Metaheurística Híbrida GRASP-MD: Novas Aplicações e Paralelização

Luis Filipe de Mello Santos

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 em Computação.

Orientador: Alexandre Plastino de Carvalho

Co-orientadora: Simone de Lima Martins

Niterói, Dezembro de 2006.

Metaheurística Híbrida GRASP-MD: Novas Aplicações e Paralelização

Luis Filipe de Mello Santos

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 em Computação.

Aprovada por:

Prof. Alexandre Plastino de Carvalho / IC-UFF (Presidente)

Profa. Simone de Lima Martins / IC-UFF

Prof. Célio Vinicius Neves de Albuquerque / IC-UFF

Prof. Eduardo Uchoa Barboza / TEP-UFF

Prof. Geraldo Robson Mateus / DCC-UFMG

Niterói, Dezembro de 2006.

Agradecimentos

Esta é mais uma conquista em minha vida e, como todas as outras, não teria conseguido alcançá-la sozinho. Gostaria de agradecer, primeiro, a Deus, por sempre me dar todas as condições que eu preciso para realizar meus objetivos. Também gostaria de agradecer à minha família, que não tenho dúvida de que é a melhor que eu poderia ter. Obrigado pela dedicação e carinho que sempre tiveram comigo, pela educação que me deram e por terem me ensinado a ter caráter.

Gostaria de agradecer também aos meus orientadores, Alexandre Plastino e Simone Martins, pela oportunidade que me deram. A realização desta dissertação foi muito facilitada devido ao bom ambiente de trabalho que me proporcionaram. Além disso, durante este período que trabalhamos juntos, aprendi e amadureci muito. Gostaria de agradecer também ao professor Célio Albuquerque, que contribuiu diretamente e de forma valiosa em parte da realização deste trabalho.

Também devo agradecer a CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pelo apoio financeiro, que foi muito útil para a realização dos meus estudos.

Eu devo um agradecimento especial para minha esposa, Renata, que é uma pessoa doce, boa, esperta, inteligente e que merece muito sucesso. Obrigado pela companhia, pelos momentos felizes e pela enorme disposição que sempre tem para me ajudar em tudo que faço.

Enfim, quero agradecer a todos que contribuíram e me ajudaram de alguma forma na preparação desta dissertação. Muito obrigado!

Resumo da Dissertação apresentada à UFF como requisito parcial para a obtenção do título de Mestre em Computação (M.Sc.)

Metaheurística Híbrida GRASP-MD: Novas Aplicações e Paralelização

Luis Filipe de Mello Santos

Dezembro/2006

Orientadores: Alexandre Plastino de Carvalho e Simone de Lima Martins
Programa de Pós-Graduação em Computação

Metaheurísticas representam uma importante ferramenta para obtenção de soluções de qualidade e em tempo viável para problemas computacionalmente intratáveis. Pesquisas demonstraram que a hibridização destes métodos com outras técnicas tem o potencial de melhorar o desempenho e robustez dos mesmos. Recentemente, foi proposta uma versão híbrida da metaheurística GRASP que incorpora técnicas de mineração de dados, chamada GRASP-MD. A aplicação deste método ao problema de empacotamento de conjuntos alcançou resultados promissores.

Nesta dissertação, objetivou-se avaliar o desempenho do método GRASP-MD para outros problemas de Otimização Combinatória. Dois problemas foram abordados: o problema da maximização da diversidade e o problema de replicação de servidores para transmissão *multicast* confiável. Os resultados demonstraram que o método é capaz de alcançar melhores soluções que o GRASP original em tempos de execução significativamente menores.

A evolução da tecnologia de computação paralela e distribuída tem proporcionado um sensível aumento de poder computacional disponível para aplicações. Outra contribuição importante desta dissertação é a avaliação de implementações paralelas do método GRASP-MD. Foram desenvolvidas versões paralelas da metaheurística híbrida para os dois problemas mencionados e os resultados experimentais evidenciaram a escalabilidade do método em relação à quantidade de processadores utilizados, especialmente quando uma estratégia dinâmica de balanceamento de carga é realizada.

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

The Hybrid Metaheuristic DM-GRASP:
New Applications and Parallelization

Luis Filipe de Mello Santos

December/2006

Advisors: Alexandre Plastino de Carvalho and Simone de Lima Martins
Department: Computer Science

Metaheuristics are among the most important tools for solving computationally intractable problems efficiently. Previous research results demonstrated that the hybridization of these methods with other techniques has the potential to improve their performance and robustness. Recently, a hybrid version of the metaheuristic GRASP that incorporates a data mining process, called DM-GRASP, was proposed. The application of this method to the Set Packing Problem achieved promising results.

One of the goals of this work was to evaluate the performance of DM-GRASP in the context of other Combinatorial Optimization problems. Two problems were considered: the maximum diversity problem and the problem of server replication for reliable multicast transmission. The results demonstrated that the method is capable of achieving better solutions than the original GRASP. In addition, the execution times are significantly reduced.

The evolution of parallel and distributed computing technology provided a great increase in computational power available to applications. Another contribution of this work is the evaluation of parallel implementations of DM-GRASP. Parallel versions of the hybrid metaheuristic were developed for both problems mentioned earlier. Experimental results evidenced the method's scalability in relation to the number of processors used, especially when a dynamic load balancing strategy is implemented.

Glossário

GRASP	:	*Greedy Randomized Adaptive Search Procedures*
GRASP-MD	:	GRASP Híbrido com Mineração de Dados
LRC	:	Lista Restrita de Candidatos
MCF	:	Mineração de Conjuntos Freqüentes
PMD	:	Problema da Maximização da Diversidade
KLD	:	*K Largest Distances* (K Maiores Distâncias)
nMM	:	Estratégia dos n Maiores Padrões Maximais

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Em Ciência da Computação, problemas são considerados intratáveis quando não se conhece algoritmo capaz de resolvê-los em tempo polinomial com relação ao tamanho da instância deste problema. Neste contexto, algoritmos de complexidade polinomial são considerados eficientes, o que significa que o tempo de processamento necessário para que eles alcancem a solução do problema é considerado tolerável.

Por serem considerados intratáveis, os problemas **NP-Completos** e **NP-Difíceis** [21, 36] têm sido alvos de grande interesse por parte da comunidade científica em computação. Muito esforço tem sido dedicado na busca por algoritmos viáveis para estes problemas. Várias técnicas eficientes, apesar de não exatas, foram propostas. Muitas delas específicas para determinados problemas, outras de características mais genéricas. Dentro destas últimas se destaca o conjunto de métodos conhecidos como metaheurísticas, que são métodos de propósito geral e que já alcançaram grande sucesso na obtenção de soluções ótimas, ou próximas à ótima, para muitos problemas intratáveis [48].

Dentre as metaheurísticas mais conhecidas se encontram: Busca Tabu [24, 25, 28], Algoritmos Genéticos ou Evolutivos [31, 35], *Simulated Annealing* [38], *Variable Neighborhood Search* [45], *Scatter Search* [29], Colônias de Formigas [15], *Greedy Randomized Adaptive Search Procedures* (GRASP) [50], entre outras. Me-

taheurísticas, em geral, possuem princípios básicos distintos. Por exemplo, os Algoritmos Genéticos, criados por John Holland na década de 70 [35], são inspirados no modelo de seleção natural proposto por Charles Darwin em meados do século XIX para descrever a lógica de sobrevivência das espécies. Colônias de Formigas, por outro lado, são inspirados na forma de cooperação com a qual formigas encontram o caminho até seus alimentos.

Metaheurísticas criadas a partir da combinação de técnicas diferentes, conhecidas como metaheurísticas híbridas, têm recebido atenção especial. Esta combinação tem o potencial de explorar as boas características das técnicas envolvidas e, ao mesmo tempo, amortizar possíveis pontos fracos das técnicas, fazendo com que os métodos resultantes sejam mais robustos. Diversas metaheurísticas híbridas já foram propostas, sendo, em geral, resultantes da combinação entre metaheurísticas clássicas [59]. Porém, também foram propostas metaheurísticas híbridas que integram uma metaheurística clássica com outros tipos de técnicas. Um destes métodos, recentemente proposto e objeto de estudo deste trabalho, é o GRASP híbrido com mineração de dados [53, 54, 55], denominado GRASP-MD.

GRASP (*Greedy Randomized Adaptive Search Procedures*) [50] é uma metaheurística que já foi aplicada com sucesso a diversos problemas [19]. Neste método, a busca por soluções é feita através de um processo iterativo, no qual cada iteração é composta de duas fases: construção e busca local. Na fase de construção, uma solução completa é gerada. Esta solução não representa garantidamente um ótimo local, o que serve de motivação para a realização da fase seguinte. Na fase de busca local, a vizinhaça desta solução é explorada até que se obtenha uma solução que seja ótima localmente. As iterações se repetem até que um critério de parada seja atingido e a melhor solução encontrada é retornada como resultado.

Mineração de dados é um tema multidisciplinar que tem em sua essência o desenvolvimento de processos para extração automática de informações úteis, implícitas em bancos de dados, sob a forma de regras e padrões [33]. Em [53, 54, 55], foi proposta a integração de técnicas de mineração de dados à metaheurística GRASP, com o objetivo de inserir aprendizado de máquina a esta metaheurística de modo a

torná-la mais eficiente. Para isto, um algoritmo de mineração de dados seria utilizado para reconhecer padrões de soluções sub-ótimas, e estes padrões seriam usados para guiar a busca por melhores soluções. O método resultante, chamado GRASP-MD, foi aplicado ao Problema de Empacotamento de Conjuntos (*Set Packing Problem*) [12] e os experimentos realizados demonstraram resultados promissores.

Uma das principais contribuições desta dissertação é a aplicação da metaheurística híbrida GRASP-MD a dois outros problemas de otimização combinatória, com o objetivo de comprovar sua eficácia e robustez. Um destes problemas é o Problema de Maximização da Diversidade (PMD) [26], que consiste em identificar, em uma população, um conjunto de indivíduos que seja o mais diverso possível. Uma implementação da metaheurística GRASP proposta em [58] alcançou excelentes resultados para o PMD, e foi escolhida como base para elaboração do método híbrido. O outro problema pertence à área de Redes de Computadores, mais precisamente à área de transmissão do tipo *multicast* confiável. Este problema consiste em selecionar, a partir de um conjunto de nós da rede, quais devem atuar como servidores de réplica para que se forneça um serviço de transmissão *multicast* confiável que utilize melhor os recursos desta rede. A implementação do GRASP-MD para este problema tomou como base uma implementação do GRASP proposta em [42]. A escolha destes dois problemas como casos de teste foi feita pois ambos satisfazem uma propriedade necessária para a utilização da versão do método híbrido GRASP-MD avaliada nesta dissertação: as soluções podem ser representadas por conjuntos de números inteiros. A razão pela qual esta propriedade deve ser satisfeita será apresentada no Capítulo 2.

Pesquisadores da área de otimização combinatória perceberam que o elevado poder computacional trazido com avanço no desenvolvimento de arquiteturas paralelas e distribuídas, como *clusters* e *grids* computacionais, pode ser usado para a obtenção de soluções para problemas intratáveis de forma muito mais rápida que nos modelos seqüenciais habituais. A utilização deste tipo de recurso computacional possibilita também a obtenção de soluções de problemas maiores e mais difíceis. Portanto, a paralelização dos métodos existentes para a solução de problemas intratáveis parece ser um caminho natural a ser seguido.

Outra contribuição importante deste trabalho é o desenvolvimento de versões paralelas da metaheurística híbrida GRASP-MD. Objetiva-se avaliar o desempenho destas versões paralelas e demonstrar a escalabilidade do método quando executado em ambientes computacionais paralelos e distribuídos.

O restante deste trabalho está organizado da seguinte forma.

No Capítulo 2, a metaheurística GRASP é descrita e uma breve introdução aos conceitos de mineração de dados é fornecida. Em seguida, apresenta-se a metaheurística híbrida GRASP-MD.

No Capítulo 3, é apresentada a definição do Problema da Maximização da Diversidade. Realiza-se uma revisão da implementação do GRASP proposto em [58] e propõe-se a implementação do GRASP-MD. Resultados computacionais são discutidos a fim de comparar os dois métodos.

O Capítulo 4 é dedicado ao problema de replicação de servidores para transmissão *multicast* confiável. Sua definição é apresentada e, em seguida, as implementações do GRASP proposto em [42] e do GRASP-MD são descritas. Também são apresentados os resultados computacionais, visando a comparação dos dois métodos.

No Capítulo 5, são apresentadas as versões paralelas do GRASP-MD. Experimentos computacionais são realizados para comparar as versões seqüenciais e paralelas desenvolvidas para os dois problemas abordados.

Finalmente, as conclusões obtidas a partir dos resultados alcançados são relatadas no Capítulo 6. Além disso, são sugeridas algumas possíveis direções para futuras pesquisas relacionadas aos temas abordados.

Capítulo 2

A Metaheurística GRASP Híbrida com Mineração de Dados

Neste capítulo, descreve-se a versão híbrida da metaheurística GRASP que incorpora técnicas de mineração de dados, proposta em [53, 54, 55]. Inicialmente, na Seção 2.1, a versão original da metaheurística GRASP será apresentada. A seguir, na Seção 2.2, alguns conceitos sobre mineração de dados serão revistos. Na Seção 2.3, será apresentado o GRASP híbrido, chamado GRASP-MD. Nesta Seção, as diferentes estratégias de implementação do método híbrido, avaliadas em [53, 54, 55], também serão descritas.

2.1 Greedy Randomized Adaptive Search Procedures

A metaheurística GRASP, originalmente proposta em [17], é um método para obtenção de soluções de boa qualidade para problemas de Otimização Combinatória. Desde sua criação, esta metaheurística tem sido utilizada, com considerável sucesso, na obtenção de boas soluções em problemas intratáveis de diversas

áreas [19].

Algoritmos gulosos, em geral, constroem soluções iterativamente. O elemento escolhido para ser incorporado à solução em cada iteração é sempre aquele que leva ao maior incremento possível na qualidade da solução. Apesar de, em muitas vezes, esta estratégia levar a boas soluções, em geral, não atingem soluções ótimas. Métodos completamente aleatórios, nos quais a escolha do elemento a ser incorporado à solução é feita aleatoriamente, apesar de alcançarem soluções de baixa qualidade média, podem ser executados consecutivas vezes e, com isto, diversificar as soluções obtidas. O GRASP pode ser entendido como sendo um método que está em um ponto intermediário destes dois extremos, procurando não ser tão restritivo quanto os métodos gulosos e nem tão desfocado quanto métodos completamente aleatórios.

O GRASP é um processo iterativo, no qual em cada iteração é gerada uma solução para o problema. Cada iteração possui duas etapas: a primeira chamada **Fase de Construção** e a outra chamada **Fase de Busca Local**. Na fase de construção, uma solução completa é gerada. Esta solução não é necessariamente um ótimo local e, por isso, a fase de busca local é realizada, quando a vizinhança da solução construída é percorrida em busca por melhores soluções. As iterações do GRASP se repetem até que um critério de parada seja atingido. Em geral, é utilizado como critério de parada um número fixo de iterações, um valor de solução alvo ou um tempo limite de processamento. A melhor solução encontrada é retornada como resultado.

A Figura 2.1 apresenta o pseudo-código do GRASP. A variável $melhor\_sol$, utilizada para armazenar a melhor solução encontrada até o momento, é inicializada na linha 1 com um conjunto vazio. No laço das linhas 2 a 8, que termina quando o critério de parada adotado é atingido, são executadas as iterações do GRASP. Na linha 3, a fase de construção é executada e a solução gerada sol é encaminhada para a fase de busca local, realizada na linha 4. A avaliação da solução alcançada é comparada com a avaliação da melhor solução até o momento na linha 5 e, caso seja necessário, a melhor solução é atualizada na linha 6. Na linha 9, finalmente, a

melhor solução encontrada é retornada.

```
procedimento GRASP(α)
1.  melhor_sol ← ∅;
2.  repita
3.    sol ← Fase_de_Construcao(α);
4.    sol ← Busca_Local(sol);
5.    se Aval(sol) > Aval(melhor_sol) então
6.      melhor_sol ← sol;
7.    fim se
8.  até criterio de parada satisfeito;
9.  retorne melhor_sol;
```

Figura 2.1: Pseudo-código da metaheurística GRASP

Na fase de construção, uma solução é totalmente construída, elemento por elemento. Suponha que o problema em questão seja um problema de minimização. Suponha também que a avaliação de uma solução s seja medida por uma função $f(s)$. Em um problema de minimização, deseja-se obter uma solução s^*, tal que $f(s^*) \leq f(s)\ \forall s \in S$, onde S é o conjunto de todas as soluções viáveis do problema. Inicialmente, todos os elementos do conjunto $E = \{1, 2, ..., n\}$, que contém todos os possíveis elementos de uma solução, são avaliados de acordo com uma função gulosa $c(x)$, que avalia a contribuição do elemento x à qualidade da solução sendo construída. Os elementos com melhor avaliação, ou seja, aqueles que quando inseridos na solução levam aos menores aumentos em seu custo, formam uma lista chamada **Lista Restrita de Candidatos (LRC)**.

A definição de quantos elementos farão parte da LRC é controlada pelo parâmetro α, que é o principal parâmetro do GRASP. Geralmente, este é usado de duas formas. Na primeira, ele define um número inteiro fixo de elementos que a LRC deve possuir. Na outra, ele é usado para estabelecer um limite de custo, de forma que os elementos que farão parte da LRC correspondam àqueles com contribuição ao custo da solução inferior a este limite[1]. Em geral, é usado um $\alpha \in [0, 1]$, e os elementos integrantes da LRC são aqueles com $c(x) \in [c^{min}, c^{min} + \alpha(c^{max} - c^{min})]$, sendo que c^{max} e c^{min} correspondem às avaliações dos elementos de maior e menor

[1]Em caso de um problema de maximização, os elementos que constituirão a LRC serão aqueles com contribuição ao custo da solução superior a este limite.

contribuição, respectivamente[2]. Note que, com α mais próximo de 0, o método tende a ser mais guloso, e com α mais próximo de 1, o método tende a ser mais aleatório. Dentre os elementos da LRC, apenas um é selecionado para integrar a solução, e esta escolha é feita de forma aleatória.

O procedimento de avaliação dos elementos ainda não presentes na solução, a geração da LRC e a escolha de um elemento para ser integrado à solução, se repetem até que uma solução seja completamente construída. Note que o GRASP tem ao mesmo tempo as características de ser guloso (*greedy*), aleatório (*randomized*) e adaptativo (*adaptive*). Isto se deve aos fatos de apenas os melhores elementos serem considerados como integrantes da LRC, de a escolha de um dos elementos da LRC para ser incorporado à solução ser feita de forma aleatória e de os elementos fora da solução serem reavaliados em cada passo, respectivamente.

A Figura 2.2 apresenta o pseudo-código da fase de construção do GRASP. A variável *sol*, que armazena o conjunto de elementos da solução a ser construída, é inicializada na linha 1 como conjunto vazio. Na linha 2, todos os elementos do conjunto E são avaliados. No laço das linhas 3 a 8, os elementos são incorporados iterativamente à solução até que esta se complete. Na linha 4, a LRC é construída e, na linha 5, um de seus elementos é escolhido aleatoriamente para ser integrado à solução, o que acontece na linha 6. Os elementos ainda não presentes na solução são reavaliados na linha 7. Por fim, na linha 9, a solução construída é retornada.

Uma vizinhança N_{sol} de uma solução *sol* é um conceito comum entre metaheurísticas. Sua definição é baseada em uma operação $\mathcal{O}$ que se realiza em *sol* de forma a gerar uma nova solução. O conjunto de soluções que podem ser geradas a partir de *sol* através de $\mathcal{O}$ constitui a vizinhança N_{sol} de *sol*. Neste contexto, uma solução é caracterizada como um ótimo local quando possui qualidade igual ou superior à qualquer outra solução pertencente a N_{sol}.

A solução gerada na fase de construção não representa, necessariamente, um ótimo local. O objetivo da fase seguinte, a busca local, é exatamente encontrar uma

[2]Em um problema de maximização, a LRC seria constituída dos elementos com $c(x) \in [c^{max} - \alpha(c^{max} - c^{min}), c^{max}]$

```
procedimento Fase_de_Construcao(α)
1.  sol ← ∅;
2.  Avalie c(x) | x ∈ E;
3.  repita
4.    LRC ← Constroi_LRC(α);
5.    s ← Escolha_Aleatoria(LRC);
6.    sol ← sol ∪ {s};
7.    Avalie c(x) | x ∈ E \ sol;
8.  até Solucao_Completa(sol);
9.  retorne sol;
```

Figura 2.2: Pseudo-código da fase de contrução do GRASP original

solução que seja ótima dentro de sua vizinhança. Para isto, a vizinhança da solução obtida na fase de construção é explorada em busca de alguma solução melhor que a original e, caso esta busca obtenha sucesso, ela é repetida tomando como ponto de partida esta nova solução. Caso contrário, conclui-se que um ótimo local foi atingido e a fase de busca local termina.

O pseudo-código da fase de busca local está apresentado na Figura 2.3. Este é constituído de um único laço, das linhas 1 a 3, que realiza, na linha 2, a substituição da solução corrente sol por uma solução melhor sol', pertencente a N_{sol}, até que um ótimo local seja atingido. Na linha 4, este ótimo local é retornado.

```
procedimento Busca_Local(sol)
1.  enquanto ∃ sol' ∈ N_sol | Aval(sol') > Aval(sol) faça
2.    sol ← sol';
3.  fim enquanto;
4.  retorne sol;
```

Figura 2.3: Pseudo-código da fase de busca local do GRASP

Os algoritmos de busca local, em geral, possuem alto custo computacional. É importante notar que quando a solução usada como ponto de partida tem boa qualidade, ou seja, é próxima a um ótimo local, o esforço necessário para que a busca local alcance um ótimo local é reduzido. Por isso, a eficiência da metaheurística GRASP está fortemente relacionada à capacidade da fase de construção de gerar soluções de boa qualidade.

2.2 Mineração de Dados

A evolução da tecnologia de armazenamento de bases de dados, assim como da capacidade de processamento de transações, permitiu um crescimento significativo do volume de dados encontrado em bancos de dados de empresas e instituições de pesquisa. Isto motivou a criação de técnicas capazes de extrair automaticamente informações implícitas nestas bases de dados. Algumas destas informações, possivelmente desconhecidas, têm o potencial de serem extremamente úteis e decisivas em processos de tomada de decisão. A linha de pesquisa e aplicação que estuda técnicas para extração automática de informações e conhecimento a partir de bancos de dados é chamada Mineração de Dados [33].

As técnicas de mineração de dados se dividem em dois grupos principais: as preditivas e as descritivas. As preditivas são usadas para se prever um valor desconhecido ou um evento, com base no histórico dos dados armazenados. As descritivas fornecem informações relevantes sobre os dados, como relações entre valores de atributos.

Os principais tipos de regras e padrões extraídos em processos de mineração de dados são: regras de associação, padrões de seqüência, modelos de classificação, entre outros. A técnica usada na hibridização da metaheurística GRASP proposta em [53, 54, 55] é uma técnica descritiva conhecida como **Mineração de Conjuntos Freqüentes (MCF)**, que é um sub-problema da tarefa de extração de regras de associação [30].

A MCF consiste em identificar, em uma base de dados de transações, todos os subconjuntos de itens que aparecem em pelo menos t transações, chamados conjuntos freqüentes. Este valor t é chamado **suporte mínimo**. O suporte de um conjunto de itens é definido como o número de transações nas quais este está contido. Por exemplo, considere a base de dados ilustrada na Tabela 2.1. Cada transação possui um campo de identificação (ID) e um conjunto de elementos. O conjunto $\{40, 50\}$ aparece em duas transações (2 e 4), portanto, seu suporte é igual a 2. Já o conjunto $\{20, 40\}$ está contido em três transações (1, 2 e 4), tendo, então, suporte

igual a 3.

ID	Itens
1	10, 20, 40
2	20, 40, 50
3	10, 50, 60
4	20, 40, 50, 60

Tabela 2.1: Exemplo de banco de dados de transações

Um conceito importante para a apresentação das estratégias de hibridização adotadas em [53, 54, 55] é o de padrão maximal. Um conjunto freqüente é dito maximal quando nenhum de seus superconjuntos também é um conjunto freqüente. Por exemplo, considerando o banco de dados da Tabela 2.1 e um suporte mínimo de 3, o conjunto $\{20, 40\}$ é um conjunto maximal, já que ele não é subconjunto de nenhum outro conjunto freqüente.

Um dos primeiros algoritmos propostos para realização da MCF, e também um dos mais importantes, é conhecido como *Apriori* [2]. Desde seu aparecimento, vários outros algoritmos foram criados, possuindo características distintas. Entre eles se destacam o DCI [47], FPgrowth [34] e FPmax* [32], sendo que este último realiza uma versão modificada da MCF na qual apenas os conjuntos freqüentes maximais são extraídos.

2.3 A metaheurística híbrida GRASP-MD

A versão básica do GRASP, apresentada na Seção 2.1, sofreu várias extensões nos últimos anos a fim aperfeiçoar seu desempenho [50], tais como: o uso de técnicas de filtragem [18], a escolha automática do valor do parâmetro α (GRASP reativo [49]), o uso de reconexão de caminhos (em inglês, *Path Relinking*) [51], hibridização com outras técnicas [3, 41, 43], entre outras.

A hibridização do GRASP com técnicas de mineração de dados proposta em [53, 54, 55] foi motivada pela ausência de uso de memória de longo prazo no

GRASP original. A inclusão de um módulo de mineração de dados à metaheurística possibilitaria o aproveitamento de informações sobre o histórico de iterações executadas. Este módulo seria responsável pela extração de padrões de soluções com valores de qualidade próximos ao valor ótimo, que seriam usados para guiar a busca por melhores soluções. Na abordagem adotada em [53, 54, 55], uma quantidade significativa de iterações é realizada para construção de um conjunto elite, composto pelas melhores soluções obtidas, que é visto como um banco de dados e suas soluções consideradas transações. A MCF é realizada neste conjunto elite e os conjuntos extraídos são considerados padrões de boas soluções.

A etapa anterior à realização do procedimento de mineração de dados é chamada **Fase de Geração do Conjunto Elite**. Na etapa seguinte, chamada **Fase Híbrida**, as iterações são guiadas pelos padrões encontrados. Nesta etapa, a fase de construção é modificada para usar os padrões encontrados como sendo soluções parciais iniciais. As Figuras 2.4 e 2.5 apresentam os pseudo-códigos do GRASP híbrido e da fase de construção adaptada, respectivamente. A fase de busca local é realizada da mesma forma que no GRASP original.

No algoritmo da Figura 2.4, a variável usada para armazenar a melhor solução encontrada é inicializada na linha 1 como um conjunto vazio. Na linha 2, a variável usada para armazenar o conjunto elite também é inicializada como um conjunto vazio. O laço das linhas 3 a 10 corresponde à fase de geração do conjunto elite. Na linha 4, a fase de construção é executada e a solução construída é encaminhada para a fase de busca local, que é realizada na linha 5. Na linha 6, a função `Atualiza_Conjunto_Elite` tenta integrar a solução encontrada ao conjunto elite. Caso o conjunto elite ainda possua menos soluções que o máximo permitido, que é estabelecido pelo parâmetro tam_elite, a solução é adicionada imediatamente. Caso o conjunto elite já possua o máximo de soluções permitido, a solução só é adicionada caso seja melhor que alguma das integrantes correntes e, neste caso, a pior solução é removida. Na linha 7, a solução obtida nesta iteração é comparada com a melhor solução até o momento. Caso seja necessário, a melhor solução é atualizada na linha 8. A extração de padrões (conjuntos freqüentes) a partir do conjunto elite é realizada na linha 11. O parâmetro $qtd\_padroes$ define a quantidade de padrões

a serem extraídos e o parâmetro *sup_min* representa o suporte mínimo que um conjunto deve ter para ser considerado freqüente. Na linha 12, um dos padrões encontrados é escolhido pela função `Proximo_Padrao`. Esta função seleciona os padrões em um esquema de rodízio (em inglês, *round robin*). O laço das linhas 13 a 20 corresponde à fase híbrida, na qual são executadas iterações que utilizam os padrões extraídos na linha 11. Esta fase possui estrutura semelhante ao GRASP original, com uma única mudança na fase de construção, que recebe um padrão como sendo uma solução parcial inicial. Note que, na linha 19, um outro padrão é selecionado para ser usado na próxima iteração. A melhor solução encontrada é retornada na linha 21.

```
procedimento GRASP_Hibrido(num_it_conj_elite, tam_elite, qtd_padroes, sup_min, α)
01.  melhor_sol ← ∅;
02.  Conjunto_Elite ← ∅;
03.  para it ← 1 até num_it_conj_elite faça
04.    sol ← Fase_de_Construcao(α);
05.    sol ← Busca_Local(sol);
06.    Atualiza_Conjunto_Elite(Conjunto_Elite, sol, tam_elite);
07.    se Aval(sol) > Aval(melhor_sol)
08.      melhor_sol ← sol;
09.    fim se
10.  fim para
11.  Padroes ← MCF(Conjunto_Elite, qtd_padroes, sup_min);
12.  p ← Proximo_Padrao(Padroes);
13.  repita
14.    sol ← Fase_de_Construcao_Adaptada(p, α);
15.    sol ← Busca_Local(sol);
16.    se Aval(sol) > Aval(melhor_sol)
17.      melhor_sol ← sol;
18.    fim se
19.    p ← Proximo_Padrao(Padroes);
20.  ate Criterio_de_Parada();
21.  retorne melhor_sol;
```

Figura 2.4: Pseudo-código do GRASP híbrido

O algoritmo da Figura 2.5 é semelhante ao algoritmo da Figura 2.2. A

única diferença está na linha 1, na qual a variável usada para armazenar a solução construída é inicializada com os elementos do padrão p recebido como parâmetro.

```
procedimento Fase_de_Construcao_Adaptada(p, α)
1.  sol ← p;
2.  Avalie c(x) | x ∈ E \ sol;
3.  repita
4.    LRC ← Constroi_LRC(α);
5.    s ← Escolha_Aleatoria(LRC);
6.    sol ← sol ∪ {s};
7.    Avalie c(x) | x ∈ E \ sol;
8.  até Solucao_Completa(sol);
9.  retorne sol;
```

Figura 2.5: Pseudo-código da fase de construção adaptada

Implementações deste método híbrido requerem a configuração de alguns parâmetros: o número de iterações da fase de geração do conjunto elite, o tamanho do conjunto elite, o suporte mínimo e a quantidade de padrões a serem utilizados. Em [53, 54, 55], o número de iterações da fase de geração do conjunto elite foi igual à metade do número total de iterações do GRASP híbrido. O tamanho do conjunto elite usado foi dez, o suporte mínimo foi igual a 2 e a quantidade de padrões utilizados foi definida de quatro maneiras diferentes, descritas a seguir.

Foram avaliadas quatro estratégias de utilização de padrões. Na primeira estratégia, um único padrão foi utilizado, o maior padrão encontrado (MP), ou seja, o maior conjunto freqüente. Na segunda, foram usados os maiores padrões de cada suporte (MPS), com suportes mínimos variando entre 2 e d, sendo d o tamanho do conjunto elite. Um padrão era selecionado para cada valor de suporte s, sendo este o maior dentre todos os padrões com suporte s. Na terceira, foram usados os n maiores padrões (nMP), com $n = 10$. Na última, foram usados os n maiores padrões maximais (nMM), também com $n = 10$. Nas três primeiras, o algoritmo usado para extração de padrões foi o DCI [47]. Os padrões maximais da última estratégia foram extraídos pelo algoritmo FPmax* [32]. Desempates na escolha dos padrões foram sempre resolvidos aleatoriamente.

Estas estratégias foram avaliadas em [53, 54, 55] no contexto do problema de empacotamento de conjuntos [12]. A estratégia nMM foi a que obteve melhores resultados em geral. Acredita-se que, com o uso de conjuntos freqüentes maximais, seja alcançada uma diversificação mais efetiva, já que um padrão não é subconjunto de nenhum outro padrão. Nas versões do GRASP-MD desenvolvidas nesta dissertação, esta estratégia também será usada.

Capítulo 3

GRASP-MD para o Problema da Maximização da Diversidade

Este capítulo é dedicado à primeira aplicação da metaheurística híbrida GRASP-MD realizada nesta dissertação. Esta aplicação foi feita para o **Problema da Maximização da Diversidade (PMD)**. Parte do conteúdo deste capítulo foi publicado em [56]. A Seção 3.1 contém a definição deste problema. Na Seção 3.2, o algoritmo **KLD**, que é uma implementação do GRASP para o PMD, proposta em [58], é apresentado. Este algoritmo é usado como base para a hibridização, que é apresentada na Seção 3.3. Resultados experimentais são apresentados e analisados na Seção 3.4.

3.1 O Problema da Maximização da Diversidade

Considere um conjunto de elementos $N = \{n_1, \ldots, n_n\}$, sendo que cada elemento n_i possui um conjunto de l caraterísticas identificadas por n_{ik}, com $k \in L = \{1, \ldots, l\}$. O problema da maximização da diversidade (PMD) [23, 26, 27, 40] consiste em identificar um subconjunto M da população N, de forma que os m elementos de M $(1 < m < n)$ possuam a maior diversidade de características possível

entre eles. A medida de diversidade d_{ij} entre um par de elementos (i, j) é calculada por uma função aplicada em suas características que mede a distância (diversidade) entre i e j. O problema pode ser formulado como:

Maximizar

$$z = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j, \tag{3.1}$$

sujeito a

$$\sum_{i=1}^{n} x_i = m, \tag{3.2}$$

onde x_i é uma variável binária indicando se o elemento n_i é selecionado para ser um membro do conjunto M.

Este problema possui várias aplicações práticas, como, por exemplo, gerenciamento de recursos humanos, avaliação de biodiversidade e montagem de circuitos VLSI [39].

Em [26], foi demonstrado que o PMD pertence à classe de problemas NP-difíceis. Desde então, vários métodos heurísticos foram propostos para obtenção de soluções de boa qualidade em tempo viável. Heurísticas construtivas e destrutivas foram apresentadas em [27], que foram avaliadas usando instâncias com diferentes tamanhos de população (com no máximo 30 elementos). As heurísticas obtiveram resultados próximos (2%) aos obtidos por um algoritmo exato, porém, de forma muito mais rápida. Em [60], foram desenvolvidas heurísticas para encontrar grupos de estudantes com as características as mais diversas possível, como nacionalidade, idade, e nível de escolaridade.

Vários trabalhos foram desenvolvidos usando a metaheurística GRASP para resolver o PMD. Bons resultados foram obtidos para instâncias pequenas do problema em [23]. Um novo procedimento de construção foi desenvolvido em [6], sendo avaliado para instâncias de até 250 elementos. Resultados demonstraram que o método foi capaz de alcançar soluções melhores que o GRASP proposto em [23]. Em [5], foi proposta a incorporação de reconexão de caminhos ao GRASP desenvolvido em [6] e o método resultante foi capaz de alcançar melhores resultados. Em [58], foram propostas várias heurísticas de construção e de busca local, sendo combinadas

para gerar diversos algoritmos GRASP. Foram utilizadas instâncias, criadas pelos autores, com no máximo 500 elementos. Resultados experimentais demonstraram que as estratégias foram capazes de alcançar resultados melhores que os métodos propostos em [6, 23].

Dentre os algoritmos apresentados em [58], a heurística KLD foi a que, em geral, obteve os melhores resultados. Nesta dissertação, esta heurística foi usada como base para a implementação da metaheurística híbrida GRASP-MD.

3.2 A heurística KLD

Devido ao elevado custo computacional da fase de busca local, é importante que a fase de construção seja capaz de lhe fornecer bons pontos de partida. Em [58], foram avaliados vários procedimentos de construção e o que obteve os melhores resultados, em geral, foi o que usou a estratégia KLD (K *Largest Distances* — K Maiores Distâncias). O algoritmo KLD é uma implementação da metaheurística GRASP que usa esta estratégia de construção combinada com a busca local proposta em [23].

O algoritmo KLD possui este nome devido à forma como a LRC é montada: para cada elemento $i \in N$, é calculada a soma s_i dos valores de d_{ij}, $j \in N \backslash \{i\}$, que é utilizada para comparar os elementos. São selecionados para formar a LRC os K elementos com maior valor de s_i, ou seja, os K elementos com as maiores distâncias (diversidades) em relação aos outros elementos. Soluções são construídas selecionando aleatoriamente elementos desta LRC.

O procedimento de construção é baseado em duas técnicas que obtiveram sucesso como extensões do GRASP: a filtragem de soluções construídas [50], e o GRASP reativo [49].

A filtragem de soluções construídas é uma técnica proposta para tentar garantir que as soluções enviadas para a fase de busca local tenham melhor qualidade. A idéia é executar o algoritmo de construção várias vezes, e fornecer para a busca

local apenas a melhor solução gerada.

O GRASP reativo é uma extensão do GRASP que tem por objetivo tornar o parâmetro α auto-ajustável. A idéia é executar um bloco inicial de iterações usando vários valores de α e, em um bloco seguinte de iterações, dar mais prioridade ao uso de valores de α que levaram, em média, a melhores soluções.

A incorporação do conceito de GRASP reativo pela heurística KLD foi feita da seguinte forma. Seja m_it o número total de iterações. O primeiro bloco de iterações B_1 deve conter $0.4m\_it$ iterações. Este bloco é dividido em quatro intervalos de mesmo tamanho, representados por $c_i, i = 1, \ldots, 4$. Quatro valores diferentes de $K \in \{K_1, K_2, K_3, K_4\}$ são avaliados nos quatro intervalos. O valor K_i é usado para todas as iterações do intervalo c_i. Os valores K_i são apresentados na Tabela 3.1, onde $\mu = (n - m)/2$.

Tabela 3.1: Valores de K para o bloco B_1

i	c_i	K
1	$[1, \ldots, 0.1m\_it]$	$m + \mu - 0.2\mu$
2	$(0.1m\_it, \ldots, 0.2m\_it]$	$m + \mu - 0.1\mu$
3	$(0.2m\_it, \ldots, 0.3m\_it]$	$m + \mu + 0.1\mu$
4	$(0.3m\_it, \ldots, 0.4m\_it]$	$m + \mu + 0.2\mu$

Depois da execução da última iteração do bloco B_1, a qualidade das soluções obtidas com a utilização de cada K_i é avaliada. O valor médio de diversidade zm_i, obtido ao longo das iterações de cada intervalo c_i, é calculado. Os valores K_i são armazenados em uma lista LK, sendo ordenados decrescentemente de acordo com seus valores zm_i correspondentes.

O próximo bloco de iterações B_2 deve conter $0.6m\_it$ iterações. Este bloco é dividido em quatro intervalos y_i, cada um com um número diferente de iterações, conforme definido na Tabela 3.2. Para cada intervalo, um valor de K é utilizado, também de acordo com a especificação da Tabela 3.2. Desta forma, os valores K_i que levaram a melhores soluções são usados em um número maior de iterações.

Tabela 3.2: Valores de K para o bloco B_2

i	y_i	K
1	$(0.4m\_it, \ldots, 0.64m\_it]$	$LK[1]$
2	$(0.64m\_it, \ldots, 0.82m\_it]$	$LK[2]$
3	$(0.82m\_it, \ldots, 0.94m\_it]$	$LK[3]$
4	$(0.94m\_it, \ldots, m\_it]$	$LK[4]$

Em cada iteração do GRASP, a técnica de filtragem de soluções é aplicada através da construção de duas soluções, sendo apenas a melhor delas enviada para a fase de busca local.

O pseudo-código incluindo a descrição do procedimento da fase de construção usando a heurística das K maiores distâncias é dado na Figura 3.1. Na linha 1, a avaliação da melhor solução encontrada na execução das max_sol_filtro iterações é inicializado. O valor de K a ser usado para construir a LRC é calculado pelo procedimento det_K na linha 2. Este procedimento define o valor de K implementando o conceito de GRASP reativo descrito anteriormente. Na linha 3, a LRC é construída. Da linha 4 até a linha 15, o procedimento de construção é executado max_sol_filtro vezes e apenas a melhor solução encontrada é retornada, na linha 22, para ser usada como ponto de partida pela fase de busca local. Da linha 6 até a linha 10, uma solução é construída com a seleção aleatória de elementos da LRC. Da linha 11 até a 14, a melhor solução encontrada é atualizada. Se a iteração GRASP pertence ao bloco B_1, a avaliação da solução encontrada é utilizado, na linha 17, para atualizar a avaliação do valor corrente de K. Quando o bloco B_1 termina, os valores K_i são colocados na lista LK, ordenados decrescentemente de acordo com suas avaliações. Este procedimento é realizado na linha 20.

Depois da fase de construção, é executado o algoritmo de busca local proposto em [23]. Neste algoritmo, a vizinhança de uma solução é o conjunto de todas as soluções obtidas com a troca de um elemento na solução por outro que ainda não pertence à solução. A solução corrente M é inicializada com a solução obtida na fase

procedimento constr_KLD($it\_GRASP, m\_it, max\_sol\_filtro, n, m$)

01. $melhor\_aval\_sol \leftarrow 0$;
02. $K \leftarrow det\_K(it\_GRASP, m\_it, LK)$;
03. $LRC \leftarrow Constroi\_LRC(K, n)$;
04. **para** $j = 1, \ldots, max\_sol\_filtro$ **faça**
05. $sol \leftarrow \{\}$;
06. **para** $k = 1, \ldots, m$ **faça**
07. Selecione aleatoriamente um indivíduo e^* da LRC;
08. $sol \leftarrow sol \cup \{e^*\}$;
09. $LRC \leftarrow LRC \setminus \{e^*\}$;
10. **fim para**;
11. **se** $(z(sol) > melhor\_aval\_sol)$ **então**
12. $sol\_constr \leftarrow sol$;
13. $melhor\_aval\_sol \leftarrow z(sol)$;
14. **fim se**
15. **fim para**;
16. **se** $(it\_GRASP < 0.4m\_it)$ **então**
17. $Atualiza\_Sol\_K(K, avaliacao\_sol, z(sol\_constr))$;
18. **fim se**;
19. **se** $(it\_GRASP == 0.4m\_it)$ **então**
20. $LK \leftarrow Constroi\_LK(avaliacao\_sol)$;
21. **fim se**;
22. **retorna** sol_constr

Figura 3.1: Procedimento de construção da heurística KLD

de construção. Para cada $i \in M$ e $j \in N \setminus M$, o melhoramento obtido com a troca de i por j, $\Delta z(i,j) = \sum_{u \in M_{\{i\}}} (d_{ju} - d_{iu})$ é calculado. Se para todo i e j, $\Delta z(i,j) < 0$, a busca local é finalizada, pois nenhuma troca irá melhorar z. Caso contrário, depois que os elementos do par (i,j), que fornece o maior $\Delta z(i,j)$ possível, forem trocados, uma nova solução corrente M é criada e a busca local é reiniciada.

3.3 A implementação do GRASP-MD

A versão híbrida da heurística KLD foi proposta com base nas idéias apresentadas na Seção 2.3.

No problema abordado neste capítulo, o PMD, uma solução é caracterizada por um conjunto de elementos. Conjuntos freqüentes, extraídos a partir de um conjunto de soluções elite, podem ser considerados como padrões de boas soluções e, por isso, podem ser utilizados para guiar a busca por melhores soluções.

As iterações da fase de geração do conjunto elite da versão híbrida são praticamente idênticas às iterações da heurística KLD, apresentada na Seção 3.2, com exceção de que, ao final de cada iteração, a solução gerada é candidata a integrar o conjunto elite. Nesta fase, são executadas n_iter iterações.

Na fase híbrida, também são executadas n_iter iterações. Nesta fase, o procedimento de construção adotado na heurística KLD é adaptado para a utilização dos padrões extraídos, de forma que a solução a ser construída contenha os elementos de um dos padrões.

O conceito de GRASP reativo foi incorporado em ambas as fases do GRASP-MD. Tanto na fase de geração do conjunto elite quanto na fase híbrida, as n_iter iterações são divididas em dois blocos, B_1 e B_2, onde os valores de K são utilizados seguindo a mesma lógica que a utilizada na heurística KLD.

O pseudo-código do novo procedimento de construção, adaptado a partir do procedimento apresentado na Figura 3.1, é apresentado na Figura 3.2. O parâmetro p irá guiar a construção da solução inicial, que irá conter todos os elementos de p. Nas linhas 2 e 3, o valor de K e a LRC são definidos da mesma forma que no algoritmo de construção original. Como no algoritmo da Figura 3.1, a fase de construção é executada max_sol_filtro vezes e apenas a melhor solução encontrada é retornada, na linha 22, para ser usada como ponto de partida da fase de busca local. Na linha 5, a solução inicial é definida com todos os elementos do padrão p. Da linha 6 até a 10, é aplicado o mesmo procedimento das linhas 6 a 10 do

```
procedimento constr_KLD_adaptada(it_GRASP, m_it, max_sol_filtro, n, m, p)
01.  melhor_aval_sol ← 0;
02.  K ← det_K(it_GRASP, m_it, LK);
03.  LRC ← Constroi_LRC(K, n);
04.  para j = 1, ..., max_sol_filtro faça
05.    sol ← p;
06.    para k = 1, ..., (m − |p|) faça
07.      Selecione aleatoriamente um indivíduo e* da LRC;
08.      sol ← sol ∪ {e*};
09.      LRC ← LRC \ {e*};
10.    fim para;
11.    se (z(sol) > melhor_aval_sol) então
12.      sol_constr ← sol;
13.      melhor_aval_sol ← z(sol);
14.    fim se
15.  fim para;
16.  se (it_GRASP < 0.4m_it) então
17.    Atualiza_Sol_K(K, avaliacao_sol, z(sol_constr));
18.  fim se;
19.  se (it_GRASP == 0.4m_it) então
20.    LK ← Constroi_LK(avaliacao_sol);
21.  fim se;
22.  retorna sol_constr
```

Figura 3.2: Procedimento de construção da heurística KLD adaptado para uso de padrões

algoritmo da Figura 3.1 para obter os outros $m-|p|$ elementos da solução. No final, a solução construída irá conter todos os elementos de p e alguns outros elementos selecionados pelo procedimento de construção. Da linha 11 até a 21, são realizadas as mesmas ações de atualização da melhor solução e de escolha do valor de K que as apresentadas nas linhas 11 a 21 do algoritmo da Figura 3.1.

A busca local usada na implementação do GRASP-MD é a mesma que é usada na heurística KLD, descrita na Seção 3.2.

A estratégia de utilização de padrões escolhida foi a nMM, avaliada em [53, 54, 55]. Nesta estratégia, é executado um algoritmo de mineração de conjuntos freqüentes maximais várias vezes, com cada execução usando um valor de suporte mínimo s diferente, $s \in S = \{2, \ldots, max\}$, onde max é o tamanho do conjunto elite. Todos os padrões encontrados (conjuntos freqüentes maximais) em cada execução foram agrupados em um conjunto F e os 10 maiores padrões obtidos de F foram escolhidos para integrar P, o conjunto de padrões utilizado na fase híbrida.

3.4 Resultados Experimentais

Os experimentos computacionais foram realizados em um conjunto de instâncias de teste utilizadas em [58], que possuem características distintas. Foram selecionadas as instâncias maiores e mais difíceis. Cada uma destas instâncias consiste em uma população N com tamanho n e uma matriz de diversidades $MatDiv$, que contém a diversidade d_{ij} entre todo par de elementos distintos i e j de N. De acordo com [58], estas instâncias foram geradas com índices de diversidade d_{ij} (com $i < j$ e $i, j \in N$) selecionados aleatoriamente em uma distribuição uniforme de números entre 0 e 9. As instâncias têm tamanhos de população n iguais a $200, 300, 400$ e 500. Também foi gerada uma instância com tamanho $n = 600$, usando o gerador desenvolvido em [58]. Para todas as instâncias, foram realizados testes para encontrar subconjuntos M de tamanhos correspondentes a 10%, 20%, 30% e 40% do tamanho da população. Partes destes resultados foram publicados em [56].

Os algoritmos foram implementados em C++, compilados com o `g++ 3.2.2` e executados em uma máquina Intel Pentium 4 de 1.7 GHz, com 256 Mb de memória RAM.

A heurística KLD foi executada usando 500 iterações. A implementação do GRASP-MD usou, na fase de geração do conjunto elite, 250 iterações e, na fase híbrida, outras 250. Cada algoritmo foi executado 10 vezes, usando sementes diferentes. O tamanho do conjunto elite, usado no GRASP-MD, foi igual a 10. Este valor foi escolhido porque obteve os melhores resultados em [53, 54, 55]. Como

já mencionado anteriormente, a estratégia de utilização de padrões escolhida foi a nMM. Os conjuntos freqüentes maximais foram minerados do conjunto elite em execuções distintas do algoritmo FPmax* com suporte mínimo variando entre 2 e 10, e apenas os dez maiores foram selecionados para serem utilizados como padrões.

Na Tabela 3.3, compara-se a qualidade das soluções obtidas por ambos os métodos. Quanto maior o valor de diversidade de uma solução, melhor sua qualidade. As duas primeiras colunas contêm os parâmetros que caracterizam as instâncias: o tamanho n da população e o número m de elementos a serem selecionados. As três colunas seguintes contêm os resultados do método KLD e as três últimas, do GRASP-MD. Para cada método, são exibidos a qualidade da melhor solução, a qualidade média e desvio padrão encontrados ao longo das 10 execuções. Para a comparação entre os dois métodos, valores em negrito indicam qual método obteve melhor desempenho.

Considerando as melhores soluções obtidas nos 20 testes, os métodos empataram em 12 casos. Nos outros oito testes, o método GRASP-MD obteve resultados melhores em sete casos. A heurística KLD só conseguiu alcançar resultado melhor em um caso. Considerando valores médios, houve empate em dois casos. Nos outros 18 testes, o método GRASP-MD obteve melhor desempenho em 13 e o KLD, em apenas cinco. Estes resultados demonstram que a metaheurística híbrida apresenta um desempenho superior em relação ao comportamento do KLD, que se mostrou um dos métodos mais eficientes da literatura.

Na Tabela 3.4, é apresentado um comparativo dos tempos de execução dos métodos. Como na Tabela 3.3, as duas primeiras colunas exibem os parâmetros que caracterizam as instâncias. As colunas 3 e 4 contêm os resultados da heurística KLD. Nas colunas 5 e 6, são apresentados os resultados do GRASP-MD. Para cada um dos métodos, são apresentados a média e o desvio padrão dos tempos de execução gastos nas 10 execuções.

Note que o GRASP-MD foi sempre mais rápido que o KLD. A última coluna exibe a redução média de tempo de execução do GRASP-MD em relação ao KLD. Observe que os ganhos variam entre 9.7% e 46.9%. Na média, este ganho foi de

Tabela 3.3: Qualidade das soluções obtidas pelos métodos KLD e GRASP-MD

Instância		KLD			GRASP-MD		
n	m	Melhor	Média	D.P.	Melhor	Média	D.P.
200	20	1247	**1246.6**	1.2	1247	1246.4	1.8
	40	4450	**4450**	0	4450	4449.8	0.6
	60	9437	9437	0	9437	9437	0
	80	16225	16225	0	16225	16225	0
300	30	2691	**2687.3**	3.7	**2694**	2686.8	4.5
	60	9689	9680.9	7.9	9689	**9687.4**	2.6
	90	20743	20735.3	5.6	20743	**20741.6**	2.8
	120	35881	**35880.3**	1.1	35881	35880	1.2
400	40	4655	4651.7	1.8	**4658**	**4656.6**	2.8
	80	16956	16943.1	11.8	16956	**16953.2**	5.6
	120	36317	36301.7	10.9	36317	**36315.1**	2.9
	160	62475	62452.7	15.5	**62487**	**62479.1**	10.5
500	50	7130	7117.3	5.8	**7141**	**7128.8**	9.5
	100	26258	26249.3	7.6	26258	**26255.7**	5.6
	150	56572	**56571.9**	0.3	56572	56571.7	0.6
	200	97344	97329.3	10.1	97344	**97342.6**	2.2
600	60	**10150**	10118.2	15.2	10149	**10147.2**	2.1
	120	37035	37008.8	15.7	**37086**	**37062.7**	17.9
	180	80410	80392.1	11.8	**80416**	**80410.6**	3.3
	240	139048	139028	11.3	**139059**	**139056**	1.6

34.6%. Esta redução substancial no tempo de execução caracteriza uma segunda contribuição do método híbrido.

Para uma investigação mais detalhada do comportamento do GRASP-MD, foram gerados os gráficos das Figuras 3.3 até 3.6. O gráfico da Figura 3.3 ilustra a variação do valor da qualidade das soluções obtidas nas fases de construção e busca local, ao longo das iterações de uma execução para a instância com $n = 400$ e $m = 120$. A Figura 3.4 ilustra os tempos de execução destas mesmas fases, para este

Tabela 3.4: Tempos de execução, em segundos, dos métodos KLD e GRASP-MD

Instância		KLD		GRASP-MD		**Redução**
n	m	Média	D.P.	Média	D.P.	**Média**
200	20	32.6	2.6	**23.2**	0.9	**28.9%**
	40	135.8	3.8	**94.8**	5.6	**30.2%**
	60	339.5	2.1	**195.6**	8.2	**42.4%**
	80	546.8	5.1	**290.3**	5.2	**46.9%**
300	30	223.1	5.3	**201.5**	90.5	**9.7%**
	60	915.2	10.7	**621.7**	22.0	**32.0%**
	90	2013.9	60.5	**1222.7**	25.6	**39.3%**
	120	3224.7	57.2	**1843.1**	66.0	**42.8%**
400	40	792.5	4.7	**569.8**	53.2	**28.1%**
	80	3317.9	22.6	**2216.2**	74.9	**33.2%**
	120	7597.7	62.1	**4497.7**	69.2	**40.8%**
	160	10792.5	97.1	**6961.8**	231.3	**35.5%**
500	50	1976.3	15.6	**1510.9**	43.3	**23.5%**
	100	9018.1	64.1	**5361.0**	270.2	**40.5%**
	150	19247.8	86.4	**10288.8**	148.7	**46.5%**
	200	28495.3	107.5	**16283.7**	274.8	**42.8%**
600	60	4132.5	25.7	**3121.7**	126.2	**24.4%**
	120	17656.8	272.8	**12014.6**	212.1	**31.9%**
	180	39804.8	589.2	**23563.8**	484.6	**40.8%**
	240	54304.9	9434.7	**36915.5**	821.8	**32.0%**
					Média	**34.6%**

mesmo teste. As Figuras 3.5 e 3.6 são análogas às Figuras 3.3 e 3.4 e correspondem a uma execução para a instância com $n = 600$ e $m = 180$. O comportamento observado nos outros testes se mostrou bastante similar a estes.

A observação dos gráficos das Figuras 3.3 e 3.5, que ilustram a qualidade das soluções obtidas, permite concluir que, após a extração dos padrões, a qualidade média das soluções geradas na fase de construção aumenta significativamente. Na

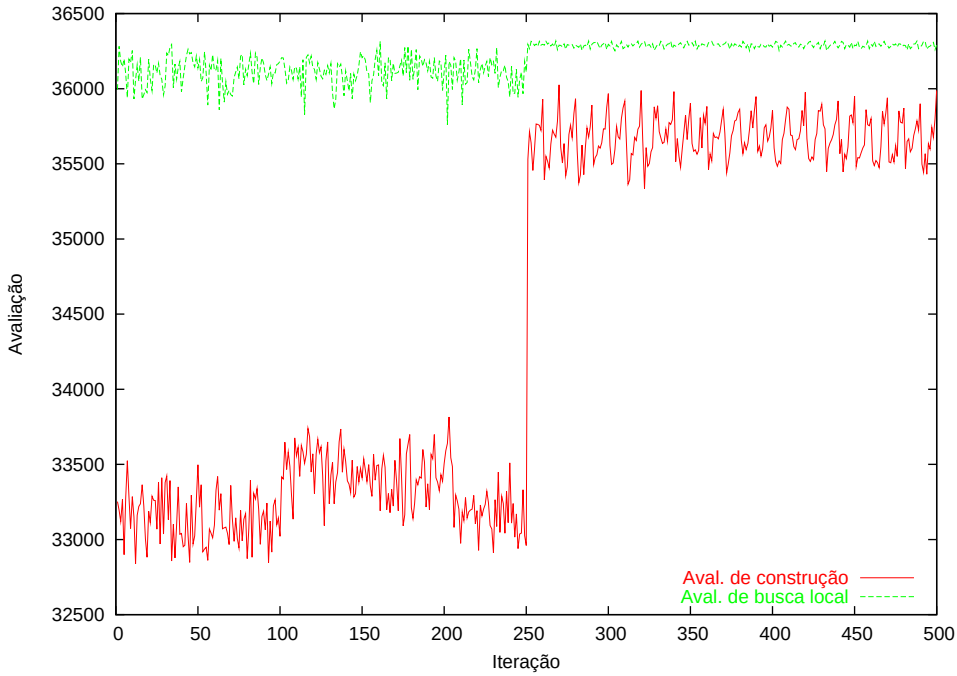

Figura 3.3: Qualidade das soluções alcançadas pelo GRASP-MD ao longo das iterações para a instância com $n = 400$ e $m = 120$ (semente $= 1$)

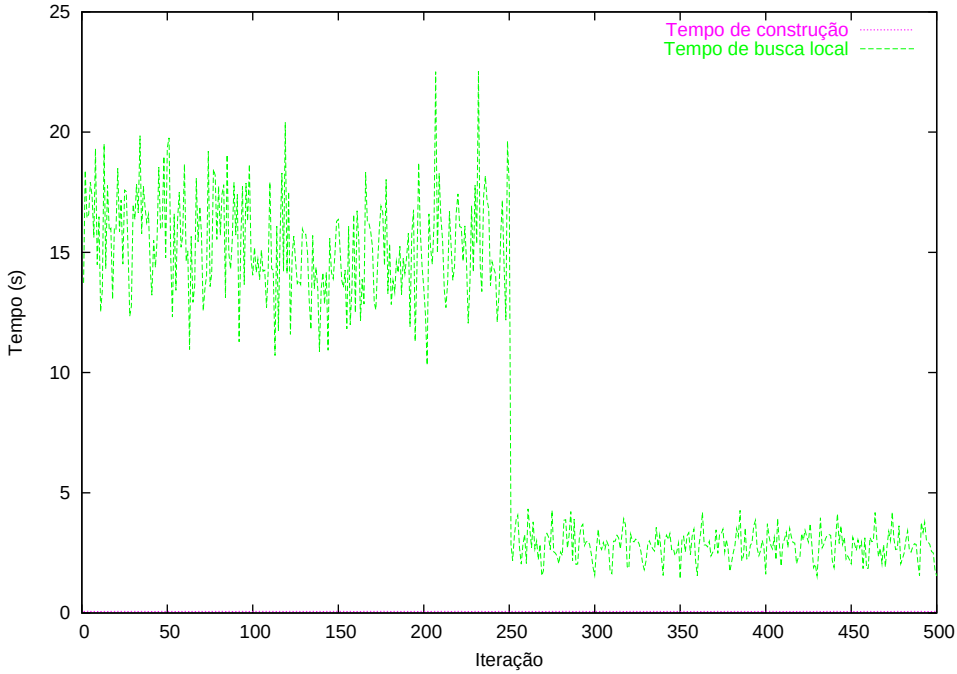

Figura 3.4: Tempos computacionais das fases do GRASP-MD ao longo das iterações para a instância com $n = 400$ e $m = 120$ (semente $= 1$)

fase de busca local, esta média de qualidade também aumenta. Isto demonstra que a utilização de padrões, que representam características de soluções de boa quali-

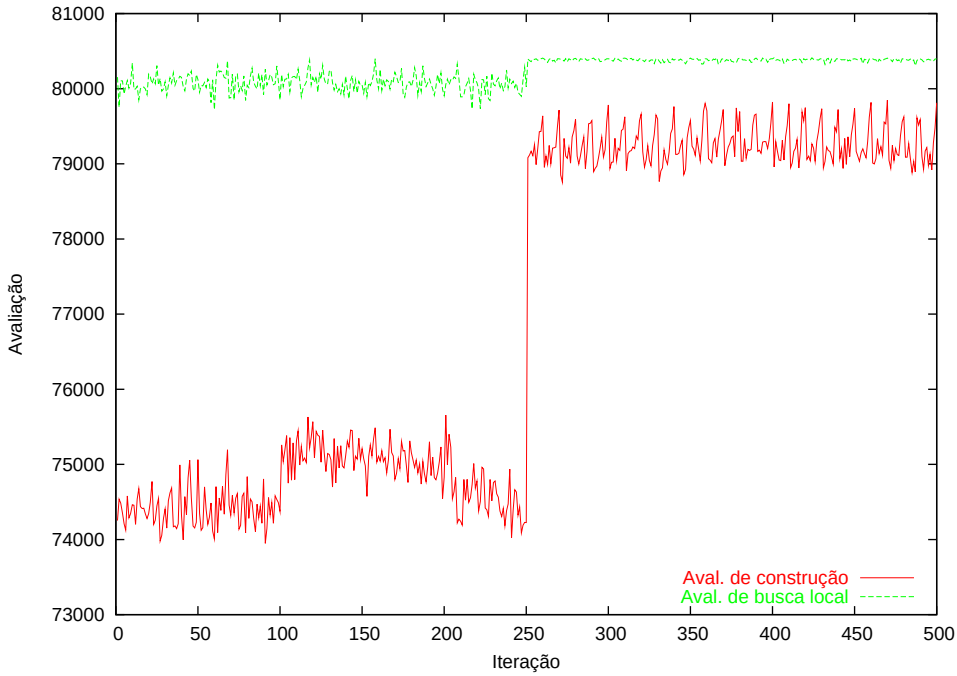

Figura 3.5: Qualidade das soluções alcançadas pelo GRASP-MD ao longo das iterações para a instância com $n = 600$ e $m = 180$ (semente $= 1$)

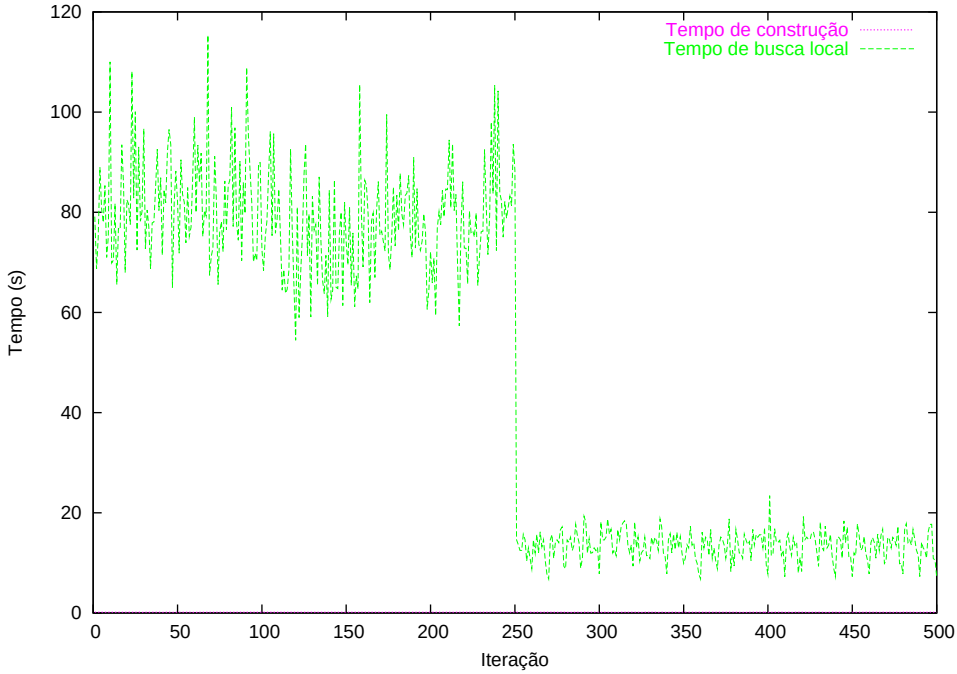

Figura 3.6: Tempos computacionais das fases do GRASP-MD ao longo das iterações para a instância com $n = 600$ e $m = 180$ (semente $= 1$)

dade, faz com que a busca se concentre em regiões mais promissoras, aumentando a probabilidade de se alcançar soluções melhores.

A partir dos gráficos das Figuras 3.4 e 3.6, que ilustram os tempos de execução, pode-se notar que, após a extração dos padrões, os tempos da fase de busca local, que predominam em relação aos tempos de construção, são significativamente reduzidos. Isto é conseqüência do fato de as soluções geradas na fase de construção serem melhores, o que reduz o esforço necessário para busca local encontrar um ótimo local. Esta é a razão principal para os ganhos em tempo de execução do GRASP-MD em relação à heurística KLD.

Capítulo 4

GRASP-MD para o Problema de Replicação de Servidores para Transmissão *Multicast* Confiável

Neste capítulo, é descrita a aplicação da metaheurística híbrida GRASP-MD em um segundo problema de Otimização Combinatória. Trata-se do problema de replicação de servidores para transmissão *multicast* confiável [42]. Parte do conteúdo deste capítulo foi publicado em [57]. Na Seção 4.1, é fornecida a definição deste problema. A implementação do GRASP-MD para este problema é descrita na Seção 4.2 e, por fim, os resultados experimentais são apresentados na Seção 4.3.

4.1 O Problema de Replicação de Servidores para Transmissão *Multicast* Confiável

Transmissões do tipo *multicast* correspondem ao envio de informações para vários receptores simultaneamente. Estas informações podem vir de uma única fonte (*multicast* um para muitos) ou de várias fontes (*multicast* muitos para muitos). Entre

as aplicações potenciais deste tipo de transmissão encontram-se: envio de notícias, distribuição de cotações de ações, atualização de *softwares*, *streaming* de áudio e vídeo, entre outras.

Os serviços de rede oferecidos para transmissão do tipo *unicast* convencional não são suficientes para dar suporte a transmissões do tipo *multicast* de forma eficiente. Para o uso destes serviços, seria necessária a criação de uma conexão *unicast* entre a fonte e cada receptor. Esta estratégia requer a transmissão de várias cópias dos mesmos dados, o que caracteriza uma ineficiência significativa no uso dos recursos da rede. O esquema utilizado em serviços de rede fornecidos correntemente para transmissões *multicast* é a criação de uma árvore de transmissão, cuja raiz representa a fonte, as folhas representam os receptores e os nós internos representam roteadores da rede. Desta forma, cópias dos dados só precisam ser criadas em ramificações desta árvore.

Desde a introdução do modelo do IP *multicast* [11], o interesse por transmissões *multicast* aumentou, tanto cientificamente, quanto comercialmente [4]. Uma rede virtual composta de roteadores capazes de fornecer serviços *multicast* foi construída. Esta rede foi chamada de *MBone* (*multicast backbone*) [16] e está se desenvolvendo continuamente. Vários protocolos já foram propostos, mas nenhum deles foi considerado, ainda, a solução definitiva [14].

Uma questão importante relacionada à transmissão *multicast* é como prover um serviço confiável. O serviço fornecido pelo modelo do IP *multicast* segue uma política de melhor esforço, o que não garante que todos os pacotes de dados enviados pela fonte sejam efetivamente entregues aos receptores. Nas transmissões *unicast* convencionais, o protocolo de transporte TCP realiza esta tarefa eficientemente. Porém, em transmissões *multicast*, esta tarefa é substancialmente mais complexa. Muitos protocolos de transporte foram desenvolvidos, cada um satisfazendo diferentes requerimentos de aplicações *multicast* [7].

O fornecimento de serviço *multicast* confiável está relacionado à forma de gerenciamento de retransmissões de dados. Em transmissões *unicast*, a fonte é responsável pela retransmissão de pacotes perdidos. Uma adaptação desta estratégia

para transmissões *multicast* sobrecarregaria a fonte, especialmente em situações em que o número de receptores é grande. Uma estratégia que tem apresentado bons resultados é a divisão dos receptores em grupos, com a realização de técnicas de recuperação local [37, 46]. Além da redução de carga na fonte, este método tem as vantagens de melhorar a vazão do sistema e de reduzir o uso de banda da rede.

A técnica abordada nesta dissertação é a **Replicação de Servidores**, que segue a estratégia de recuperação local. Nesta técnica, os dados são replicados em alguns dos nós internos da rede e cada um destes é responsável pela retransmissão de pacotes aos receptores de seu grupo. Estes nós são chamados servidores de réplica ou servidores de reparo.

Em [42], foi proposto um GRASP para solucionar o problema de selecionar o melhor subconjunto dos nós intermediários para agirem como servidores de réplica em uma transmissão *multicast* confiável. Os bons resultados obtidos por este método serviram de motivação para a investigação do desempenho da aplicação do GRASP-MD neste problema.

4.1.1 Definição do Problema

Nesta subseção, será descrita a formulação do problema, apresentada em [42]. Inicialmente, a função que será usada para modelar o custo de uma árvore de transmissão *multicast* será definida. A seguir, a formulação é apresentada com base nesta função de custo.

Formulação da Função de Custo

Para que a replicação de servidores seja efetiva, é necessário encontrar a alocação de servidores de réplica que leve ao melhor desempenho da transmissão *multicast*. A função apresentada a seguir fornece a medida do custo de uma árvore de transmissão *multicast*. Então, o objetivo é definir quais nós devem funcionar como servidores de réplica para que se alcance uma árvore que minimize esta função.

Suponha que um nó s seja a fonte de uma transmissão *multicast* e que os nós do conjunto R sejam os receptores. Neste caso, uma árvore de transmissão T é construída com raiz em s, os nós em R como folhas e os nós em MR como nós internos. Existe uma probabilidade de perda de pacotes associada à cada aresta de T. Agora, suponha que os dados sejam replicados em $P \subseteq MR$ nós. Então, P é o subconjunto dos nós internos que funcionarão como servidores de réplica. A Figura 4.1 ilustra uma árvore de transmissão T com raiz na fonte s. Os receptores, $R = \{r_1, r_2, ..., r_9\}$, são as folhas. Os nós intermediários estão no conjunto $MR = \{n_1, n_2, n_3, p_1, p_2, p_3\}$. O conjunto $P = \{p_1, p_2, p_3\}$ corresponde aos servidores de réplica.

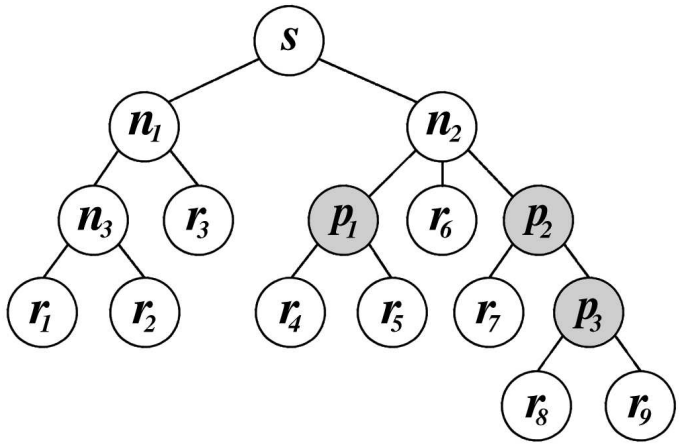

Figura 4.1: Uma árvore de transmissão. Nós acinzentados representam os servidores de réplica

Seja T_v a subárvore de T com a raiz no nó v. A formulação que será apresentada é baseada na seguinte terminologia:

- R_v O conjunto de receptores em T_v
- R'_v O subconjunto de elementos em R_v cujo caminho entre eles e o nó v não inclui um servidor de réplica
- H_v O número total de arestas em T_v
- P_v O conjunto de servidores de réplica em T_v
- P'_v O subconjunto de elementos em P_v cujo caminho entre eles e o nó v não inclui um servidor de réplica

Por exemplo, considere T_{n_1}, a subárvore com raiz em n_1 na Figura 4.1. Nesta subárvore, $R_v = \{r_1, r_2, r_3\}$, $R'_v = R_v$, $H_v = 4$, e $P'_v = P_v = \varnothing$. Em T_{n_2}, subárvore com raiz em n_2, $R_v = \{r_4, r_5, ..., r_9\}$, $R'_v = \{r_6\}$, $H_v = 9$, $P_v = \{p_1, p_2, p_3\}$, e $P'_v = \{p_1, p_2\}$.

A idéia fundamental da formulação proposta em [42] é considerar o custo de transmitir um pacote como sendo o produto do seu número total de transmissões (a original mais as retransmissões subseqüentes) e o número de arestas percorridas até que o pacote chegue em todos os receptores [46]. Então, de acordo com esta idéia, a árvore na qual este custo é mínimo é aquela que proporciona o melhor uso dos recursos da rede, o que leva ao melhor desempenho da transmissão *multicast*.

Seja K_v a variável aleatória que representa o número total de transmissões de um pacote de um nó v requeridas para garantir sua entrega para todos os receptores em T_v, dado que o pacote foi enviado com sucesso para v pelo seu pai. Note que, no caso da fonte s, K_s é o número de transmissões requeridas para garantir a entrega de um pacote para todos os receptores. Se T_v não inclui servidores de réplica, toda transmissão de um pacote irá percorrer todas as arestas de T_v. Então, a função de custo de T_v pode ser definida como se segue.

$$custo(T_v) = E[K_v] * H_v, \tag{4.1}$$

onde $E[K_v]$ é o valor esperado de K_v. No caso em que T_v contém servidores de réplica, a fonte v será responsável apenas pela entrega para os nós em $R'_v \cup P'_v$. Isto porque todo nó $w \in P'_v$ é responsável pela transmissão em T_w. Então, a função de custo pode ser definida como se segue:

$$custo(T_v) = E[K_v] * (H_v - \sum_{r \in P'_v} H_r) + \sum_{r \in P'_v} custo(T_r) \tag{4.2}$$

Note que a Equação (4.1) é derivada a partir de (4.2) no caso de uma árvore sem servidores de réplica. Note, também, que uma subárvore T_w com raiz no nó $w \in P'_v$ também pode conter servidores de réplica. Então, trata-se de uma definição recursiva.

Para completar a formulação, ainda é necessário derivar uma expressão para $E[K_v]$.

Seja p_n a probabilidade de perda de pacotes na aresta entre o nó n e seu pai. Considere um nó folha n de uma árvore T_v. A probabilidade de que k transmissões de um pacote sejam suficientes para garantir sua entrega para o nó n pelo seu pai, ou seja, a função de distribuição de probabilidades de K_n, é:

$$F_n(k) = P[K_n \leq k] = 1 - (p_n)^k \tag{4.3}$$

Note que a equação anterior também pode ser usada para todo nó $w \in P'_v$. Dado que um nó $w \in P'_v$ é responsável pela entrega em T_w, é suficiente que os pacotes cheguem a w para garantir suas entregas aos receptores em R_w.

Agora considere um nó interno n de uma árvore T_v. Neste caso, a função de distribuição de probabilidades K_n é:

$$F_n(k) = \sum_{l=0}^{k-1} \left(\binom{k}{l} (p_n)^l (1-p_n)^{k-l} * \prod_{c \in filhos(n)} F_c(k-l) \right), \tag{4.4}$$

onde $filhos(n)$ é o conjunto dos filhos de n. O primeiro termo do produto corresponde à probabilidade de ocorrerem l perdas em k transmissões para o nó n pelo seu pai. O segundo termo corresponde à probabilidade de que um pacote irá chegar em todos os nós filhos de n com no máximo $k-l$ transmissões a partir de n e depois entregues com sucesso a todos os receptores de T_n. O somatório considera todos os casos em que exista pelo menos uma transmissão com sucesso para n pelo seu pai.

Para o nó raiz v, a função de distribuição é a probabilidade de um pacote chegar a todos os seus nós filhos com nó máximo k transmissões e depois serem entregues a todos os receptores:

$$F_v(k) = \prod_{c \in filhos(v)} F_c(k) \tag{4.5}$$

Na Equação (4.6), $E[K_v]$ é apresentado em termos de sua função de distri-

buição. A substituição desta expressão na Equação (4.2) completa a formulação da função de custo $custo(T_v)$.

$$E[K_v] = \sum_{k=0}^{\infty} \big(1 - F_v(k)\big) \tag{4.6}$$

Note que a função de custo é baseada em um termo que é um somatório infinito (Equação 4.6). As implementações dos programas desenvolvidos nesta dissertação utilizaram um valor aproximado para este termo. A computação do somatório era interrompida sempre que o último valor de k avaliado não contribuía para o somatório com um valor maior do que 10^{-4}.

A função de custo pode ser facilmente generalizada para o caso onde múltiplos servidores coexistem. Neste caso, as árvores de entrega, agrupadas, formarão um grafo G. Dado que o roteamento e a entrega de requisições de retransmissão são independentes para cada fonte, a função de custo do grafo G pode ser definida como:

$$custo(G) = \sum_{s} custo(T_s), \tag{4.7}$$

onde $custo(T_s)$ é o custo da árvore de entrega com raiz em s, dado pela Equação (4.2).

Agora, o problema pode ser facilmente definido: para um dado valor m, encontrar o melhor conjunto $P \subseteq MR$, com $|P| = m$, tal que o grafo de transmissão resultante tenha custo mínimo de acordo com a função da Equação (4.7).

4.2 A implementação do GRASP-MD

Três técnicas para resolução deste problema foram propostas em [42]. A primeira é uma estratégia puramente gulosa, a segunda é um algoritmo evolutivo e a outra é uma implementação do GRASP. Os experimentos demonstraram que o GRASP atingiu os melhores resultados em todos os casos, enquanto que a estratégia

gulosa foi sempre a pior. Este GRASP foi adotado para a hibridização realizada nesta dissertação.

Soluções deste problema são caracterizadas por um conjunto de nós internos que devem funcionar como servidores de réplica. Como ocorreu no problema abordado no Capítulo 3, conjuntos freqüentes, extraídos a partir de um conjunto de soluções elite, podem ser considerados como padrões de boas soluções e, por isso, podem ser utilizados para guiar a busca por melhores soluções.

Na fase de construção do GRASP implementado em [42], nós intermediários são adicionados à solução iterativamente. Em cada iteração, todo nó interno $n \in MR$ que ainda não faz parte da solução é avaliado. Para o caso de uma transmissão *multicast* com uma única fonte s, cada um destes nós é avaliado da seguinte forma. Seja c_{sol} o custo da árvore de transmissão T_s que possui os nós da solução parcial corrente *sol* como servidores de réplica, avaliado de acordo com a Equação 4.2. O nó n é adicionado a este conjunto de servidores de réplica e o custo $c_{sol \cup n}$ da árvore T_n é avaliado. A avaliação do nó n é dada pela diferença entre c_{sol} e $c_{sol \cup n}$. Quanto maior a redução no custo c_{sol} proporcionada pelo nó n, melhor será sua avaliação. Para o caso de uma transmissão *multicast* com várias fontes, existe uma árvore de transmissão para cada fonte. A avaliação de um nó n é, então, igual ao somatório de sua avaliação em cada uma destas árvores.

Seja $aval_{max}$ a avaliação do melhor nó interno, a LRC é constituída de todos os nós internos que possuem avaliação melhor que uma porcentagem de $aval_{max}$. Esta porcentagem é definida pelo parâmetro α, ou seja, um roteador só será incluído na LRC caso sua avaliação seja melhor ou igual a $\alpha * aval_{max}$ $(0 < \alpha < 1)$. A cada iteração da fase de construção, um elemento da LRC é escolhido aleatoriamente para ser integrado à solução.

Na fase de busca local, a estrutura de vizinhança utilizada é baseada em um procedimento chamado k-trocas. Neste procedimento, cada solução sol', que pode ser obtida através da troca de k elementos a partir de uma solução *sol*, faz parte da vizinhança de *sol*. O valor de k escolhido foi $k = 1$, fazendo com que o procedimento tenha um custo computacional viável. Além disso, é utilizada uma

outra restrição: um nó interno n só pode ser substituído por outro nó n' desde que exista exatamente uma aresta que ligue n a n'.

No GRASP proposto em [42], utilizado como base para hibridização, a realização dos procedimentos de construção e busca local é repetida ao longo de um total de n_iter iterações. Da mesma forma que na implementação do GRASP-MD realizada para o PMD, descrita no Capítulo 3, as iterações da fase de geração do conjunto elite são praticamente idênticas às iterações do GRASP original. A exceção é que, no GRASP-MD, ao final de cada iteração, a solução gerada é candidata a integrar o conjunto elite. Além disso, na fase híbrida, o procedimento de construção também é adaptado. Ao invés de construir a solução a partir de um conjunto vazio de elementos, este procedimento inicializa a solução com os elementos de um dos padrões extraídos e utiliza a mesma lógica do procedimento de construção acima para obter os outros elementos da solução. O GRASP-MD também executa um total de n_iter iterações, sendo que a primeira metade corresponde à fase de geração do conjunto elite, e a segunda metade à fase híbrida.

A estratégia de utilização de padrões, da mesma forma que na implementação do GRASP-MD para o PMD, descrita no Capítulo 3, também foi a nMM, que obteve os melhores resultados em [53, 54, 55].

4.3 Resultados Experimentais

Ambientes de transmissão *multicast* podem ser bastante diferentes. Eles podem possuir diferentes números de participantes, número de fontes, volume de tráfego, etc. No entanto, eles são normalmente classificados em dois grupos principais: ***broadcasts*** e **conferências virtuais** [9]. O primeiro é caracterizado por ter uma única fonte e um grande número de receptores. O segundo possui um número menor de membros, mas estes, em geral, atuam tanto como fontes quanto receptores.

As avaliações dos algoritmos foram feitas em cinco cenários de transmissão *multicast* simulados, com três destes representando conferências virtuais e os outros dois, transmissões do tipo *broadcast*. Estes cenários foram gerados da seguinte forma.

Primeiramente, as topologias de rede foram criadas com a ferramenta GT-ITM (*Georgia Tech Internetwork Topology Models*) [8][1]. A seguir, uma probabilidade de perda foi associada a cada aresta destas redes. Depois disso, foram definidos os nós que representariam as fontes e os receptores das transmissões *multicast* e, finalmente, uma árvore de transmissão foi contruída para cada fonte[2].

A Tabela 4.1 descreve os cenários de transmissão *multicast* simulados. Foram criados três cenários representando conferências virtuais ($CONF_1$, $CONF_2$ e $CONF_3$) e dois representando *broadcasts* ($BROAD_1$ e $BROAD_2$). Para cada um, é descrita sua topologia e a transmissão *multicast* que acontece sobre esta. As topologias são descritas pelo número de nós de trânsito[3], número de arestas e intervalo de probabilidade de perda destas. A probabilidade de perda de todas as arestas são distribuídas uniformemente neste intervalo. As transmissões *multicast* são descritas pelo número de fontes, número de receptores e número de nós candidatos, que correspondem aos nós que podem funcionar como servidores de réplica. São considerados candidatos os nós de trânsito que fazem parte de pelo menos uma das árvores de transmissão.

Para os cenários $CONF_1$, $BROAD_1$ e $BROAD_2$, os algoritmos foram executados para encontrar subconjuntos dos nós candidatos de tamanhos correspondentes a 2,5%, 5%, 7,5% e 10% do total de nós de trânsito, para funcionarem como servidores de réplica. Para os outros dois cenários, estes tamanhos corresponderam a

[1]Foi utilizado o modelo *transit-stub*. Neste modelo, nós de trânsito (*transit nodes*) são nós internos da infra-estrutura principal da rede. Domínios *stub* (*stub domains*) são subredes locais que se conectam a esta infra-estrutura. As instâncias foram criadas com um domínio *stub* associado a cada nó de trânsito (nó intermediário) e cada domínio *stub* contendo apenas um nó. Estes nós representaram os terminais (nós folhas) da rede e apenas eles poderiam representar fontes e/ou receptores nas transmissões *multicast*.

[2]O algoritmo de Dijkstra [13] foi utilizado para encontrar caminhos mínimos entre a fonte e cada receptor com a probabilidade de perda como o peso das arestas. A união das arestas de cada um destes caminhos formou a árvore de transmissão desta fonte. Note que este algoritmo foi utilizado apenas para a geração de árvores de transmissão básicas, não necessariamente as de menor custo.

[3]Como existe exatamente um nó terminal (folha) conectado a cada nó de trânsito (intermediário), o número de nós de trânsito e de terminais é o mesmo. Note que o total de nós da topologia é, então, o dobro do número de nós de trânsito.

Tabela 4.1: Descrição dos cenários de transmissão *multicast* simulados

Cenário	Topologia			Transmissão *multicast*		
	#Nós de Trânsito	#Arestas	Faixa de Prob. de Perda	#Fontes	#Receptores	#Nós Candidatos
$CONF_1$	200	4194	1-70%	50	50	164
$CONF_2$	200	1931	1-15%	50	50	130
$CONF_3$	400	7497	1-15%	50	50	186
$BROAD_1$	1000	10612	1-15%	1	500	682
$BROAD_2$	2000	40495	1-15%	1	500	797

5%, 10%, 15% e 20%.

Os algoritmos GRASP e GRASP-MD foram implementados em C++ e compilados com o `g++ 3.2.2`. Os testes foram realizados em um Intel Pentium 4 de 1.7 GHz com 256 Mb de memória RAM. Parte dos resultados apresentados neste capítulo foram publicados em [57].

As duas implementações foram executadas dez vezes para cada instância, cada uma utilizando uma semente de números aleatórios diferente. Para que as comparações entre o GRASP e o GRASP-MD fossem justas, o valor do parâmetro α escolhido para ambas as implementações foi de $\alpha = 0.7$, o mesmo utilizado em [42]. O GRASP básico foi executado utilizando 500 iterações. No GRASP-MD, a fase de geração do conjunto elite foi executada por 250 iterações, e a fase híbrida, por outras 250. O tamanho do conjunto elite, utilizado no GRASP-MD, foi igual a 10. Este valor foi escolhido porque obteve os melhores resultados em [53, 54, 55]. Conforme mencionado anteriormente, a estratégia de utilização de padrões escolhida foi a nMM. Nesta estratégia, conjuntos freqüentes maximais são minerados do conjunto elite em execuções distintas do algoritmo FPmax* com suporte mínimo variando entre 2 e 10, e apenas os dez maiores são selecionados para serem utilizados como padrões.

A qualidade das soluções obtidas é apresentada na Tabela 4.2. As duas primeiras colunas descrevem informações sobre as instâncias, que são o cenário de

transmissão e o número de nós a serem definidos como servidores de réplica. As próximas três colunas contêm os resultados alcançados pelo GRASP proposto em [42] e as três colunas finais, os resultados do GRASP-MD. Para os dois métodos, são exibidos o custo da melhor solução, o valor médio e o desvio padrão encontrados ao longo das dez execuções. Para comparação entre os dois métodos, valores em negrito indicam qual método obteve melhor desempenho.

Note que o desempenho do GRASP-MD foi superior ao do GRASP. Considerando as 20 instâncias, o GRASP-MD alcançou soluções melhores em 12 e as soluções foram as mesmas em oito instâncias. Comparando os valores médios, o GRASP-MD foi superior em 13, o GRASP em quatro e houve empate em três instâncias.

Na Tabela 4.3, comparam-se os tempos de execução dos métodos. Para cada método, são apresentados a média e o desvio padrão dos tempos de execução gastos nas dez execuções. O GRASP-MD foi consideravelmente mais rápido em todos os testes. A última coluna apresenta a redução média de tempo de execução do GRASP-MD em relação ao do GRASP. Considerando a média de todas as instâncias, o GRASP-MD foi 36.8% mais rápido que o GRASP.

Os gráficos das Figuras 4.2 até 4.5 permitem uma avaliação mais detalhada do comportamento do GRASP-MD. O gráfico da Figura 4.2 ilustra a variação do valor da qualidade das soluções obtidas nas fases de construção e busca local, ao longo das iterações de uma execução para a instância com cenário CONF_1 e $m = 20$. A Figura 4.3 ilustra os tempos de execução destas mesmas fases, para este mesmo teste. As Figuras 4.4 e 4.5 são análogas às Figuras 4.2 e 4.3 e correspondem a uma execução para a instância com cenário BROAD_1 e $m = 100$. O comportamento do GRASP-MD para as outras instâncias foi bastante similar.

Como ocorreu na aplicação do GRASP-MD ao PMD, a análise dos gráficos das Figuras 4.2 e 4.4, relativos às qualidades das soluções obtidas, permite concluir que, após a extração dos padrões, a qualidade médias das soluções geradas na fase de construção aumenta significativamente. Na fase de busca local, esta média de qualidade também aumenta. Isto reforça a idéia de que a utilização de padrões, que

Tabela 4.2: Qualidade das soluções obtidas pelos métodos GRASP e GRASP-MD

Instância		GRASP			GRASP-MD		
Cenário	m	Melhor	Média	D.P.	Melhor	Média	D.P.
$CONF_1$	5	63762.2	63762.2	0.0	63762.2	63762.2	0.0
	10	44480.7	44480.7	0.0	44480.7	44480.7	0.0
	15	31328.6	31347.2	59.0	31328.6	**31328.6**	0.0
	20	23625.2	23775.9	112.0	23625.2	**23763.7**	83.3
$CONF_2$	10	11894.1	11894.1	0.0	11894.1	11894.1	0.0
	20	10076.3	10076.3	0.0	**10047.1**	**10047.1**	0.0
	30	9207.8	**9208.7**	2.0	9207.8	9211.7	6.5
	40	8668.5	8676.6	7.0	**8642.3**	**8646.3**	9.4
$CONF_3$	20	11130.1	11177.4	16.6	**11114.5**	**11114.5**	0.0
	40	9631.7	9652.9	11.9	**9584.3**	**9596.5**	9.2
	60	8855.9	**8869.3**	9.4	**8848.1**	8869.4	11.0
	80	8550.4	**8557.8**	4.3	8550.4	8559.1	5.6
$BROAD_1$	25	2818.9	2818.9	0.0	**2807.2**	**2807.2**	0.0
	50	2296.6	2299.0	2.1	**2281.8**	**2287.4**	3.9
	75	2039.3	2045.9	4.6	**2020.9**	**2030.8**	7.9
	100	1873.6	**1877.5**	2.0	1873.6	1877.9	2.1
$BROAD_2$	50	2444.0	2444.2	0.1	**2425.6**	**2431.4**	3.4
	100	2019.0	2020.2	0.7	**2018.9**	**2020.1**	0.8
	150	1836.3	1837.0	0.4	**1836.2**	**1836.9**	0.5
	200	1727.9	1729.6	0.9	**1726.5**	**1729.3**	1.3

representam características de soluções de boa qualidade, faz com que a busca se concentre em regiões mais promissoras, aumentando a probabilidade de se alcançar soluções melhores.

Um fato importante a observar é que, no gráfico da Figura 4.4, durante a fase híbrida, as soluções obtidas ao longo das iterações se repetiam em ciclos. Apesar destas soluções possuírem boa qualidade média, estes ciclos mostram que a busca ficou restrita a uma pequena região do espaço de soluções, o que pode levar a

Tabela 4.3: Tempos de execução, em segundos, dos métodos GRASP e GRASP-MD

Instância		GRASP		GRASP-MD		**Redução**
Cenário	m	Média	D.P.	Média	D.P.	**Média**
$CONF_1$	5	28231.5	417.0	**20292.0**	522.0	**28.1%**
	10	43826.0	480.5	**30881.8**	821.9	**29.5%**
	15	43374.8	698.1	**30058.0**	1313.7	**30.7%**
	20	43314.2	661.5	**26831.3**	512.8	**38.0%**
$CONF_2$	10	3083.9	24.2	**2631.8**	28.4	**14.7%**
	20	5239.0	25.9	**3280.1**	22.5	**37.4%**
	30	7211.5	16.0	**4196.7**	92.8	**41.9%**
	40	6787.9	36.9	**4418.2**	106.8	**34.9%**
$CONF_3$	20	7518.6	39.2	**5661.6**	36.9	**24.7%**
	40	15077.0	50.9	**8724.9**	279.7	**42.1%**
	60	19683.5	142.0	**11312.0**	346.2	**42.5%**
	80	17747.8	95.5	**10628.1**	121.2	**40.1%**
$BROAD_1$	25	1555.1	2.3	**1004.8**	3.6	**35.4%**
	50	3709.2	9.2	**2301.7**	40.9	**38.0%**
	75	5530.9	14.7	**3366.7**	160.7	**39.1%**
	100	7183.0	21.0	**4160.2**	78.5	**42.0%**
$BROAD_2$	50	5096.8	8.3	**2994.0**	42.0	**41.3%**
	100	9246.2	18.4	**5188.1**	87.9	**43.9%**
	150	11482.1	24.4	**6098.7**	65.3	**46.9%**
	200	14047.3	24.3	**7705.9**	124.0	**45.1%**
					Média	**36.8%**

obtenção de uma solução que não represente um ótimo global. Este resultado serve de motivação para avaliações de estratégias para garantir uma melhor diversificação na utilização dos padrões.

Em relação aos gráficos das Figuras 4.3 e 4.5, de tempos de execução, também se pode notar que, após a extração dos padrões, os tempos da fase de busca local são significativamente reduzidos. Como na aplicação para o PMD, acredita-se

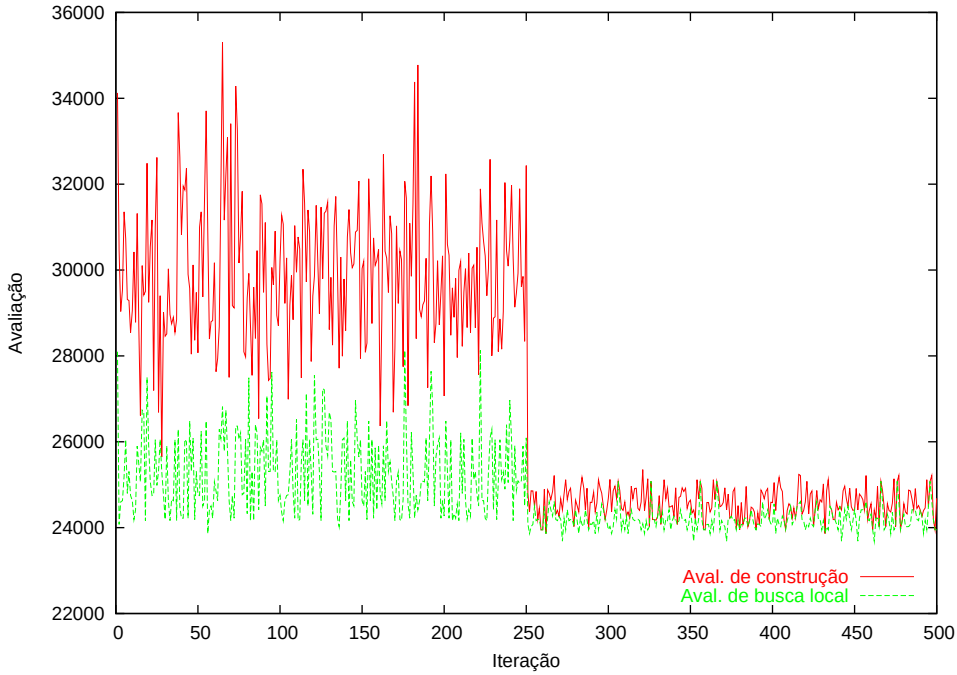

Figura 4.2: Qualidade das soluções alcançadas pelo GRASP-MD ao longo das iterações para a instância com cenário CONF_1 e $m = 20$ (semente $= 1$)

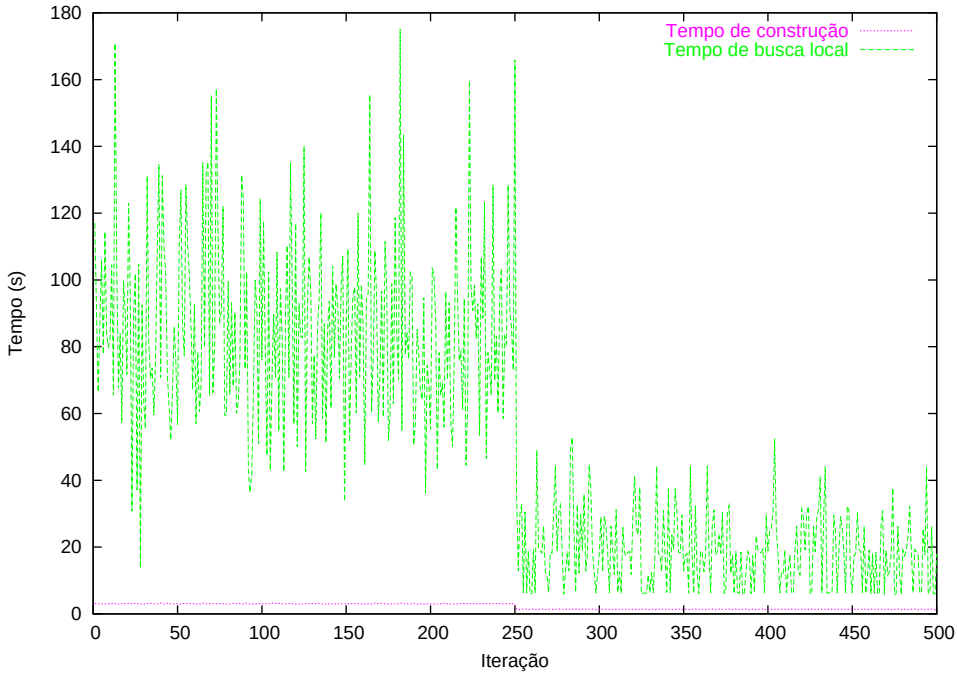

Figura 4.3: Tempos computacionais das fases do GRASP-MD ao longo das iterações para a instância com cenário CONF_1 e $m = 20$ (semente $= 1$)

que isto é conseqüência do fato de as soluções geradas nas fases de construção serem melhores, o que reduz o esforço necessário para a busca local encontrar um ótimo

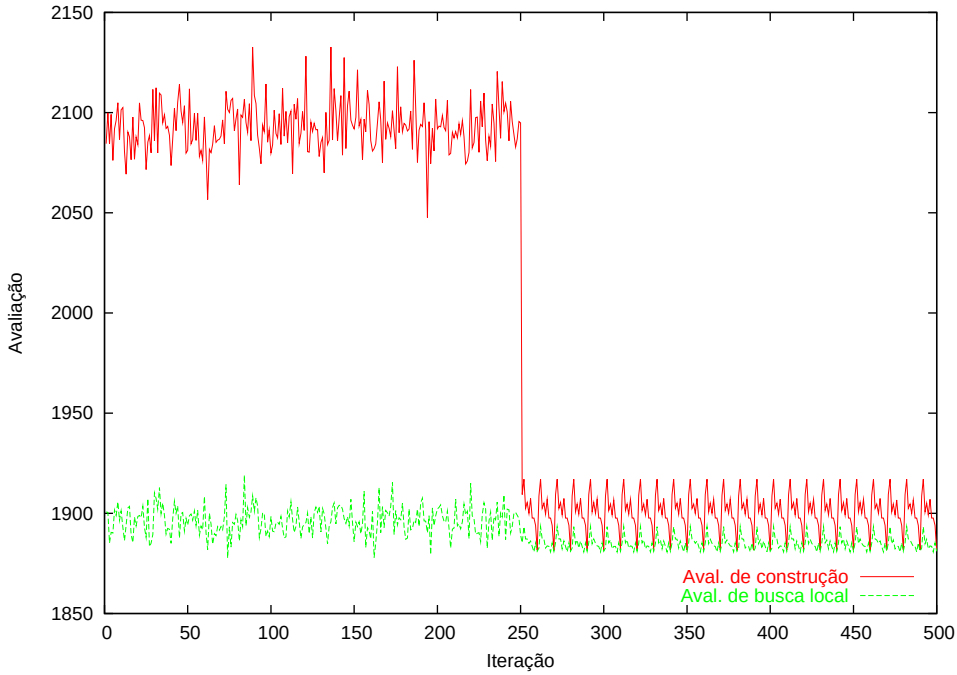

Figura 4.4: Qualidade das soluções alcançadas pelo GRASP-MD ao longo das iterações para a instância com cenário BROAD$_1$ e $m = 100$ (semente $= 1$)

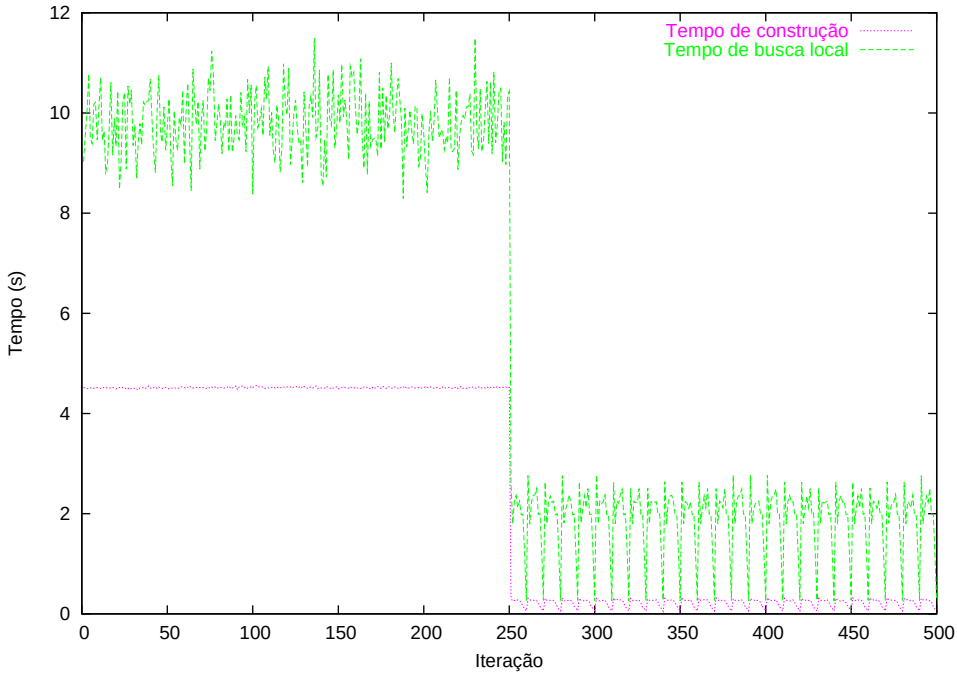

Figura 4.5: Tempos computacionais das fases do GRASP-MD ao longo das iterações para a instância com cenário BROAD$_1$ e $m = 100$ (semente $= 1$)

local.

A análise da Tabela 4.2 também permite concluir que o custo das transmis-

sões *multicast* não decresce linearmente com o aumento do número de servidores de réplica. Por exemplo, observe os resultados para o cenário $CONF_3$. Com a utilização de 20 servidores de réplica, a melhor solução encontrada possui custo igual a 11.114,5. Já com a utilização de 40 servidores de réplica, o custo da melhor solução encontrada é igual a 9.584,3. Ou seja, com o dobro do número de servidores de réplica, o custo foi reduzido apenas em aproximadamente 13,8%. Caso o decréscimo do custo da transmissão *multicast* fosse linear com relação ao aumento do número de servidores de réplica, poderia-se esperar que a utilização de 80 servidores de réplica levaria a uma transmissão com custo 13,8% inferior em relação ao da transmissão utilizando 40 servidores de réplica, o que seria igual a aproximadamente 8.261,6. No entanto, o custo da melhor solução encontrada para 80 servidores de réplica foi de 8.550,4, o que corresponde a uma redução de aproximadamente 10,8%.

Para uma análise mais detalhada do comportamento do custo das transmissões *multicast* com o aumento do número de servidores de réplica, o GRASP-MD foi executado para cada cenário utilizando várias quantidades de servidores de réplica, que chegaram a até 45% do total de nós candidatos. No gráfico da Figura 4.6, as curvas representam a redução dos custos das transmissões *multicast* com relação ao aumento do número de servidores de réplica, com uma curva para cada cenário simulado. Note que o custo converge para um valor assintótico. Isto indica que a inclusão de um novo servidor de réplica só é recomendado quando os benefícios compensam os gastos envolvidos. Estes resultados mostram que seria útil uma adaptação da formulação do problema que levasse em consideração o custo de inclusão de um servidor de réplica. Desta forma, a escolha do número ideal de servidores de réplica poderia ser automaticamente identificado pelo algoritmo.

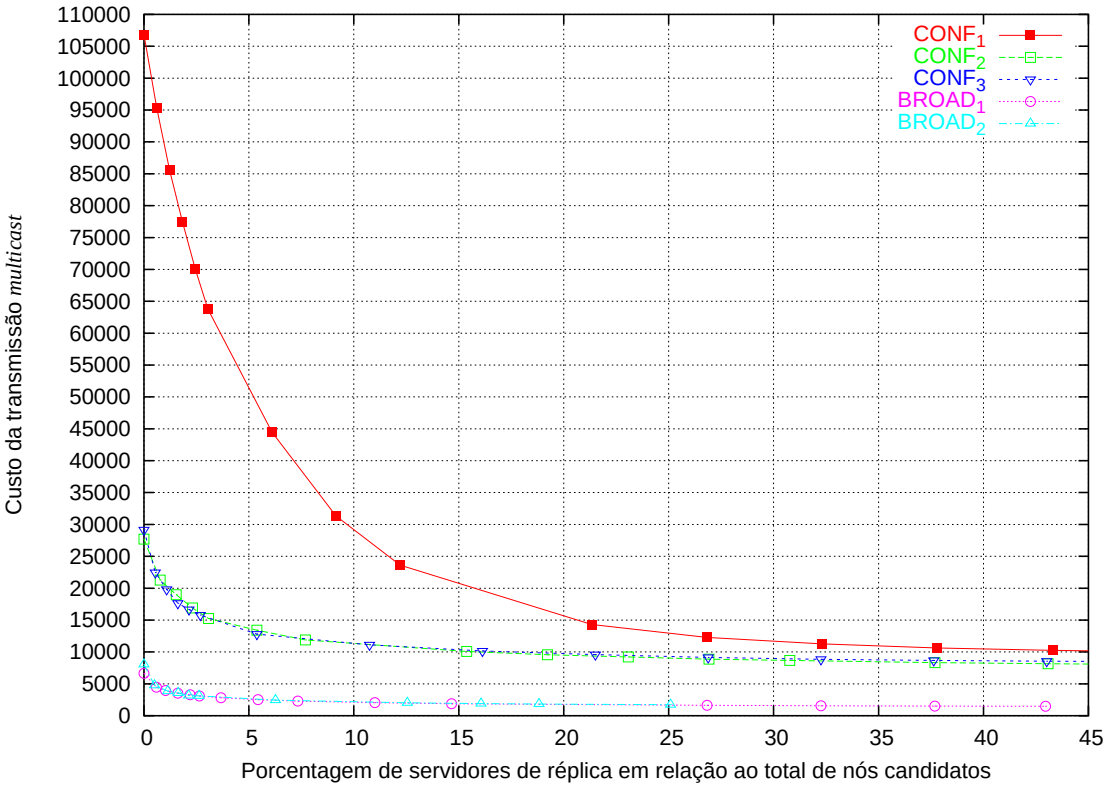

Figura 4.6: Variação do custo das transmissões *multicast* de acordo com o número de servidores de réplica

Capítulo 5

Paralelização do GRASP-MD

Neste capítulo, são apresentadas as versões paralelas do GRASP-MD desenvolvidas nesta dissertação. Inicialmente, na Seção 5.1, são apresentadas as razões que motivaram o desenvolvimento e a avaliação de versões paralelas do método híbrido GRASP-MD. A seguir, na Seção 5.2, são descritas as estratégias utilizadas para o desenvolvimento das versões paralelas. Na Seção 5.3, são apresentados e analisados os resultados dos experimentos realizados para o problema da maximização da diversidade e para o problema de replicação de servidores para transmissão *multicast* confiável.

5.1 Motivação

A tecnologia dos computadores atravessa um período de grande avanço, principalmente devido à exploração de paralelismo. Supercomputadores com centenas de processadores foram lançados, alcançando TeraFlops de capacidade de processamento[1] [1]. Devido ao elevado custo de construção e manutenção destes sistemas computacionais, sua utilização ficou restrita a poucas empresas e centros de pesquisa.

[1]Flops – *Floating point operations per second* (Operações de ponto flutuante por segundo) – é uma das principais medidas de capacidade de processamento de sistemas computacionais. 1 TeraFlops é igual a 10^{12} Flops.

No entanto, a redução do custo de dispositivos computacionais de menor porte e o avanço significativo nas tecnologias de transmissão de redes de computadores possibilitaram a criação de arquiteturas de baixo custo capazes de oferecer alto poder computacional. Dentre estas arquiteturas se destacam os *clusters* de computadores e os mais recentes *grids* computacionais [20].

O desenvolvimento de aplicações para serem executadas nestes sistemas computacionais pode exigir um grande esforço de programação. Por isso, várias ferramentas surgiram para amenizar esta dificuldade. Por exemplo, linguagens de programação com instruções de comunicação de processos embutidas foram propostas. Além disso, sistemas operacionais implementaram o conceito de *threads*, que são especialmente úteis para arquiteturas que possuem múltiplos processadores compartilhando a mesma memória. As ferramentas mais utilizadas em ambientes de memória distribuída, como *clusters* e *grids*, são as bibliotecas que implementam a comunicação entre processos por troca de mensagens. Nesta classe de ferramentas, se tornaram populares o PVM (*Parallel Virtual Machine* – Máquina Virtual Paralela) [22], e o MPI (*Message Passing Interface* – Interface para Envio de Mensagens) [44].

Problemas de Otimização Combinatória, em geral, requerem alto poder computacional para serem resolvidos. Metaheurísticas, apesar de proporcionarem a obtenção de soluções de boa qualidade para estes problemas em tempo significativamente menor que métodos exatos, podem necessitar de tempos de execução extensos, especialmente quando se busca a solução de instâncias grandes ou difíceis de um determinado problema. Pesquisadores desta área perceberam que o poder computacional oferecido pelas arquiteturas paralelas deve ser utilizado para acelerar as técnicas de obtenção de soluções para os problemas intratáveis e, nos últimos anos, foram propostas e avaliadas paralelizações de várias metaheurísticas para problemas de diversas áreas [10]. A paralelização das metaheurísticas não só reduz seus tempos de execução como também tem o potencial de torná-las mais robustas, pois possibilita uma exploração mais diversificada do espaço de busca.

Define-se aceleração de uma aplicação paralela em n processadores, a razão

entre o tempo seqüencial da aplicação e seu tempo de execução em n processadores.

Com a paralelização de aplicações seqüenciais, espera-se obter um comportamento escalável, ou seja, uma variação linear da aceleração em relação ao número de processadores utilizados. No entanto, isto nem sempre é alcançado devido às relações de dependência entre partes do programa.

Implementações paralelas de metaheurísticas, em geral, são realizadas com o particionamento do espaço de busca entre os processadores e, com isso, processos são executados em cada processador de forma independente. Desta forma, acelerações lineares são normalmente alcançadas. Paralelizações da metaheurística GRASP original, seguindo esta estratégia, em geral, obtêm resultados satisfatórios [52].

A paralelização da metaheurística híbrida GRASP-MD, que será apresentada a seguir, requer um ponto de sincronização entre os processos para que a mineração dos padrões a partir do conjunto elite seja realizada. Portanto, os diferentes processos não são executados de forma completamente independente, como, em geral, ocorre em paralelizações do GRASP original. Uma avaliação de implementações paralelas do GRASP-MD torna-se então importante para que se verifique o impacto desta característica na escalabilidade do método híbrido quando executado em ambientes paralelos e distribuídos.

5.2 Estratégias de Paralelização do GRASP-MD

Para que o desenvolvimento de aplicações paralelas seja bem sucedido, algumas característcas da arquitetura computacional na qual as aplicações serão realizadas devem ser consideradas. Por exemplo, *clusters* de computadores, em geral, se caracterizam por possuírem recursos computacionais homogêneos, alta confiabilidade e disponibilidade. Já os *grids* computacionais, se caracterizam por possuírem recursos extremamente heterogêneos, serem altamente compartilhados e por apresentarem maior probabilidade de falhas.

Foram desenvolvidas duas versões paralelas do GRASP-MD nesta disser-

tação, que se diferenciam pela estratégia de balanceamento de carga adotada. Na primeira delas, chamada de paralelização estática, objetiva-se distribuir a carga de processamento de forma equivalente entre todos os processos. Esta estratégia é adequada para ambientes com recursos homogêneos e não compartilhados. Na segunda estratégia, a distribuição de carga é feita de forma dinâmica entre os processos, seguindo o modelo Mestre/Escravo. Neste modelo, um dos processos, chamado **mestre**, é responsável pela distribuição das tarefas, que são entregues aos outros processos, chamados **escravos**, sob demanda. Esta estratégia é mais indicada para ambientes heterogêneos ou compartilhados.

Em ambas as estratégias, uma tarefa é definida como uma iteração do GRASP-MD. Cada iteração é identificada por uma semente de números aleatórios e, no caso das iterações da fase híbrida, também pelo padrão utilizado na fase de construção.

Em ambas as versões, o critério de parada utilizado é o número total de iterações n_iter, sendo que a fase de geração do conjunto elite corresponde à metade destas iterações e a fase híbrida, à outra metade.

Na versão estática, cada um dos p processos recebe uma carga equivalente a n_iter/p iterações. Tanto na fase de geração do conjunto elite quanto na fase híbrida, cada processo executa um total de $(n\_iter/2)/p$ iterações. Quando um processo termina a execução das suas iterações da primeira fase, este envia todas as soluções correntes em seu conjunto elite local, que contém tam_elite soluções, para todos os outros processos e espera pelo envio das soluções destes outros processos. Quando um processo recebe as soluções de todos os outros processos, este está apto a selecionar as tam_elite melhores soluções recebidas, gerando o conjunto elite global, e a realizar o procedimento de mineração de dados. Desta forma, todas os processos passam a compartilhar o mesmo conjunto elite global e, conseqüentemente, realizam o procedimento de mineração de dados a partir do mesmo conjunto elite, obtendo, então, o mesmo conjunto de padrões a ser utilizado nas iterações da fase híbrida.

Na versão mestre/escravo, o processo mestre distribui cada uma das n_iter

iterações (tarefas) para os p processos escravos sob demanda[2]. Inicialmente, o processo mestre envia uma iteração para cada processo escravo. Um processo escravo, ao receber uma iteração, realiza a execução desta iteração e retorna o resultado para o mestre, que, por sua vez, percebe que este processo escravo está apto a executar uma nova tarefa e, então, lhe envia uma outra iteração.

Na fase de geração do conjunto elite, o processo mestre envia iterações (tarefas) para os processos escravos até obter os resultados das $n\_iter/2$ iterações. Neste momento, o mestre realiza o procedimento de mineração de dados, e a partir de então, todas as iterações enviadas aos processos escravos são acompanhadas de um dos padrões minerados. Quando o processo mestre recebe os resultados de todas as n_iter iterações, este envia uma sinalização para todos os processos escravos para informar que o processamento terminou.

5.3 Resultados Experimentais

As implementações paralelas foram desenvolvidas para ambos os problemas abordados nos Capítulos 3 e 4. Os parâmetros utilizados foram exatamente os mesmos das respectivas versões seqüenciais, como, por exemplo, valores de α, tamanho do conjunto elite, suporte mínimo, estratégia de utilização de padrões. Os experimentos foram realizados com 4, 8, e 16 processadores. A quantidade total de iterações foi igual a 1000 para que houvesse carga significativa para cada processo mesmo quando um número maior de processadores fosse utilizado.

Os programas foram desenvolvidos em C++ e a troca de mensagens entre os processos foi feita com as rotinas da biblioteca `LAM MPI 7.0.6`, que é uma implementação do MPI [44]. Foram compilados com o `g++ 3.2.2` e executados em um *cluster* homogêneo composto de computadores Intel Pentium 4 de 1.7 GHz com 256

[2]Nas implementações da versão mestre/escravo, o processo mestre não foi executado exclusivamente em um processador, já que este ficaria ocioso a maior parte do tempo. Então, nos experimentos realizados com p processadores, foram criados p processos escravos, sendo um para cada processador, além do processo mestre, que era executado em um destes p processadores.

Mb de memória RAM, interconectados por uma rede Ethernet de alta velocidade.

Pava avaliar a escalabilidade das implementações paralelas em ambientes homogêneos e heterogêneos, foi realizado um conjunto de testes livre de cargas externas (outros processos executando concorrentemente), e outro conjunto com a presença de carga externa simulada. Para esta simulação, foi criado um processo infinito $\mathcal{P}$, que realiza apenas algumas operações lógicas e matemáticas. Em cada grupo de 4 processadores, a distribuição da carga externa é feita da seguinte forma: o primeiro processador fica livre, o segundo executa uma instância do processo $\mathcal{P}$, o terceiro executa duas destas instâncias e o quarto, três.

O tempo de processamento utilizando um único processador, necessário para as avaliações de aceleração das versões paralelas, foi considerado, em todos os experimentos, como sendo o tempo da versão estática executada em um único processador livre de carga externa.

5.3.1 Paralelização do GRASP-MD para o Problema da Maximização da Diversidade

As versões paralelas desenvolvidas para o problema da maximização da diversidade possuem alguns detalhes adicionais em relação às descrições apresentadas na Seção 5.2. Estes detalhes correspondem à implementação do conceito de GRASP reativo. As iterações da fase de geração do conjunto elite e da fase híbrida são divididas nos blocos B_1 e B_2 para a avaliação dos valores de K, da mesma forma que na versão seqüencial. As versões necessitam, então, de dois pontos de sincronização adicionais ao fim do bloco B_1 de cada fase. Na versão estática, os processos se comunicam nestes pontos para trocarem as informações de avaliação de K. Na versão mestre/escravo, o processo mestre é responsável por distribuir as iterações de cada bloco aos processos escravos e por realizar a avaliação dos valores de K nos momentos adequados.

Nos experimentos realizados, foram consideradas apenas as instâncias mais difíceis entre as que foram utilizadas nos experimentos do Capítulo 3. Estas instân-

cias correspondem às de tamanho de população $n = 400, 500$ e 600.

As Tabelas 5.1 e 5.2 ilustram os tempos de execução obtidos pelas versões paralelas. As duas primeiras colunas contêm os dados que caracterizam as instâncias. A terceira coluna indica a quantidade de processadores utilizados. As duas colunas seguintes apresentam os resultados obtidos sem a presença de carga externa, sendo uma coluna para os resultados da versão estática e a outra para a versão mestre/escravo. Valores em negrito indicam qual versão foi mais rápida. As duas últimas colunas contêm os resultados com a presença de carga externa. Note que, nestas duas últimas colunas, o tempo de execução para um único processador não é exibido. Para que a avaliação da aceleração das versões paralelas sob a presença de carga externa fosse consistente, os testes feitos com 4 processadores foram utilizados como base tanto para a versão estática quanto para a versão mestre/escravo e, portanto, não foram feitos testes com um único processador.

Note que, para os testes realizados sem a presença de carga externa, houve um equilibrio entre os tempos de execução gastos pelas versões estática e mestre/escravo, com ligeira vantagem para a última. Na versão estática, os processos possuem cargas equivalentes e a comunicação entre eles é menor que a necessária na versão mestre/escravo. No entanto, a vantagem da versão mestre/escravo pode ser explicada pelo fato de uma iteração ser considerada uma tarefa e algumas iterações requerem mais tempo computacional que outras, tornando as cargas dos processos da versão estática não perfeitamente equivalentes. Por isso, processos que alcançam os pontos de sincronização ficam ociosos até que os outros processos façam o mesmo. O impacto dos pontos de sincronização na versão mestre/escravo é menor, devido à distribuição de carga sob demanda. Para os testes com presença de carga externa, a versão mestre/escravo obteve desempenho substancialmente superior ao da versão estática, como esperado.

Para avaliar as acelerações obtidas pelas versões paralelas, foram gerados os gráfico das Figuras 5.1 e 5.2. No gráfico da Figura 5.1, cada curva representa a aceleração média (considerando as 12 instâncias) das versões estática e mestre/escravo, sem a presença de carga externa. O gráfico da Figura 5.2 é análogo ao anterior, só

Tabela 5.1: Tempos de execução, em segundos, das versões paralelas do GRASP-MD para o Problema da Maximização da Diversidade

Instância			Sem carga externa		Com carga externa	
n	m	#Processadores	Estática	Mestre/Escravo	Estática	Mestre/Escravo
400	40	1	1518.6	1518.6	-	-
		4	392.2	**382.6**	1439.0	**749.1**
		8	201.7	**193.8**	754.6	**397.3**
		16	103.2	**99.1**	383.6	**205.7**
	80	1	6942.6	6942.6	-	-
		4	1772.4	**1746.5**	6670.8	**3385.8**
		8	903.9	**880.3**	3524.9	**1707.6**
		16	473.6	**383.7**	1810.5	**906.9**
	120	1	14505.6	14505.6	-	-
		4	3744.4	**3648.9**	14301.4	**7018.2**
		8	1912.2	**1842.5**	7397.2	**3640.1**
		16	984.5	**946.0**	3791.9	**1896.0**
	160	1	22858.9	22858.9	-	-
		4	**5814.1**	6040.4	22943.0	**11129.4**
		8	3037.4	**2948.2**	11746.0	**5712.8**
		16	1595.8	**1536.5**	6111.5	**2999.8**
500	50	1	3729.63	3729.63	-	-
		4	957.1	**938.2**	3693.7	**1820.5**
		8	488.7	**476.6**	1877.8	**929.5**
		16	250.9	**243.2**	957.9	**483.5**
	100	1	16876.1	16876.1	-	-
		4	4363.2	**4324.1**	16547.9	**8323.9**
		8	2216.7	**2164.5**	8347.5	**4157.6**
		16	1146.3	**1111.3**	4338.5	**2183.1**

que corresponde aos testes realizados com a presença de carga externa.

Observe que, em ambos os casos, a escalabilidade do método é praticamente

Tabela 5.2: Tempos de execução, em segundos, das versões paralelas do GRASP-MD para o Problema da Maximização da Diversidade — continuação

Instância			Sem carga externa		Com carga externa	
n	m	#Processadores	Estática	Mestre/Escravo	Estática	Mestre/Escravo
500	150	1	31895.6	31895.6	-	-
		4	**8175.3**	8443.4	32063.3	**15535.4**
		8	4165.6	**4054.8**	16327.6	**8080.4**
		16	2172.7	**2085.7**	8534.0	**4139.8**
	200	1	51929.9	51929.9	-	-
		4	**13216.0**	15374.3	51228.2	**25431.5**
		8	**6754.0**	7178.9	26225.3	**13058.0**
		16	3470.4	**3364.6**	13585.4	**6796.9**
600	60	1	8317.2	8317.2	-	-
		4	2854.1	**2098.3**	8165.1	**4056.0**
		8	**1104.0**	1134.2	4120.1	**2093.4**
		16	575.2	**542.2**	2110.0	**1093.6**
	120	1	37780.9	37780.9	-	-
		4	**9700.5**	9911.8	37963.4	**18416.5**
		8	4895.3	**4819.2**	19357.0	**9471.4**
		16	2512.6	**2439.2**	9850.0	**4902.3**
	180	1	74258.7	74258.7	-	-
		4	**18945.7**	21790.5	73814.3	**36084.2**
		8	9742.0	**9553.0**	37959.7	**18385.6**
		16	5110.9	**4909.7**	19707.6	**9716.1**
	240	1	119189	119189	-	-
		4	**30839.8**	35728.9	118917.0	**58533.2**
		8	15992.7	**15440.4**	61196.7	**29939.8**
		16	8551.7	**8017.5**	32114.9	**15867.4**

linear, tanto para a versão estática quanto para a versão mestre/escravo.

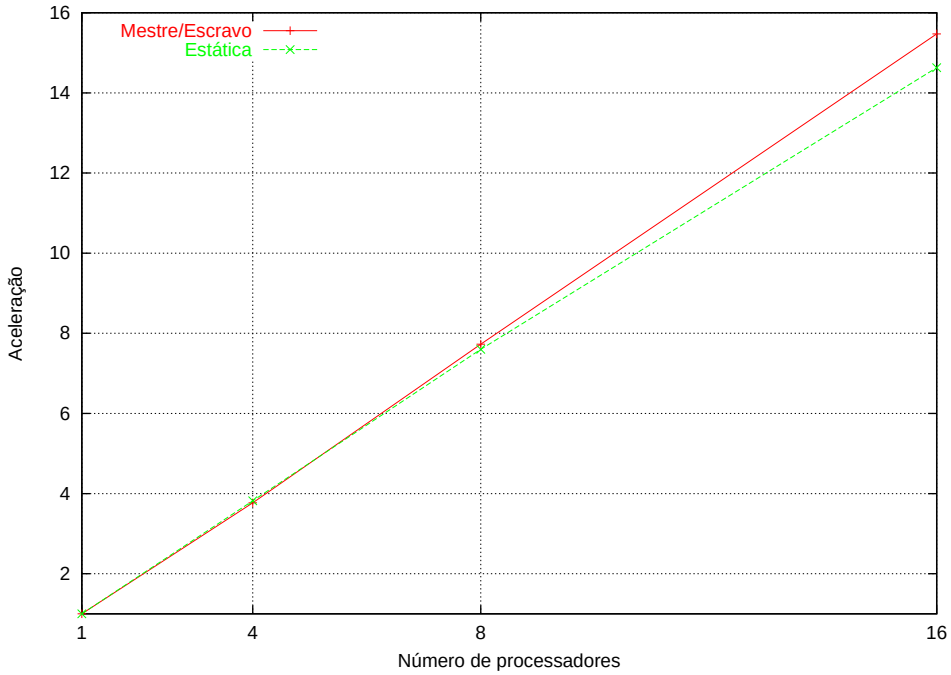

Figura 5.1: Aceleração das implementações paralelas do GRASP-MD para o problema da maximização da diversidade sem a presença de carga externa

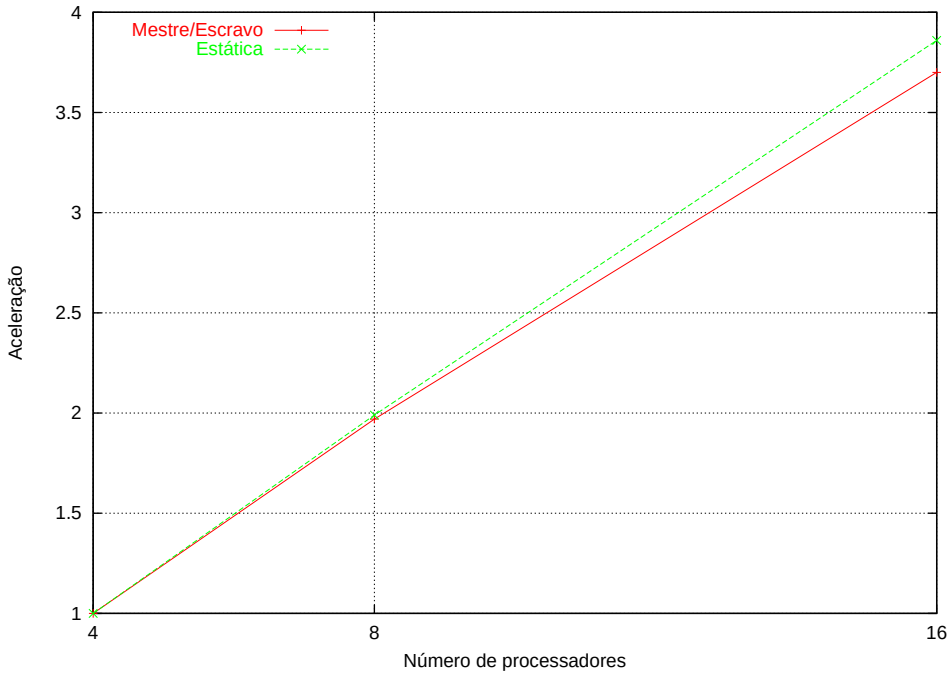

Figura 5.2: Aceleração das implementações paralelas do GRASP-MD para o problema da maximização da diversidade com a presença de carga externa

5.3.2 Paralelização do GRASP-MD para o Problema de Replicação de Servidores para Transmissão *Multicast* Confiável

Nas paralelizações desenvolvidas para o problema de replicação de servidores para transmissão *multicast* confiável, não foram necessários outros pontos de sincronização além do previsto na Seção 5.2 e, por isso, foram realizadas seguindo exatamente os modelos contidos naquela Seção.

Nos experimentos realizados, foram consideradas todas as instâncias utilizadas na avaliação da versão seqüencial do GRASP-MD.

As Tabelas 5.3, 5.4, 5.5 e 5.6 ilustram os tempos de execução obtidos pelas versões paralelas. As duas primeiras colunas contêm os dados que caracterizam as instâncias. A terceira coluna indica a quantidade de processadores utilizados. As duas colunas seguintes apresentam os resultados obtidos sem a presença de carga externa, sendo uma coluna para os resultados da versão estática e a outra para a versão mestre/escravo. Valores em negrito indicam qual versão foi mais rápida. As duas últimas colunas contêm os resultados com a presença de carga externa. Note que, da mesma forma que ocorreu na Subseção 5.3.1, o tempo de execução para um único processador não é exibido, pois, para a avaliação da aceleração das versões paralelas na presença de carga externa, foram considerados como base os tempos obtidos com a utilização de 4 processadores para ambas as versões.

Da mesma forma que ocorreu para o caso do problema da maximização da diversidade, os resultados obtidos pelas versões estática e mestre/escravo sem a presença de carga externa foram bastante similares. Porém, neste caso, a vantagem da versão mestre/escravo foi ainda maior. Isto reforça a idéia de que a distribuição de um mesmo número de tarefas para cada processador não proporciona um balanceamento de carga equivalente entre os processos da versão estática. Na presença de carga externa, novamente a versão mestre/escravo obteve resultados significativamente superiores.

Os gráficos das Figuras 5.3 e 5.4 ilustram as acelerações médias (conside-

Tabela 5.3: Tempos de execução, em segundos, das versões paralelas do GRASP-MD para o Problema de Replicação de Servidores para Transmissão *Multicast* Confiável

Instância			Sem carga externa		Com carga externa	
Cenário	m	#Processadores	Estática	Mestre/Escravo	Estática	Mestre/Escravo
$CONF_1$	5	1	40490.3	40490.3	-	-
		4	10257.3	**10193.8**	38857.6	**19617.2**
		8	5379.4	**5174.9**	19673.2	**10054.0**
		16	2878.1	**2655.7**	10462.1	**5682.8**
	10	1	60689.1	60689.1	-	-
		4	15818.2	**15407.8**	60374.9	**29657.2**
		8	8405.0	**7734.5**	30704.7	**15127.5**
		16	4315.9	**3920.1**	18046.3	**8456.3**
	15	1	61449.1	61449.1	-	-
		4	15942.9	**15636.4**	61635.0	**30050.5**
		8	8089.3	**7886.6**	31357.8	**15250.6**
		16	4244.0	**4008.7**	17426.3	**7997.8**
	20	1	55006.2	55006.2	-	-
		4	14257.7	**13876.0**	53762.4	**26838.8**
		8	7405.2	**6966.8**	27139.3	**13487.7**
		16	3890.1	**3524.4**	14028.9	**7022.6**
$CONF_2$	10	1	5394.5	5394.5	-	-
		4	1470.9	**1358.2**	5798.3	**2648.3**
		8	753.9	**683.9**	3001.7	**1336.4**
		16	392.9	**347.5**	1507.1	**694.7**
	20	1	6702.8	6702.8	-	-
		4	1746.8	**1681.1**	6470.1	**3248.3**
		8	889.3	**844.3**	3354.2	**1645.5**
		16	453.4	**427.5**	1662.2	**840.7**

rando as 20 instâncias) obtidas pelas versões paralelas na presença e na ausência de carga externa, respectivamente. Novamente, a escalabilidade do método foi pratica-

Tabela 5.4: Tempos de execução, em segundos, das versões paralelas do GRASP-MD para o Problema de Replicação de Servidores para Transmissão *Multicast* Confiável — continuação

Instância			Sem carga externa		Com carga externa	
Cenário	m	#Processadores	Estática	Mestre/Escravo	Estática	Mestre/Escravo
$CONF_2$	30	1	8642.8	8642.8	-	-
		4	2241.2	**2163.1**	8437.7	**4179.9**
		8	1128.6	**1087.3**	4257.2	**2117.2**
		16	579.6	**551.8**	2159.4	**1089.6**
	40	1	9318.8	9318.8	-	-
		4	2382.9	**2332.1**	9264.4	**4501.9**
		8	1213.3	**1173.6**	4687.9	**2265.1**
		16	625.1	**592.1**	2380.6	**1152.9**
$CONF_3$	20	1	11695.4	11695.4	-	-
		4	3192.4	**2943.9**	10659.4	**5673.8**
		8	1625.1	**1472.9**	5617.0	**2870.6**
		16	820.1	**744.8**	2855.9	**1466.9**
	40	1	18448.4	18448.4	-	-
		4	4792.8	**4616.9**	17734.2	**8876.5**
		8	2434.1	**2322.9**	9242.7	**4516.9**
		16	1237.0	**1175.3**	4656.8	**2281.5**
	60	1	22886.3	22886.3	-	-
		4	5849.3	**5732.8**	22540.0	**11008.3**
		8	2971.1	**2882.4**	11461.6	**5540.1**
		16	1507.7	**1460.6**	5857.3	**2832.4**
	80	1	21647.4	21647.4	-	-
		4	5506.3	**5394.4**	21201.6	**10334.9**
		8	2786.0	**2717.4**	10891.1	**5264.2**
		16	1402.9	**1371.7**	5488.4	**2696.5**

Tabela 5.5: Tempos de execução, em segundos, das versões paralelas do GRASP-MD para o Problema de Replicação de Servidores para Transmissão *Multicast* Confiável — continuação

Instância			Sem carga externa		Com carga externa	
Cenário	m	#Processadores	Estática	Mestre/Escravo	Estática	Mestre/Escravo
$BROAD_1$	25	1	2235.1	2235.1	-	-
		4	569.1	**558.4**	2194.4	**1086.3**
		8	286.5	**280.9**	1103.4	**549.2**
		16	146.4	**142.3**	563.4	**275.3**
	50	1	5131.4	5131.4	-	-
		4	1298.5	**1270.2**	5071.1	**2447.2**
		8	656.7	**638.5**	2580.1	**1232.2**
		16	337.4	**322.8**	1293.1	**629.1**
	75	1	7283.3	7283.3	-	-
		4	1845.6	**1790.8**	7226.4	**3456.1**
		8	931.3	**899.9**	3625.5	**1730.2**
		16	473.1	**456.2**	1840.7	**901.1**
	100	1	9210.1	9210.1	-	-
		4	2349.7	**2261.8**	8940.1	**4346.5**
		8	1180.6	**1138.9**	4503.3	**2203.6**
		16	600.5	**574.9**	2284.9	**1104.1**
$BROAD_2$	50	1	6786.6	6786.6	-	-
		4	1713.9	**1680.8**	6706.1	**3233.9**
		8	866.2	**846.5**	3380.5	**1623.2**
		16	442.8	**429.1**	1717.2	**821.9**
	100	1	11801.8	11801.8	-	-
		4	2989.5	**2907.1**	11549.3	**5564.9**
		8	1510.1	**1463.2**	5849.1	**2828.1**
		16	768.7	**740.4**	2950.6	**1464.7**

Tabela 5.6: Tempos de execução, em segundos, das versões paralelas do GRASP-MD para o Problema de Replicação de Servidores para Transmissão *Multicast* Confiável — continuação

Instância			Sem carga externa		Com carga externa	
Cenário	m	#Processadores	Estática	Mestre/Escravo	Estática	Mestre/Escravo
$BROAD_2$	150	1	14143.0	14143.0	-	-
		4	3571.1	**3465.4**	13840.2	**6681.8**
		8	1799.1	**1744.2**	6973.1	**3390.4**
		16	916.5	**885.4**	3542.6	**1673.3**
	200	1	17952.9	17952.9	-	-
		4	4492.6	**4387.9**	17709.1	**8474.1**
		8	2269.8	**2209.6**	8901.8	**4266.8**
		16	1156.1	**1122.7**	4552.3	**2119.5**

mente linear para ambas as versões paralelas desenvolvidas.

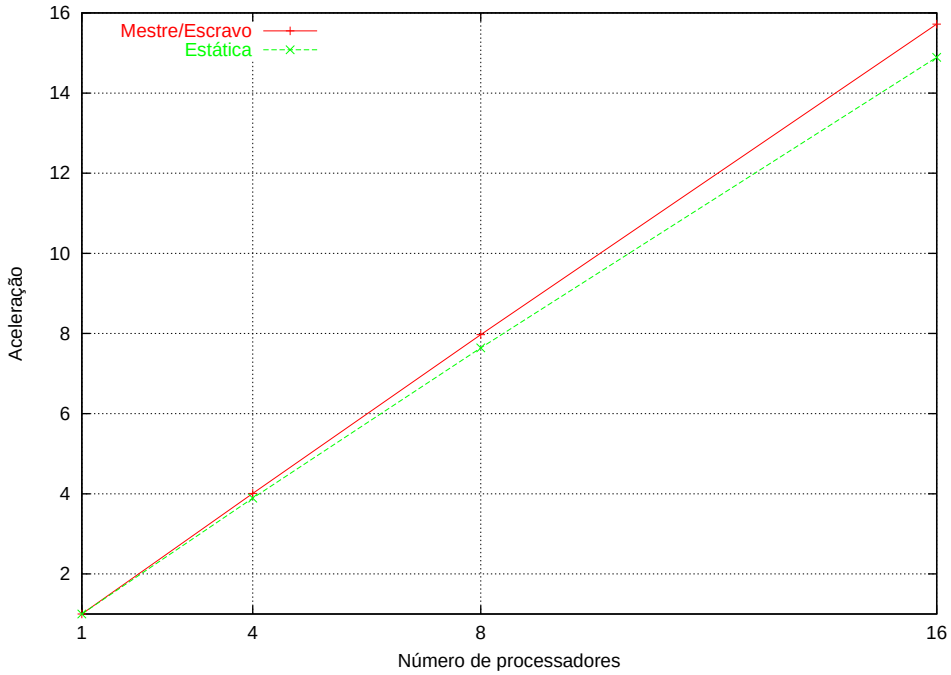

Figura 5.3: Aceleração das implementações paralelas do GRASP-MD para o problema de replicação de servidores para transmissão *multicast* confiável sem a presença de carga externa

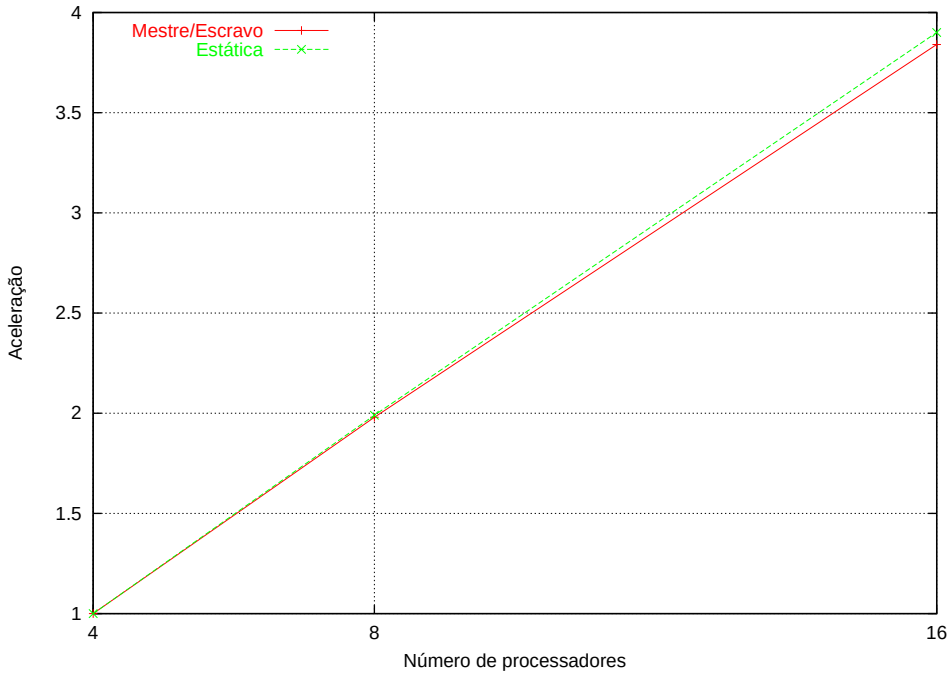

Figura 5.4: Aceleração das implementações paralelas do GRASP-MD para o problema de replicação de servidores para transmissão *multicast* confiável com a presença de carga externa

Capítulo 6

Conclusões

A incorporação de técnicas de mineração de dados à metaheuristica GRASP foi proposta em [53, 54, 55] com o objetivo de adicionar a esta metaheurística uma forma de aprendizado de máquina. A idéia fundamental era utilizar uma técnica de mineração de dados para identificar padrões de soluções de boa qualidade e, em seguida, utilizar esses padrões para guiar a busca por melhores soluções. Nestes trabalhos, o método resultante, chamado GRASP-MD, foi aplicado ao problema de empacotamento de conjuntos e os resultados foram bastante satisfatórios.

O primeiro objetivo desta dissertação foi evidenciar a eficiência e robustez do método híbrido. Para isto, foram realizadas aplicações do GRASP-MD em dois outros problemas de Otimização Combinatória: o problema da maximização da diversidade e o problema de replicação de servidores para transmissão *multicast* confiável. Os resultados de ambas as aplicações demonstraram que o método híbrido foi capaz de alcançar soluções melhores que o GRASP original em tempo computacional menor, o que comprova que a incorporação de técnicas de mineração de dados à metaheurística GRASP contribui para aperfeiçoar sua eficiência.

Sistemas computacionais paralelos e distribuídos oferecem poder computacional significativamente maior que arquiteturas seqüenciais tradicionais. Uma tendência na área de Otimização Combinatória é a paralelização das técnicas existentes

para obtenção de soluções para problemas intratáveis. O segundo objetivo desta dissertação foi avaliar o desempenho de implementações paralelas do método híbrido GRASP-MD. Para cada um dos problemas mencionados, foram desenvolvidas duas versões paralelas utilizando estratégias de balanceamento de carga diferentes. Em uma delas, a carga foi distribuída estaticamente entre os processos e, na outra, esta distribuição foi dinâmica. Para avaliar o comportamento de cada estratégia, foi realizado um conjunto de experimentos livre de cargas externas e outro conjunto com a presença de carga externa simulada. Apesar de a realização do procedimento de mineração de dados requerer um ponto de sincronização entre os processos, os resultados experimentais demonstraram que o método tem escalabilidade linear em relação ao número de processadores utilizados. Os resultados demonstraram ainda que a estratégia dinâmica de balanceamento de carga obteve desempenho superior à estratégia estática.

A análise dos resultados obtidos permite concluir que a redução de tempo computacional do GRASP-MD em relação ao GRASP original é conseqüência da melhora significativa da qualidade das soluções geradas na fase de construção, que é proporcionada pela utilização dos padrões extraídos a partir de um conjunto elite de soluções. Com isso, a fase de busca local, que, em geral, consome a maior parte do tempo de execução do GRASP, necessita de tempo computacional senvivelmente menor para alcançar ótimos locais. Também foi possível notar que a qualidade média das soluções do conjunto elite aumentava durante as primeiras iterações, mas se estabilizava, na maioria dos casos, antes do término das iterações da fase de geração deste conjunto. Uma importante investigação futura seria a identificação de um número ideal de iterações para a fase de geração do conjunto elite. Isto evitaria a execução de iterações desnecessárias, fazendo com que os tempos de execução do método híbrido fossem ainda menores.

A avaliação do desempenho das versões paralelas do GRASP-MD realizada nesta dissertação objetivou apenas a análise da escalabilidade do método quando executado em ambientes paralelos. Outra possível investigação futura seria no sentido de avaliar estratégias cooperativas de paralelização que, além de possibilitar acelerações lineares, também fossem capazes de obter soluções de melhor qualidade que a versão seqüencial.

Referências Bibliográficas

[1] *TOP500 Supercomputing Sites*. http://www.top500.org, Novembro 2006.

[2] Agrawal, R., e Srikant, R. Fast algorithms for mining association rules in large databases. Em *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases* (San Francisco, CA, USA, 1994), Morgan Kaufmann Publishers Inc., pp. 487–499.

[3] Ahuja, R., Orlin, J., e Tiwari, A. A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research 27* (2000), 917–934.

[4] Almeroth, K. C. The evolution of multicast: From the MBone to interdomain multicast to Internet2 deployment. *IEEE Network* (Janeiro/Fevereiro 2000), 10–20.

[5] Andrade, M., Andrade, P., Martins, S., e Plastino, A. GRASP with path-relinking for the maximum diversity problem. Em *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms, LNCS 3503* (2005), pp. 558–569.

[6] Andrade, P., Plastino, A., Ochi, L., e Martins, S. GRASP for the maximum diversity problem. Em *Proceedings of MIC 2003* (2003).

[7] Atwood, J. W. A classification of reliable multicast protocols. *IEEE Network* (Maio/Junho 2004), 24–34.

[8] Calvert, K., Doar, M., e Zegura, E. Modeling Internet topology. *IEEE Communications Magazine 35*, 6 (Junho 1997), 160–163.

[9] Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M., e Pinkas, B. Multicast Security: A taxonomy and some efficient constructions. Em *Proceedings of the IEEE INFOCOM'99* (1999), vol. 2, pp. 708–716.

[10] Cung, V.-D., Martins, S., Ribeiro, C., e Roucairol, C. Strategies for the parallel implementation of metaheuristics. Em *Essays and Surveys in Metaheuristics*, C. R. e P. Hansen, Ed. Kluwer, 2001, pp. 263–308.

[11] Deering, S. E. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, 1991.

[12] Delorme, X., Gandibleux, X., e Rodriguez, J. GRASP for set packing problems. *European Journal of Operational Research 153* (2003), 564–580.

[13] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik 1* (1959), 269–271.

[14] Diot, C., Dabbous, W., e Crowcroft, J. Multipoint communication: a survey of protocols, functions and mechanisms. *IEEE Journal on Selected Areas in Communications 15:3* (Abril 1997), 277–290.

[15] Dorigo, M., e Stützle, T. *Ant Colony Optimization*. MIT Press, 2004.

[16] Eriksson, H. MBONE: The multicast backbone. *Communications of the ACM 37:8* (Agosto 1994), 54–60.

[17] Feo, T., e Resende, M. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization 6* (1995), 109–133.

[18] Feo, T., Resende, M., e Smith, S. A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. *Operations Research 42* (1994), 860–878.

[19] Festa, P., e Resende, M. GRASP: An annotated bibliography. Em *Essays and Surveys in Metaheuristics*, C. C. Ribeiro e P. Hansen, Eds. Kluwer Academic Publishers, 2002, pp. 325–367.

[20] Foster, I. *The Grid: Blueprint for a New Computing Infrastructure*, Second ed. Morgan Kaufmann, 2004.

[21] Garey, M., e Johnson, D. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[22] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., e Sunderam, V. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

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

[24] Glover, F. Tabu Search – Part I. *ORSA Journal on Computing 1*, 3 (1989), 190–206.

[25] Glover, F. Tabu Search – Part II. *ORSA Journal on Computing 2*, 1 (1990), 4–32.

[26] Glover, F., Hersh, G., e McMillan, C. Selecting subsets of maximum diversity. MS/IS Report 77-9, University of Colorado at Boulder, 1977.

[27] Glover, F., Kuo, C.-C., e Dhir, K. Integer programming and heuristic approaches to the minimum diversity problem. *Journal of Business and Management 4* (1996), 93–111.

[28] Glover, F., e Laguna, M. *Tabu Search*. Kluwer Academic Publishers, 1997.

[29] Glover, F., Laguna, M., e Martí, R. Scatter Search. Em *Advances in evolutionary computing: theory and applications*. Springer-Verlag, 2003, pp. 519–537.

[30] Goethals, B., e Zaki, M. Advances in frequent itemset mining implementations: report on FIMI'03. *SIGKDD Explorations Newsletter 6*, 1 (2004), 109–117.

[31] Goldberg, D. *Genetic Algorithms in Search, Optimization and Machine Learning*. Kluwer Academic Publishers, 1989.

[32] Grahne, G., e Zhu, J. Efficiently using prefix-trees in mining frequent itemsets. Em *Proceedings of the First IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)* (2003).

[33] Han, J., e Kamber, M. *Data Mining: Concepts and Techniques*, Second ed. Morgan Kaufmann, 2006.

[34] Han, J., Pei, J., e Yin, Y. Mining frequent patterns without candidate generation. Em *SIGMOD'00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2000), ACM Press, pp. 1–12.

[35] Holland, J. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[36] Hopcroft, J., Motwani, R., e Ullman, J. *Introduction to Automata Theory, Languages, and Computation*, Second ed. Addison-Wesley, 2001.

[37] Kasera, S. K., Kurose, J., e Towsley, D. A comparison of server-based and receiver-based local recovery approaches for scalable reliable multicast. Em *Proceedings of the IEEE INFOCOM'98* (1998), vol. 3, pp. 988–995.

[38] Kirkpatrick, S., Gelatt, C., e Vecchi, M. Optimization by Simulated Annealing. *Science 220*, 4598 (1983), 671–680.

[39] Kochenberger, G., e Glover, F. Diversity data mining. Working paper, 1999.

[40] Kuo, C.-C., Glover, F., e Dhir, K. Analyzing and Modeling the Maximum Diversity Problem by Zero-One Programming. *Decision Sciences 24*, 6 (1993), 1171–1185.

[41] Laguna, M., e González-Velarde, J. A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing 2* (1991), 253–260.

[42] Li, B., F.Chen, e Yin, L. Server Replication and its Placement for Reliable Multicast. Em *Proceedings of the 9th International Conference on Computer Communications and Networks (IC3N'2000)* (Outubro 2000), pp. 396–401.

[43] Martins, S., Resende, M., Ribeiro, C., e Pardalos, P. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization 17* (2000), 267–283.

[44] Message Passing Interface Forum. MPI: A message-passing interface standard. Relatório técnico, 1994.

[45] Mladenović, N., e Hansen, P. Variable Neighborhood Search. *Computers And Operations Research 24* (1997), 1097–1100.

[46] Nonnenmacher, J., Lacher, M. S., Jung, M., Biersack, E., e Carle, G. How bad is reliable multicast without local recovery? Em *Proceedings of the IEEE INFOCOM'98* (1998), vol. 3, pp. 972–979.

[47] Orlando, S., Palmerini, P., e Perego, R. Enhancing the apriori algorithm for frequent set counting. Em *DaWaK '01: Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery* (London, UK, 2001), Springer-Verlag, pp. 71–82.

[48] Osman, I., e Laporte, G. Metaheuristics: A bibliography. *Annals of Operations Research 63* (1996), 513–623.

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

[50] Resende, M., e Ribeiro, C. Greedy randomized adaptive search procedures. Em *Handbook of Metaheuristics*, F. Glover e G. Kochenberger, Eds. 2003, pp. 219–249.

[51] Resende, M., e Ribeiro, C. GRASP with path-relinking: Recent advances and applications. Em *Metaheuristics: Progress as Real Problem Solvers*, T. Ibaraki, K. Nonobe, e M. Yagiura, Eds. Kluwer Academic Publishers, 2005.

[52] Resende, M., e Ribeiro, C. Parallel greedy randomized adaptive search procedures. Em *Parallel Metaheuristics: A New Class of Algorithms*, E. Alba, Ed. Wiley, 2005, pp. 315–346.

[53] Ribeiro, M. Incorporando Técnicas de Mineração de Dados à Metaheurística GRASP. *Dissertação de Mestrado*, Instituto de Computação – Universidade Federal Fluminense (2005).

[54] Ribeiro, M., Plastino, A., e Martins, S. Hybridization of GRASP metaheuristic with data mining techniques. *Journal of Mathematical Modelling and Algorithms 5*, 1 (2006), 23–41.

[55] Ribeiro, M., Trindade, V., Plastino, A., e Martins, S. Hybridization of GRASP metaheuristic with data mining techniques. Em *Proceedings of the 1st International Workshop on Hybrid Metaheuristics* (2004), pp. 69–78.

[56] Santos, L. F., Ribeiro, M. H., Plastino, A., e Martins, S. A hybrid GRASP with data mining for the maximum diversity problem. Em *Proceedings of the 2nd International Workshop on Hybrid Metaheuristics* (2005), pp. 116–127.

[57] Santos, L. F. M., Milagres, R., Albuquerque, C., Plastino, A., e Martins, S. A hybrid GRASP with data mining for efficient server replication for reliable multicast. Em *Proceedings of the IEEE GLOBECOM'06 Conference* (San Francisco, Novembro 2006).

[58] Silva, G. C., Ochi, L. S., e Martins, S. L. Experimental Comparison of Greedy Randomized Adaptive Search Procedures for the Maximum Diversity Problem. *Lecture Notes on Computer Science 3059* (2004), 498–512.

[59] Talbi, E.-G. A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics 8*, 5 (2002), 541–564.

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