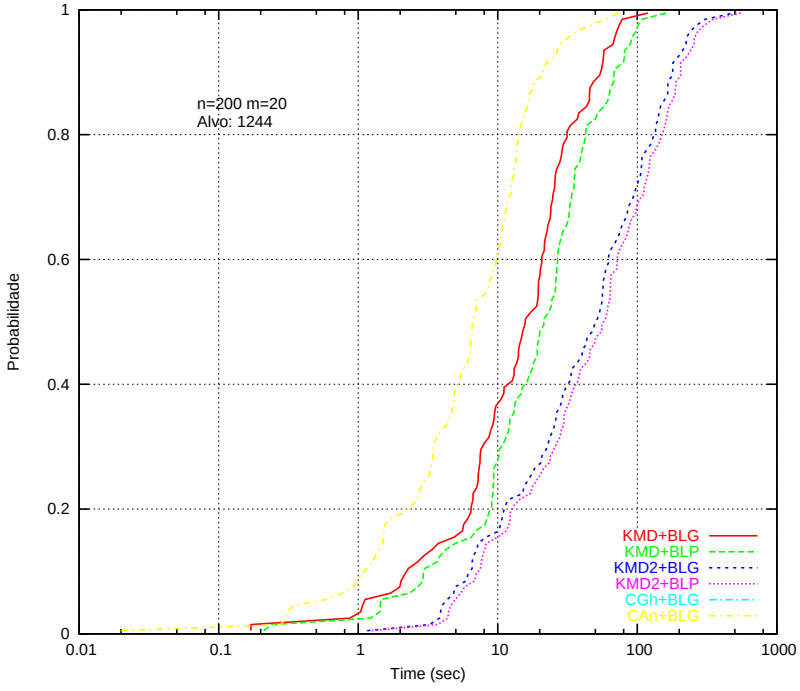

Figura 6.5: Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 20$ com alvo igual a 1244

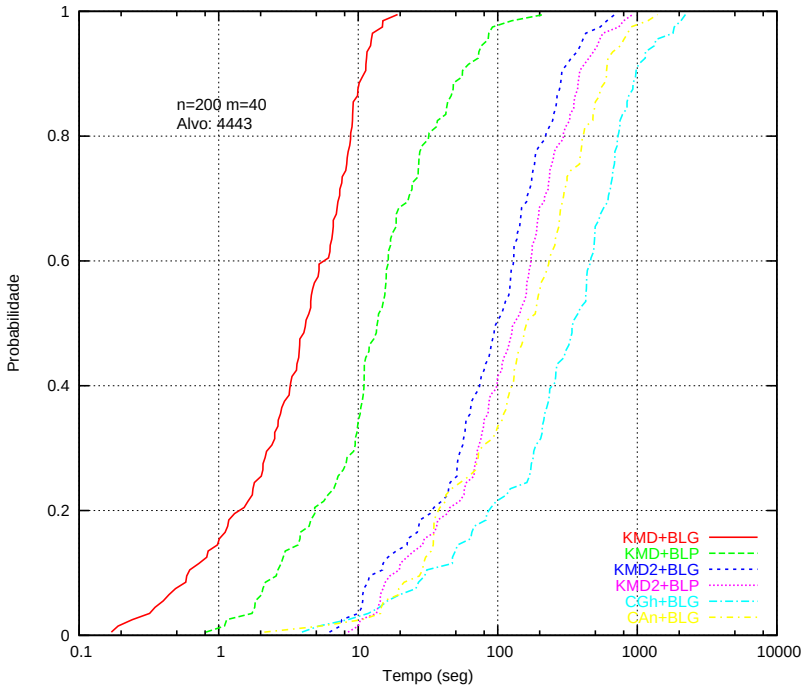

Figura 6.6: Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 40$ com alvo igual a 4443

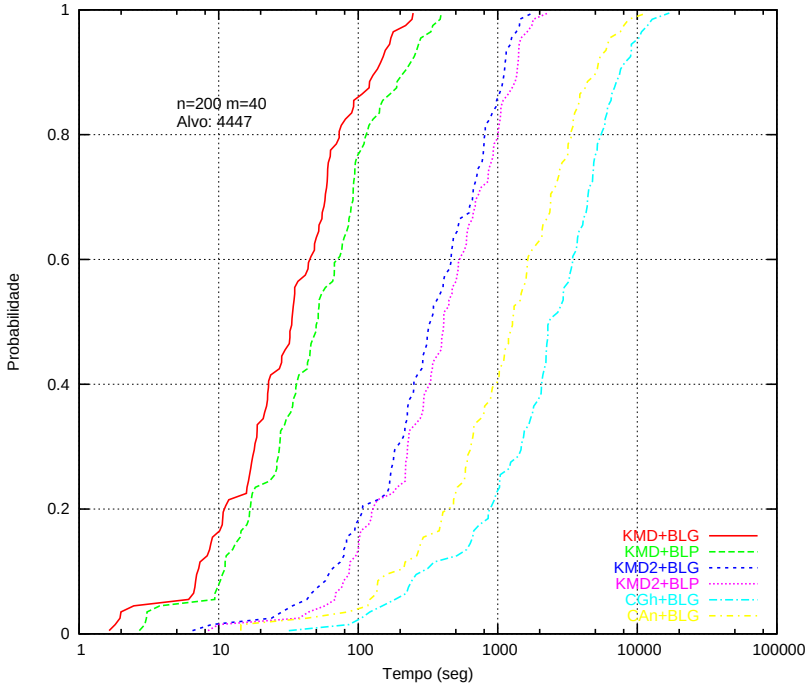

Figura 6.7: Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 40$ com alvo igual a 4447

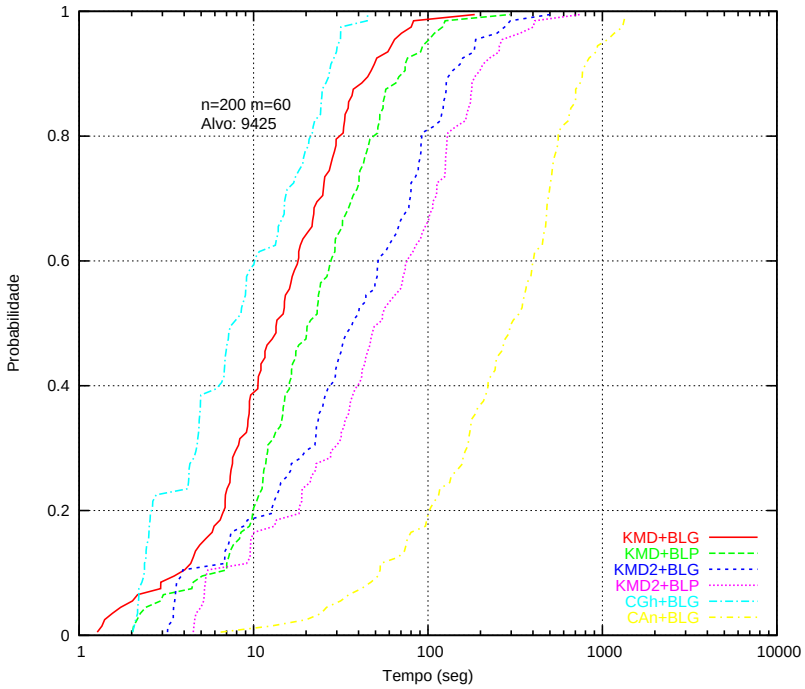

Figura 6.8: Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 60$ com alvo igual a 9425

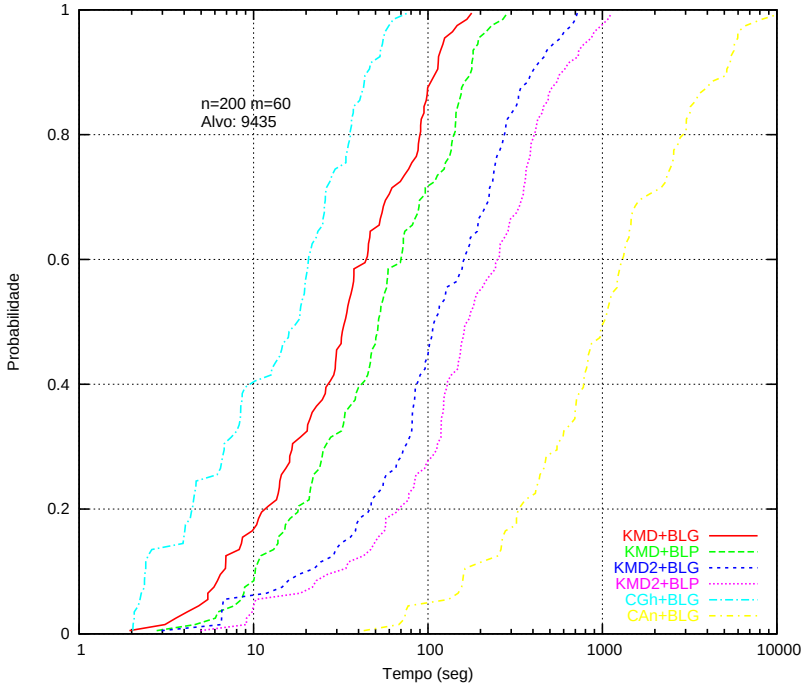

Figura 6.9: Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 60$ com alvo igual a 9435

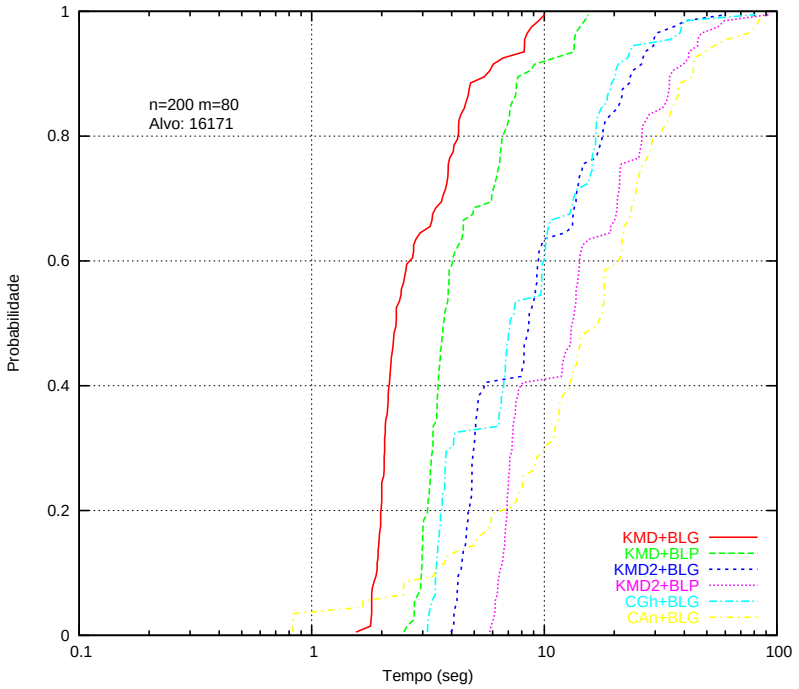

Figura 6.10: Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 80$ com alvo igual a 16171

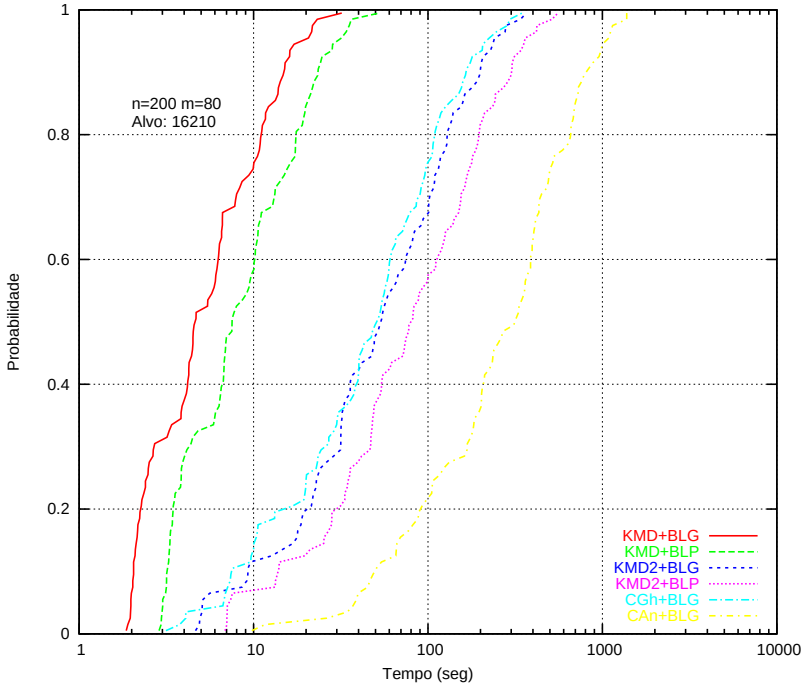

Figura 6.11: Comparação entre os algoritmos GRASP para a instância $n = 200$ e $m = 80$ com alvo igual a 16210

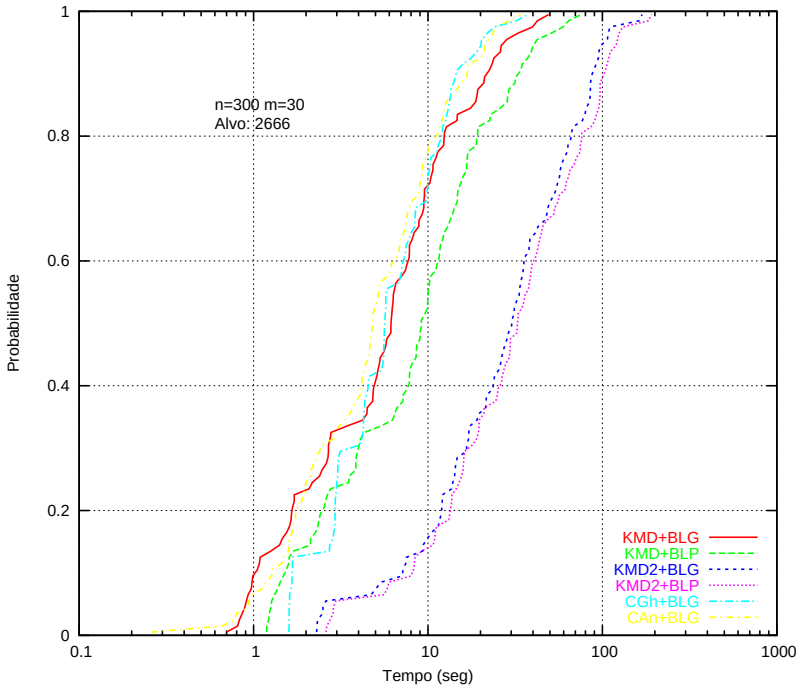

Figura 6.12: Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 30$ com alvo igual a 2666

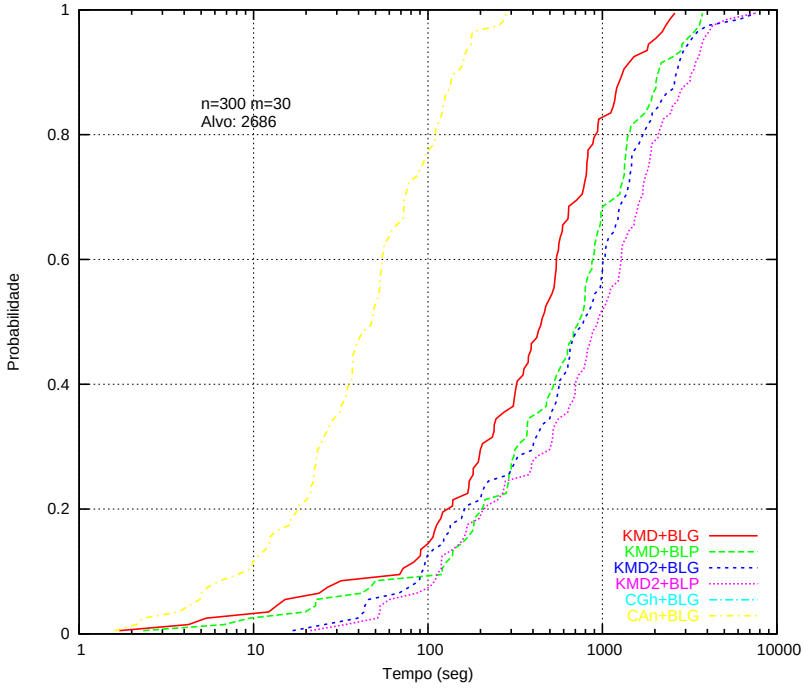

Figura 6.13: Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 30$ com alvo igual a 2686

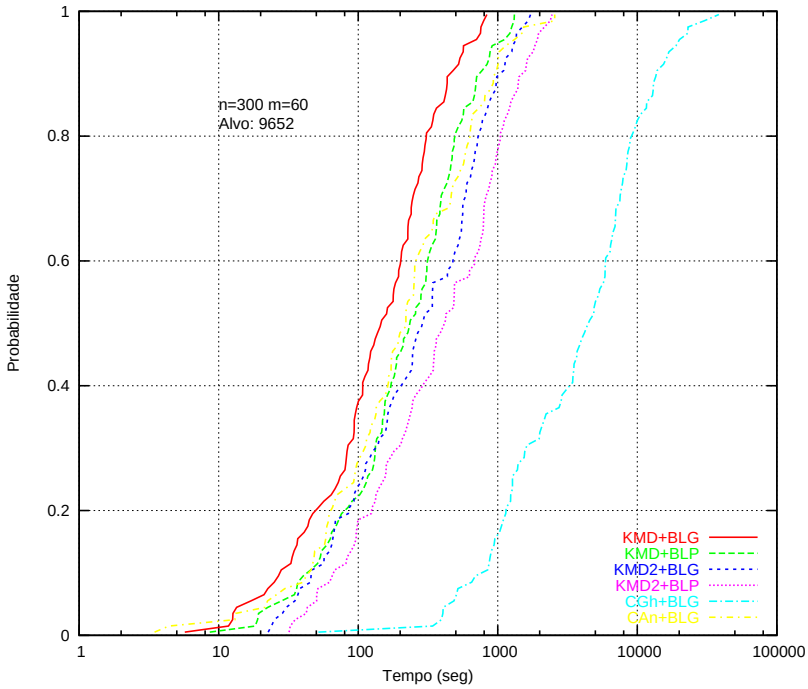

Figura 6.14: Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 60$ com alvo igual a 9652

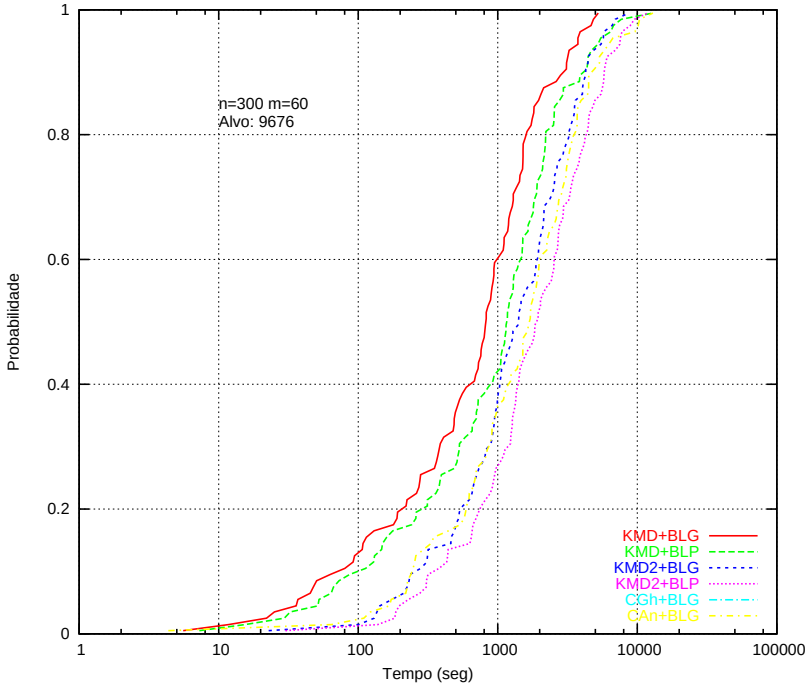

Figura 6.15: Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 60$ com alvo igual a 9676

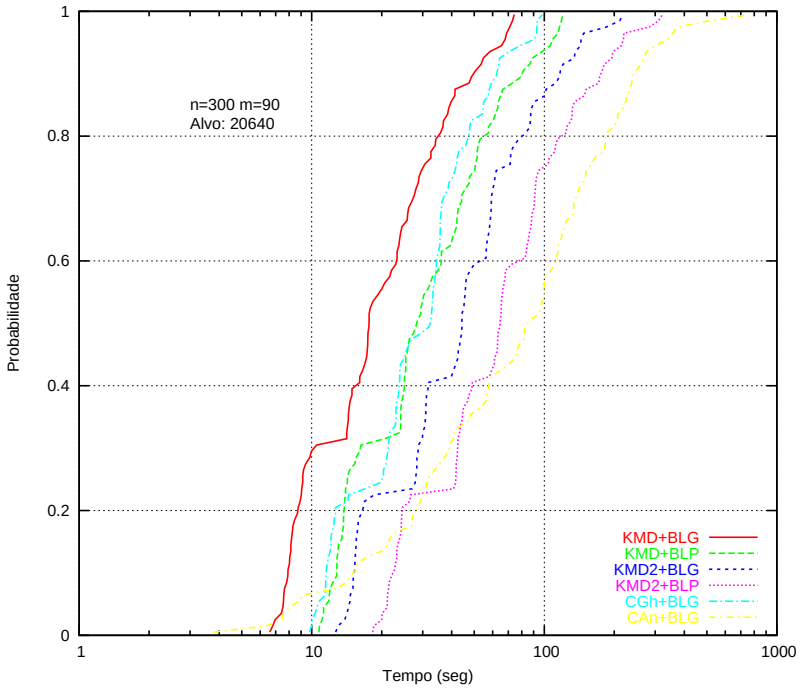

Figura 6.16: Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 90$ com alvo igual a 20640

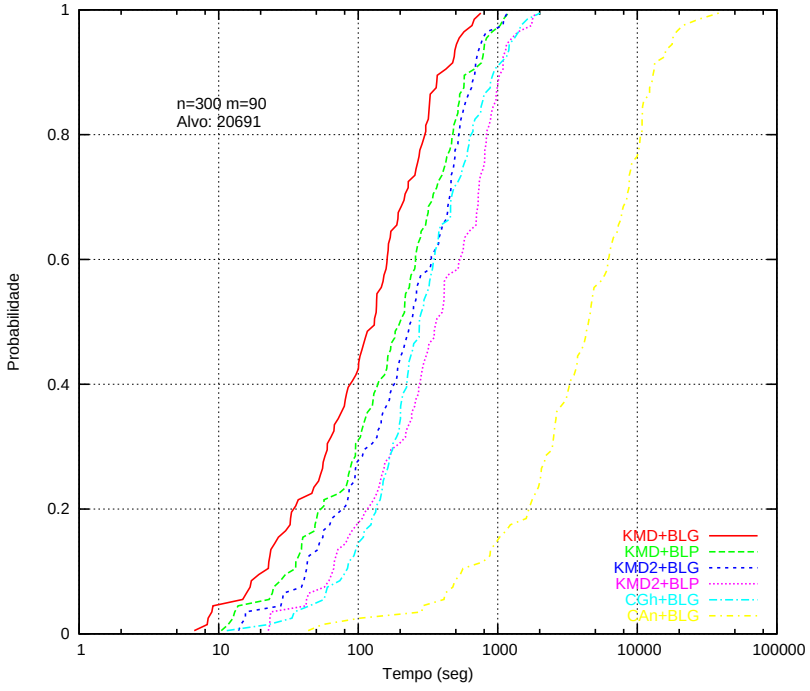

Figura 6.17: Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 90$ com alvo igual a 20691

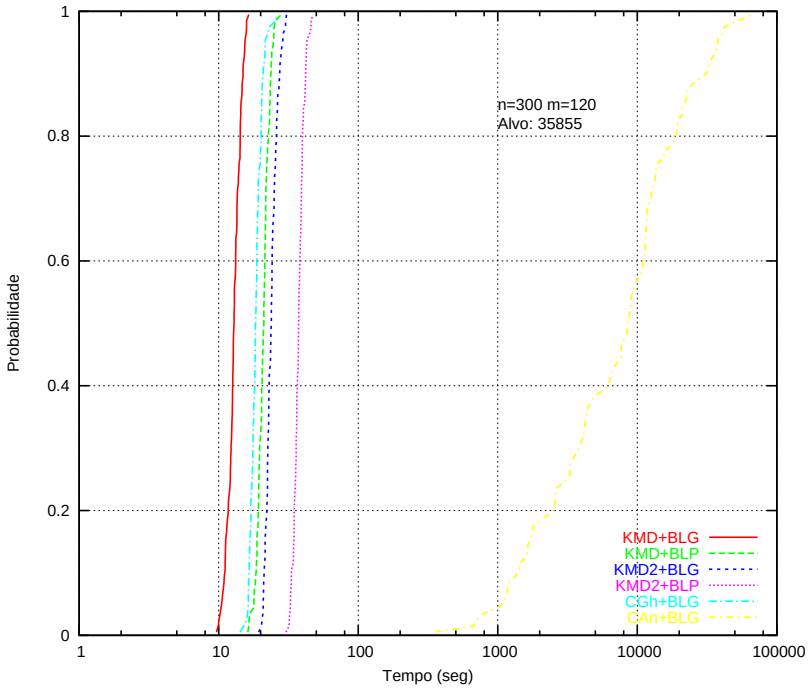

Figura 6.18: Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 120$ com alvo igual a 35855

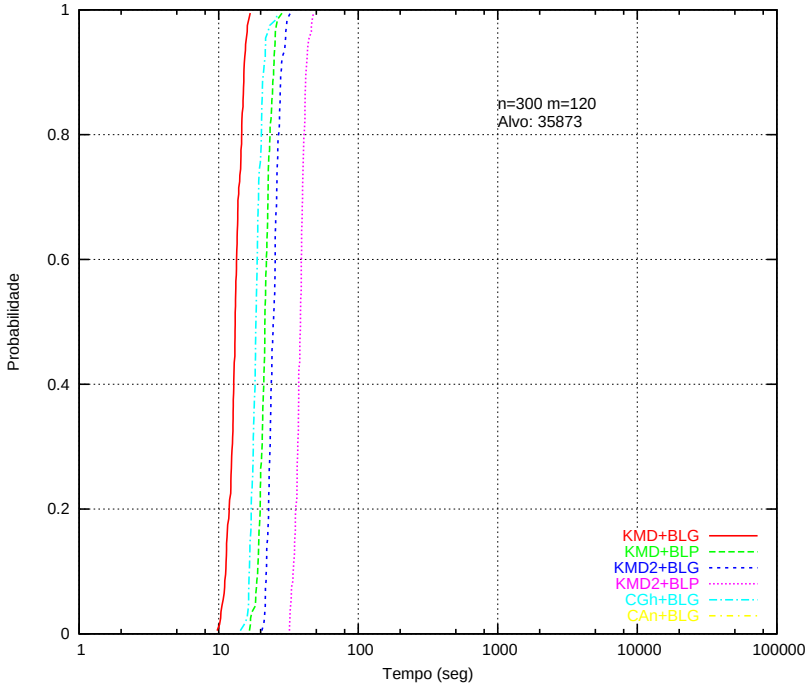

Figura 6.19: Comparação entre os algoritmos GRASP para a instância $n = 300$ e $m = 120$ com alvo igual a 35873

6.3 Considerações

Através dos resultados obtidos nos experimentos realizados pode-se concluir que os algoritmos se mostram extremamente promissores obtendo, nos testes, as melhores soluções para o problema, sobretudo quando se tratam de instâncias de tamanhos maiores comparadas às que foram experimentadas nos trabalhos [3] e [10].

Dentre os algoritmos propostos, destacam-se os algoritmos `KMD+BLG` e `KMD+BLP` por ter os menores tempos de computação e `KMD2+BLG` e `KMD2+BLP` pela qualidade das soluções.

Pode ser observado que os resultados computacionais obtidos pelos algoritmos propostos demostram robustez, que é uma característica importante em boas heurísticas. Isto pode ser visto através da análise probabilística, discutida na seção 6.2.3.

Capítulo 7

Conclusões e Trabalhos Futuros

O objetivo principal desta dissertação foi o de analisar o impacto de diferentes heurísticas de construção e de busca local quando inseridas na heurística GRASP para a solução aproximada do Problema da Diversidade Máxima. Para tanto, foram selecionadas algumas heurísticas da literatura e outras foram propostas no sentido de viabilizar as metas programadas.

Nas heurísticas propostas para o GRASP foi apresentada uma maneira para se utilizar o conceito de GRASP Reativo. Além disso, fez-se uso de um procedimento de filtro da etapa de construção do GRASP. Estas duas estratégias aliadas permitiram que as heurísticas obtivessem resultados satisfatórios.

Uma extensa bateria de testes foi realizada com as diferentes heurísticas GRASP propostas neste trabalho e as da literatura. Nestes experimentos, foram geradas instâncias de dimensões bem maiores que as citadas em [3, 10] e através destes experimentos foi possível mostrar que as heurísticas propostas promoveram uma melhoria na qualidade das soluções encontradas, se comparadas aos algoritmos da literatura.

Foi utilizado o GLPK para obter soluções exatas para parte das instâncias testadas e os algoritmos propostos puderam chegar a soluções de valores de diversidade iguais ao do algoritmo exato em todas estas instâncias. Além disso, para algumas instâncias de porte maior, o GLPK teve seu tempo de processamento limitado em

36.000 segundos. Neste caso, as heurísticas propostas foram capazes de encontrar soluções tão boas ou melhores. Este resultado parece bastante satisfatório pois mostra que para estas instâncias os algoritmos propostos são capazes de achar soluções de valores similares aos obtidos pelo método exato mas com um tempo aproximado de até 346 vezes menor que o do algoritmo exato. Os algoritmos da literatura, no entanto, não foram capazes de chegar ao ótimo em todas as instâncias.

Os gráficos obtidos com o teste probabilístico demostram que os algoritmos propostos se portaram de maneira semelhante ao longo das instâncias testadas e conforme se varia o valor alvo. Como ocorreu que, nos testes realizados, as heurísticas propostas chegaram sempre aos valores alvo estabelecidos, pode-se supor que elas possuem uma característica desejável em heurísticas: robustez. O mesmo parece não ocorrer com os algoritmos da literatura que, em algumas instâncias, foram incapazes de chegar aos alvos.

Dentre os algoritmos propostos ressaltam-se as versões que utilizam as heurísticas de construção `HA_KMD` e `HA_KMD2`. Além disso, os resultados mostram que `KMD+BLG` e `KMD+BLP` apresentam bons resultados sendo os mais rápidos dentre os algoritmos propostos enquanto, `KMD+BLG` e `KMD+BLP` produzem melhores resultados, no entanto, demandam um pouco mais de tempo computacional.

Como sugestões para trabalhos futuros podem ser incluídos os seguintes tópicos:

- uso de outras metaheurísticas (busca tabu, algoritmos evolutivos e outras) para o PDM;
- versões paralelas de metaheurísticas para o PDM;
- estudo de variantes do PDM.

Apêndice A

Publicações Associadas

Alguns resultados parciais desta dissertação foram publicados/submetidos para os seguintes eventos/revistas:

1. Experimental Comparison of Greedy Randomized Adaptative Search Procedures for the Maximum Diversity Problem, SILVA, G. C., OCHI, L. S., MARTINS, S. L., Lecture Notes in Computer Science (LNCS). Springer Verlag, 2004. v. 3059. pp. 498 - 512. (Trabalho apresentado no Third Internacional Workshop on Efficient and Experimental Algorithms, Angra dos Reis, Brasil).

2. O Problema da Diversidade Máxima: Proposta e Análise de Metaheurística GRASP, SILVA, G. C., OCHI, L. S., MARTINS, S. L., Trabalho submetido para Tendências em Matemática Aplicada e Computacional da SBMAC (TEMA) em 2003.

3. O Problema da Diversidade Máxima: Propopsta e Análise de Metaheurística GRASP (Resumo), SILVA, G. C., OCHI, L. S., MARTINS, S. L., Anais do XXVI Congresso Nacional de Matemática Aplicada e Computacional (CNMAC), 2003. (Trabalho apresentado no XXVI Congresso Nacional de Matemática Aplicada e Computacional).

Apêndice B

Tabelas

A seguir são apresentadas outras tabelas geradas no experimentos feitos nesta dissertação.

B.1 Conjunto de Instâncias 2

As tabelas B.1, B.2 e B.3 foram utilizadas como base para cálculo das tabelas apresentadas na seção 6.2.2.

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	And+BLG
50	5	**86733,0**	**86733,0**	**86733,0**	**86733,0**	**86733,0**	**86733,0**	**86733,0**	**86733,0**	**86733,0**
50	10	**334976,0**	**334976,0**	**334976,0**	**334976,0**	**334976,0**	**334976,0**	**334976,0**	**334976,0**	**334976,0**
50	15	**692704,0**	**692704,0**	**692704,0**	**692704,0**	**692704,0**	**692704,0**	**692704,0**	**692704,0**	**692704,0**
50	20	**1171416,0**	**1171416,0**	**1171416,0**	**1171416,0**	**1171416,0**	**1171416,0**	**1171416,0**	**1171416,0**	**1171416,0**
100	10	**353730,0**	**353730,0**	**353730,0**	**353730,0**	**353730,0**	**353730,0**	352286,0	352286,0	**353730,0**
100	20	**1267277,0**	**1267277,0**	**1267277,0**	**1267277,0**	**1267277,0**	**1267277,0**	**1267277,0**	**1267277,0**	**1267277,0**
100	30	**2674152,0**	**2674152,0**	**2674152,0**	**2674152,0**	**2674152,0**	**2674152,0**	**2674152,0**	**2674152,0**	**2674152,0**
100	40	**4544642,0**	**4544642,0**	**4544642,0**	**4544642,0**	**4544642,0**	**4544642,0**	**4544642,0**	**4544642,0**	**4544642,0**
150	15	**772982,0**	**772982,0**	**772982,0**	**772982,0**	**772982,0**	**772982,0**	753041,0	753041,0	**772982,0**
150	30	**2758381,0**	**2758381,0**	**2758381,0**	**2758381,0**	**2758381,0**	**2758381,0**	**2758381,0**	**2758381,0**	**2758381,0**
150	45	5825149,5	**5825682,0**	**5825682,0**	**5825682,0**	**5825682,0**	**5825682,0**	**5825682,0**	**5825682,0**	5820475,0
150	60	**9960461,0**	9960013,0	**9960461,0**	**9960461,0**	**9960461,0**	9960013,0	9959068,0	9960013,0	9954841,0
200	20	1330354,6	1329425,4	1330146,6	1330354,6	**1331076,0**	1330354,6	1321476,6	1321476,6	1325944,0
200	40	**4788086,0**	**4788086,0**	4788078,0	4788078,0	4786265,0	4786273,0	**4788086,0**	**4788086,0**	4787819,0
200	60	**10211323,0**	**10211323,0**	10207669,0	10207669,0	**10211323,0**	**10211323,0**	**10211323,0**	**10211323,0**	10193317,0
200	80	**17544448,0**	**17544448,0**	**17544448,0**	**17544448,0**	**17544448,0**	**17544448,0**	**17544448,0**	**17544448,0**	17526848,0
250	25	**2047828,0**	**2047828,0**	2046995,4	2046995,4	**2047828,0**	**2047828,0**	2041049,0	2041049,0	2042761,3
250	50	7382340,5	7378458,5	7385769,5	7384253,5	7385247,5	7387481,5	7388210,5	**7388997,5**	7357309,6
250	75	15795352,0	15789600,0	**15801103,0**	**15801103,0**	15778641,0	15786333,0	15788177,0	15787915,0	15762014,6
250	100	27161422,0	27162304,0	27159518,0	**27162906,0**	27156670,0	27158770,0	27148842,0	27149402,0	27134802,6

Tabela B.1: Valores dos custos das soluções obtidas nas instâncias do Grupo A

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	And+BLG
50	5	**84517,0**	**84517,0**	**84517,0**	**84517,0**	**84517,0**	**84517,0**	80120,0	80120,0	**84517,0**
50	10	**316409,0**	**316409,0**	**316409,0**	**316409,0**	**316409,0**	**316409,0**	**316409,0**	**316409,0**	**316409,0**
50	15	**659250,0**	**659250,0**	**659250,0**	**659250,0**	**659250,0**	**659250,0**	**659250,0**	**659250,0**	**659250,0**
50	20	**1094343,0**	**1094343,0**	**1094343,0**	**1094343,0**	**1094343,0**	**1094343,0**	**1094343,0**	**1094343,0**	**1094343,0**
100	10	**1094343,0**	**1094343,0**	**1094343,0**	**1094343,0**	**1094343,0**	**1094343,0**	**1094343,0**	**1094343,0**	**1094343,0**
100	20	**1207522,0**	**1207522,0**	**1207522,0**	**1207522,0**	**1207522,0**	**1207522,0**	**1207522,0**	**1207522,0**	**1207522,0**
100	30	**2509045,0**	**2509045,0**	**2509045,0**	**2509045,0**	**2509045,0**	**2509045,0**	**2509045,0**	**2509045,0**	**2509045,0**
100	40	4219476,0	4219476,0	4219476,0	4219476,0	4219476,0	4219476,0	4219476,0	4219476,0	4219476,0
150	15	**737342,0**	**737342,0**	**737342,0**	**737342,0**	**737342,0**	**737342,0**	**737342,0**	**737342,0**	**737342,0**
150	30	**2613286,0**	**2613286,0**	**2613286,0**	**2613286,0**	**2613286,0**	**2613286,0**	**2613286,0**	**2613286,0**	**2613286,0**
150	45	**5519104,0**	**5519104,0**	**5519104,0**	**5519104,0**	**5519104,0**	**5519104,0**	**5519104,0**	**5519104,0**	5515769,0
150	60	**9374611,0**	**9374611,0**	**9374611,0**	**9374611,0**	**9374611,0**	**9374611,0**	**9374611,0**	**9374611,0**	**9374611,0**
200	20	**1294817,0**	**1294817,0**	**1294817,0**	**1294817,0**	**1294817,0**	**1294817,0**	**1294817,0**	**1294817,0**	1291071,0
200	40	**4630545,0**	**4630545,0**	**4630545,0**	**4630545,0**	**4630545,0**	**4630545,0**	**4630545,0**	**4630545,0**	4626180,0
200	60	**9811296,0**	**9811296,0**	**9811296,0**	**9811296,0**	**9811296,0**	**9811296,0**	**9811296,0**	**9811296,0**	9809944,3
200	80	**16759895,0**	**16759895,0**	**16759895,0**	**16759895,0**	**16759895,0**	**16759895,0**	**16759895,0**	**16759895,0**	**16759895,0**
250	25	**1983723,0**	**1983723,0**	**1983723,0**	**1983723,0**	**1983723,0**	**1983723,0**	**1983723,0**	**1983723,0**	1975432,0
250	50	7178042,5	7178042,5	7178042,5	7178042,5	7178042,5	7178042,5	**7178043,0**	7178042,5	7172509,0
250	75	**15303499,0**	**15303499,0**	**15303499,0**	**15303499,0**	**15303499,0**	**15303499,0**	**15303499,0**	**15303499,0**	15299981,3
250	100	**26047022,0**	**26047022,0**	**26047022,0**	**26047022,0**	**26047022,0**	**26047022,0**	**26047022,0**	**26047022,0**	26046496,0

Tabela B.2: Média dos valores das soluções obtidas nas instâncias do Grupo B

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	And+BLG
50	5	**93007,0**	**93007,0**	**93007,0**	**93007,0**	**93007,0**	**93007,0**	**93007,0**	**93007,0**	**93007,0**
50	10	**381379,0**	**381379,0**	**381379,0**	**381379,0**	**381379,0**	**381379,0**	380750,3	**381379,0**	380750,3
50	15	**851353,0**	**851353,0**	**851353,0**	**851353,0**	**851353,0**	**851353,0**	**851353,0**	851344,0	**851353,0**
50	20	**1502908,0**	**1502908,0**	**1502908,0**	**1502908,0**	**1502908,0**	**1502908,0**	**1502908,0**	**1502908,0**	**1502908,0**
100	10	**399449,0**	**399449,0**	**399449,0**	**399449,0**	**399449,0**	**399449,0**	**399449,0**	**399449,0**	398422,0
100	20	**1570800,0**	**1570800,0**	**1570800,0**	**1570800,0**	**1570800,0**	**1570800,0**	**1570800,0**	**1570800,0**	**1570800,0**
100	30	**3475608,0**	**3475608,0**	**3475608,0**	**3475608,0**	**3475608,0**	**3475608,0**	**3475608,0**	**3475608,0**	**3475608,0**
100	40	**6067776,0**	**6067776,0**	**6067776,0**	**6067776,0**	**6067776,0**	**6067776,0**	**6067776,0**	**6067776,0**	**6067776,0**
150	15	**898585,0**	**898585,0**	**898585,0**	**898585,0**	**898585,0**	**898585,0**	896529,3	**898585,0**	896529,3
150	30	**3502567,0**	**3502567,0**	**3502567,0**	**3502567,0**	**3502567,0**	**3502567,0**	3498361,0	**3502567,0**	3498361,0
150	45	**7748205,0**	**7748205,0**	**7748205,0**	**7748205,0**	7747708,0	7747708,0	**7748205,0**	**7748205,0**	**7748205,0**
150	60	**13611261,0**	**13611261,0**	**13611261,0**	**13611261,0**	**13611261,0**	**13611261,0**	13609447,6	**13611261,0**	13609447,6
200	20	**1595768,0**	**1595768,0**	**1595768,0**	**1595768,0**	**1595768,0**	**1595768,0**	1590285,3	**1595768,0**	1590285,3
200	40	**6207580,0**	**6207580,0**	**6207580,0**	**6207580,0**	**6207580,0**	**6207580,0**	6199891,3	**6207580,0**	6199891,3
200	60	**13749403,0**	**13749403,0**	**13749403,0**	**13749403,0**	**13749403,0**	**13749403,0**	13744341,6	**13749403,0**	13744341,6
200	80	**24133320,0**	**24133320,0**	**24133320,0**	**24133320,0**	**24133320,0**	**24133320,0**	24124858,0	24132662,0	24124858,0
250	25	**2488888,0**	**2488888,0**	**2488888,0**	**2488888,0**	2488272,8	2488272,8	2483263,0	2488272,8	2483263,0
250	50	9685273,0	9685352,0	**9685430,0**	**9685430,0**	9685273,0	9685273,0	9668626,6	9685195,0	9668626,6
250	75	**21464360,0**	**21464360,0**	**21464360,0**	**21464360,0**	**21464360,0**	**21464360,0**	21440150,3	**21464360,0**	21440150,3
250	100	**37753120,0**	**37753120,0**	**37753120,0**	**37753120,0**	**37753120,0**	**37753120,0**	37718428,0	**37753120,0**	37718428,0

Tabela B.3: Média dos valores das soluções obtidas nas instâncias do Grupo C

B.2 Conjunto de Instâncias 3

A tabela B.4 apresenta os valores das melhores soluções encontradas pelos algoritmos para instâncias do conjunto 3.

n	m	KMD+BLG	KMD+BLP	KMD2+BLG	KMD2+BLP	IMD+BLG	IMD+BLP	CGh+BLG	CGh+BLP	CAn+BLG
100	10	**333,0**	**333,0**	**333,0**	**333,0**	**333,0**	**333,0**	318,0	318,0	**333,0**
100	20	**1195,0**	**1195,0**	**1195,0**	**1195,0**	**1195,0**	**1195,0**	1178,0	1178,0	1191,7
100	30	**2457,0**	**2457,0**	**2457,0**	**2457,0**	**2457,0**	**2457,0**	**2457,0**	**2457,0**	2454,7
100	40	**4142,0**	**4142,0**	**4142,0**	**4142,0**	**4142,0**	**4142,0**	**4142,0**	**4142,0**	4140,3
200	20	**1247,0**	**1247,0**	**1247,0**	**1247,0**	1245,7	**1247,0**	1233,0	1233,0	1243,0
200	40	4446,3	**4449,3**	4447,0	4448,0	4446,7	4446,0	4443,0	4442,3	4443,3
200	60	**9437,0**	**9437,0**	**9437,0**	**9437,0**	9434,3	**9437,0**	**9437,0**	**9437,0**	9424,0
200	80	16182,3	16170,0	16225,0	16225,0	16224,7	16225,0	16225,0	16225,0	16200,0
300	30	2686,0	2683,0	2688,5	**2692,0**	2684,0	2679,0	2666,0	2666,0	2681,3
300	60	9667,7	9658,0	**9682,5**	9676,0	9678,0	9678,3	9652,0	9644,3	9658,0
300	90	20640,0	20640,0	20726,0	**20727,0**	20721,3	20722,3	20725,0	20725,0	20673,7
300	120	35871,0	35871,0	35878,0	**35881,0**	35880,0	35877,0	35880,0	35880,3	35853,5
400	40	4635,3	4634,7	4647,0	4654,0	4648,0	**4654,7**	4615,0	4611,0	4629,7
400	80	16916,3	16902,0	16930,0	16938,0	**16946,7**	16944,7	16900,5	16905,3	16873,3
400	120	36175,0	36175,0	36272,5	36290,0	36301,0	**36301,7**	36298,7	36301,0	36187,7
400	160	62313,0	62313,0	**62478,0**	62451,5	62450,0	62439,3	62442,3	62432,0	62345,0
500	50	**7124,0**	7116,0	7112,5	7107,0	7110,7	7105,3	7079,3	7079,0	7082,7
500	100	26197,0	26221,0	26229,0	**26229,5**	26213,0	26220,0	26219,0	26219,0	26129,0
500	150	57055,7	56571,7	**56572,0**	56571,0	56549,0	56554,7	56571,0	56571,5	56360,0
500	200	97213,0	**97333,0**	97316,5	97310,0	97316,3	97325,5	97255,0	97322,3	97166,7

Tabela B.4: Médias dos valores das soluções obtidas nas instâncias do Conjunto 3

Referências Bibliográficas

[1] Agrafiotis, D. K. Stochastic algorithms for maximizing molecular diversity. *Journal Chem. Inf. Comput. Sci. 37* (1997), 841–851.

[2] Aiex, R., Resende, M., e Ribeiro, C. Probability distribution of solution time in GRASP: An experimental investigation. *J. of Heuristics 8* (2002), 343–373.

[3] Andrade, P. M. F., Plastino, A., Ochi, L. S., e Martins, S. L. GRASP for the maximum diversity problem. In *Fifth Metaheuristics International Conference (MIC)* (2003).

[4] Bhadury, J., Mighty, E. J., e Damar, H. Maximing workforce diversity in project teams: a network flow approach. *Omega 28* (2000), 143–153.

[5] Cutler, M., e Klastorin, T. The max diversity problem. Relatório técnico, Department of Management Science, University of Washington, 1997.

[6] Dhir, K., Glover, F., e Kuo, C.-C. Optimizing diversity for engineering management. In *Proceedings of the IEEE International Engineering Management Conference* (1994).

[7] Feo, T., e Resende, M. Greedy randomized adaptive search procedures. *J. of Global Optimization 6* (1995), 109–133.

[8] Fleurent, C., e Glover, F. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing 11* (1999), 198–204.

[9] Fogel, D. B. An introduction to simulated evolutionary optimization. *IEEE Trans. on Neural Networks: Special Issue on Evolutionary Computation 5* (1994), 3–14.

[10] Ghosh, J. B. Computational aspects of maximum diversity problem. *Operations Research Letters 19* (1996), 175–181.

[11] Glover, F. Tabu search: A tutorial. *Interfaces 20* (1990), 74–94.

[12] Glover, F., Hersh, G., e McMillan, C. Selecting subsets of maximum diversity. Relatório técnico, University of Colorado at Boulder, 1977.

[13] Glover, F., Kuo, C.-C., e Dhir, K. Heuristic algorithms for the maximum diversity problem. *Journal of Information & Optimization Sciences 19* (1998), 109–132.

[14] Glover, F., Kuo, C.-C., e Dhir, K. S. A discrete optimization model for preserving biological diversity. *Applied Mathematical Modelling 19* (1995), 696–701.

[15] Glover, F., e Laguna, M. *Tabu Search.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[16] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co. Inc., 1989.

[17] Hansen, P. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization* (1986).

[18] Hassin, R., Rubinstein, S., e Tamir, A. Approximation algorithms for maximum dispersion. *Operations Research Letters 21* (1997), 133–137.

[19] Kirkpatrick, S., Gelatt, C. D., e Vecchi, M. P. Optimization by simulated annealing. *Science 220* (1983), 671–680.

[20] Kochenberger, G., e Glover, F. Diversity data mining. Relatório técnico, The University of Mississipi, 1999.

[21] Kuo, C.-C., Glover, F., e Dhir, K. Analysing and modeling the maximum diversity problem by zero-one programming. *Decision Science 24* (1993), 1171–1185.

[22] Laguna, M., e Martí, R. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing 11* (1999), 44–52.

[23] Lea, D. The GNU C++ library. In *C++ Report* (1993).

[24] Makhorin, A. GNU linear programming kit, 2004.

[25] Maser, C. *Ecological Diversity in Sustainable Development: The Vital and Forgotten Dimension*. Saint Lucie Press, 1999.

[26] Mladenovic, N., e Hansen, P. Variable neighborhood search. *Comput. Oper. Res. 24*, 11 (1997), 1097–1100.

[27] Motwani, R. Lecture notes on approximation algorithms: Volume I. Relatório Técnico CS-TR-92-1435, Cambridge University Press, 1992.

[28] Pardalos, P., Pitsoulis, L., Mavridou, T., e Resende, M. Parallel search for combinatorial optimization: Genetic algorithms, simulated annealing and GRASP. In *Parallel Algorithms for Irregularly Structured Problems*, vol. 980 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995, pp. 317–331.

[29] Pearce, D. Economics and genetic diversity. *Futures 19* (1987), 710–712.

[30] Prais, M., e Ribeiro, C. C. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing 12*, 3 (2000), 164–176.

[31] Reinhold, V. N. *Handbook of Genetic Algorithms*. Lawrence Davis, 1991.

[32] Sammon, J. W. A non-linear mapping for data structure analysis. *IEEE Trans. Comput. C-18* (1969), 401–409.

[33] Sidwell, N. How to get the best from `g++`. In *First Annual GCC Developers' Summit* (2003), pp. 229–242.

[34] Unkel, W. C. Natural diversity and national forest. *Natural Areas Journal 5* (1985), 8–13.

[35] Weitz, R., e Lakshminarayanan, S. An empirical comparison of heuristic methods for creating maximally diverse group. *Journal of the operational Research Society 49* (1998), 635–646.

[36] Zhou, S., e Giannakis, G. B. Space-time coded transmissions with maximum diversity gains over frequency-selective multipath fading channels. In *GLOBECOM 2001 - IEEE Global Telecommunications Conference* (2001).