UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

CRITÉRIOS DE PARADA BASEADOS EM PROBABILIDADE BAYESIANA APLICADOS A HEURÍSTICAS GRASP
UM ESTUDO EXPERIMENTAL

Layni Andrade Neves

Orientadores
Adriana Cesário de Faria Alvim
Mauricio Guilherme de Carvalho Resende

RIO DE JANEIRO, RJ - BRASIL
MAIO DE 2009

CRITÉRIOS DE PARADA BASEADOS EM PROBABILIDADE BAYESIANA APLICADOS A HEURÍSTICAS GRASP

UM ESTUDO EXPERIMENTAL

Layni Andrade Neves

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM INFORMÁTICA.

Aprovada por:

Prof. Adriana Cesário de Faria Alvim, D.Sc.

Dr. Mauricio Guilherme de Carvalho Resende, Ph.D.

Prof. Alexandre Albino Andreatta, D.Sc.

Prof. Reinaldo Castro Souza, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

MAIO DE 2009

Neves, Layni Andrade.
N518 Critérios de parada baseados em probabilidade bayesiana aplicados a heuristica GRASP : um estudo experimental / Layni Andrade Neves, 2009.
ix, 83f.

Orientador: Adriana Cesário de Faria Alvim.
Co-orientador: Mauricio Guilherme de Carvalho Resende.
Dissertação (Mestrado em Informática) – Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2009.

1. GRASP. 2. Estatistica bayesiana. 3. Heuristica. 4. Otimização combinatória. 5. Critério de parada. I. Alvim, Adriana Cesário de Faria. II. Resende, Mauricio Guilherme de Carvalho. III. Universidade Federal do Estado do Rio de Janeiro (2003-). Centro de Ciências Exatas e Tecnologia. Curso de Mestrado em Informática. IV. Título.

CDD – 005.1

Agradecimentos

A minha orientadora Prof. Adriana Alvim pela orientação criteriosa, pela dedicação e pelos ensinamentos.

Ao meu co-orientador Mauricio Resende pela orientação, pela disponibilização de material de estudo e pela contribuição no artigo desenvolvido durante o trabalho.

A minha família pela paciência, apoio e encorajamento.

Aos professores e funcionários do Departamento de Informática da UNIRIO pelo auxílio neste perído de estudo.

Aos professores Alexandre Andreatta e Reinaldo Castro Souza pelas contribuições na versão final da dissertação.

NEVES, Layni Andrade. **Critérios de Parada baseados em Probabilidade Bayesiana aplicados a Heurísticas GRASP - Um Estudo Experimental.** UNIRIO, 2009. 83 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

RESUMO

O uso de algoritmos baseados em metaheurísticas é de relevante importância para a resolução de diversos problemas de otimização combinatória. Estes algoritmos procuram uma solução viável de boa qualidade quando é difícil encontrar a solução exata devido à complexidade computacional do problema. Uma das questões fundamentais em heurísticas baseadas em metaheurísticas é equilibrar tempo de processamento e qualidade da solução. Este trabalho apresenta a implementação e teste de critérios de parada baseados em estatística bayesiana em heurísticas baseadas na metaheurística GRASP. O objetivo é escolher um critério de parada que resulte em um tempo de processamento curto da heurística, e que, ao mesmo tempo, alcance uma solução de boa qualidade. No experimento computacional foram comparadas cinco implementações de heurísticas GRASP previamente publicadas que utilizam como critério de parada um número fixo de iterações, com novas versões que utilizam o critério de parada bayesiano.

Palavras-chave: GRASP; Estatística Bayesiana; Heurística; Otimização Combinatória; Critério de Parada.

NEVES, Layni Andrade. **Critérios de Parada baseados em Probabilidade Bayesiana aplicados a Heurísticas GRASP - Um Estudo Experimental.** UNIRIO, 2009. 83 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

ABSTRACT

The use of algorithms based on the GRASP metaheuristic is relevant for the solution of many combinatorial optimization problems. These algorithms find a feasible solution of good quality when is difficult to find an exact solution because of the problem's computational complexity. One of the key issues in designing heuristics is the balance between processing time and solution quality. This thesis presents the implementation and testing of Bayesian stopping rules in GRASP heuristics. The objective is to choose a stopping rule that results in a short running time of the heuristic while at the same time achieving good solution quality. In the computational experiment, we compared, on five previously published GRASP heuristics, the original version using a stopping rule based on a fixed number of iterations with versions that use Bayesian stopping rules.

Keywords: GRASP; Bayesian Statistic; Heuristic; Combinatorial Optimization; Stopping Rule.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Problemas de otimização combinatória aparecem quando temos que selecionar de um conjunto discreto e finito de dados o melhor subconjunto que satisfaz a determinados critérios. Esta tarefa é frequentemente muito complexa e o tempo é limitado. Em muitas situações, pode ser impossível garantir a obtenção de soluções exatas (ótimas). Nestes casos, a melhor opção tem sido usar heurísticas. Uma heurística é um método que busca soluções *viáveis* (que satisfazem os critérios de restrição), geralmente satisfatórias, para um problema de otimização específico.

Uma metaheurística é um *framework* (estrutura) genérico, composto pela combinação de procedimentos heurísticos genéricos, feito para resolver problemas de otimização combinatória em geral. O projeto de implementação de algoritmos baseados nesta estrutura é uma tarefa muito relevante. Tais heurísticas encontram soluções viáveis de boa qualidade quando é difícil encontrar soluções ótimas devido à complexidade do problema.

Uma das questões fundamentais em heurísticas derivadas de metaheurísticas é achar uma solução de qualidade cujo valor seja o mais próximo possível do valor da solução ótima, no menor tempo de processamento possível, e dentro de um limite de tempo. Uma pergunta natural é como saber quando a qualidade da solução é suficientemente boa, uma vez que se desconhece o valor da solução ótima. Esta é a principal motivação para o estudo de um critério de parada estatístico.

O problema que se deseja tratar nesta dissertação é o de equilibrar tempo de processamento e qualidade da solução, pela determinação dinâmica de um bom momento de parada de uma dada heurística. Em particular, propõe-se fazer um estudo comparativo da

aplicação de critérios de parada baseados em estatística bayesiana propostos por Boender e Rinnooy Kan [1] aplicados a heurísticas baseadas na metaheurística GRASP [4, 5]. O estudo comparativo tem a finalidade de avaliar o *desempenho* destas técnicas e corroborar sua utilidade em metaheuristicas que utilizam técnicas de multi-inicialização e busca local como o GRASP.

Alguns conceitos relevantes para o estudo em questão sobre a área de otimização combinatória são revisados no Capítulo 2 e sobre a metaheurística GRASP são revisados no Capítulo 3. Em seguida, abordamos no Capítulo 4 a parte teórica dos critérios de parada inteligentes. Na Seção 4.3 é apresentado o *framework* `GRASP_BSR`, do inglês *GRASP with Bayesian Stopping Rule*, que implementa o critério de parada bayesiano proposto por Boender e Rinnooy Kan [1]. O Capítulo 5 apresenta as cinco heurísticas baseadas na metaheurística GRASP sobre as quais são aplicados os critérios de parada estudados nesta dissertação. No Capítulo 6 são apresentados o projeto experimental e os resultados computacionais. Por fim, no Capítulo 7 apresentam-se as contribuições baseadas nas análises dos resultados computacionais.

2 Conceitos básicos

Este capítulo aborda fundamentos da área de otimização combinatória e revisa alguns conceitos básicos da área de heurísticas para problemas de otimização combinatória, segundo Hoos e Stützle [2].

2.1 Problemas combinatórios

Problemas combinatórios aparecem em muitas áreas da Ciência da Computação, Inteligência Artificial, Pesquisa Operacional, Bio-informática, etc. Tais problemas tipicamente envolvem a procura de grupos, ordenações ou atribuições de um conjunto discreto e finito de objetos que satisfaçam certas condições ou restrições. A combinação destes *componentes de solução* constituem uma solução potencial para o problema combinatório.

Problema *abstrato* refere-se à forma generalizada do problema, como, por exemplo, "encontrar o trajeto mais curto entre quaisquer duas cidades dentro de qualquer conjunto de cidades". *Instância do problema* corresponde ao problema específico, como por exemplo, "encontrar um caminho mais curto entre a cidade Petrópolis e a cidade Cuiabá utilizando o conjunto de rotas viárias do Brasil".

Um *problema de otimização combinatória* é um problema combinatório onde a solução do problema combinatório (solução candidata do problema de otimização) é avaliada por uma função objetivo e a finalidade é encontrar uma solução com valor da função objetivo ótimo. O valor da *função objetivo* de uma dada solução é a qualidade da solução candidata. Um problema de otimização pode ser de maximização ou minimização. Algoritmicamente ambos são tratados de forma equivalente. Problemas de otimização combinatória

podem ser distinguidos em duas variantes: variante de busca (o objetivo é encontrar uma solução ótima) e variante de avaliação (o objetivo é encontrar o valor ótimo da função objetivo, isto é, a qualidade de uma solução ótima). Geralmente os dois objetivos são concomitantemente perseguidos.

Soluções candidatas que satisfazem integralmente as restrições do problema combinatório são chamadas *soluções viáveis*, e entre estas, *soluções ótimas* são distinguidas através do valor da função objetivo.

Problemas combinatórios geralmente podem ter um grande número de soluções candidatas, conforme o número de objetos envolvidos. De fato, o espaço de soluções potenciais para uma dada instância de um problema é no mínimo exponencial em relação ao tamanho da instância. A complexidade computacional de um algoritmo é caracterizada pela dependência funcional entre o tamanho de uma instância e o tempo e espaço requerido para resolver esta instância. *Problemas de decisão* são aqueles cujo objetivo é determinar se existe ou não um elemento de um conjunto que atende ou não a determinadas restrições. A classe de complexidade P é a classe dos problemas de decisão que podem ser resolvidos de forma determinística em tempo polinomial e a classe NP é a classe dos problemas que podem ser resolvidos de forma não-determinística em tempo polinomial. Todos os problemas em P estão também em NP, isto é, $P \subseteq NP$. Entretanto a questão se também $NP \subseteq P$, e conseqüentemente $P = NP$, é um dos problemas em aberto mais proeminentes da Ciência da Computação. Conjectura-se que $P \neq NP$. Muitos problemas relevantes estão em NP mas não se sabe se estão em P. Heurísticas para problemas de otimização são usadas quando a complexidade do problema, tamanhos de instâncias, e tempo disponível, não permitem que os algoritmos exatos obtenham soluções ótimas em tempo factível.

2.2 Métodos de busca

Segundo Hoos e Stützle [2], basicamente todas as abordagens computacionais para resolver problemas combinatórios difíceis podem ser caracterizadas como algoritmos de busca e a idéia fundamental por trás disso é iterativamente gerar e avaliar soluções candi-

datas.

Soluções candidatas são constituídas por *componentes de solução*. Portanto, alterando-se uma ou mais componentes da solução cria-se uma nova solução. É comum designar como *solução candidata parcial* quando se tem uma estrutura com componentes de solução, mas que ainda não atende as restrições do problema pela ausência do número mínimo de componentes necessárias para configurar-se uma solução candidata. *Métodos de busca construtivos* geram soluções candidatas completas por iterativamente expandir soluções candidatas parciais. O método construtivo pode ser formulado como um problema de busca na qual o objetivo é obter uma solução candidata boa, onde a qualidade corresponde ao valor da função objetivo (no caso de problemas de otimização).

Métodos de busca sistemáticos (exatos) exploram sistematicamente o conjunto finito de soluções candidatas (*o espaço de busca*) até encontrar uma solução ótima, caso exista, ou até percorrer todo o espaço em caso contrário. Esta propriedade é denominada integralidade de busca (*completeness*).

Métodos de busca local iniciam a procura em um dado ponto no espaço de busca (solução candidata inicial) e movem-se para um ponto vizinho (solução vizinha), de forma iterativa. Uma solução vizinha se diferencia da solução inicial em algum(s) componente(s) conforme critério pré-estabelecido. Nesta relação de vizinhança pré-determinada, cada ponto (solução candidata) tem um pequeno grupo de pontos vizinhos (soluções vizinhas). A relação pode ser simétrica, mas não é reflexiva nem transitiva. A escolha desta relação de vizinhança é crucial para o *desempenho* de um algoritmo de busca local. Um procedimento de busca local parte sempre de uma solução inicial $s_0 \in$ espaço de busca S e gera uma seqüência de soluções $s_1, s_2, \ldots, s_k$. Um *movimento* comum é o movimento que troca elementos em uma solução. Neste trabalho, denota-se por *troca((p,q),s)* o conjunto de movimentos que, considerando uma solução s, remove p componentes de s e insere q novos componentes em s, p e q constantes. A aplicação deste conjunto de movimentos resulta em um conjunto $N(s) \subseteq S$ de soluções, chamado de "vizinhança de s". Por exemplo, seja a solução $s = (1,2,3)$ e a vizinhança $N(s)$ gerada por movimentos de *troca((2,2),s)* aplicados a um vetor de permutação onde as componentes são as posições neste vetor, os

vizinhos serão $(3,2,1),(2,1,3),(1,3,2)$. A escolha do movimento é uma decisão baseada no exame da vizinhança da solução corrente. Esta decisão, tanto quanto a escolha do ponto (solução) inicial pode ser feita de forma aleatória.

Qualquer relação de vizinhança N em S pode ser representada por um grafo implícito sobre o espaço de busca S. Neste *grafo direcionado de vizinhança* G_N, dois vértices s, s' são conectados por uma aresta (s,s') se, e somente se, $(s,s') \in N$. Ou seja, para uma dada instância π do problema, o grafo de vizinhança induzido pela relação de vizinhança $N(\pi)$ no espaço de busca $S(\pi)$ é definido como $G_N(\pi) := (S(\pi), N(\pi))$.

Uma busca local pode visitar o mesmo ponto no espaço de busca mais de uma vez. Existem métodos de busca que armazenam um número determinado de soluções candidatas visitadas para tentar controlar a varredura do espaço de busca, para não visitar sempre os mesmos pontos.

Os métodos de busca local são considerados *incompletos*, isto é, não existe nenhuma garantia que caso uma solução ótima exista, esta seja encontrada.

2.2.1 Estratégias de busca

Uma das estratégias de busca mais simples é o *Passeio Aleatório* (*Random Walk*). Esta estratégia seleciona, de forma aleatória, com a mesma probabilidade, qualquer elemento no espaço de busca como ponto inicial para um processo de busca.

Com o objetivo de melhorar a qualidade das soluções encontradas pela estratégia do *Passeio Aleatório* utiliza-se uma *função de avaliação* como mecanismo para guiar a busca. Esta função é utilizada para avaliar ou ranquear soluções candidatas na vizinhança. Normalmente, a função objetivo é usada como função de avaliação.

Uma das estratégias básicas que usam uma função de avaliação é a de *Melhoria Iterativa*. Dado um espaço de busca S, uma relação de vizinhança N e a função de avaliação g; a Melhoria Iterativa inicia selecionando aleatoriamente um ponto no espaço de busca - a solução candidata $s \in S$ - e procura melhorar a qualidade da solução buscando no conjunto de todas as soluções candidatas vizinhas $s' \in N(s)$ uma solução para qual $g(s') < g(s)$. Dois métodos de Melhoria Iterativa muito utilizados são *Melhor Melhoria* e *Primeira Melhoria*.

A estratégia *Melhor Melhoria* é baseada na idéia de, em cada passo da busca, selecionar de forma aleatória uma das soluções candidatas vizinhas que alcance a melhoria máxima na função de avaliação. Dado uma solução $s \in S$ e seja $g^* = min\{g(s') \mid s' \in N(s)\}$ o melhor valor da função de avaliação na vizinhança de s. O conjunto de vizinhos que maximiza a melhoria de s é $\{s' \in N(s) \mid g(s') = g^*\}$. Note que a estratégia de Melhor Melhoria requer uma avaliação completa de todos os vizinhos em cada passo da busca.

A estratégia *Primeira Melhoria* reduz o tempo necessário para selecionar uma solução vizinha, pois executa o movimento da primeira melhoria encontrada durante a inspeção da vizinhança. Na solução corrente s, avaliam-se as soluções candidatas vizinhas em $N(s)$ em uma ordem fixa particular, e a primeira $s' \in N(S)$ para qual $g(s') < g(s)$ é selecionada. A ordem na qual os vizinhos são avaliados pode influenciar significativamente na eficiência desta estratégia. Também é importante lembrar que iniciar a busca de uma mesma solução inicial usando ordem fixa resulta no mesmo ótimo local. Ao invés de utilizar uma ordem fixa para avaliar os vizinhos de uma dada solução, é possível usar ordens aleatórias para alcançar diferentes soluções em cada iteração.

2.2.2 Ótimo local

Dado um espaço de busca S, um conjunto de soluções $S' \subseteq S$, uma relação de vizinhança $N \subseteq S \times S$ e uma função de avaliação $g : S \longmapsto \Re$, um *ótimo local* é uma solução candidata $s \in S$ tal que para todo $s' \in N(s)$, $g(s) \leq g(s')$ se o que se quer é minimizar a função de avaliação, ou $g(s) \geq g(s')$ se o que se quer é maximizar a função de avaliação. Ótimos locais ocorrem com muita freqüência e geralmente não trazem a qualidade desejada em um problema de otimização, por isto são utilizadas diversas técnicas para evitar ou escapar destas soluções.

Uma solução s' pertence à *região de atração* de um ótimo local s_0 quando, a partir de uma busca local determinística iniciada em s', o ótimo local s_0 será alcançado. Caso uma das soluções iniciais esteja na região de atração de um ótimo global, a busca local irá encontrar este ótimo global. Para aumentar a possibilidade de cair na região de atração do ótimo global, a geração de soluções iniciais deve ser mais aleatória, procurando abranger o máximo possível todo o espaço de busca, todas as regiões de atração. Por outro lado,

ao gerar soluções iniciais muito aleatórias, estas podem se distanciar muito da região de atração do ótimo global e com isso favorecer soluções finais de baixa qualidade. Algoritmos gulosos geralmente produzem soluções de melhor qualidade do que as geradas aleatoriamente, porém a diversidade de soluções finais é muito pequena. O desafio é garantir a diversidade de soluções e ao mesmo tempo um controle na qualidade das soluções produzidas.

```
procedimento GRASP(semente);
1  Ler_Dados();
2  k ← 1;
3  melhor_sol ← ∞;
4  faça
5      sol[k] ← Construção_Solução(semente);
6      se (sol[k] não é uma solução viável) então
7          sol[k] ← Reparo(sol[k]);
8      fim-se;
9      sol[k] ← Busca_Local(sol[k]);
10     Atualiza_Solução(sol[k], melhor_sol);
11     k ← k + 1;
12 enquanto (Critério_de_Parada (sol, melhor_sol, k));
13 fim-GRASP.
```

Figura 3.1: Metaheurística GRASP.

3 Metaheurística GRASP

Neste capítulo descrevem-se os fundamentos da metaheurística GRASP.

A metaheurística GRASP (greedy randomized adaptative search procedure) proposta por Feo e Resende (1989,1995) [4, 5] e comentada por Resende e Ribeiro (2003,2009) [6, 7] é um procedimento de busca adaptativa aleatória gulosa com múltiplas inicializações para problemas combinatórios difíceis, composta basicamente por duas fases: uma fase construtiva que produz uma solução e uma fase de busca local que explora a vizinhança da solução construída (Figura 3.1). Como a fase construtiva não necessariamente cria uma solução viável, às vezes faz-se necessário ter um procedimento que repara a solução para adequá-la as restrições do problema específico. O procedimento da Figura 3.1 será detalhado nas seções seguintes.

3.1 Fase de construção

Nesta fase, conforme ilustra o algoritmo representado na Figura 3.2, tenta-se construir iterativamente uma solução viável da seguinte forma. Constrói-se um conjunto de componentes da solução ordenado conforme uma função gulosa adaptativa que mede os benefícios de cada componente para a qualidade da solução do problema (o valor da função objetivo). A cada iteração faz-se uma seleção aleatória de um dos melhores componentes candidatos da solução dentro deste conjunto, chamado *lista restrita de candidatos* (LRC). Estes componentes só podem ser adicionados se não violarem as restrições de viabilidade. A função gulosa é de caráter adaptativo, pois se atualiza a cada iteração para incorporar as mudanças causadas pela escolha do último componente.

Existem duas estratégias básicas usadas para construir a lista restrita de candidatos. Uma delas é baseada em cardinalidade, onde para um valor inteiro k pré-estabelecido, coloca-se na LRC os k melhores elementos da lista de candidatos. A outra abordagem é baseada no valor associado a cada elemento candidato. Sejam c uma componente do conjunto de componentes da solução C e $g(c)$ o custo do benefício desta componente. Considere $\overline{g} = \max\{g(c) \mid c \in C\}, \underline{g} = \min\{g(c) \mid c \in C\}$, e um parâmetro $\alpha \in [0,1]$. Em um problema de minimização, uma LRC baseada em valores será determinada por LRC $= \{c \in C \mid g(c) \leq \underline{g} + \alpha(\overline{g} - \underline{g})\}$. No caso $\alpha = 0$, o algoritmo semi-guloso corresponde ao algoritmo guloso puro, enquanto que para $\alpha = 1$ são construídas soluções aleatórias [8]. As primeiras implementações do método usavam valores fixos para o α. Em [9] foi proposto o uso de um α gerado aleatoriamente no intervalo [0,1] em cada iteração do algoritmo.

Com base nesta solução inicial construída, avalia-se sua vizinhança através de uma busca local à procura da melhor solução local.

3.2 Fase de busca local

Na fase de busca local é realizada uma busca iterativa onde, em cada iteração, a solução corrente é substituida pela melhor solução da sua vizinhança. O procedimento

```
procedimento Construção_Solução(semente);
1  solucao ← Ø;
2  Avaliar o custo incremental das componentes candidatas;
3  faça
4      Construir a lista restrita de candidatos LRC;
5      Selecionar aleatoriamente uma componente c da LRC;
6      solucao ← solucao ∪ {c};
7      Atualizar os custos incrementais das componentes candidatas;
8  enquanto (houver componente candidata);
9  retorne solucao;
10 fim-GRASP.
```

Figura 3.2: Pseudo-código da fase Construtiva.

```
procedimento Busca_Local(sol);
1  faça
2      Busca sol' ∈ Vizinhança(sol) com f(sol') < f(sol);
3      sol ← sol';
4  enquanto (sol não é ótima local);
5  fim-BuscaLocal.
```

Figura 3.3: Pseudo-código da BuscaLocal.

pára quando não houver mais melhoria, isto é, quando a busca encontra um ótimo local.

De acordo com o algoritmo representado na Figura 3.3, o procedimento tem como parâmetro uma solução viável, construída na primeira fase ou pela rotina de reparo: $sol[k]$. A cada iteração da busca local, é selecionada uma solução vizinha $sol' \in \texttt{Vizinhança}(sol)$ e se esta é melhor que a solução corrente, ou seja, melhor até o momento (a linha 2 do algoritmo refere-se a minimização), a solução corrente é atualizada (linha 3) até que não se encontrem soluções melhores (linha 4).

3.3 Componente aleatória

Em uma heurística baseada na metaheurística GRASP o que se quer é encontrar uma solução cujo valor seja o mais próximo do valor da solução ótima, com o menor tempo de processamento possível.

O GRASP usa uma seqüência de números pseudo-aleatórios para a fase constru-

tiva. A semente da seqüência é um parâmetro da heurística. Para um número fixo de iterações, a qualidade da solução obtida pela heurística GRASP irá depender da seqüência de números pseudo-aleatórios usada. Fixando-se a qualidade da solução (o valor da função objetivo que se deseja alcançar), o tempo de execução do algoritmo também dependerá da seqüência de números pseudo-aleatórios utilizada. Esse fato leva à obtenção de conclusões imprecisas quando variantes da heurística desenvolvidas para um mesmo problema são comparadas.

Segundo Resende e Ribeiro [8], a metaheurística GRASP pode ser vista como uma técnica de amostragem repetitiva. Uma solução produzida em uma iteração do método é uma amostra de alguma distribuição desconhecida, onde a média e a variância são funções da natureza restritiva da LRC. Por exemplo, caso a LRC seja formada por apenas um elemento, apenas uma solução será produzida e a média será o valor da solução gulosa com variância zero. Caso a LRC seja formada por um número maior de elementos, então várias soluções serão produzidas e a variância será maior. Neste segundo caso, a média dos custos das soluções obtidas deverá ser de qualidade inferior à solução gulosa. Porém a melhor solução obtida deverá ser de qualidade superior à solução obtida pelo algoritmo guloso.

Quanto maior a variância dos valores das soluções obtidas durante a fase construtiva, maior será a variância das soluções obtidas após a busca local [8]. A probabilidade de um algoritmo GRASP encontrar uma solução ótima é maior quando a variância das soluções construídas é grande. Entretanto, como a diferença entre o valor médio das soluções construídas e o valor da melhor solução é grande, a busca local precisará de um tempo computacional maior em média para encontrar uma solução vizinha de melhor qualidade. Portanto, a escolha correta do parâmetro α é critica para se obter um bom equilíbrio entre tempo computacional e qualidade da solução.

Em um procedimento de GRASP Reativo [10] o parâmetro α é ajustado em tempo de execução de acordo com informações de soluções visitadas anteriormente. O procedimento GRASP Reativo atualiza em tempo de execução as probabilidades de α em cada iteração para favorecer valores de α que tenham produzido soluções de qualidade.

O GRASP tem implementação simples, pois pode fazer uso de técnicas para construção de soluções e de busca local já usados em outras abordagens.

4 Critérios de parada

Neste capítulo, na primeira seção, são apresentados alguns conceitos de estatística básica importantes para o entendimento deste trabalho. Na Seção 4.2 comentam-se estudos realizados sobre o problema de parada em metaheurísticas, e em seguida, na Seção 4.3, se faz um detalhamento do critério proposto por Boender e Rinnooy Kan. Por fim, apresenta-se na Seção 4.4 o *framework* que implementa o critério de parada proposto por Boender e Rinnooy Kan [1] aplicado a metaheurísticas GRASP.

4.1 Alguns conceitos de probabilidade

A seguir, são apresentados alguns conceitos importantes sobre probabilidade. Considere uma experiência cujo resultado é aleatório, ou seja, não pode ser conhecido a *priori* antes da realização da experiência. Seja S o conjunto de todos os resultados possíveis desta experiência. O conjunto S é chamado de *espaço amostral*. Um *evento* A é qualquer subconjunto do espaço amostral S. Ou seja, um evento é apenas um conjunto de resultados possíveis. Diz-se que um evento A ocorreu se o resultado da experiência é um ponto no espaço amostral que é elemento do conjunto A.

Seja S o espaço amostral e A um subconjunto qualquer deste espaço. Uma função de probabilidade que atua sobre o espaço amostral S satisfaz:

$$0 \leq P(A) \leq 1 \text{ para todo } A \subseteq S$$

$$P(S) = 1$$

$$P(A_1 \cup A_2 \cup A_3 \cup \ldots) = P(A_1) + P(A_2) + P(A_3) + \ldots$$

onde os A_i são mutuamente exclusivos. Uma probabilidade é uma função que mapeia elementos do espaço amostral em $[0,1]$ tal que: a probabilidade do espaço amostral inteiro é um, a probabilidade da união de eventos mutuamente exclusivos é a soma das probabilidades de cada um dos conjuntos na união e, finalmente, a probabilidade de qualquer subconjunto do espaço amostral é um número no intervalo $[0,1]$.

A abordagem bayesiana é fundamentada na probabilidade condicional, ou seja, dado que é conhecida a distribuição *a priori* do evento A, $P(A)$, pode-se derivar a probabilidade *posteriori* do evento B dado A, $P(B|A)$.

4.1.1 Teorema de Bayes

Sejam $B_1, B_2, \ldots, B_k$ uma partição de S e A um evento qualquer em S. Então:

$$P(B_i|A) = \frac{P(A|B_i)P(B_i)}{\sum_{j=1}^{k} P(A|B_j)P(B_j)}.$$

O teorema de Bayes é importante pois permite inverter probabilidades condicionais. Às vezes é fácil calcular $P(A|B)$, mas o que se deseja conhecer é $P(B|A)$. O teorema de Bayes permite calcular $P(B|A)$ em termos de $P(A|B)$.

Na abordagem bayesiana é possível expressar algum conhecimento prévio através da distribuição *a priori*. Dados resultados experimentais da aplicação de uma determinada heurística a um determinado problema, pode-se derivar a distribuição *posteriori*, que reflete o caminho no qual o conhecimento prévio (*a priori*) afeta os resultados do experimento.

4.1.2 Variável aleatória discreta

Considere uma experiência aleatória com espaço amostral S. *Variável aleatória* é a função que associa cada elemento s_i do espaço amostral S a um número real x_i. Uma função X definida no espaço amostral S, que tem valores em um conjunto enumerável de pontos da reta, é dita uma *variável aleatória discreta*. A probabilidade de ocorrência de cada valor x_i da variável aleatória X é dada pela probabilidade do evento A de S, cujos

elementos s_i correspondem ao valor x_i. Matematicamente esta definição é representada abaixo:

$$P(X = x_i) = P(A), \text{ onde } A = \{s_1, s_2, \ldots\} \subset S$$

$$\text{é tal que } X(s_i) = x_i, \text{ se } s_i \in A \text{ e } X(s_i) \neq x_i, \text{ se } s_i \in \{S \setminus A\}.$$

4.1.3 Distribuição de probabilidade

Chama-se função de probabilidade da variável aleatória discreta X, que assume os valores $x_1, x_2, \ldots, x_n$, a função $\{(x_i, p(x_i)), i = 1, 2, \ldots\}$, que a cada valor de x_i associa a sua probabilidade de ocorrência, isto é, $p(x_i) = P(X = x_i) = p_i, i = 1, 2, \ldots$.

Algumas variáveis aleatórias adaptam-se a uma série de problemas práticos. Portanto, um estudo pormenorizado dessas variáveis é de grande importância para a construção de modelos probabilísticos para situações reais e a conseqüente estimação de seus parâmetros.

4.1.4 Distribuição Binomial

Uma experiência tem apenas 2 resultados possíveis: "sucesso" e "falha", onde a probabilidade de "sucesso" é p e a probabilidade de "falha" é $q = 1 - p$. A experiência é repetida um número fixo n de vezes, sempre nas mesmas condições, de tal forma que as probabilidades de "sucesso" (p) e "falha" $(q = 1 - p)$ se mantêm inalteradas a cada repetição. As diversas repetições da experiência são feitas de maneira independente. A variável aleatória X que mede o número de "sucessos" nas n repetições da experiência é uma variável discreta. Esta variável tem distribuição binomial com parâmetros n e p e função de probabilidade:

$$P(X = k \mid n, p) = \binom{n}{k} p^k q^{n-k}, k = 0, 1, \ldots, n.$$

4.1.5 Distribuição Multinomial

A distribuição multinomial é uma generalização da distribuição binomial quando há mais de dois resultados possíveis em cada prova. As probabilidades dos vários resultados permanecem as mesmas para cada prova e as provas são todas independentes. Se há k re-

sultados possíveis para cada prova (em n provas) e suas probabilidades são $p_1, p_2, \ldots, p_k$, a função de probabilidade é dada por:

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_k = x_k \mid n, p_1, p_2, \ldots, p_k) = \frac{n!}{x_1!x_2!\ldots x_k!} p_1^{x_1} p_2^{x_2} p_k^{x_k}.$$

$$\text{considerando as restrições } \sum_{n=i}^{k} x_i = n \text{ e } \sum_{n=i}^{k} p_i = 1.$$

4.2 Critérios de parada abordados em outras pesquisas

Orsenigo e Vercellis [11] investigam como utilizar regras de parada baseadas na teoria de probabilidade bayesiana. Dado que a solução encontrada em cada execução de um algoritmo pode ser considerada uma realização de uma variável aleatória independente, definindo-se sua distribuição, pode-se calcular a probabilidade da solução encontrada estar próxima da solução ótima a um nível de confiança determinado. Seja A um algoritmo para gerar uma solução viável com valor da função objetivo z^A. M execuções de A resultam numa seqüência de realizações independentes $z_i^A, i = 1, \ldots, M$, da variável aleatória Z^A. O melhor valor observado $\overline{z} = \min_{1 \leq i \leq M} z_i^A$ (no caso de um problema de minimização) na seqüência de M execuções independentes é mantido como uma aproximação para o valor ótimo z^* e é uma realização da variável aleatória $\overline{Z}$. Sejam z^L e z^U limites inferior e superior conhecidos para o valor ótimo z^*. A regra de parada proposta pelos autores é derivada da hipótese de que o usuário está satisfeito se a aproximação da solução ótima tem um erro relativo inferior a um nível de tolerância especificado $0 \leq \psi \leq 1$; ou seja: $(\overline{z} - z^*)/(z^U - z^L) \leq \psi$. Então, dado que p_z é a função densidade de probabilidade e $F(z)$ é a função de distribuição acumulativa, a execução pára se a probabilidade de melhora do valor da melhor solução corrente $\overline{z}$ após M execuções é inferior a um nível de confiança definido ξ, ou seja, $F(\overline{z}) < \xi$.

A abordagem proposta por Hart [12] sugere regras de parada baseadas em métodos estatísticos para algoritmos de múltiplas inicializações com busca local simples ou busca local estratificada. A busca local simples seleciona pontos $(X_1, \ldots, X_n)$ usando uma distribui-

ção comum em um domínio D e estima o ótimo global $Y = min\{f(X_1), \ldots, f(X_n)\}$. A busca local estratificada particiona o domínio D em um conjunto finito de sub-domínios D_i, onde $D = \cup_{i=1}^{t} D_i$, $D_i \cap D_j = \emptyset$, se $i \neq j$, seleciona pontos de todos os sub-domínios de acordo com uma distribuição fixa e estima o ótimo global $Y = min_{i=1,\ldots,t}(f(argmin_i \{f(X_1^i), \ldots, f(X_n^i)\}))$. O método proposto usa estatística de valores extremos da função objetivo para construir o intervalo de confiança para a estimativa do ótimo global, e assim, determinar sua parada se a solução encontrada estiver neste intervalo.

Um método de descoberta da distribuição empírica dos tempos de processamento das heurísticas GRASP é proposto em Aiex [8] e Aiex, Resende e Ribeiro [13]. Segundo os autores, dado o conhecimento desta distribuição é possível a estimação do tempo de processamento ideal (momento de parada) para se chegar a uma solução de qualidade suficientemente boa para um dado problema (que alcance um valor alvo).

Bartkuté et al. abordam em diversos artigos [14, 15, 16, 17, 18] o uso do método de ordens estatísticas na estimação do intervalo de confiança para o valor mínimo da função objetivo (no caso de uma minimização) de um algoritmo estocástico. Em [14], Bartkuté, Felinskas e Sakalauskas determinam o critério de parada quando o intervalo de confiança do valor mínimo da função objetivo torna-se tão pequeno quanto um valor estipulado como suficiente.

A proposta de Boender e Rinnooy Kan [1] utiliza o número de diferentes ótimos locais observados e a freqüência que estes ocorrem em um determinado número de execuções como variáveis para estimar uma distribuição baseada na estatística bayesiana. Com base nesta distribuição será definido o critério de parada, considerando o custo (a perda) da execução ser parada antes de todos os ótimos locais terem sido encontrados ou da execução continuar sem a localização de um novo ótimo local. Os autores propuseram uma extensão deste trabalho em [19]. A adaptação proposta considera também o valor da função objetivo dos ótimos locais como informação de entrada na definição do critério de parada. Na Seção 4.3, apresenta-se, com mais detalhes, o trabalho de Boender e Rinnooy Kan [1].

Foi demonstrado em [1, 11, 12, 13, 14, 15, 16, 17, 18, 19] que o uso da teoria estatística

pode determinar um critério de parada mais inteligente do que o critério que simplesmente limita um número fixo de iterações para um algoritmo. Existem abordagens que utilizam conceitos teóricos da estatística clássica, outras que se fundamentam na análise empírica dos dados observados, e as que se baseiam na teoria da probabilidade bayesiana (que de certa forma agrega a teoria e a análise dos dados).

O presente trabalho propõe estudar critérios de parada dando ênfase na abordagem bayesiana. A próxima seção segue o trabalho de Boender e Rinnooy Kan [1].

4.3 Critérios de parada proposto por Boender e Rinnooy Kan

O método GRASP [4, 5, 6, 7] utiliza múltiplas inicializações e uma busca local aplicada em cada ponto inicial. Cada iteração GRASP produz uma amostra de soluções em uma distribuição desconhecida. Na literatura estatística é comum admitir a distribuição *a priori* como uniforme em casos em que esta é desconhecida.

A distribição binomial é caracterizada pelo número de vezes que um distinto ótimo local é encontrado nas iterações executadas, ou seja, o número de vezes que na amostragem ocorre o sucesso deste distinto ótimo local.

Então, pode-se obter a informação do número de diferentes ótimos locais W encontrados em n buscas locais e determinar a distribuição da variável W. Um diferente ótimo local tem distribuição binomial. Esta distribição é caracterizada pelo número de vezes que um distinto ótimo local é encontrado nas iterações executadas, ou seja, o número de vezes que na amostragem ocorre o sucesso deste distinto ótimo local.

Desta forma, um conjunto de diferentes ótimos locais segue a distribuição multinomial. Logo, tendo-se a distribuição multinomial *a priori* do conjunto de ótimos locais, condicionada a um conjunto de ótimos locais observados, pode-se estimar o número de ótimos locais ideal, considerando-se uma *função de perda* que agrega a perda de tempo de execução e a perda de qualidade da solução.

Considera-se que os métodos de múltiplas inicializações fazem uma busca local L inicializada em cada ponto em uma amostra de distribuição uniforme sobre o espaço de soluções viáveis S. Seja R_{x^*} a região de atração de um ótimo local x^* dada a busca local

L, ou seja, um conjunto de pontos no espaço de soluções viáveis S a partir dos quais, iniciando-se a busca local L, o ótimo local x^* é encontrado. Seja k o número de ótimos locais no espaço de soluções viáveis e θ_i o volume relativo da i-ésima $(i = 1,\ldots,k)$ região de atração (geralmente desconhecidos). O volume relativo θ_i da região de atração i é a probabilidade de cada i-ésimo ótimo local ser encontrado em cada execução. Dado um número de buscas locais n, cada ótimo local observado pode ser considerado como uma amostra de distribuição multinomial de k elementos com probabilidades iguais aos volumes relativos $\theta_i,\ldots,\theta_k$. Definindo-se as variáveis aleatórias N_i, com realizações $n_i (i = 1,\ldots,k)$, como o número de vezes que o i-ésimo ótimo local é encontrado em n buscas locais, a probabilidade do evento $\{(N_1,\ldots,N_k) = (n_1,\ldots,n_k)\}$ é dada pela distribuição multinomial:

$$P(n_1,\ldots,n_k) = \frac{n!}{\prod_{i=1}^{k} n_i!} \times \prod_{i=1}^{k} \theta_i^{n_i}, \tag{4.1}$$

onde as seguintes restrições devem ser satisfeitas: $\sum_{i=1}^{k} n_i = n$ e $\sum_{i=1}^{k} \theta_i = 1$.

Aplicando-se a função de verossimilhança, poderia-se obter informação sobre $k, \theta_1,\ldots,\theta_k$ e substituindo valores da amostra observada $n_1,\ldots,n_k$ em (4.1). Porém, não é possível decidir para quais ótimos locais um θ_i corresponde e distingüir a ordem em que os eventos $n_1,\ldots,n_k$ ocorrem. Por exemplo, se em 8 buscas locais um ótimo local é encontrado 5 vezes, outro duas vezes e outro uma vez, é imposível distingüir entre os eventos $\{(N_1,N_2,N_3) = (5,2,1)\}$, $\{(N_1,N_2,N_3) = (1,2,5)\}$, $\{(N_1,N_2,N_3,N_4,N_5,N_6) = (0,5,0,1,2,0)\}$, etc, para calcular a equação (4.1).

A solução proposta por Boender e Rinnooy Kan [1] é definir mutuamente agregações de eventos individuais $n_1,\ldots,n_k$ desconsiderando a ordem e omitindo os n_i's que são iguais a 0. Para o exemplo acima o evento agregado correspondente é $\{N_1,N_2,N_3\} = \{1,2,5\}$.

Definida a variável aleatória W, com realizações w, como o número de diferentes ótimos locais observados em n buscas locais, a probabilidade dos agregados $\{N_1,N_2,\ldots,N_w\}$ é dada pela distribuição multinomial generalizada:

$$P(\{n_1, n_2, \ldots, n_w\}) = \left(\frac{1}{\prod_{j=1}^{n} h_j!} \right) \times \frac{n!}{\prod_{i=1}^{w} n_i!} \times \sum_{(g_1,\ldots,g_w) \in S_k[w]} \prod_{i=1}^{w} \theta_{g_i}^{n_i} \qquad (4.2)$$

onde h_j é o número de $n_i's$ que são iguais a j e $S_k[w]$ é o conjunto de todas as permutações de w diferentes elementos de $\{1,\ldots,k\}$ ($\sum_{i=1}^{w} n_i = n, n_i > 0$ para $i = 1,\ldots,w$).

A expressão (4.2) não é suficiente, porém, para obter a estimativa de máxima verossimilhança do número de ótimos locais k, pois esta, conforme Boender e Rinnooy Kan [1] seria igual a ∞ para todos os possíveis resultados $\{n_1,\ldots,n_w\}$.

Na abordagem proposta pelos autores, os valores desconhecidos de $k, \theta_1, \theta_2, \ldots, \theta_k$ serão assumidos como as variáveis aleatórias $K, \Theta_1, \Theta_2, \ldots, \Theta_k$. Boender e Rinnooy Kan [1] assumem que os tamanhos relativos das regiões de atração, as variáveis $\Theta_i, \ldots, \Theta_k$, seguem uma distribuição uniforme em um simplex unitário de dimensão $(k-1)$ $I_{k-1} = \{(\theta_1,\ldots,\theta_k) \mid \theta_i \geq 0\ (i = 1,\ldots,k), \sum_{i=1}^{k} \theta_i = 1\}$. Então, dado o resultado $n_1,\ldots,n_w$ de um número n de buscas locais, utiliza-se o *Teorema de Bayes* para incorporar a idéia da distribuição a *priori* e calcular a distribuição *posteriori* das variáveis $K, \Theta_i, \ldots, \Theta_k$:

Teorema 1 *[1] Dada a função de densidade de probabilidade a priori uniforme* $p(k, \theta_1, \ldots, \theta_k) \propto (k-1)$ *(onde* $\propto$ *denota proporcionalidade) e o resultado* $\{n_1,\ldots,n_w\}$*, a função de densidade posteriori do número de ótimos locais K e do volume relativo das regiões de atração* $\Theta_1,\ldots,\Theta_k$ *para* $n \geq w+2$ *é dada por:*

$$\begin{aligned} &P(k, \theta_1, \theta_2, \ldots, \theta_k | \{n_1, n_2, \ldots, n_w\}) \\ &= \frac{(n-1)!(n-2)!(k-1)!}{w!(w-1)(n-w-2)! \prod_{i=1}^{w} n_i!} \times \sum_{(g_1,\ldots,g_w) \in S_k[w]} \prod_{i=1}^{w} \theta_{g_i}^{n_i} \end{aligned} \qquad (4.3)$$

A função de densidade *posteriori* é encontrada substituindo a multinomial generalizada (4.2) e a *priori* uniforme assumida $p(k, \theta_1, \ldots, \theta_k) \propto (k-1)$ no Teorema de Bayes:

$$
\begin{aligned}
&P(k,\theta_1,\theta_2,\ldots,\theta_k|\{n_1,n_2,\ldots,n_w\}) \\
&= \frac{p(\{n_1,\ldots,n_w\})p(k,\theta_1,\ldots,\theta_k}{\sum_{m=w}^{\infty}\int_{I_{m-1}} p(\{n_1,\ldots,n_w\})p(\{m,\psi_1,\ldots,\psi_m\})\prod_{i=1}^{m} d\psi_i} \\
&= \frac{(k-1)!\sum_{(g_1,\ldots,g_w)\in S_k[w]}\prod_{i=1}^{w}\theta_{g_i}^{n_i}}{\sum_{m=w}^{\infty}\int_{I_{m-1}}(m-1)!\sum_{(g_1,\ldots,g_w)\in S_m[w]}\prod_{i=1}^{w}\psi_{g_i}^{n_i}\prod_{i=1}^{m} d\psi_i}
\end{aligned}
\tag{4.4}
$$

Observa-se que

$$
\frac{(n+m-1)!}{\prod_{i=1}^{w} n_i!}\prod_{i=1}^{w}\psi_{g_i}^{n_i} \tag{4.5}
$$

é uma função de densidade *Dirichlet* m-dimensional com parâmetros $n_1+1,\ldots,n_w+1,1,\ldots,1$ tal que simplificando (4.4):

$$
\begin{aligned}
&P(k,\theta_1,\theta_2,\ldots,\theta_k|\{n_1,n_2,\ldots,n_w\}) \\
&= \frac{(k-1)!}{\prod_{i=1}^{w} n_i!\sum_{m=w}^{\infty}\frac{(m-1)!m!}{(m-w)!(n+m-1)!}\sum_{(g_1,\ldots,g_w)\in S_k[w]}\prod_{i=1}^{w}\theta_{g_i}^{n_i}}
\end{aligned}
\tag{4.6}
$$

A prova é finalizada incorporando a igualdade:

$$
\sum_{m=w}^{\infty}\frac{(m-1)!m!}{(m-w)!(n+m-1)!} = \frac{w!(w-1)!(n-w-2)!}{(n-1)!(n-2)!}(n\geq w+2) \tag{4.7}
$$

Os corolários apresentados a seguir foram estimados por Boender e Rinnooy Kan [1] em decorrência do Teorema 1.

Corolário 1 *[1, Corolário 1.2] O valor esperado, a moda e a variância do número de ótimos locais K*

$$
E(K|\{n_1,\ldots,n_w\}) = \frac{w(n-1)}{n-w-2}, n\geq w+3 \tag{4.8}
$$

$$
M(K|\{n_1,\ldots,n_w\}) = \frac{w(n-1)}{n-w}, n\geq w+1 \tag{4.9}
$$

$$\sigma^2(K|\{n_1,\ldots,n_w\}) = \frac{w(w+1)(n-1)(n-2)}{(n-w-2)^2(n-w-3)}, n \geq w+4 \tag{4.10}$$

Corolário 2 *[1, Corolário 1.3] O valor esperado posteriori do volume relativo* Θ *de uma região de atração de um ótimo local que foi encontrado* n_j *vezes:*

$$E(\Theta_{n_j}|\{n_1,\ldots,n_w\}) = \frac{(n_j+1)(n+w)}{n(n-1)}, n \geq w+2 \tag{4.11}$$

Corolário 3 *[1, Corolário 1.4] O valor esperado posteriori do volume total das regiões de atração observadas* Ω*:*

$$E(\Omega|\{n_1,\ldots,n_w\}) = \frac{(n-w-1)(n+w)}{n(n-1)}, n \geq w+2 \tag{4.12}$$

$$\sigma^2(\Omega|\{n_1,\ldots,n_w\}) = \frac{2(n+w)(n-w-1)w(w+1)}{(n-1)^2n^2(n+1)}, n \geq w+2 \tag{4.13}$$

Conforme Corolário 1, a estimativa bayesiana do número de ótimos locais é $\frac{w(n-1)}{n-w-2}$. Como a estimativa do número de ótimos locais deve ser um número inteiro, e para a maioria dos pares (n,w) esta expressão resulta em um número real, o seu valor deve ser truncado subtraindo-se $\frac{1}{2}$. Portanto, o critério de parada é definido por:

$$w \times \left(\frac{n-1}{n-w-2}\right) - \frac{1}{2} \leq w. \tag{4.14}$$

A solução poderia estar em (4.14), ou seja, parar a execução da heurística quando o número de diferentes ótimos locais observados w superar o valor da estimativa bayesiana inteira do número de ótimos locais. Entretanto, se o critério de parada é satisfeito, a estimativa do número de ótimos locais não observados é igual a zero, o que implica na execução extremamente longa do algoritmo se o valor da função objetivo tiver muitos ótimos locais com pequenas regiões de atração. Nesta situação, não é uma boa idéia executar a heurística em busca de todos os ótimos locais. Os autores sugerem levar em conta o custo da amostragem na definição do critério de parada, considerando a perda de terminação (quando a execução é parada antes de todos os ótimos locais terem sido

encontrados) e a perda de execução (o custo de uma nova execução de busca local) em uma regra que minimize a perda *posteriori* esperada.

Boender e Rinnooy Kan [1] introduzem quatro funções de perda, denominadas L_1, L_2, L_3 e L_4. A função de perda L_2 não será abordada devido sua inviabilidade, que será explicada mais a diante. A seguir apresentam-se as funções L_1, L_3 e L_4:

– A perda de terminação L_1 é igual a uma constante fixa se a amostragem é parada antes de todos os ótimos locais serem descobertos, ou seja, se o número de diferentes ótimos locais observados W for menor que o número de ótimos locais existentes K:

$$L_1 = \begin{cases} n + c_1, & \text{se } K > W \text{ onde } c_1 \text{ é uma constante fixa;} \\ n & \text{se } K = W; \end{cases}$$

Nesta função, a única forma de diminuir a perda de terminação é se todos os mínimos locais forem descobertos.

– A perda da terminação L_3 é proporcional à fração de ótimos locais não observados, ou seja, proporcional a diferença entre o número de ótimos locais existentes K e o número de diferentes ótimos locais observados W:

$$L_3 = c_3 \left(\frac{K - W}{K} \right) + n \text{ onde } c_3 \text{ é uma constante fixa.}$$

Neste caso, a função de perda tenta envolver diretamente o mínimo global na análise: se o verdadeiro número de ótimos locais é igual a k e w diferentes mínimos locais forem descobertos, a chance de que o mínimo global esteja entre os observados é igual a $(k-w)/k$.

– A perda da terminação L_4 é proporcional ao volume total relativo das regiões de atração não observadas:

$$L_4 = c_4(1 - \Omega) + n \text{ onde } c_4 \text{ é uma constante fixa.}$$

A função de perda L_4, baseada no volume total relativo de regiões de atração não observadas, reflete o caso em que se está pouco interessado em encontrar mínimos locais com regiões de atração extremamente pequenas.

Não será abordado neste trabalho a função L_2 em que a perda de terminação é proporcional ao número de ótimos locais não observados (a perda é reduzida por uma constante fixa cada vez que um novo ótimo local é observado). Segundo [1], para as regras de parada baseadas nesta função de perda, não é possível determinar que a partir de uma determinada iteração a execução do programa vai parar. Ou seja, não é possível calcular um limite máximo para o número de iterações que garanta que a partir daí a perda só vai decrescer. Este assunto será melhor abordado adiante, quando for apresentado a definição de limite máximo n_j^*.

Uma vez que, para um dado resultado $\{n_1, \ldots, n_w\}$, as perdas *posteriori* dependem exclusivamente do número n de buscas locais e do número w de mínimos locais observados, daqui em diante, denota-se por (n,w) o resultado $\{n_1, \ldots, n_w\}$.

Considerando as funções de perda $L_j, j \in \{1,3,4\}$, dada uma amostra de tamanho n, a perda *posteriori* depois de $n' > n$ observações é uma variável aleatória $E(L_j|(n',W))$. O objetivo é encontrar, para cada função de perda, o critério de parada que minimize o valor esperado da seqüência de perdas *posteriori*

$$\{E(L_j|(n',W))\}_{n'=n+1}^{\infty}.$$

Dado um par corrente (n,w) e considerando uma busca local adicional, apenas dois resultados podem acontecer: um ótimo local diferente é observado ou não. A probabilidade *posteriori* de que a próxima busca não resultará na descoberta de um ótimo local não observado é igual ao volume esperado *posteriori* das regiões de atração observadas [1]. Então dado o par corrente (n,w), pode-se calcular o valor esperado condicional da perda *posteriori* de $n+1$ observações de acordo com a relação de recorrência

$$E(E(L_j|(n+1,W))|(n,w)),$$

isto é, deseja-se estimar a próxima perda no passo corrente e avaliar se deve-se continuar a busca ou não.

Desta forma, uma possível regra de parada *de um passo* é terminar a busca se a perda *posteriori* esperada de $n+1$ observações for maior que a perda *posteriori* atual, isto é, se:

$$E(E(L_j|(n+1,W))|(n,w)) > E(L_j|(n,w)). \tag{4.15}$$

A seguir, mostra-se o desmembramento da desigualdade (4.15):

$$\begin{aligned} &E(E(L_j|(n+1,W))|(n,w)) && > E(L_j|(n,w)) \\ &\frac{(n-w-1)(n+w)}{n(n-1)}E(L_j|(n+1,w)) + \frac{w(w+1)}{n(n-1)}E(L_j|(n+1,w+1)) && > E(L_j|(n,w)) \end{aligned} \tag{4.16}$$

Observe que esta decisão é baseada na premissa que a busca é imediatamente parada uma vez que a ocorrência da próxima observação teria sido executada. Esta regra checa sempre um passo a frente sem ter certeza se haverá um limite máximo a partir do qual a perda do passo seguinte sempre crescerá. Em contraste, (segundo De Groot [20]), *critérios de parada ótimos* são obtidos comparando-se a perda *posteriori* corrente com a perda *posteriori* esperada de outra observação assumindo-se que *a melhor estratégia é adotada daí em diante*. Para esta abordagem ser possível, deve-se encontrar um valor n_j^* tal que para todos os pares (n,w) com $n \geq n_j^*$ a seqüencia de perdas *posteriori* é um **submartingale**, i.e. a perda *posteriori* esperada para todos pares (n,w) com $n \geq n_j^*$ nunca diminui se uma nova observação é realizada. Sabe-se então que para todos os pares (n,w) com $n = n_j^*$ a decisão ótima é parar, e é possível calcular a perda *posteriori* para $E(L_j|(n,w))$.

A aplicação da desigualdade (4.16) nas funções de perda L_1, L_3 e L_4 é mostrada a seguir [1]:

[1] A demonstração desta desigualdade foi feita aplicando-se respectivamente as equações (21), (25) e (27) na desigualdade (28) de Boender e Rinnooy Kan [1] para calcular $E(E(L_j|(n+1,W))|(n,w)), j \in \{1,3,4\}$ e verificando-se se este é maior ou igual às equações (21), (25) e (27), respectivamente, que correspondem

$$
\begin{aligned}
&E(E(L_1|(n+1,W))|(n,w)) \geq E(L_1|(n,w)), n > n_1^* = c_1 + 1 - \sqrt{4c_1+1} \\
&= \left[\frac{(n-w-1)(n+w)}{n(n-1)}\right] \times \left[c_1\left(1 - \prod_{i=1}^{w} \frac{n-i}{n+i}\right) + n + 1\right] \\
&+ \left[\frac{w(w+1)}{n(n-1)}\right] \times \left[c_1\left(1 - \prod_{i=1}^{w+1} \frac{n-i}{n+i}\right) + n + 1\right] \\
&\geq c_1\left(1 - \prod_{i=1}^{w} \frac{n-1-i}{n-1+i}\right) + n
\end{aligned} \tag{4.17}
$$

$$
\begin{aligned}
&E(E(L_3|(n+1,W))|(n,w)) \geq E(L_3|(n,w)), n > n_3^* = \frac{c_3}{4} \\
&= \left[\frac{(n-w-1)(n+w)}{n(n-1)}\right] \times \left[c_3\frac{w}{n} + n + 1\right] \\
&+ \left[\frac{w(w+1)}{n(n-1)}\right] \left[c_3\frac{w+1}{n} + n + 1\right] \\
&\geq c_3\frac{w}{n-1} + n
\end{aligned} \tag{4.18}
$$

$$
\begin{aligned}
&E(E(L_4|(n+1,W))|(n,w)) \geq E(L_4|(n,w)), n > n_4^* = \frac{c_4}{3} \\
&= \left[\frac{(n-w-1)(n+w)}{n(n-1)}\right] \times \left[c_4\frac{w(w+1)}{(n^2+n)} + n + 1\right] \\
&+ \left[\frac{w(w+1)}{n(n-1)}\right] \left[c_4\frac{(w+1)^2}{(n^2+n)} + n + 1\right] \\
&\geq c_4\frac{w(w+1)}{n(n-1)} + n
\end{aligned} \tag{4.19}
$$

A partir destes resultados, aplicam-se as desigualdades (4.17), (4.18) e (4.19), recursivamente, na *ordem inversa*, isto é, de $n = n_j^*$ até $n = 1$. Uma vez que para todos os pares (n,w) com $n = n_j^*$ a decisão ótima é parar, tem-se que para todos os pares (n,w) com $n = n_j^* - 1$ a regra de um passo explicada anteriormente é a decisão ótima. Comparando-se a perda *posteriori* esperada de $n+1$ observações e a perda *posteriori* cor-

a $E(L_j|(n,w)), j \in \{1,3,4\}$.

rente (4.17), (4.18) e (4.19), pode-se determinar para cada par (n,w) com $n = n_j^* - 1$ se a decisão ótima é continuar ou não. Então prossegue-se para o passo $n_j^* - 2$. Mais uma vez utiliza-se (4.17), (4.18) e (4.19) para calcular a perda *posteriori* esperada da continuação. Agora, porém, se a decisão ótima em $(n+1,w)$ ou $(n+1,w+1)$ é continuar mais uma vez, substitui-se a perda *posteriori* esperada calculada anteriormente em (4.17), (4.18) e (4.19). Trabalhando-se para trás, desta forma, até o primeiro passo, pode-se calcular para cada par (n,w) se a decisão ótima é parar ou não. Então, nota-se que, dado c_j, é possível construir uma tabela sumarizando a estratégia de parada ótima para todas as possíveis funções objetivo. A Figura 4.1 ilustra graficamente a aplicação da estratégia de parada ótima às funções de perda L_1, L_3 e L_4 considerando $c_j = 1000$. Em resumo, enquanto um algoritmo, durante sua execução, tiver pares (n,w) dentro da região convexa (região sombreada) do gráfico (Figura 4.1), o algoritmo deve continuar. Quando o algoritmo sair desta área, ele deve parar.

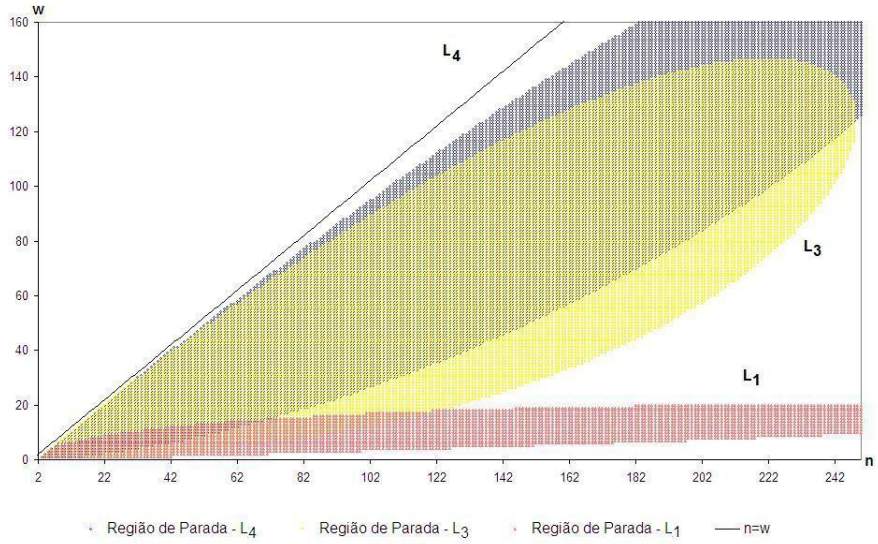

Figura 4.1: Estratégia de Parada Comparativa

Para ilustrar o cálculo do critério de parada, considere o exemplo a seguir sobre a função de perda L_3:

Calcula-se a estratégia de parada ótima para todos os pares (n,w) entre $n = 1$ e $n = n_3^* = \frac{c_3}{4}$. Boender e Rinnooy Kan [1] considera $c_3 = 1000$, então $n_3^* = 250$. Voltando um passo para trás, checa-se $E(E(L_3|(n,W))|(n-1,w)) > E(L_3|(n-1,w))$ para n decrescendo a partir de 250. Supondo $\{(n-1 = 249, w = 3) \rightarrow (n = 250, w = 3)\}$,

a comparação é representada por $E(E(L_3|(250,W))|(249,3)) > E(L_3|(249,3))$. Substituindo em (4.18) conforme (4.20), dado que $E(E(L_3|(250,W))|(249,3)) = 262,0$ e $E(L_3|(249,3)) = 261,1$, o critério para o par $(n = 249, w = 3)$ é parar.

$$
\begin{aligned}
&E(E(L_3|(249+1,W))|(249,3)) \geq E(L_3|(249,3)), n > n_3^* = \frac{1000}{4} \\
&= \left[\frac{(249-3-1)(249+3)}{249(249-1)}\right] \times \left[1000\frac{3}{249} + 249 + 1\right] \\
&+ \left[\frac{3(3+1)}{249(249-1)}\right] \left[1000\frac{3+1}{249} + 249 + 1\right] \\
&\geq 1000\frac{3}{249-1} + 249 \\
&= 262,0 \geq 261,1
\end{aligned} \tag{4.20}
$$

4.4 O *framework* GRASP_BSR

Nesta seção será apresentado o *framework* GRASP_BSR, do inglês *GRASP with Bayesian Stopping Rule*, que implementa o critério de parada proposto por Boender e Rinnooy Kan [1] aplicado a metaheurísticas GRASP. O *framework* GRASP_BSR pode ser visto na Figura 4.2.

Os parâmetros de entrada são: função de perda $L_j, j \in \{1,3,4\}$, controle c e um conjunto P de parâmetros GRASP. O parâmetro c é usado para calcular a perda *posteriori* correspondente às desigualdades (4.17), (4.18) e (4.19) para $E(L_j|(n,w))$ e calcular o limite superior de iterações n_j^*.

A seguir, relata-se o significado das variáveis usadas. A variável n representa o número corrente de iterações GRASP; a variável w indica o número de diferentes ótimos locais encontrados; o conjunto O armazena os w diferentes ótimos locais encontrados durante a busca; a variável n_j^* é o limite superior de iterações definido pelo critério de parada j; a variável *melhor_sol* armazena a melhor solução encontrada até o momento; a variável *parada* indica se a busca deve parar ou continuar; a variável s armazena a solução corrente e as variáveis x_1 e x_2 são usadas para armazenar os valores das perdas *posteriori*.

As variáveis são inicializadas nas linhas 1-6. Na linha 4 é chamada a função que cal-

cula o limite superior de iterações n_j^* conforme os critérios de parada definidos anteriormente em (4.17), (4.18) e (4.19) (Figura 4.3). A variável que guarda a melhor solução até o momento é atualizada na linha 5. O número de iterações corrente n é atualizado na linha 8. O procedimento `Uma_Iteração_GRASP` chama uma iteração GRASP completa (fases construtiva e de busca local) de uma específica implementação GRASP para o problema tratado com o conjunto P de parâmetros, e retorna a solução ótima local s. A melhor solução encontrada até o momento é atualizada na linha 10. Na linha 11, verifica-se se a solução ótima local s não existe em O, o conjunto de ótimos locais. Caso verdadeiro, o número w de diferentes ótimos locais w é atualizado na linha 12 e a solução s é incluída no conjunto O na linha 13. Na linha 15 é testado se a iteração corrente n é maior ou igual à condição necessária para o teste do critério de parada $(w+2)$. Se verdadeiro, a perda *posteriori* esperada de $n+1$ iterações (observações) x_1 é calculada na linha 16 e a perda *posteriori* corrente x_2 é calculada na linha 17. A busca pára se $x_1 \geq x_2$ (para iteração corrente n maior ou igual à condição necessária para o teste do critério de parada $(w+2)$) ou se o número de iterações n é maior ou igual ao limite superior de iterações n_j^* (linhas 19-21). A melhor solução encontrada na busca é retornada na linha 23.

Para finalizar, deve-se considerar o custo adicional de se contabilizar os diferentes ótimos locais w (linhas 11 a 14 do pseudo código). Considerando-se uma solução representada como um conjunto de m elementos (o tamanho do problema), verificar a igualdade de duas soluções custa $O(m)$. Como, a cada iteração, esta verificação é realizada para w ótimos locais e, no pior caso, $w = n_j^*$, tem-se que a complexidade de uma iteração da busca é adicionada em $O(n_j^* m)$, sendo o valor de n_j^* calculado em função da constante c (Figura 4.3). É importante salientar que o *framework* proposto é genérico e duas dadas soluções são consideradas dois ótimos locais distintos quando suas respectivas configurações são distintas. Casos especiais como, por exemplo, o de soluções simétricas que acontece, por exemplo, em coloração de vértices, necessita um tratamento diferenciado.

```
Framework GRASP_BSR(L_j, j ∈ {1,3,4}; c, P);
1   w ← 0;
2   n ← 0;
3   O ← ∅
4   n*_j ← Calcula_limite_sup_iteracao(c, j);
5   melhor_sol ← Inicializa_Melhor_Solucao;
6   parada ← FALSO;
7   enquanto (NÃO parada) faca
8       n ← n + 1;
9       s ← Uma_Iteração_GRASP(P);
10      Atualiza_Melhor_Solucao(s, melhor_sol);
11      se (s ∉ O) então
12          w ← w + 1;
13          O ← O ∪ {s};
14      fim-se
15      se (n ≥ w + 2) então
16          x1 ← E(E(L_j|(n+1, w))|(n, w));
17          x2 ← E(L_j|(n, w));
18      fim-se
19      se ((x1 ≥ x2 e n ≥ w + 2) ou n ≥ n*_j) então
20          parada ← VERDADEIRO;
21      fim-se
22  enquanto
23  retorna melhor_sol
24  fim-GRASP_BSR.
```

Figura 4.2: O *framework* do procedimento GRASP_BSR usado para implementar os critérios de parada proposta por Boender e Rinnooy Kan [1], aplicado a metaheurísticas GRASP.

```
Função Calcula_limite_sup_iteracao(c, j);
1   caso j igual a
2       1: n* = c1 + 1 − √(4c + 1)
3       3: n* = c/4
4       4: n* = c/3
5   fim-caso
6   retorna n*
7   fim-Calcula_limite_sup_iteracao.
```

Figura 4.3: Pseudo código da função que calcula o limite superior de iterações n_j^*.

5 Heurísticas baseadas na metaheurística GRASP

Neste capítulo são abordadas cinco heurísticas baseadas na metaheurística GRASP [4, 5, 6, 7] para solucionar problemas de otimização combinatória, nas quais serão aplicados os critérios de parada estudados nesta dissertação.

5.1 GRASP para o problema de conjunto independente máximo

Segundo Feo, Resende e Smith [21] aplicações práticas do problema de conjunto independente máximo são abundantes como, por exemplo, recuperação de informações, análises de transmissão de sinal, teoria da classificação, algumas aplicações de economia, programação de escala, desenho experimental e visão computacional, além de ser uma sub-rotina em heurísticas de coloração de grafos.

O problema e o espaço de soluções viáveis são definidos desta forma: seja $G = (V, E)$ um grafo não-direcionado com conjunto de vértices V e conjunto de arestas E. Vértices $u, v \in V$ são não-adjacentes se $(u, v) \notin E$. Um conjunto de vértices $S \subseteq V$ é independente se todos os vértices em S não tem pares adjacentes. No problema de conjunto independente máximo o que se quer é encontrar um conjunto independente com cardinalidade máxima.

Encontrar um conjunto independente máximo é equivalente a encontrar um clique máximo ou uma cobertura de vértice mínimo de um grafo.

Aiex, Resende e Ribeiro [13] resume bem a proposta do algoritmo abordado em Feo, Resende e Smith [21], baseado na metaheurística GRASP, para problemas de conjunto independente máximo, conforme mostrado a seguir:

5.1.1 Fase de construção

O algoritmo inicializa um grafo $G' = (V', E')$ igual o grafo original e o conjunto independente S como o conjunto vazio. O conjunto independente é construído colocando-se em S um vértice a cada iteração. A função gulosa que guia a fase de construção é o grau do vértice em relação ao grafo em construção. A heurística seleciona entre os vértices do grafo o vértice independente com grau mínimo. A função gulosa é adaptada após a seleção de cada vértice para o conjunto independente.

A seleção do vértice é aleatória, restrita aos vértices na LRC. Sejam $\underline{d}$ e $\overline{d}$ respectivamente o grau mínimo e máximo de todos os vértices de trabalho (ainda não levados para o conjunto S). A lista restrita de candidatos LRC $= \{v \in V' \mid d(v, G') \leq \underline{d} + \alpha(\overline{d} - \underline{d})\}$ onde o parâmetro α controla o tamanho da LRC e é tal que $0 \leq \alpha \leq 1$.

5.1.2 Fase de busca local

A busca local usa o movimento *troca(p,q)*, definido na Seção 2.2. Um componente é um vértice do conjunto independente S. Neste movimento remove-se p vértices x do conjunto independente S e o substitui por q vértices não-adjacentes tal que estes são não-adjacentes para qualquer vértice em $S \setminus \{x\}$. Uma solução ótima local é detectada quando mais nenhuma troca é possível.

5.2 GRASP para o problema quadrático de atribuição

Resende, Pardalos e Li [22] propõem uma heurística para o caso especial do problema quadrático de atribuição "denso". Estes problemas tem aplicações em instalações elétricas/eletrônicas, onde cada componente de um computador deve estar localizado de tal forma que minimize-se a quantidade de fios para conectá-los; na arquitetura de um hospital, para localizar seus serviços de forma a minimizar o tempo ou a distância para comunicação entre as salas; no design de um teclado; entre outros [2].

Dado um conjunto $N = \{1, 2, \ldots, n\}$ e matrizes $F = (f_{ij})$ e $D = (d_{kl})$ de dimensões $n \times n$, o problema quadrático de atribuição, em ingês *Quadratic Assignment Problem*

(QAP), pode ser definido como:

$$min_{p\in\prod_N} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{p(i)p(j)}, \tag{5.1}$$

onde $\prod_N$ é o conjunto de todas as permutações dos elementos de N. Uma das maiores aplicações de QAP está na teoria da locação onde a matriz $F = (f_{ij})$ é a matriz de fluxo de materiais do serviço i para o serviço j, e $D = (d_{kl})$ é a matriz de distância da localidade k para a localidade l. A contribuição para o custo total de simultaneamente instalar o serviço i na localidade k e o serviço j na localidade l é $f_{ij}.d_{kl}$. O objetivo é encontrar uma instalação de todos os serviços em todas as localidades, tal que o custo total da atribuição seja minimizada. Esta formulação tem uma limitação que as matrizes F e D sejam simétricas.

5.2.1 Fase de construção

Esta fase é composta por duas etapas. A idéia é atribuir serviços com alta interação (valores de f_{ij} altos) a locações próximas (valores de d_{kl} baixos). Então é necessário ordenar as distâncias entre locações em ordem crescente e os serviços entre as mesmas em ordem decrescente. Seja $d_{k1,l1} \leq d_{k2,l2} \leq \ldots \leq d_{kp,lp}$ e $f_{i1,j1} \geq f_{i2,j2} \geq \ldots \geq f_{ip,jp}$, onde $p = n^2 - n$. Os produtos $d_{k1,l1}.f_{i1,j1}, d_{k2,l2}.f_{i2,j2}, \ldots, d_{kp,lp}.f_{ip,jp}$ são dispostos em ordem crescente. Entre os menores produtos $d_{k,l}.f_{i,j}$, um é selecionado aleatoriamente na primeira etapa da fase de construção. Como ordenar todas as $p = n^2 - n$ distâncias e fluxos é ineficiente, somente é ordenado as $n_\beta = \beta p$, onde β é um parâmetro tal que $0 < \beta < 1$. Entre estes $n\beta = \beta p$ pares de atribuições, um par é selecionado aleatoriamente do conjunto de αn_β atribuições com os menores produtos $d_{kl}.f_{ij}$, onde α é um parâmetro tal que $0 < \alpha < 1$.

Na segunda etapa da fase de construção, a idéia é favorecer, entre os $n-2$ serviços remanescentes a serem instalados, aquelas atribui-ções que agregadas ao conjunto de atribuições já feitas, incorporem baixo custo a solução parcial corrente. Seja Γ o conjunto de q atribuições em um dado ponto da fase de construção, isto é, $\Gamma = \{(i_1,k_1),(i_2,k_2),\ldots,(i_q,k_q)\}$. O custo da atribuição do serviço j à localidade l, considerando as atribuições já

feitas, é dado por:

$$c_{ij} = \sum_{(i,k)\in\Gamma} f_{ij} d_{kl} \tag{5.2}$$

Todos os pares serviço-localidade não associados (j,l) são ordenados em ordem crescente de custo. Entre os pares que tem os menores $\alpha.|\Gamma|$ custos, um é selecionado aleatoriamente e adicionado ao conjunto Γ. O procedimento é repetido até todas as atribuições serem feitas.

5.2.2 Fase de busca local

Utiliza busca local na vizinhança *troca(2,2)*, ou seja, todas as possíveis trocas 2 a 2 dos pares serviço-localidade são consideradas. Se uma troca melhora o custo, esta troca é aceita. O procedimento continua até nenhuma troca melhorar o valor da solução.

5.3 GRASP para o problema de satisfabilidade máxima ponderada

Resende, Pitsoulis e Pardalos [9] propõem uma heurística para o problema de satisfabilidade máxima ponderada. Este problema pode ser aplicado em diversas áreas tais como diagnósticos de sistemas, banco de dados, "time-tabling", etc.

Sejam m cláusulas $C_1, C_2, \ldots, C_m$ envolvendo n variáveis booleanas $x_1, x_2, \ldots, x_n$ com atribuição de verdadeiro ou falso (1 ou 0 respectivamente). Existe um peso não-negativo w_i associado à cada cláusula C_i. A definição da cláusula i é:

$$C_{i,j} = \bigcup_{j=1}^{n_i} l_{ij} \tag{5.3}$$

onde n_i é o número de literais na cláusula C_i e os literais $l_{ij} \in \{x_i, \bar{x}_i \mid i = 1, \ldots, n\}$. Em um problema de satisfabilidade máxima ponderada (MAX-SAT) o que se faz é determinar a atribuição de valores "verdade" para as n variáveis que maximizam a soma de pesos de cláusulas satisfeitas.

O problema de satisfabilidade clássico (SAT) é um caso especial do MAX-SAT na qual todas as cláusulas tem peso unitário e o que se quer é decidir se existe uma atribuição

de peso total m. MAX-SAT é NP-Completo mesmo quando cada cláusula contém exatamente dois literais (MAX-2SAT) [9].

5.3.1 Fase de construção

A fase de construção consiste de n iterações. Sejam $N = \{1,2,\ldots,n\}$ e $M = \{1,2,\ldots, m\}$ o conjunto de índices das variáveis e de cláusulas, respectivamente. Soluções são construídas atribuindo-se verdadeiro (código 1) ou falso (código 0) para cada variável. Desta forma, para definir uma lista restrita de candidatos, há duas potenciais atribuições para cada variável: 1 ou 0. O objetivo da função gulosa é maximizar o peso total de cláusulas ainda não satisfeitas que tornam-se satisfeitas depois da atribuição em cada iteração. Para $i \in N$, seja Γ_i^+ o conjunto de cláusulas ainda não satisfeitas que poderiam ser satisfeitas se a variável $x_i = 1$. Da mesma forma, seja Γ_i^- o conjunto de cláusulas não atribuídas que poderiam ser satisfeitas se a variável x_i for falsa.

Sejam γ_i^+ e γ_i^- o somatório dos pesos das cláusulas satisfeitas pelas atribuições $x_i = 1$ e $x_i = 0$, respectivamente $\gamma_i^+ = \sum_{j \in \Gamma_i^+} w_j$ e $\gamma_i^- = \sum_{j \in \Gamma_i^-} w_j$. A escolha gulosa é selecionar a variável x_k com os maiores valores γ_k^+ ou γ_k^-. Se $\gamma_k^+ > \gamma_k^-$, então a atribuição $x_k = 1$ é feita, senão $x_k = 0$. Note que com todas as atribuições feitas, o conjunto Γ_i^+ e Γ_i^- muda para todo i, para refletir a nova atribuição. Isto conseqüentemente muda os valores de γ_i^+ e γ_i^-, caracterizando a componente adaptativa da heurística.

É assegurado que uma seleção aleatória será feita entre os melhores candidatos em qualquer dada atribuição. Sejam

$$\gamma^* = \max\{\gamma_i^+, \gamma_i^- \mid x_i \text{ ainda não atribuído}\}$$
$$\gamma_* = \min\{\gamma_i^+, \gamma_i^- \mid x_i \text{ ainda não atribuído}\}$$

seja $\alpha (0 \leq \alpha \leq 1)$ o parâmetro da lista restrita de candidatos. Um candidato x_i = verdadeiro é inserido dentro da LRC se $\gamma_i^+ \geq \gamma_* + \alpha.(\gamma^* - \gamma_*)$. Da mesma forma, um candidato x_i = falso é inserido dentro da LRC se $\gamma_i^- \geq \gamma_* + \alpha.(\gamma^* - \gamma_*)$.

5.3.2 Fase de busca local

Dado uma atribuição $x \in \{0,1\}^n$, definimos a vizinhança $N(x)$ = *troca(1,1)* como o conjunto de todos os vetores $y \in \{0,1\}^n$ tal que $||x-y||_2 = 1$. Denota-se por $w(x)$ o peso total de cláusulas satisfeitas pela atribuição verdade x, então a atribuição verdade x é um máximo local se e somente se $w(x) \geq w(y)$, para todo $y \in N(x)$. Começando com uma atribuição x, a busca local encontra o máximo local y em $N(x)$. Se $y \neq x$, faz-se $x = y$.

5.4 GRASP para o problema de planarização de grafos

Conforme [23], um grafo é considerado planar se ele pode ser desenhado em um plano, de tal forma que as arestas não se cruzem. A planarização de grafos tem grande aplicação em projetos de circuitos VLSI (vem do inglês "Very Large Scale Integration", integração de grande escala). Dado um grafo $G = (V,E)$ com conjunto de vértices V e conjunto de arestas E, o objetivo da planarização de grafos é encontrar um subconjunto de cardinalidade mínima de arestas $F \subseteq E$ tal que o grafo $G' = (V, E \setminus F)$ (resultante da remoção das arestas F de G) é planar.

Este problema é também conhecido como o problema do sub-grafo planar máximo. Um sub-grafo planar máximo é um sub-grafo planar $G' = (V', E')$ de $G = (V,E)$, tal que a adição de qualquer aresta $e \in E \setminus E'$ para G' torna-o não planar.

5.4.1 Fase de construção

Sejam $g_G(v)$ o grau do vértice v com respeito a G, o grau mínimo $\underline{g} = min_{v \in V}\{g_G(v)\}$ e o grau máximo $\overline{g} = max_{v \in V}\{g_G(v)\}$. A lista restrita de candidatos (LRC) é composta pelos vértices com grau no intervalo $[\underline{g}, \alpha(\overline{g} - \underline{g}) + \underline{g}]$.

Seja G_k o grafo induzido em G por $V \setminus \{v_1, v_2, \ldots, v_k\}$ na k-ésima iteração. Seja $ADJ_{G_{k-1}}(v_{k-1})$ o conjunto de vértices de G_{k-1} adjacentes de v_{k-1} em G. A LRC é composta de todos os vértices em $ADJ_{G_{k-1}}(v_{k-1})$ com o grau no intervalo $[\underline{g}, \alpha(\overline{g} - \underline{g}) + \underline{g}]$ em G_k. Por outro lado, se $ADJ_{G_{k-1}}(v_{k-1}) = 0$, a LRC é composta de todos os vértices não selecionados com o grau no intervalo $[\underline{g}, \alpha(\overline{g} - \underline{g}) + \underline{g}]$ em G_k.

Considere a seqüencia de vértices $\Pi = (v_1, v_2, \ldots, v_{|V|})$ e $\pi(v)$ a posição relativa do vértice $v \in V$ na seqüência Π. Seja $e_1 = (a,b)$ e $e_2 = (c,d)$ duas arestas de G, tal que, sem perda de generalidade, $\pi(a) < \pi(b)$ e $\pi(c) < \pi(d)$. Estas arestas são cruzadas com respeito a seqüência Π se $\pi(a) < \pi(c) < \pi(b) < \pi(d)$ ou $\pi(c) < \pi(a) < \pi(d) < \pi(b)$.

5.4.2 Fase de busca local

A busca local é feita na vizinhança $N(\Pi)$ formada por todos os vértices na seqüência Π' diferindo em exatamente duas posições: $N(\Pi) = \{\Pi' = (v'_1, v'_2, \ldots, v'_{|V|}) : v'_i = v_i, \forall i \neq j,k, v'_j = v_k, v'_k = v_j\}$.

5.4.3 Pós-otimização (procedimento de coloração dupla)

Seja $H = (E,I)$ um grafo onde cada um dos vértices corresponde a uma aresta do grafo G. Vértices de e_1 e e_2 de H são conectados por uma aresta se a aresta correspondente de G cruza com respeito a seqüência Π. Um grafo sobreposto tem seus vértices localizados em uma correspondência um-a-um com uma família de intervalos na linha. Dois intervalos são considerados sobrepostos se eles cruzam e nenhum é contido no outro. Dois vértices do grafo sobreposto são conectados por uma aresta se e somente se eles correspondem a intervalos sobrepostos. Desta forma, o grafo H é um grafo sobreposto associado com a representação de G definido pela seqüência Π.

O procedimento de coloração dupla consiste na coloração de um número máximo de vértices do grafo sobreposto H tal que cada uma das duas cores A (azul) e V (vermelho) formam um conjunto independente. Este procedimento equivale a busca de um sub-grafo bipartido tendo o maior número de vértices. Portanto, equivale a desenhar as arestas do grafo G acima ou abaixo da linha, onde os vértices tem sido localizado, de acordo com a seqüência Π.

5.5 GRASP para o problema de recobrimento

O problema de recobrimento, abordado por Feo e Resende [4] visa encontrar, dentro de um conjunto, um subconjunto de cardinalidade mínima composto por um conjunto selecionado de elementos que satisfazem ("cobrem") um conjunto de restrições.

Sejam n conjuntos finitos $P_1, P_2, \ldots, P_n$ representados pelos conjuntos $I = \cup(P_j : 1 \leq j \leq n) = \{1, \ldots, m\}$ e $J = \{1, \ldots, n\}$. Um subconjunto J^* de J é chamado de *cobertura* se $\cup(P_j : j \in J^*) = I$. O problema de recobrimento é encontrar uma *cobertura* de cardinalidade mínima. Definido A como a matriz $m \times n$ composta de elementos $(0,1)$, tal que $a_{ij} = 1$ se e somente se $i \in P_j$, uma formulação de programação inteira para o problema de recobrimento é

$$\text{minimizar } e_n x \tag{5.4}$$

$$\text{sujeito a } Ax \geq e_m, x = 0,1 \tag{5.5}$$

onde e_k é um vetor de 1´s de tamanho k, e x é um vetor de $(0,1)$ de tamanho n com $x_j = 1$ se e somente se $j \in J^*$.

A matriz A de entrada utilizada em [4] pertence à classe dos sistemas triplos de Steiner com largura igual a um. A largura β da matriz de $(0,1)$ A é o número mínimo de colunas que pode ser selecionado de A tal que todas as somas de linhas resultantes de uma submatriz de A tem no mínimo β elementos. Desta forma, para todos os pares de coluna j e k existem exatamente uma linha i na qual $a_{ij} = a_{ik} = 1, (i, j, k)$.

5.5.1 Fase de construção

Cada cobertura J^0 é construída iterativamente. Seja $\bar{\Gamma}$ a cardinalidade máxima dos conjuntos $P_1^0, \ldots, P_n^0$, a lista restrita de candidatos (LRC) $\wp$ é o conjunto de índices j correspondentes aos conjuntos P_j^0 com cardinalidade maior ou igual a $\alpha \times \bar{\Gamma}$. Um índice k é escolhido aleatoriamente da LRC e adicionado ao conjunto J^0. Então o conjunto correspondente ao índice selecionado P_k^0 é adicionado ao conjunto $(P_1^0, \ldots, P_n^0)$.

5.5.2 Fase de busca local

Dado uma cobertura J^0, a busca local procura remover, um a um, os seus elementos, com a finalidade de verificar se o elemento é supérfluo ou não. Caso o elemento seja supérfluo, ele é mantido fora da solução. Caso contrário, ele é colocado de volta na cobertura.

A cada nova cobertura J^0, avalia-se se a cobertura corrente contém a menor cardinalidade comparado as coberturas geradas anteriormente. Caso o fato seja verdadeiro, a cobertura corrente J^0 é armazenada como a melhor cobertura J^*.

6 Resultados computacionais

Neste capítulo serão apresentados os resultados computacionais. Será descrito o ambiente computacional usado para conduzir os experimentos e as instâncias selecionadas para cada uma das cinco heurísticas GRASP. Serão apresentados para cada combinação "critério de parada-heurística-instância" o tempo de parada (número de iterações) e a qualidade da solução e analisados os seus resultados.

6.1 Projeto experimental

Segundo Rardin e Uzsoy [3], a avaliação dos algoritmos baseados em metaheurísticas geralmente é feita empiricamente: aplicam-se procedimentos em uma coleção de instâncias específicas e compara-se a qualidade da solução observada e o custo computacional entre as abordagens avaliadas.

Na Seção 4.4 foi apresentado o *framework* `GRASP_BSR` usado para implementar os três critérios de parada propostos por Boender e Rinnooy Kan (detalhes na Seção 4.3) em heurísticas baseadas em GRASP. Experimentos computacionais foram executados com o *framework* `GRASP_BSR` nas cinco heurísticas GRASP mostradas no Capítulo 5, a saber: GRASP para o problema de conjunto independente máximo (programa `gmis` de Resende, Feo e Smith [24]); GRASP para o problema quadrático de atribuição (programa `gqapd` de Resende, Pardalos e Li. [22]); GRASP para o problema de satisfabilidade máxima ponderada (programa `maxsat` de Resende, Pitsoulis e Pardalos [9]); GRASP para o problema de planarização de grafos (programa `gmpsg` de Ribeiro e Resende [23]) e GRASP para o problema de recobrimento (programa `sc` de Feo e Resende [4]). O projeto experimental

totaliza 15 abordagens do tipo "heurística versus critério de parada", daqui em diante chamada de "heurística-parada". A heurística para o problema quadrático de atribuição foi testada com 18 instâncias e as demais heurísticas foram testadas com 5 instâncias cada, em um processo de compilação, detecção de erros, validação, organização e análise dos resultados.

Nos experimentos computacionais foram utilizadas instâncias com diferentes níveis de dificuldade conforme parâmetros descritos na Seção 6.3. As instâncias para os programas `gmpsg`, `maxsat` e `sc` foram fornecidas por um dos autores das heurísticas abordadas, as instâncias para o programa `gqapd` estavam disponíveis na biblioteca QAPLIB [25] e as instâncias para o programa `gmis` foram criadas por gerador de instâncias aleatórias. Dados uma semente aleatória e uma lista de parâmetros, estes geradores criam instâncias aleatórias consistentes com estes parâmetros. Na Seção 6.3 encontram-se mais detalhes sobre as fontes das instâncias.

Para cada uma das 114 combinações "heurística-parada" versus "instância'", daqui em diante "heurística-parada-instância", foram feitas 10 execuções independentes e mensurados o número de iterações, a qualidade da solução e o número da iteração onde foi encontrada a melhor solução. As sementes aleatórias que inicializam a execução variam de 1 a 10 e o gerador de números aleatórios utilizado foi baseado em Schrage [26].

Para a comparação dos critérios de parada será armazenado o tempo de parada (número de iterações) de cada abordagem "heurística-parada" e a qualidade da solução para este dado tempo. Desta forma, será possível avaliar as abordagens mais rápidas e as de melhor qualidade (através do valor da função objetivo da melhor solução encontrada). É necessário perceber se há uma convergência entre rapidez e qualidade para alguma abordagem ou se há conflitos onde uma abordagem supera a outra em um só quesito, mas não nos dois. Os experimentos indicarão também a eficiência ou não de critérios de parada "inteligentes" em relação a opção de se utilizar um número fixo de iterações. Com este objetivo será feito um estudo comparativo com as heurísticas executando um número fixo de iterações.

Em um estudo comparativo, o ideal é se obter o código-fonte original das heurísticas

abordadas, e assim, poder relatar os resultados produzidos tanto pelo programa original quanto pelo programa implementado com os critérios de parada, sem perder a comparabilidade por causa do efeito máquina (processador, sistema operacional, etc). No presente trabalho isto foi possível, pois os códigos-fontes das heurísticas GRASP abordadas [9, 22, 23, 4, 24] foram disponibilizados por um dos autores.

Sobre a documentação é importante ressaltar que foram apresentadas as informações sobre as métricas de desempenho (número de iterações e qualidade da solução encontrada), o nome e a fonte dos programas (heurísticas), a configuração do computador (processador, sistema operacional, linguagem de programação e compilador), a especificação das instâncias, o gerador de números aleatórios, as sementes e os parâmetros utilizados. Todas estas informações são necessárias para garantir a reprodutibilidade dos resultados [27].

Ressalta-se que as análises empíricas estão limitadas ao cenário proposto neste projeto experimental, ou seja, todas as conclusões referem-se, a priori, apenas para as instâncias avaliadas no estudo em questão. Pode-se afirmar que uma abordagem é melhor que as outras avaliadas, porém não é plausível generalizá-la como a melhor de todas as existentes.

6.2 Ambiente computacional

Os experimentos foram feitos em um computador HP Pavillion dv6000 com processador AMD Turion 64 X2 com 2GB de memória em um sistema operacional Ubuntu Linux 7.1.

Os programas foram escritos em Fortran (as versões originais dos programas utilizados foram escritos em Fortran e por isso as novas versões estão em Fortran) e os códigos foram compilados com o compilador `g77` versão 3.4.6. Os programas para o problema de conjunto independente máximo, para satisfabilidade máxima ponderada, para o problema quadrático de atribuição e para planarização de grafos foram compilados usando a opção `-o -static` e para recobrimento foram compilados usando as opções `-o`.

6.3 Instâncias

As instâncias utilizadas no programa para o problema de conjunto independente máximo foram escolhidas de uma das classes mais estudadas de grafos aleatórios em Bollobas [28] denotado por $G_{n,p}$. Estes grafos possuem n nós e cada arco (i, j), para $i, j = 1, \ldots, n$ e $i \neq j$, existe segundo uma probabilidade p. Foram utilizados grafos com os seguintes parâmetros: (i) $G_{n,p}(n = 100, p = 0,2)$ (991 arcos); (ii) $G_{n,p}(n = 100, p = 0,5)$ (2505 arcos); (iii) $G_{n,p}(n = 150, p = 0,2)$ (2173 arcos); (iv) $G_{n,p}(n = 150, p = 0,5)$ (5556 arcos) e (v) $G_{n,p}(n = 500, p = 0,5)$ (62523 arcos). Tais grafos foram gerados aleatoriamente através do programa `gngrf` de Resende, Feo e Smith [24].

Para o problema quadrático de atribuição, foram selecionadas 18 instâncias da coleção QAPLIB de Burkard, Karisch e Rendl [25]. Essas instâncias possuem pelo menos uma das matrizes de distância ou de fluxo simétricas. O valor ótimo de todas as instâncias deste grupo é conhecido. São elas: chr12a, chr15a, chr20a, chr25a, els19, esc16a, esc32a, kra30a, nug18, nug25, nug30, rou20, sko42, ste36a, tai25a, tai30a, tho30 e wil50.

Para o programa de planarização de grafos, foram utilizadas instâncias de Ribeiro e Resende [23] (originalmente apresentadas em Cimikowski [29] e Goldschmidt e Takvorian [30]). São elas: g10, rg50.1, rg200.1, tg100.1 e tg200.1.

As instâncias para satisfabilidade máxima ponderada foram escolhidas entre as instâncias apresentadas em Resende, Pitsoulis e Pardalos [9]. São elas: jnh8, jnh16, jnh201, jnh218 e jnh307.

Para o programa de recobrimento, foram adquiridas instâncias em Feo e Resende [4] originárias de Fulkerson, Nemhauser e Trotter [31] e Ramakrishnan [34]. São elas: data.9, data.45, data.81, data.135 e data.243.

6.4 Análises dos Resultados

Relembrando a Seção 4.3, para dado valor c_j uma simples tabela resume a estratégia de parada ótima para as funções de perda L_1, L_3 e L_4, onde o limite superior de iterações de cada função de perda L_j é dado pelo valor da variável n_j^*. A execução de um programa

Tabela 6.1: Relação entre c_j e n_j^*.

	$n_j^* \cong$	1.000	10.000	100.000
			c_j	
L_1	$c_j + 1 - \sqrt{4c+1}$	1.000	10.000	100.000
L_3	$c_j/4$	4.000	40.000	400.000
L_4	$c_j/3$	3.000	30.000	300.000

pára ou porque a perda posteriori esperada de $n+1$ iterações é maior ou igual a perda posteriori corrente ou porque o número de iterações alcançou o limite superior n_j^*.

Os três gráficos teóricos da Figura 6.1 ilustram a região de continuidade do algoritmo para $c_j \in \{1.000, 2.000, 5.000\}$. Observa-se que quanto maior o valor do parâmetro c_j, maior a região de continuidade do algoritmo.

Desta forma, considerando-se que pretende-se executar, no máximo, um dado número de n iterações, deseja-se investigar o comportamento do *framework* `GRASP_BSR` para cada um dos critérios de parada fazendo $n_j^* = n$. Para tal, foi idealizado o experimento que compara a qualidade das soluções obtidas pelo *framework* `GRASP_BSR` com aquelas obtidas pelo GRASP original com n iterações para $n \in \{1.000, 10.000, 100.000\}$. O objetivo é perceber se dado um limite máximo de iterações que o usuário possa esperar, o que é mais vantajoso: executar este limite como número fixo de iterações ou utilizar um critério de parada inteligente que demore até no máximo este limite. A escolha favorecerá o critério de parada inteligente se a qualidade da sua solução for igual ao do número fixo de iterações com um número de iterações menor. A Tabela 6.1 relaciona, para cada função de perda L_j, os valores do parâmetro c_j com os valores do limite superior de iterações n_j^* usados no experimento.

A métrica usada para avaliar a qualidade da solução q será calculada como a diferença relativa entre o valor da solução obtida em `GRASP_BSR` (s_{GRASP_BSR}), e o valor da melhor solução conhecida (s_{melhor}). A qualidade q é um índice de escala de 100 pontos, onde quanto mais próximo de 100, melhor a qualidade da solução:

$$q = 100 - \left(\frac{|s_{GRASP\_BSR} - s_{melhor}|}{s_{melhor}} \times 100\right). \tag{6.1}$$

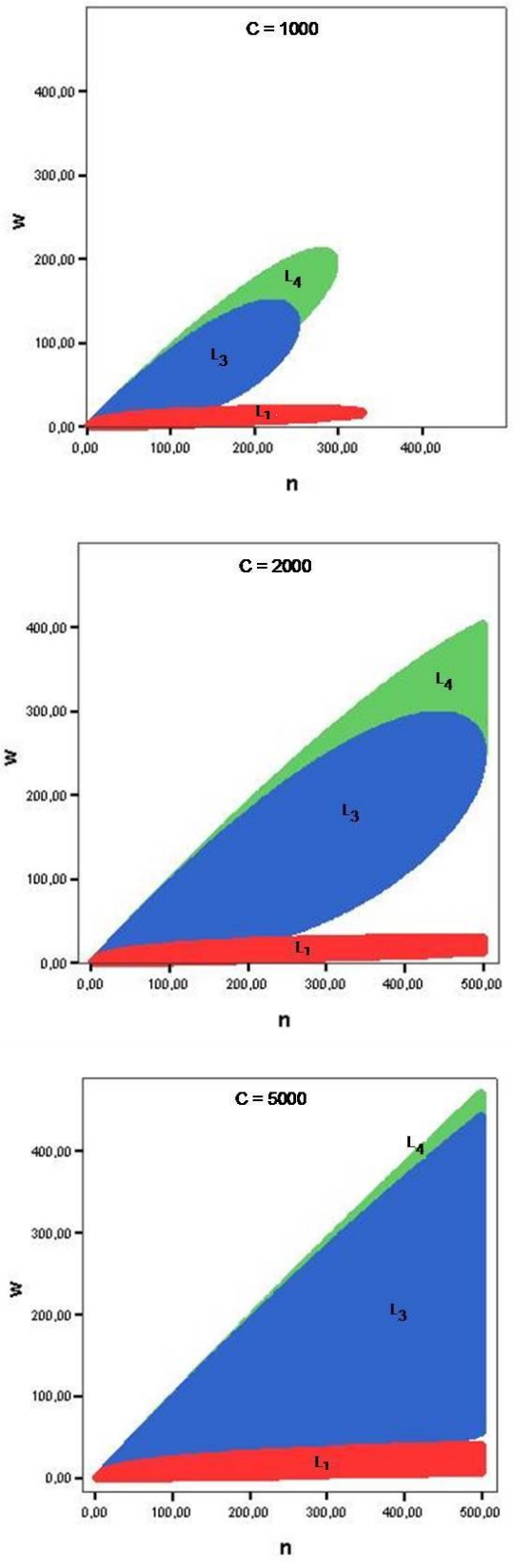

Figura 6.1: Gráfico teórico com o parâmetro c_j igual a 1000, 2000 e 5000.

As Tabelas 6.2, 6.3 e 6.4 mostram os resultados detalhados do experimento para os problemas de conjunto independente máximo, de satisfabilidade máxima ponderada, de planarização de grafos e de recobrimento. Existem quatro grupos de linha, um para cada heurística GRASP e suas cinco instâncias. A primeira coluna identifica o nome da instância. Em seguida, um grupo com três colunas apresenta os resultados para heurística GRASP original com um número fixo de n iterações. Para cada critério de parada L_1, L_3 e L_4 existe um grupo com cinco colunas que apresentam os resultados para `GRASP_BSR`. Para todos os grupos, a coluna w mostra o número de diferentes ótimos locais, a coluna s a melhor solução encontrada, e a coluna mi reporta o número da iteração onde a melhor solução foi encontrada. Para os últimos três grupos, a coluna n representa o número de iterações executadas e a coluna q representa a qualidade da solução de acordo com (6.1), onde s_{melhor} é igual ao valor de s para o GRASP original com n iterações. Todos os resultados são médias de 10 execuções independentes.

As Tabelas 6.5, 6.6 e 6.7 mostram os resultados detalhados do experimento para o problema quadrático de atribuição. Para este problema o valor da melhor solução é o valor ótimo s^*. A primeira coluna identifica o nome da instância e a segunda coluna mostra o valor da solução ótima s^*. Em seguida, um grupo com quatro colunas apresenta os resultados para heurística GRASP original com um número fixo de n iterações. Para cada critério de parada L_1, L_3 e L_4 existe um grupo com cinco colunas que apresentam os resultados para `GRASP_BSR`. Para todos os grupos, a coluna w mostra o número de diferentes ótimos locais, a coluna s a melhor solução encontrada, a coluna mi reporta o número da iteração onde a melhor solução foi encontrada e a coluna q representa a qualidade da solução de acordo com (6.1), onde $s_{melhor} = s^*$. Para os últimos três grupos, a coluna n representa o número de iterações executadas. Todos os resultados são médias de 10 execuções independentes. Ao todo, foram realizadas 4.560 execuções.

A Tabela 6.8 sumariza os resultados. Para cada uma das cinco heurísticas e para cada um dos três valores de n_j^* existe uma linha na tabela com as médias dos resultados apresentados nas Tabelas de 6.2 a 6.7, a quarta linha do grupo mostra a média dos três valores de n_j^* para a mesma combinação heurística e critério de parada. O último grupo

Tabela 6.2: GRASP com número fixo de iterações comparado com GRASP_BSR para L_1, L_3 e L_4 com $n_j^* \cong 1.000$.

	$n = 1.000$			L_1, $c = 1.000$, $n^* = 938$					L_3, $c = 4.000$, $n^* = 1.000$					L_4, $c = 3.000$, $n^* = 1.000$				
instância	*w*	*s*	*mi*	*n*	*w*	*s*	*mi*	*q*	*n*	*w*	*s*	*mi*	*q*	*n*	*w*	*s*	*mi*	*q*
GRASP para o problema de conjunto independente máximo (programa `gmis`)																		
$G_{100,0.2}$	285,9	20,0	74,1	17,3	15,3	19,0	1,6	95,0	853,3	263,3	20,0	74,1	100,0	532,1	200,7	20,0	74,1	100,0
$G_{100,0.5}$	3,0	8,0	1,0	100,0	3,0	8,0	1,0	100,0	108,0	3,0	8,0	1,0	100,0	41,0	3,0	8,0	1,0	100,0
$G_{150,0.2}$	482,8	22,0	23,4	25,3	23,3	21,6	5,6	98,2	997,9	481,8	22,0	23,4	100,0	778,0	404,2	22,0	23,4	100,0
$G_{150,0.5}$	66,8	10,0	10,5	11,2	9,2	9,6	2,7	96,0	454,3	59,4	10,0	10,5	100,0	225,2	48,8	10,0	10,5	100,0
$G_{500,0.5}$	411,8	13,0	17,0	19,9	17,9	12,6	5,7	96,9	971,7	405,3	13,0	17,0	100,0	709,7	336,5	13,0	17,0	100,0
média	250,1		25,2	34,7	13,7		3,3	97,2	677,0	242,6		25,2	100,0	457,2	198,6		25,2	100,0
GRASP para o problema de satisfabilidade máxima ponderada (programa `maxsat`)																		
jnh8.sat	1000,0	776987,0	553,1	938,0	938,0	776987,0	553,1	100,0	1000,0	1000,0	776987,0	553,1	100,0	1000,0	1000,0	776987,0	553,1	100,0
jnh16.sat	1000,0	729918,2	445,1	938,0	938,0	729918,2	445,1	100,0	1000,0	1000,0	729918,2	445,1	100,0	1000,0	1000,0	729918,2	445,1	100,0
jnh201.sat	999,9	461326,3	435,1	938,0	937,9	461264,2	410,1	100,0	1000,0	999,9	461326,3	435,1	100,0	1000,0	999,9	461326,3	435,1	100,0
jnh218.sat	1000,0	435146,7	274,3	938,0	938,0	435146,7	274,3	100,0	1000,0	1000,0	435146,7	274,3	100,0	1000,0	1000,0	435146,7	274,3	100,0
jnh307.sat	1000,0	856669,7	541,4	938,0	938,0	856669,7	541,4	100,0	1000,0	1000,0	856669,7	541,4	100,0	1000,0	1000,0	856669,7	541,4	100,0
média	1000,0		449,8	938,0	938,0		444,8	100,0	1000,0	1000,0		449,8	100,0	1000,0	1000,0		449,8	100,0
GRASP para problema de planarização de grafos (programa `gmpsg`)																		
g10	999,9	68,6	472,0	938,0	937,9	68,6	472,0	100,0	1000,0	999,9	68,6	472,0	100,0	1000,0	999,9	68,6	472,0	100,0
rg50.1	1000,0	87,9	366,3	938,0	938,0	87,9	366,3	100,0	1000,0	1000,0	87,9	366,3	100,0	1000,0	1000,0	87,9	366,3	100,0
rg200.1	1000,0	275,7	375,8	938,0	938,0	275,7	375,8	100,0	1000,0	1000,0	275,7	375,8	100,0	1000,0	1000,0	275,7	375,8	100,0
tg100.1	1000,0	242,0	478,9	938,0	938,0	240,9	308,9	99,5	1000,0	1000,0	242,0	478,9	100,0	1000,0	1000,0	242,0	478,9	100,0
tg200.1	1000,0	448,7	354,6	938,0	938,0	448,6	306,7	100,0	1000,0	1000,0	448,7	354,6	100,0	1000,0	1000,0	448,7	354,6	100,0
média	1000,0		409,5	938,0	938,0		365,9	99,9	1000,0	1000,0		409,5	100,0	1000,0	1000,0		409,5	100,0
GRASP para o problema de recobrimento (programa `sc`)																		
data.9	54,0	5,0	1,0	16,3	14,3	5,0	1,0	100,0	435,0	54,0	5,0	1,0	100,0	239,7	53,9	5,0	1,0	100,0
data.45	998,9	30,5	262,6	766,4	765,3	30,6	164,0	99,7	803,6	802,5	30,6	164,0	99,7	803,6	802,5	30,6	164,0	99,7
data.81	1000,0	61,4	400,3	938,0	938,0	61,4	400,3	100,0	1000,0	1000,0	61,4	400,3	100,0	1000,0	1000,0	61,4	400,3	100,0
data.135	1000,0	105,6	218,9	938,0	938,0	105,7	132,7	99,9	1000,0	1000,0	105,6	218,9	100,0	1000,0	1000,0	105,6	218,9	100,0
data.243	1000,0	204,6	320,2	938,0	938,0	204,6	320,2	100,0	1000,0	1000,0	204,6	320,2	100,0	1000,0	1000,0	204,6	320,2	100,0
média	810,6		240,6	719,3	718,7		203,6	99,9	847,7	771,3		220,9	99,9	808,7	771,3		220,9	99,9

Tabela 6.3: GRASP com número fixo de iterações comparado com GRASP_BSR para L_1, L_3 e L_4 com $n_j^* \cong 10.000$.

	$n = 10.000$			L_1, $c = 10.000$, $n^* = 9.801$					L_3, $c = 40.000$, $n^* = 10.000$					L_4, $c = 30.000$, $n^* = 10.000$				
instância	*w*	*s*	*mi*	*n*	*w*	*s*	*mi*	*q*	*n*	*w*	*s*	*mi*	*q*	*n*	*w*	*s*	*mi*	*q*
GRASP para o problema de conjunto independente máximo (programa `gmis`)																		
$G_{100,0.2}$	896,6	20,0	74,1	17,5	15,5	19,0	1,6	95,0	4691,4	637,2	20,0	74,1	100,0	2050,6	425,3	20,0	74,1	100,0
$G_{100,0.5}$	3,0	8,0	1,0	337,0	3,0	8,0	1,0	100,0	345,0	3,0	8,0	1,0	100,0	89,0	3,0	8,0	1,0	100,0
$G_{150,0.2}$	2116,0	22,1	1007,0	25,3	23,3	21,6	5,6	97,7	7282,4	1747,7	22,0	23,4	99,5	3896,6	1191,7	22,0	23,4	99,5
$G_{150,0.5}$	79,6	10,0	10,5	11,4	9,4	9,6	2,7	96,0	1651,0	71,2	10,0	10,5	100,0	596,0	62,3	10,0	10,5	100,0
$G_{500,0.5}$	1015,1	13,0	17,0	20,1	18,1	12,6	5,7	96,9	5350,2	851,8	13,0	17,0	100,0	2721,1	666,5	13,0	17,0	100,0
média	822,1		221,9	82,3	13,9		3,3	97,1	3864,0	662,2		25,2	99,9	1870,7	469,8		25,2	99,9
GRASP para o problema de satisfabilidade máxima ponderada (programa `maxsat`)																		
jnh8.sat	10000,0	821062,4	3626,0	9801,0	9801,0	821062,4	3626,0	100,0	10000,0	10000,0	821062,4	3626,0	100,0	10000,0	10000,0	821062,4	3626,0	100,0
jnh16.sat	10000,0	788786,3	5313,0	9801,0	9801,0	788786,3	5313,0	100,0	10000,0	10000,0	788786,3	5313,0	100,0	10000,0	10000,0	788786,3	5313,0	100,0
jnh201.sat	9994,0	462018,0	6149,7	5676,4	5674,4	461599,1	2100,9	99,9	5676,4	5674,4	461599,1	2100,9	99,9	5676,4	5674,4	461599,1	2100,9	99,9
jnh218.sat	10000,0	437068,3	5050,8	9801,0	9801,0	437068,3	5050,8	100,0	10000,0	10000,0	437068,3	5050,8	100,0	10000,0	10000,0	437068,3	5050,8	100,0
jnh307.sat	10000,0	921336,3	4904,1	9801,0	9801,0	921336,3	4904,1	100,0	10000,0	10000,0	921336,3	4904,1	100,0	10000,0	10000,0	921336,3	4904,1	100,0
média	9998,8		5008,7	8976,1	8975,7		4199,0	100,0	9135,3	9134,9		4199,0	100,0	9135,3	9134,9		4199,0	100,0
GRASP para problema de planarização de grafos (programa `gmpsg`)																		
g10	9976,2	69,0	1160,3	2597,1	2595,1	68,9	899,4	99,9	2597,1	2595,1	68,9	899,4	99,9	2597,1	2595,1	68,9	899,4	99,9
rg50.1	10000,0	89,2	2974,1	9801,0	9801,0	89,2	2974,1	100,0	10000,0	10000,0	89,2	2974,1	100,0	10000,0	10000,0	89,2	2974,1	100,0
rg200.1	10000,0	278,6	3744,9	9801,0	9801,0	278,6	3744,9	100,0	10000,0	10000,0	278,6	3744,9	100,0	10000,0	10000,0	278,6	3744,9	100,0
tg100.1	10000,0	247,3	4842,6	9801,0	9801,0	247,3	4842,6	100,0	10000,0	10000,0	247,3	4842,6	100,0	10000,0	10000,0	247,3	4842,6	100,0
tg200.1	10000,0	456,7	4994,5	9801,0	9801,0	456,4	4275,5	99,9	10000,0	10000,0	456,7	4994,5	100,0	10000,0	10000,0	456,7	4994,5	100,0
média	9995,2		3543,3	8360,2	8359,8		3347,3	100,0	8519,4	8519,0		3491,1	100,0	8519,4	8519,0		3491,1	100,0
GRASP para o problema de recobrimento (programa `sc`)																		
data.9	54,0	5,0	1,0	16,5	14,5	5,0	1,0	100,0	1442,0	54,0	5,0	1,0	100,0	544,0	54,0	5,0	1,0	100,0
data.45	9881,0	30,0	1276,0	1094,1	1092,1	30,5	298,4	98,3	1094,1	1092,1	30,5	298,4	98,3	1094,1	1092,1	30,5	298,4	98,3
data.81	10000,0	61,0	824,0	9801,0	9801,0	61,0	824,0	100,0	10000,0	10000,0	61,0	824,0	100,0	10000,0	10000,0	61,0	824,0	100,0
data.135	10000,0	104,8	2632,3	9801,0	9801,0	104,8	2632,3	100,0	10000,0	10000,0	104,8	2632,3	100,0	10000,0	10000,0	104,8	2632,3	100,0
data.243	10000,0	203,8	2545,8	9801,0	9801,0	203,8	2545,8	100,0	10000,0	10000,0	203,8	2545,8	100,0	10000,0	10000,0	203,8	2545,8	100,0
média	7987,0		1455,8	6102,7	6101,9		1260,3	99,7	6507,2	6229,2		1260,3	99,7	6327,6	6229,2		1260,3	99,7

Tabela 6.4: GRASP com número fixo de iterações comparado com GRASP_BSR para L_1, L_3 e L_4 com $n_j^* \cong 100.000$.

	$n = 100.000$			L_1, $c = 100.000$, $n^* = 99.369$					L_3, $c = 400.000$, $n^* = 100.000$					L_4, $c = 300.000$, $n^* = 100.000$				
instância	*w*	*s*	*mi*	*n*	*w*	*s*	*mi*	*q*	*n*	*w*	*s*	*mi*	*q*	*n*	*w*	*s*	*mi*	*q*
GRASP para o problema de conjunto independente máximo (programa gmis)																		
$G_{100,0.2}$	2126,1	20,0	74,1	17,6	15,6	19,0	1,6	95,0	21420,0	1222,5	20,0	74,1	100,0	6704,7	752,1	20,0	74,1	100,0
$G_{100,0.5}$	3,0	8,0	1,0	1084,0	3,0	8,0	1,0	100,0	1094,0	3,0	8,0	1,0	100,0	192,0	3,0	8,0	1,0	100,0
$G_{150,0.2}$	6836,6	22,3	7443,3	25,3	23,3	21,6	5,6	96,9	39343,7	4407,2	22,3	7443,3	100,0	15307,1	2696,3	22,1	1007,0	99,1
$G_{150,0.5}$	80,1	10,0	10,5	11,6	9,6	9,6	2,7	96,0	5519,0	77,3	10,0	10,5	100,0	1414,0	69,9	10,0	10,5	100,0
$G_{500,0.5}$	1413,8	13,0	17,0	20,2	18,2	12,6	5,7	96,9	21070,2	1181,3	13,0	17,0	100,0	7812,7	951,0	13,0	17,0	100,0
média	2091,9		1509,2	231,7	13,9		3,3	97,0	17689,4	1378,3		1509,2	100,0	6286,1	894,5		221,9	99,8
GRASP para o problema de satisfabilidade máxima ponderada (programa maxsat)																		
jnh8.sat	100000,0	834797,3	55577,6	99369,0	99369,0	834797,3	55577,6	100,0	100000,0	100000,0	834797,3	55577,6	100,0	100000,0	100000,0	834797,3	55577,6	100,0
jnh16.sat	100000,0	800353,5	51852,2	99369,0	99369,0	800353,5	51852,2	100,0	100000,0	100000,0	800353,5	51852,2	100,0	100000,0	100000,0	800353,5	51852,2	100,0
jnh201.sat	99728,7	462664,0	54813,2	5676,4	5674,4	461599,1	2100,9	99,8	5676,4	5674,4	461599,1	2100,9	99,8	5676,4	5674,4	461599,1	2100,9	99,8
jnh218.sat	99998,5	437838,3	44083,5	88100,8	88099,4	437702,8	41014,7	100,0	88353,2	88351,8	437702,8	41014,7	100,0	88353,2	88351,8	437702,8	41014,7	100,0
jnh307.sat	100000,0	939935,5	50165,7	99369,0	99369,0	939935,5	50165,7	100,0	100000,0	100000,0	939935,5	50165,7	100,0	100000,0	100000,0	939935,5	50165,7	100,0
média	99945,4		51298,4	78376,8	78376,2		40142,2	99,9	78805,9	78805,2		40142,2	99,9	78805,9	78805,2		40142,2	99,9
GRASP para problema de planarização de grafos (programa gmpsg)																		
g10	97667,0	69,0	1395,7	2597,1	2595,1	68,9	899,4	99,9	2597,1	2595,1	68,9	899,4	99,9	2597,1	2595,1	68,9	899,4	99,9
rg50.1	99999,0	90,7	15837,3	98218,6	98217,9	90,4	29992,4	99,7	98786,5	98785,8	90,4	29992,4	99,7	98786,5	98785,8	90,4	29992,4	99,7
rg200.1	100000,0	280,4	42147,7	99369,0	99369,0	280,4	42147,7	100,0	100000,0	100000,0	280,4	42147,7	100,0	100000,0	100000,0	280,4	42147,7	100,0
tg100.1	100000,0	251,2	36654,0	99369,0	99369,0	251,2	36654,0	100,0	100000,0	100000,0	251,2	36654,0	100,0	100000,0	100000,0	251,2	36654,0	100,0
tg200.1	100000,0	466,0	41328,8	99369,0	99369,0	466,0	41328,8	100,0	100000,0	100000,0	466,0	41328,8	100,0	100000,0	100000,0	466,0	41328,8	100,0
média	99533,2		27472,7	79784,5	79784,0		30204,5	99,9	80276,7	80276,2		30204,5	99,9	80276,7	80276,2		30204,5	99,9
GRASP para o problema de recobrimento (programa sc)																		
data.9	54,0	5,0	1,0	16,6	14,6	5,0	1,0	100,0	4619,0	54,0	5,0	1,0	100,0	1194,0	54,0	5,0	1,0	100,0
data.45	91178,8	30,0	1276,0	1094,1	1092,1	30,5	298,4	98,3	1140,6	1138,6	30,5	298,4	98,3	1196,1	1194,1	30,5	298,4	98,3
data.81	99999,9	61,0	824,0	99369,0	99368,9	61,0	758,1	100,0	100000,0	99999,9	61,0	824,0	100,0	100000,0	99999,9	61,0	824,0	100,0
data.135	100000,0	104,7	5898,3	99369,0	99369,0	104,7	5898,3	100,0	100000,0	100000,0	104,7	5898,3	100,0	100000,0	100000,0	104,7	5898,3	100,0
data.243	100000,0	203,0	24504,5	99369,0	99369,0	203,0	24504,5	100,0	100000,0	100000,0	203,0	24504,5	100,0	100000,0	100000,0	203,0	24504,5	100,0
média	78246,5		6500,8	59843,5	59842,7		6292,1	99,7	61151,9	60238,5		6305,2	99,7	60478,0	60249,6		6305,2	99,7

Tabela 6.5: GRASP para o problema quadrático de atribuição (programa gqapd) com $n_j^* \cong 1.000$.

		$n = 1.000$				L_1, $c = 1.000$, $n^* = 938$					L_3, $c = 4.000$, $n^* = 1.000$					L_4, $c = 3.000$, $n^* = 1.000$				
instância	$s*$	w	s	mi	q	n	w	s	mi	q	n	w	s	mi	q	n	w	s	mi	q
chr12a	9552	531,0	9552,0	182,2	100,0	40,6	38,6	10061,1	16,9	94,7	787,6	425,1	9592,4	151,0	99,6	747,8	425,6	9552,0	182,2	100,0
chr15a	9896	954,8	9924,8	309,8	99,7	197,0	195,0	10132,2	106,3	97,6	197,0	195,0	10132,2	106,3	97,6	197,0	195,0	10132,2	106,3	97,6
chr20a	2192	999,9	2356,6	499,0	92,5	938,0	937,9	2356,6	499,0	92,5	1000,0	999,9	2356,6	499,0	92,5	1000,0	999,9	2356,6	499,0	92,5
chr25a	3796	1000,0	4405,8	609,6	83,9	938,0	938,0	4405,8	609,6	83,9	1000,0	1000,0	4405,8	609,6	83,9	1000,0	1000,0	4405,8	609,6	83,9
els19	17212548	454,6	17212548,0	233,7	100,0	25,9	23,9	17729900,2	9,5	97,0	991,2	451,5	17212548,0	233,7	100,0	749,9	374,9	17212548,0	233,7	100,0
esc16a	68	1000,0	68,0	4,1	100,0	938,0	938,0	68,0	4,1	100,0	1000,0	1000,0	68,0	4,1	100,0	1000,0	1000,0	68,0	4,1	100,0
esc32a	130	1000,0	135,6	594,9	95,7	938,0	938,0	135,6	594,9	95,7	1000,0	1000,0	135,6	594,9	95,7	1000,0	1000,0	135,6	594,9	95,7
kra30a	88900	1000,0	90197,0	400,3	98,5	938,0	938,0	90197,0	400,3	98,5	1000,0	1000,0	90197,0	400,3	98,5	1000,0	1000,0	90197,0	400,3	98,5
nug18	1930	975,2	1932,0	362,4	99,9	293,3	291,3	1938,8	163,0	99,5	293,3	291,3	1938,8	163,0	99,5	293,3	291,3	1938,8	163,0	99,5
nug25	3744	998,8	3746,2	465,7	99,9	897,3	896,3	3746,8	386,6	99,9	946,4	945,3	3746,2	465,7	99,9	946,4	945,3	3746,2	465,7	99,9
nug30	6124	1000,0	6155,4	402,4	99,5	938,0	938,0	6155,4	402,4	99,5	1000,0	1000,0	6155,4	402,4	99,5	1000,0	1000,0	6155,4	402,4	99,5
rou20	725522	999,1	727457,4	644,0	99,7	923,8	922,9	727603,0	552,6	99,7	973,4	972,5	727550,2	637,5	99,7	973,4	972,5	727550,2	637,5	99,7
sko42	15812	1000,0	15921,6	483,0	99,3	938,0	938,0	15922,4	389,2	99,3	1000,0	1000,0	15921,6	483,0	99,3	1000,0	1000,0	15921,6	483,0	99,3
ste36a	9526	1000,0	9730,2	586,2	97,9	938,0	938,0	9731,6	501,3	97,8	1000,0	1000,0	9730,2	586,2	97,9	1000,0	1000,0	9730,2	586,2	97,9
tai25a	1167256	1000,0	1186543,0	424,7	98,3	938,0	938,0	1188718,4	290,3	98,2	1000,0	1000,0	1186543,0	424,7	98,3	1000,0	1000,0	1186543,0	424,7	98,3
tai30a	1818146	1000,0	1847617,4	497,3	98,4	938,0	938,0	1847896,6	451,8	98,4	1000,0	1000,0	1847617,4	497,3	98,4	1000,0	1000,0	1847617,4	497,3	98,4
tho30	149936	1000,0	150764,4	523,1	99,4	938,0	938,0	150764,4	523,1	99,4	1000,0	1000,0	150764,4	523,1	99,4	1000,0	1000,0	150764,4	523,1	99,4
wil50	48816	1000,0	48947,8	601,1	99,7	938,0	938,0	48947,8	601,1	99,7	1000,0	1000,0	48947,8	601,1	99,7	1000,0	1000,0	48947,8	601,1	99,7
média		939,6		434,6	97,9	757,4	756,9		361,2	97,3	899,4	848,9		410,2	97,8	883,8	844,7		411,9	97,8

Tabela 6.6: GRASP para o problema quadrático de atribuição (programa gqapd) com $n_j^* \cong 10.000$.

		$n = 10.000$				L_1, $c = 10.000$, $n^* = 9.801$					L_3, $c = 40.000$, $n^* = 10.000$					L_4, $c = 30.000$, $n^* = 10.000$				
instância	$s*$	w	s	mi	q	n	w	s	mi	q	n	w	s	mi	q	n	w	s	mi	q
chr12a	9552	1453,8	9552,0	165,2	100,0	39,4	37,4	10010,2	16,4	95,2	6378,3	1271,3	9552,0	165,2	100,0	3551,1	1023,8	9552,0	165,2	100,0
chr15a	9896	7211,2	9896,0	1368,6	100,0	197,0	195,0	10132,2	106,3	97,6	2216,1	1767,9	10006,4	248,5	98,9	5978,3	4501,5	9951,2	1155,1	99,4
chr20a	2192	9991,2	2227,6	4578,4	98,4	4708,8	4706,8	2265,8	2423,9	96,6	4708,8	4706,8	2265,8	2423,9	96,6	4708,8	4706,8	2265,8	2423,9	96,6
chr25a	3796	9999,8	4022,4	4815,0	94,0	9801,0	9800,8	4022,4	4815,0	94,0	10000,0	9999,8	4022,4	4815,0	94,0	10000,0	9999,8	4022,4	4815,0	94,0
els19	17212548	1551,2	17212548,0	233,7	100,0	25,9	23,9	17729900,2	9,5	97,0	6363,5	1264,3	17212548,0	233,7	100,0	3341,4	927,3	17212548,0	233,7	100,0
esc16a	68	9998,8	68,0	4,1	100,0	9239,2	9238,3	68,0	4,1	100,0	9378,5	9377,6	68,0	4,1	100,0	9378,5	9377,6	68,0	4,1	100,0
esc32a	130	10000,0	132,2	3616,7	98,3	9801,0	9801,0	132,2	3616,7	98,3	10000,0	10000,0	132,2	3616,7	98,3	10000,0	10000,0	132,2	3616,7	98,3
kra30a	88900	9999,8	88920,0	3933,3	100,0	9801,0	9800,8	88920,0	3933,3	100,0	10000,0	9999,8	88920,0	3933,3	100,0	10000,0	9999,8	88920,0	3933,3	100,0
nug18	1930	8504,3	1930,0	585,3	100,0	293,3	291,3	1938,8	163,0	99,5	392,9	388,8	1936,6	175,5	99,7	1026,0	970,8	1936,0	187,4	99,7
nug25	3744	9915,7	3744,0	1822,2	100,0	1344,4	1342,4	3745,4	694,1	100,0	1344,4	1342,4	3745,4	694,1	100,0	1344,4	1342,4	3745,4	694,1	100,0
nug30	6124	9998,2	6134,4	5728,4	99,8	8574,7	8573,2	6134,4	5728,4	99,8	8674,2	8672,7	6134,4	5728,4	99,8	8674,2	8672,7	6134,4	5728,4	99,8
rou20	725522	9937,2	725542,0	4794,2	100,0	1382,1	1380,1	727311,2	704,7	99,8	1382,1	1380,1	727311,2	704,7	99,8	1382,1	1380,1	727311,2	704,7	99,8
sko42	15812	10000,0	15876,2	5433,4	99,6	9801,0	9801,0	15876,2	5433,4	99,6	10000,0	10000,0	15876,2	5433,4	99,6	10000,0	10000,0	15876,2	5433,4	99,6
ste36a	9526	9999,6	9627,8	6810,1	98,9	9801,0	9800,6	9628,8	6578,1	98,9	10000,0	9999,6	9627,8	6810,1	98,9	10000,0	9999,6	9627,8	6810,1	98,9
tai25a	1167256	10000,0	1179041,4	3797,8	99,0	9801,0	9801,0	1179041,4	3797,8	99,0	10000,0	10000,0	1179041,4	3797,8	99,0	10000,0	10000,0	1179041,4	3797,8	99,0
tai30a	1818146	10000,0	1839444,6	5043,6	98,8	9801,0	9801,0	1839444,6	5043,6	98,8	10000,0	10000,0	1839444,6	5043,6	98,8	10000,0	10000,0	1839444,6	5043,6	98,8
tho30	149936	9993,5	150341,6	5462,1	99,7	4783,1	4781,2	150448,0	2553,6	99,7	4802,7	4800,7	150448,0	2553,6	99,7	4802,7	4800,7	150448,0	2553,6	99,7
wil50	48816	10000,0	48900,4	5560,4	99,8	9801,0	9801,0	48900,4	5560,4	99,8	10000,0	10000,0	48900,4	5560,4	99,8	10000,0	10000,0	48900,4	5560,4	99,8
média		8808,6	1183774,9	3541,8	99,2	6055,3	6054,3	1212662,2	2843,5	98,5	6980,1	6387,3	1183887,8	2885,7	99,0	6899,3	6539,1	1183884,7	2936,7	99,1

Tabela 6.7: GRASP para o problema quadrático de atribuição (programa `gqapd`) com $n_j^* \cong 100.000$.

		$n = 100.000$				L_1, $c = 100.000$, $n^* = 99.369$					L_3, $c = 400.000$, $n^* = 100.000$					L_4, $c = 300.000$, $n^* = 100.000$				
instância	$s*$	w	s	mi	q	n	w	s	mi	q	n	w	s	mi	q	n	w	s	mi	q
chr12a	9552	2080,2	9552,0	165,2	100,0	197,0	195,0	10132,2	106,3	93,9	25687,1	1782,8	9552,0	165,2	100,0	10334,3	1464,5	9552,0	165,2	100,0
chr15a	9896	29595,1	9896,0	1368,6	100,0	197,0	195,0	10132,2	106,3	97,6	85151,4	27443,6	9896,0	1368,6	100,0	56105,9	22216,0	9896,0	1368,6	100,0
chr20a	2192	99083,5	2196,8	30807,2	99,8	4708,8	4706,8	2265,8	2423,9	96,6	4708,8	4706,8	2265,8	2423,9	96,6	4708,8	4706,8	2265,8	2423,9	96,6
chr25a	3796	99964,5	3877,4	46697,0	97,9	24519,4	24517,4	3994,6	9526,3	94,8	24519,4	24517,4	3994,6	9526,3	94,8	24519,4	24517,4	3994,6	9526,3	94,8
els19	17212548	3174,0	17212548,0	233,7	100,0	25,9	23,9	17729900,2	9,5	97,0	28877,4	2276,7	17212548,0	233,7	100,0	10986,6	1610,3	17212548,0	233,7	100,0
esc16a	68	99780,0	68,0	4,1	100,0	11735,5	11733,5	68,0	4,1	100,0	11735,5	11733,5	68,0	4,1	100,0	11735,5	11733,5	68,0	4,1	100,0
esc32a	130	100000,0	130,2	38910,0	99,8	99369,0	99369,0	130,2	38910,0	99,8	100000,0	100000,0	130,2	38910,0	99,8	100000,0	100000,0	130,2	38910,0	99,8
kra30a	88900	99980,0	88900,0	4627,9	100,0	30194,3	30192,3	88900,0	4627,9	100,0	30194,3	30192,3	88900,0	4627,9	100,0	30194,3	30192,3	88900,0	4627,9	100,0
nug18	1930	54092,6	1930,0	585,3	100,0	293,3	291,3	1938,8	163,0	99,5	93897,4	51720,0	1930,0	585,3	100,0	81839,4	46852,4	1930,0	585,3	100,0
nug25	3744	94558,2	3744,0	1822,2	100,0	1344,4	1342,4	3745,4	694,1	100,0	1344,4	1342,4	3745,4	694,1	100,0	1353,2	1351,2	3745,4	694,1	100,0
nug30	6124	99689,6	6124,4	26381,2	100,0	9614,3	9612,3	6134,2	6240,5	99,8	9614,3	9612,3	6134,2	6240,5	99,8	9614,3	9612,3	6134,2	6240,5	99,8
rou20	725522	95260,5	725522,0	7180,2	100,0	1382,1	1380,1	727311,2	704,7	99,8	1382,1	1380,1	727311,2	704,7	99,8	1382,1	1380,1	727311,2	704,7	99,8
sko42	15812	99999,7	15832,8	45497,4	99,9	99369,0	99368,7	15832,8	45497,4	99,9	100000,0	99999,7	15832,8	45497,4	99,9	100000,0	99999,7	15832,8	45497,4	99,9
ste36a	9526	99943,2	9573,0	44456,5	99,5	17959,9	17957,9	9615,0	9310,2	99,1	17959,9	17957,9	9615,0	9310,2	99,1	17959,9	17957,9	9615,0	9310,2	99,1
tai25a	1167256	99990,0	1169454,2	35514,9	99,8	46184,7	46182,7	1171027,8	27664,1	99,7	46184,7	46182,7	1171027,8	27664,1	99,7	46184,7	46182,7	1171027,8	27664,1	99,7
tai30a	1818146	100000,0	1829918,6	40472,6	99,4	99369,0	99369,0	1829918,6	40472,6	99,4	100000,0	100000,0	1829918,6	40472,6	99,4	100000,0	100000,0	1829918,6	40472,6	99,4
tho30	149936	99443,9	149936,0	30053,3	100,0	4802,7	4800,7	150448,0	2553,6	99,7	4802,7	4800,7	150448,0	2553,6	99,7	4802,7	4800,7	150448,0	2553,6	99,7
wil50	48816	100000,0	48868,2	32736,2	99,9	99369,0	99369,0	48868,2	32736,2	99,9	100000,0	100000,0	48868,2	32736,2	99,9	100000,0	100000,0	48868,2	32736,2	99,9
média		82035,3	1182670,6	21528,5	99,8	30590,9	30589,3	1211686,8	12319,5	98,7	43670,0	35313,8	1182899,2	12428,8	99,4	39540,1	34698,8	1182899,2	12428,8	99,4

de quatro linhas mostra as médias das cinco heurísticas. A primeira coluna indica o limite superior de iterações n_j^*. As próximas duas colunas apresentam os resultados para heurística GRASP original com um número fixo de n iterações. Para cada critério de parada L_1, L_3 e L_4 existe um grupo com quatro colunas que apresentam os resultados para GRASP_BSR. Para todos os grupos, a coluna *w* mostra o número de diferentes ótimos locais, e a coluna *mi* reporta o número da iteração onde a melhor solução foi encontrada. Para os últimos três grupos, a coluna *n* representa o número de iterações executadas e a coluna *q* representa a qualidade da solução.

Considere o gráfico comparativo da Figura 6.2 dividido em 12 regiões. O eixo da abscissa (horizontal) indica o número n de iterações executadas e o eixo da ordenada (vertical) indica a qualidade q da solução encontrada até esta iteração. Em relação à qualidade da solução, soluções na área "superior" do gráfico são aquelas cuja qualidade encontra-se na faixa entre 99 e 100. Soluções cuja qualidade encontra-se abaixo de 97 estão localizadas na área "inferior" e soluções cuja qualidade está na faixa entre 97 e 99 estão localizadas na área "média". Em relação ao número de iterações executadas (tempo de processamento), soluções muito rápidas (com até 20% do tempo estipulado pelo limite superior de iteração) estão localizadas na área "extremo esquerda", soluções muito lentas (acima de 80% do tempo estipulado pelo limite superior de iteração) estão localizadas na área "extremo direita" e soluções intermediárias podem estar localizadas nas áreas "esquerda" (entre 20% e 50% do tempo estipulado pelo limite superior de iteração) ou "direita" (entre 50% e 80% do tempo estipulado pelo limite superior de iteração).

O cenário ideal é aquele onde as soluções estão localizadas na área "superior extremo esquerda", em oposição ao cenário menos favorável que é aquele onde as soluções estão localizadas na área "inferior extremo direita". Considera-se que as "melhores soluções" encontram-se nas quatro áreas mais sombreadas do gráfico (região "sombreada"), a saber: "superior extremo esquerda", "superior esquerda", "média extremo esquerda" e "média esquerda".

Os gráficos das Figuras 6.3 a 6.5 mostram um panorama comparativo para todas os critérios de parada aplicados às cinco heurísticas GRASP. Em cada gráfico existem 18

Tabela 6.8: Resumo com as médias dos resultados apresentados nas Tabelas de 6.2 a 6.7.

	$n = n^*$		L_1				L_3				L_4			
n^*	w	mi	n	w	mi	q	n	w	mi	q	n	w	mi	q
GRASP para o problema de conjunto independente máximo (programa `gmis`)														
1.000	250,1	25,2	34,7	13,7	3,3	97,2	677,0	242,6	25,2	100,0	457,2	198,6	25,2	100,0
10.000	822,1	221,9	82,3	13,9	3,3	97,1	3864,0	662,2	25,2	99,9	1870,7	469,8	25,2	99,9
100.000	2091,9	1509,2	231,7	13,9	3,3	97,0	17689,4	1378,3	1509,2	100,0	6286,1	894,5	221,9	99,8
médias	1054,7	585,4	116,2	13,8	3,3	97,1	7410,1	761,0	519,9	100,0	2871,3	521,0	90,8	99,9
GRASP para o problema de satisfabilidade máxima ponderada (programa `maxsat`)														
1.000	1000,0	449,8	938,0	938,0	444,8	100,0	1000,0	1000,0	449,8	100,0	1000,0	1000,0	449,8	100,0
10.000	9998,8	5008,7	8976,1	8975,7	4199,0	100,0	9135,3	9134,9	4199,0	100,0	9135,3	9134,9	4199,0	100,0
100.000	99945,4	51298,4	78376,8	78376,2	40142,2	99,9	78805,9	78805,2	40142,2	99,9	78805,9	78805,2	40142,2	99,9
médias	36981,4	18919,0	29430,3	29430,0	14928,7	100,0	29647,1	29646,7	14930,3	100,0	29647,1	29646,7	14930,3	100,0
GRASP para problema de planarização de grafos (programa `gmpsg`)														
1.000	1000,0	409,5	938,0	938,0	365,9	99,9	1000,0	1000,0	409,5	100,0	1000,0	1000,0	409,5	100,0
10.000	9995,2	3543,3	8360,2	8359,8	3347,3	100,0	8519,4	8519,0	3491,1	100,0	8519,4	8519,0	3491,1	100,0
100.000	99533,2	27472,7	79784,5	79784,0	30204,5	99,9	80276,7	80276,2	30204,5	99,9	80276,7	80276,2	30204,5	99,9
médias	36842,8	10475,2	29694,2	29693,9	11305,9	99,9	29932,0	29931,7	11368,4	100,0	29932,0	29931,7	11368,4	100,0
GRASP para o problema de recobrimento (programa `sc`)														
1.000	810,6	240,6	719,3	718,7	203,6	99,9	847,7	771,3	220,9	99,9	808,7	771,3	220,9	99,9
10.000	7987,0	1455,8	6102,7	6101,9	1260,3	99,7	6507,2	6229,2	1260,3	99,7	6327,6	6229,2	1260,3	99,7
100.000	78246,5	6500,8	59843,5	59842,7	6292,1	99,7	61151,9	60238,5	6305,2	99,7	60478,0	60249,6	6305,2	99,7
médias	29014,7	2732,4	22221,8	22221,1	2585,3	99,8	22835,6	22413,0	2595,5	99,8	22538,1	22416,7	2595,5	99,8
GRASP para o problema quadrático de atribuição (programa `gqapd`)														
1.000	939,6	434,6	757,4	756,9	361,2	97,3	899,4	848,9	410,2	97,8	883,8	844,7	411,9	97,8
10.000	8808,6	3541,8	6055,3	6054,3	2843,5	98,5	6980,1	6387,3	2885,7	99,0	6899,3	6539,1	2936,7	99,1
100.000	82035,3	21528,5	30590,9	30589,3	12319,5	98,7	43670,0	35313,8	12428,8	99,4	39540,1	34698,8	12428,8	99,4
médias	30594,5	8501,3	12467,9	12466,8	5174,7	98,2	17183,2	14183,3	5241,6	98,7	15774,6	14027,6	5258,8	98,8
Resumo das cinco heurísticas														
1.000	800,0	312,0	677,5	673,1	275,8	98,9	884,8	772,5	303,1	99,5	829,9	762,9	303,5	99,5
10.000	7522,3	2754,3	5915,3	5901,1	2330,7	99,1	7001,2	6186,5	2372,2	99,7	6550,5	6178,4	2382,5	99,7
100.000	72370,5	21661,9	49765,5	49721,2	17792,3	99,0	56318,8	51202,4	18118,0	99,8	53077,4	50984,8	17860,5	99,7
médias	26897,6	8242,7	18786,1	18765,1	6799,6	99,0	21401,6	19387,1	6931,1	99,7	20152,6	19308,7	6848,8	99,6

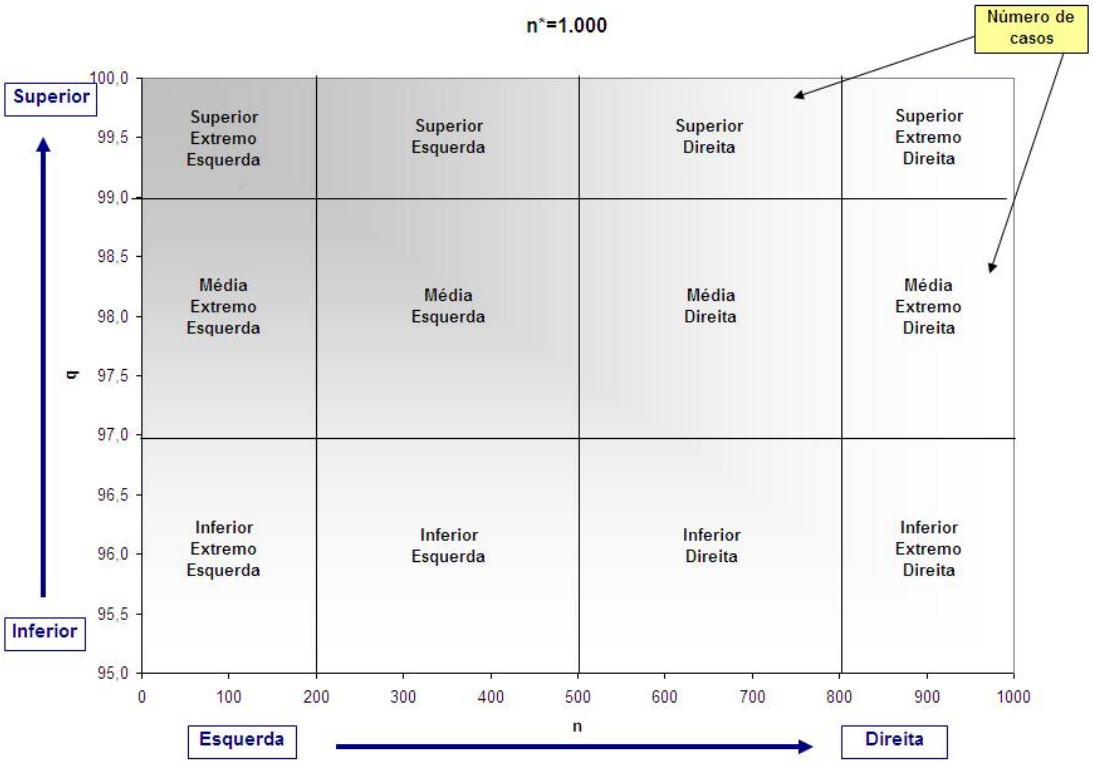

Figura 6.2: Gráfico comparativo dividido em 12 regiões.

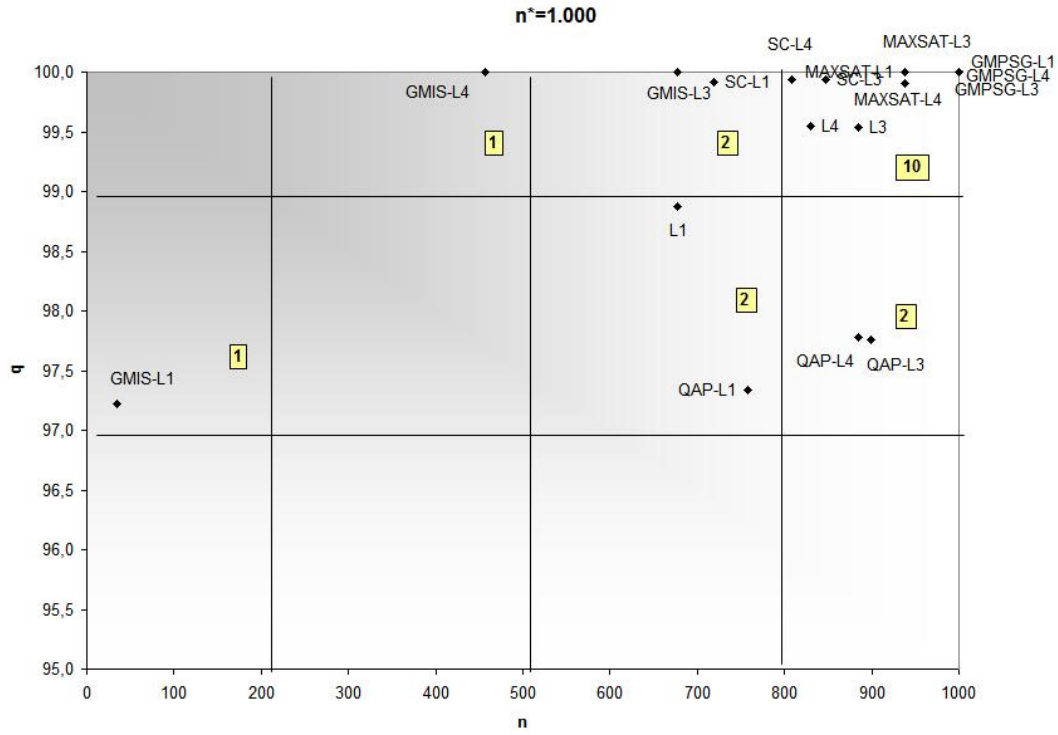

Figura 6.3: Comparação dos critérios de parada L_1, L_3 e L_4, para $n_j^* \cong 1.000$.

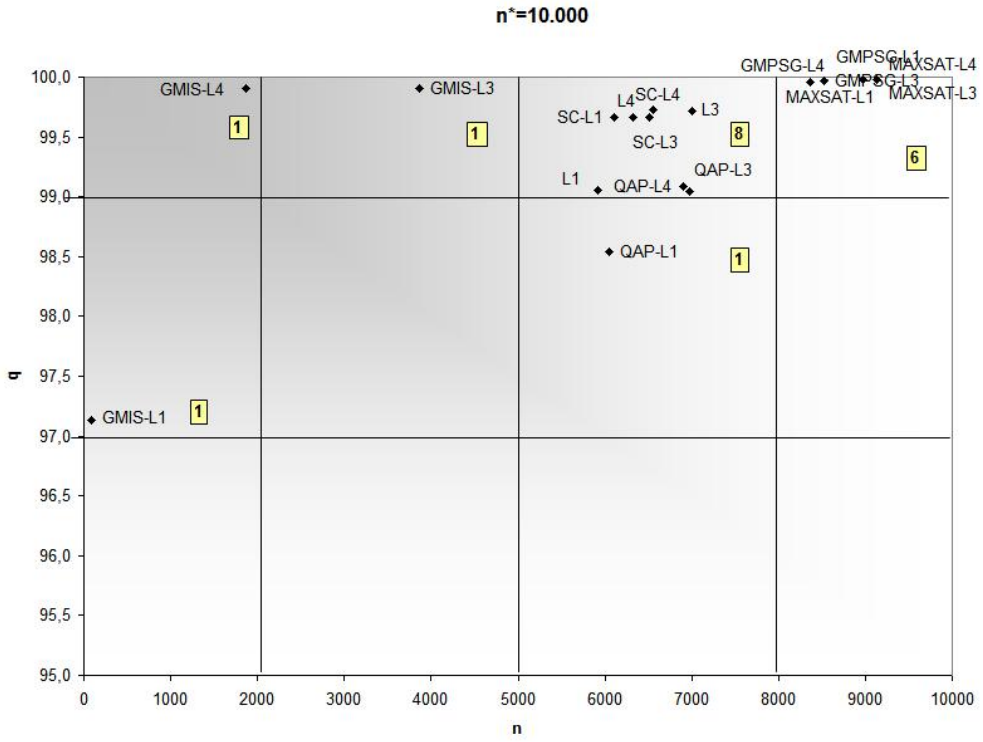

Figura 6.4: Comparação dos critérios de parada L_1, L_3 e L_4, para $n_j^* \cong 10.000$.

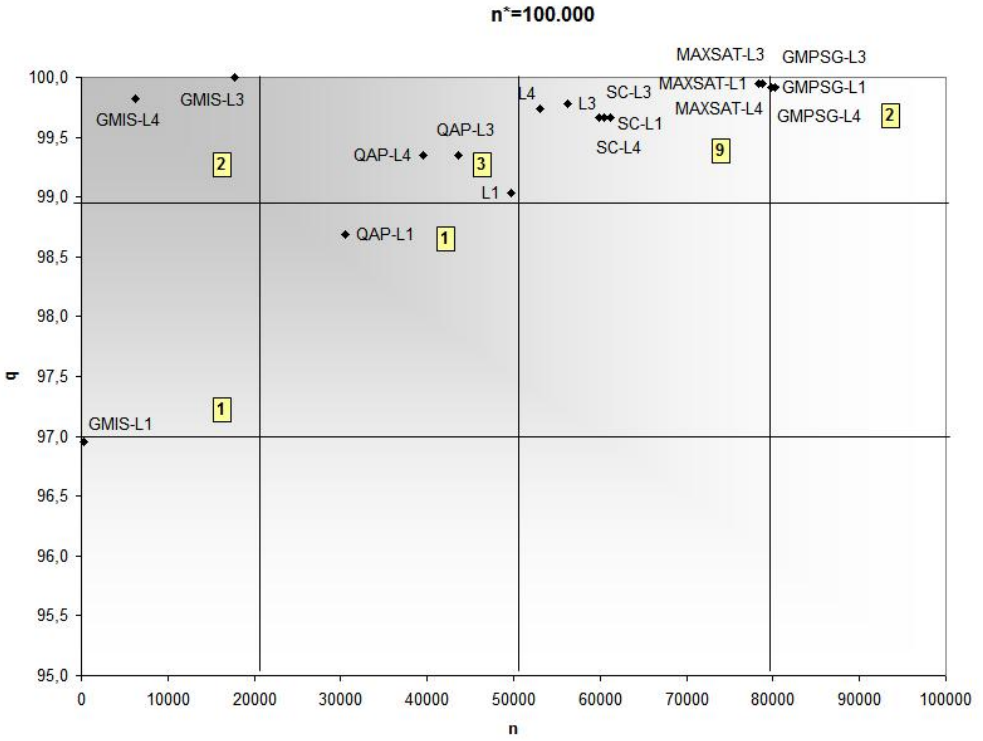

Figura 6.5: Comparação dos critérios de parada L_1, L_3 e L_4, para $n_j^* \cong 100.000$.

pontos, 15 deles representam a média dos resultados de cada combinação "heurística-parada" correspondente às primeiras 15 linhas da Tabela 6.8 das colunas $n_j^* \in \{1.000, 10.000, 100.000\}$ e os demais três pontos são relativos à média geral de cada critério de parada relativos às últimas três linhas das colunas $n_j^* \in \{1.000, 10.000, 100.000\}$ da Tabela 6.8.

Considerando-se o critério número de iterações executadas, percebe-se que a heurística identificada por `maxsat` possui nos três gráficos (Figuras 6.3 a 6.5) seis pontos concentrados na região "superior extremo direita" e três pontos na região "superior direita". Isto significa que a heurística executou um número alto de iterações. De fato, dos 45 casos individuais (mostrados nas Tabelas 6.2-6.4), em praticamente 36 deles o número n de iterações executadas é igual ao número w de ótimos locais encontrados e ao limite superior n_j^*. Isto indica que se o programa continuasse a execução, ele provavelmente continuaria encontrando novos ótimos locais diferentes. O mesmo acontece com a heurística mais lenta, a `gmpsg`, com oito pontos na área "superior extremo direita" e um ponto na "superior direita" A heurística mais rápida é a identificada por `gmis` (seis pontos na região "extremo esquerda"). A heurística `gqapd` executou em tempo intermediário, conforme n_j^* cresce ela fica relativamente mais rápida: três pontos na região "esquerda" para $n_j^* = 100.000$, três pontos na região "direita" para $n_j^* = 10.000$ e três casos nas regiões "direita" e "extremo direita" para $n_j^* = 1.000$. Em seguida vem a heurística `sc`, sete pontos na região "direita" e dois na "extremo direita".

Considerando-se a relação tempo de processamento e qualidade da solução, a heurística que apresentou o melhor comportamento é a identificada por `gmis`. Dos nove pontos, oito deles encontram-se na região "sombreada", sendo que três deles encontram-se na melhor região a "superior extremo esquerda". De fato, pode-se observar na Tabela 6.8 que, para o problema do conjunto independente máximo, a média da qualidade das soluções encontra-se acima de 97 e a média do número máximo de iterações em apenas um caso em nove casos (L_3 com $n_j^* = 1.000$) ultrapassou 50% do limite superior de iterações n_j^*. Em segundo lugar, encontra-se a heurística para o problema quadrático de atribuição (`gqapd`): quatro casos na região "superior" e cinco casos na região "média", sendo três

deles na região "sombreada". Todos os noves pontos de cada um dos problemas de satisfabilidade máxima ponderada (maxsat), de recobrimento (sc) e de planarização de grafos (gmpsg), encontram-se na área "superior", o que indica que as soluções são de qualidade, entretanto, o tempo de processamento é alto.

Ainda na Tabela 6.8, pode-se analisar o comportamento considerando cada um dos problemas individualmente. Para o problema de conjunto independente máximo (gmis), o critério L_1 é o mais rápido e o que produz soluções de pior qualidade. O critério L_4 é o melhor, produz soluções de qualidade equivalentes aos do critério L_3 e em menos tempo. Tanto o critério L_3 quanto o L_4 produzem soluções de qualidade equivalentes às produzidas pelo GRASP original com $n = n_j^*$ e com um número de iterações bem menor. Para este problema o uso dos critérios de parada mostrou-se eficiente. Os problemas de satisfabilidade máxima ponderada (maxsat) e de planarização de grafos (gmpsg) têm comportamento similar, em termos de qualidade de solução e de tempo de processamento: os três critérios são equivalentes. Em ambos os casos o programa mostrou potencial para melhorar a melhor solução encontrada caso executasse mais iterações (em muitos casos valor de n igual ao limite superior n_j^*). Por este motivo, o uso de critérios de parada também é válido pois ele reconhece que a busca não deve parar. O problema de recobrimento (sc) também é beneficiado pelo uso de critérios de parada. A qualidade média dos três critérios são equivalentes e altas ($99,8\%$) e o número médio de iterações é em torno de 60% do limite superior. De forma geral, para o problema quadrático de atribuição (gqapd) novamente os três critérios são equivalentes em termos de qualidade de solução. A ordem de eficiência em relação ao tempo é L_1, L_4 e L_3, com pequenas diferenças entre elas.

Os gráficos das Figuras 6.6 a 6.8 permitem fazer uma análise detalhada do comportamento de todas as instâncias do problema quadrático de atribuição para todos os critérios de parada. Esta verificação detalhada torna-se interessante pelo fato de a qualidade ser medida em relação a solução ótima (conhecida para estas instâncias através do QAPLIB). Em cada gráfico existem 57 pontos, sendo 54 deles relativos aos três critérios de parada de cada uma das 18 instâncias e 3 pontos relativos às médias de cada critério de

parada. Na Figura 6.6 com $n_j^* = 1.000$, dos 54 casos das instâncias, 46 deles (85,2%) encontram-se nas regiões "direita" e "extremo direita". Em oposição, na Figura 6.8 com $n_j^* = 100.000$, dos 54 casos, 38 (70,4%) encontram-se nas regiões "esquerda" e "extremo esquerda". Pode-se dizer que para valores pequenos de n_j^* o limite do número máximo de iterações não é suficinte para se encontrar soluções de qualidade (28 soluções, 51,8%, na área "superior" com $n_j^* = 1.000$), pois o programa tem potencial de melhorar a qualidade das soluções caso fosse permitido executar mais iterações (o critério mais demorado usou 89,9% do tempo máximo disponível). De forma contrária, para valores altos de n_j^* o limite do número máximo de iterações é suficiente para o programa parar com soluções de qualidade (45 soluções, 83,3%, na área "superior" com $n_j^* = 100.000$) e em menos tempo do que gastaria caso executasse um número de iterações igual ao limite superior (o critério mais demorado usou em média 43,7% do tempo máximo disponível). Por outro lado, com valores intermediários de n_j^* o programa pára com soluções de qualidade razoável (32 soluções, 59,2%, na área "superior" com $n_j^* = 10.000$) e em tempo razoável (o critério mais demorado usou em média 69,8% do tempo máximo disponível).

Observando-se o posicionamento da média geral dos critérios de parada L_1, L_3 e L_4 nos gráficos das Figuras 6.3 a 6.5, percebe-se que de maneira geral o critério de parada L_1 é mais rápido e produz soluções de qualidade inferior às demais (embora ainda seja muito boa). Enquanto que os critérios de parada L_3 e L_4 produzem soluções de qualidade similares, porém com uma pequena vantagem de tempo para o L_4 (um pouco mais rápido do que L_3). Esta ordem de parada está ligada diretamente a região de continuidade, que é controlada pelo parâmetro c. Neste experimento, para manter o limite superior de iteração n_j^* igual entre os critérios de parada, o parâmetro c variou de critério para critério, conforme Tabela 6.1. Desta forma, o critério de parada L_1, com o parâmetro c de menor valor, pára mais rápido; o critério de parada L_3, com o parâmetro c de maior valor, demora mais; e o critério de parada L_4, com o parâmetro c de valor intermediário, pára em tempo intermediário.

A seguir, será abordado o aspecto do custo adicional de se contabilizar os diferentes ótimos locais w embutido no *framework* `GRASP_BSR`. Considerando que se tem disponível

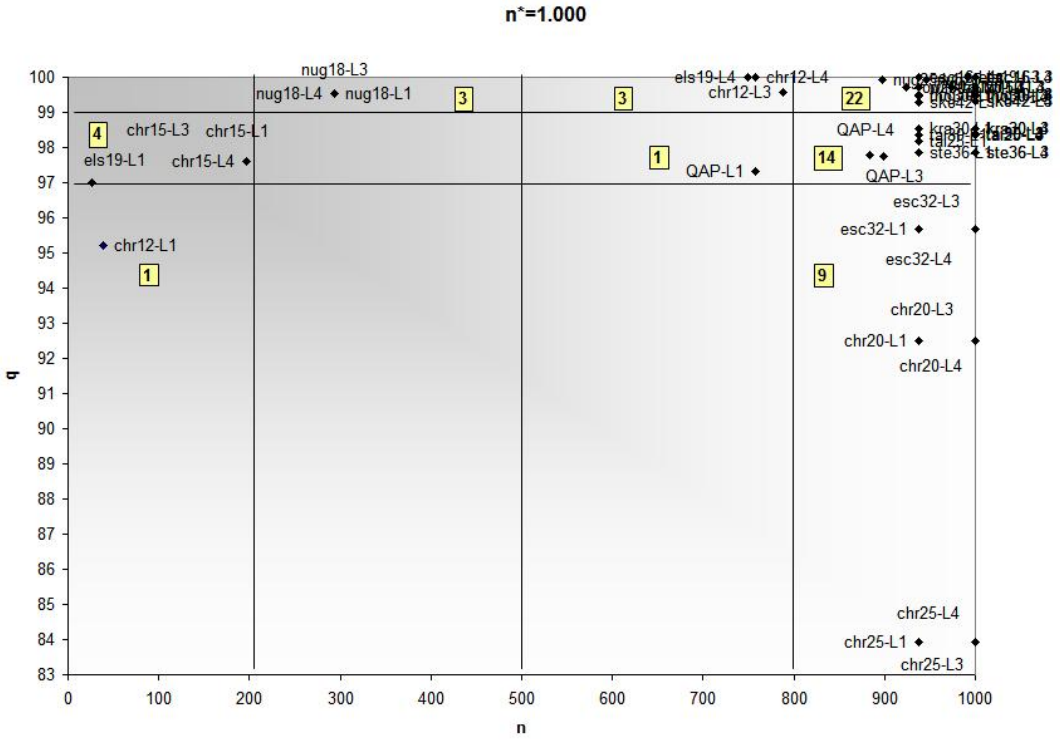

Figura 6.6: Comparação dos critérios de parada L_1, L_3 e L_4 com todas as instâncias do problema quadrático de atribuição, para $n_j^* \cong 1.000$.

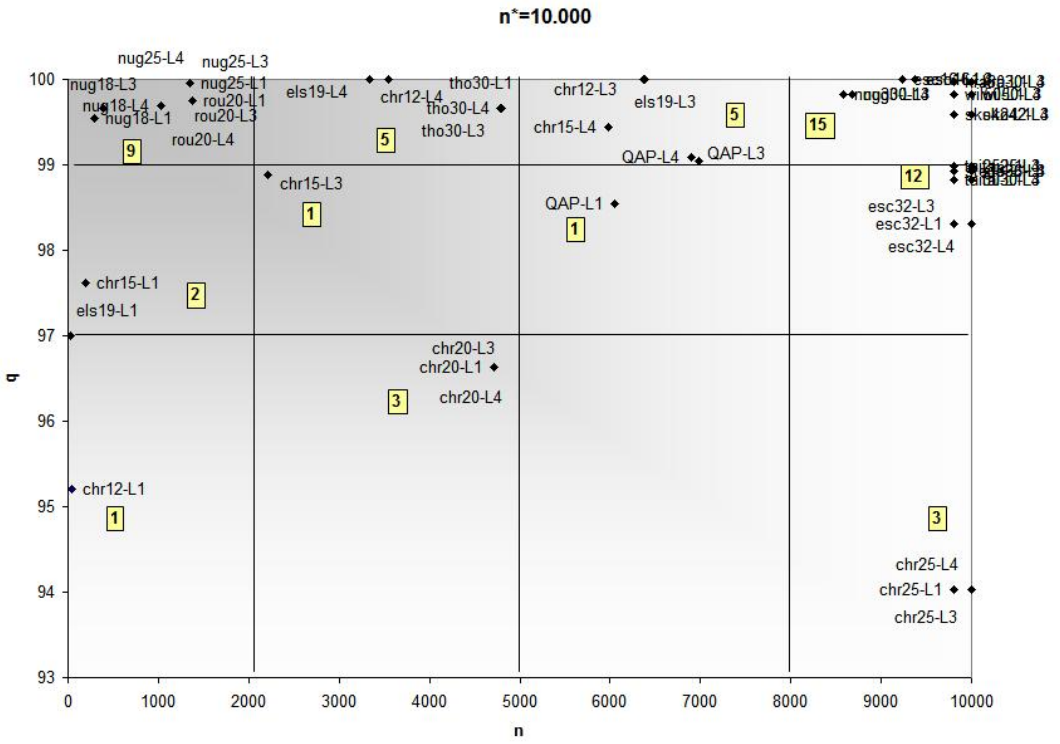

Figura 6.7: Comparação dos critérios de parada L_1, L_3 e L_4 com todas as instâncias do problema quadrático de atribuição, para $n_j^* \cong 10.000$.

n_j^* iterações para realizar, comparando-se a versão original das heurísticas com o *framework*, o pior caso acontece quando o *framework* atinge este limite. Na prática, entretanto, este limite nem sempre é atingido. O número de iterações realizadas n varia muito em função da dificuldade do problema e da instância. Isso pode ser verificado nos experimentos. Por exemplo, na Tabela 6.4, para um mesmo problema, o de recobrimento, a instância `data.9` parou com o número de iterações n igual a $0,017\%$ de n_j^* e 100% de qualidade da solução e outra instância (`data.243`) parou com o número de iterações n igual a $99,4\%$ de n_j^* e a mesma qualidade de solução. Este comportamento, em que a instância `data.9` requer menos tempo computacional do que a instância `data.243`, se mantém independente do critério de parada. A instância `data.9` requer $0,5\%$ e $0,1\%$ de n_j^* para L_3 e L_4 respectivamente, enquanto que `data.243` requer 100%. Outro exemplo de instâncias com níveis de dificuldade diferentes é demonstrado no problema de planarização de grafos. A instância `g10` é mais fácil do que as demais: pára com $0,3\%$ de n_j^* para qualquer critério de parada (Tabela 6.4) produzindo solução com $99,9\%$ de qualidade, enquanto que as demais executam iterações até o limite superior. Entretanto, já para as instâncias do problema de conjunto independente máximo, a relação entre o número de iterações realizadas é mais significativa com o critério de parada específico do que com a dificuldade da instância (o número de iterações varia muito de critério para critério). Apesar deste fato, pode-se dizer que o número de diferentes ótimos locais encontrados, que é uma informação relacionada à dificuldade da instância, interfere diretamente na definição do critério de parada. Por exemplo, o número de diferentes ótimos locais da instância $G_{100,0.5}$ é pequeno ($w = 3$) e a ordem dos critérios do mais rápido para o mais lento é: L_4, L_1 e L_3. Para as demais instâncias ($w > 9$) a ordem dos critérios foi L_1, L_4 e L_3. No problema quadrático de atribuição a maioria das instâncias (13 de um total de 18) param com o mesmo número de iterações independente do critério utilizado. De forma geral, pode-se concluir que para a maioria dos casos estudados, a instância determina o número de iterações executadas e a qualidade alcançada na solução encontrada.

No primeiro experimento os resultados obtidos para os diferentes critérios foram comparados considerando-se um mesmo valor de n_j^*. No próximo experimento, os resul-

tados serão comparados considerando-se o mesmo valor do parâmetro c para todos os critérios de parada. Trabalhou-se com $c = 1.000$ conforme sugerido por Boender e Rinnooy Kan [1]. Sabe-se que o limite superior de iterações n^* dos critérios de parada L_1, L_3 e L_4, respectivamente, iguais a 938, 250 e 333, são inferiores ao número fixo de iterações determinado de $n = 1.000$ e portanto a melhor solução conhecida no cálculo da qualidade será a solução obtida pela execução com $n = 1.000$ iterações.

A Tabela 6.9 mostra os resultados detalhados do experimento para os problemas de conjunto independente máximo, de satisfabilidade máxima ponderada, de planarização de grafos e de recobrimento e é estruturada da mesma forma que as Tabelas 6.2, 6.3 e 6.4. A Tabela 6.10 mostra os resultados para o problema quadrático de atribuição e estrutura igual às Tabelas 6.5, 6.6 e 6.7. Neste experimento, foram realizadas 1.520 execuções.

Nota-se, na Tabela 6.9, que para a heurística `gmis` o critério de parada L_1 é o mais rápido ($3,5\%$ do número de iterações de um GRASP com o número de iterações fixas), produzindo soluções com 97% de qualidade. O critério L_3 encontra as melhores soluções (100% de qualidade), assim como o L_4, porém com um tempo médio menor (16% do número de iterações fixas). O `gmis` é o programa em que os critérios de parada obtiveram melhor desempenho, conforme o gráfico da Figura 6.9, onde aparece com dois de três critérios na região "superior extremo esquerda".

Como avaliado no primeiro experimento, as heurísticas `maxsat`, `gmpsg` e `sc` têm comportamento similar. Todas param com $n = w = n_j^*$. Por isso, o tempo de processamento e a qualidade da solução são definidos diretamente pelo limite superior de iteração n_j^*: quanto maior o limite, maior o tempo e a qualidade.

Na Tabela 6.10, observa-se que para o problema quadrático de atribuição, `gqapd`, o critério L_3 é o mais rápido, podendo executar cerca de 23% do tempo do GRASP original e obter uma solução, em média, aproximadamente 95% boa. Para os critérios L_1 e L_4 é possível obter soluções em média 97% boas utilizando-se, respectivamente, 76% e 30% do tempo do GRASP original.

No gráfico da Figura 6.9 percebe-se que de forma geral o critério de parada L_1 demora mais porém produz soluções de melhor qualidade. Caso reduzir tempo seja impres-

cindível, L_3 e L_4 produzem soluções cerca de 97% boas com, respectivamente, 23% e 29% do tempo do GRASP original.

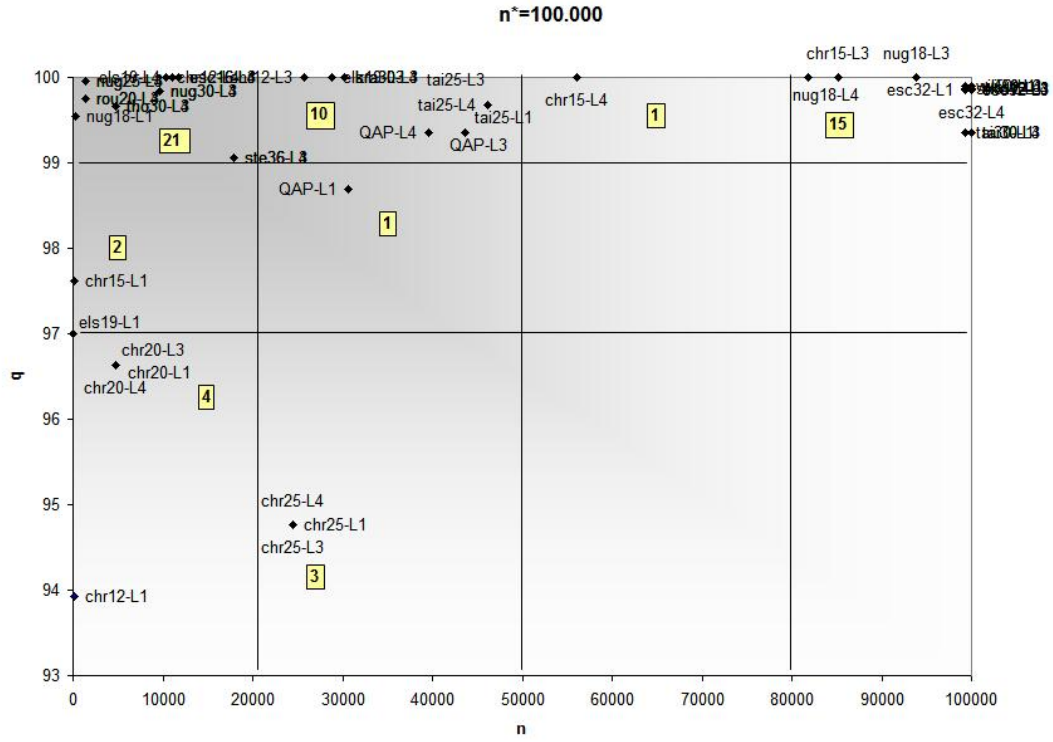

Figura 6.8: Comparação dos critérios de parada L_1, L_3 e L_4 com todas as instâncias do problema quadrático de atribuição, para $n_j^* \cong 100.000$.

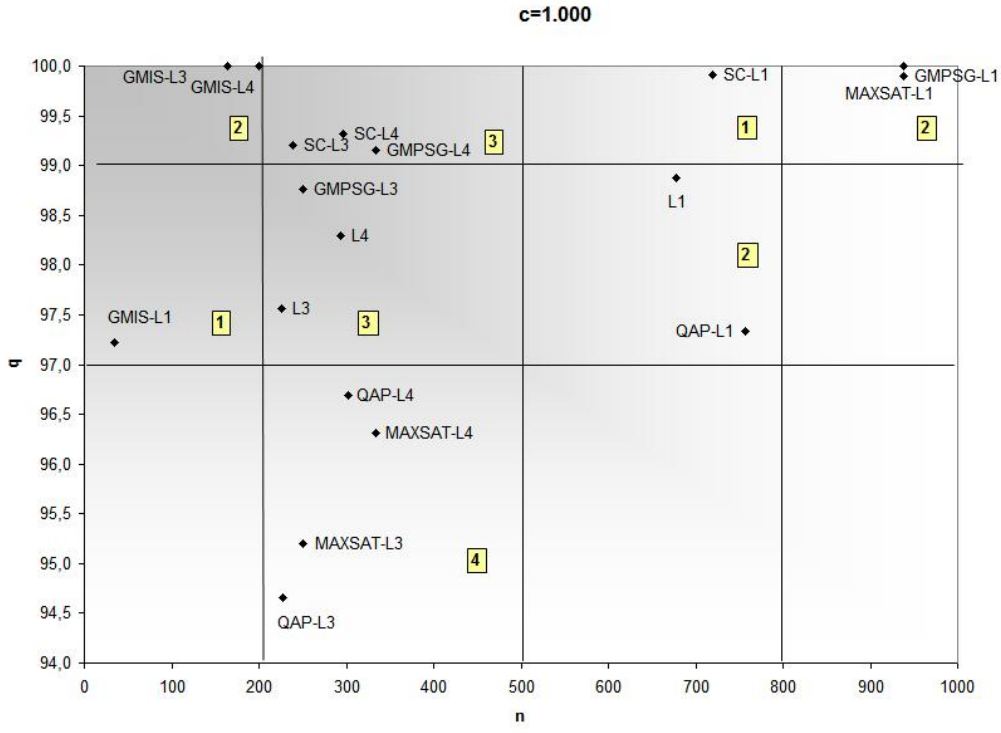

Figura 6.9: Comparação dos critérios de parada L_1, L_3 e L_4, para $c = 1.000$.

Tabela 6.9: GRASP com número fixo de iterações comparado com GRASP_BSR para L_1, L_3 e L_4 com $c = 1.000$.

	$n = 1.000$			L_1, $c = 1.000$, $n^* = 938$					L_3, $c = 1.000$, $n^* = 250$					L_4, $c = 1.000$, $n^* = 333$				
instância	*w*	*s*	*mi*	*n*	*w*	*s*	*mi*	*q*	*n*	*w*	*s*	*mi*	*q*	*n*	*w*	*s*	*mi*	*q*
GRASP para o problema de conjunto independente máximo (programa gmis)																		
$G_{100,0.2}$	285,9	20,0	74,1	17,3	15,3	19,0	1,6	95,0	249,0	125,8	20,0	74,1	100,0	252,3	127,3	20,0	74,1	100,0
$G_{100,0.5}$	3,0	8,0	1,0	100,0	3,0	8,0	1,0	100,0	54,0	3,0	8,0	1,0	100,0	28,0	3,0	8,0	1,0	100,0
$G_{150,0.2}$	482,8	22,0	23,4	25,3	23,3	21,6	5,6	98,2	163,7	117,9	22,0	23,4	100,0	294,1	193,8	22,0	23,4	100,0
$G_{150,0.5}$	66,8	10,0	10,5	11,2	9,2	9,6	2,7	96,0	187,6	46,8	10,0	10,5	100,0	133,6	41,0	10,0	10,5	100,0
$G_{500,0.5}$	411,8	13,0	17,0	19,9	17,9	12,6	5,7	96,9	166,3	111,7	13,0	17,0	100,0	291,1	180,3	13,0	17,0	100,0
média	250,1		25,2	34,7	13,7		3,3	97,2	164,1	81,0		25,2	100,0	199,8	109,1		25,2	100,0
GRASP para o problema de satisfabilidade máxima ponderada (programa maxsat)																		
jnh8.sat	1000,0	776987,0	553,1	938,0	938,0	776987,0	553,1	100,0	250,0	250,0	725790,5	121,7	93,4	333,0	333,0	747405,8	226,9	96,2
jnh16.sat	1000,0	729918,2	445,1	938,0	938,0	729918,2	445,1	100,0	250,0	250,0	668587,1	138,6	91,6	333,0	333,0	670868,1	147,8	91,9
jnh201.sat	999,9	461326,3	435,1	938,0	937,9	461264,2	410,1	100,0	250,0	250,0	460899,0	83,8	99,9	333,0	333,0	460940,4	111,7	99,9
jnh218.sat	1000,0	435146,7	274,3	938,0	938,0	435146,7	274,3	100,0	250,0	250,0	432570,0	120,2	99,4	333,0	333,0	433760,1	170,2	99,7
jnh307.sat	1000,0	856669,7	541,4	938,0	938,0	856669,7	541,4	100,0	250,0	250,0	785540,1	123,1	91,7	333,0	333,0	803782,7	177,2	93,8
média	1000,0		449,8	938,0	938,0		444,8	100,0	250,0	250,0		117,5	95,2	333,0	333,0		166,8	96,3
GRASP para problema de planarização de grafos (programa gmpsg)																		
g10	999,9	68,6	472,0	938,0	937,9	68,6	472,0	100,0	250,0	250,0	67,1	112,1	97,8	333,0	333,0	67,8	166,0	98,8
rg50.1	1000,0	87,9	366,3	938,0	938,0	87,9	366,3	100,0	250,0	250,0	87,1	128,9	99,1	333,0	333,0	87,2	157,4	99,2
rg200.1	1000,0	275,7	375,8	938,0	938,0	275,7	375,8	100,0	250,0	250,0	273,6	115,3	99,2	333,0	333,0	274,1	166,3	99,4
tg100.1	1000,0	242,0	478,9	938,0	938,0	240,9	308,9	99,5	250,0	250,0	238,3	116,5	98,5	333,0	333,0	239,1	130,1	98,8
tg200.1	1000,0	448,7	354,6	938,0	938,0	448,6	306,7	100,0	250,0	250,0	445,0	131,9	99,2	333,0	333,0	446,4	168,1	99,5
média	1000,0		409,5	938,0	938,0		365,9	99,9	250,0	250,0		120,9	98,8	333,0	333,0		157,6	99,1
GRASP para o problema de recobrimento (programa sc)																		
data.9	54,0	5,0	1,0	16,3	14,3	5,0	1,0	100,0	197,6	53,6	5,0	1,0	100,0	154,3	52,2	5,0	1,0	100,0
data.45	998,9	30,5	262,6	766,4	765,3	30,6	164,0	99,7	250,0	249,8	30,9	18,0	98,7	331,9	331,6	30,8	47,9	99,0
data.81	1000,0	61,4	400,3	938,0	938,0	61,4	400,3	100,0	250,0	250,0	62,7	21,5	97,9	333,0	333,0	62,7	21,5	97,9
data.135	1000,0	105,6	218,9	938,0	938,0	105,7	132,7	99,9	250,0	250,0	105,8	105,4	99,8	333,0	333,0	105,7	132,7	99,9
data.243	1000,0	204,6	320,2	938,0	938,0	204,6	320,2	100,0	250,0	250,0	205,3	95,6	99,7	333,0	333,0	205,1	144,2	99,8
média	810,6		240,6	719,3	718,7		203,6	99,9	239,5	210,7		48,3	99,2	297,0	276,6		69,5	99,3

Tabela 6.10: GRASP para o problema quadrático de atribuição (programa gqapd) com $c = 1.000$.

		$n = 1.000$				L_1, $c = 1.000$, $n^* = 938$					L_3, $c = 1.000$, $n^* = 250$					L_4, $c = 1.000$, $n^* = 333$				
instância	$s*$	w	s	mi	q	n	w	s	mi	q	n	w	s	mi	q	n	w	s	mi	q
chr12a	9552	531,0	9552,0	182,2	100,0	40,6	38,6	10061,1	16,9	94,7	44	41,8	9992,2	20,7	95,4	97,2	83,1	9860,4	39,2	96,8
chr15a	9896	954,8	9924,8	309,8	99,7	197,0	195,0	10132,2	106,3	97,6	186,4	184,6	10132,2	106,3	97,6	197,0	195,0	10132,2	106,3	97,6
chr20a	2192	999,9	2356,6	499,0	92,5	938,0	937,9	2356,6	499,0	92,5	250,0	250,0	2476,4	122,8	87,0	333,0	333,0	2460,2	161,0	87,8
chr25a	3796	1000,0	4405,8	609,6	83,9	938,0	938,0	4405,8	609,6	83,9	250,0	250,0	4613,0	104,8	78,5	333,0	333,0	4602,2	183,3	78,8
els19	17212548	454,6	17212548,0	233,7	100,0	25,9	23,9	17729900,2	9,5	97,0	141,9	99,8	17528035,4	40,4	98,2	217,5	141,1	17460846,8	53,0	98,6
esc16a	68	1000,0	68,0	4,1	100,0	938,0	938,0	68,0	4,1	100,0	250,0	250,0	68,0	4,1	100,0	333,0	333,0	68,0	4,1	100,0
esc32a	130	1000,0	135,6	594,9	95,7	938,0	938,0	135,6	594,9	95,7	250,0	250,0	139,0	131,4	93,1	333,0	333,0	138,4	153,5	93,5
kra30a	88900	1000,0	90197,0	400,3	98,5	938,0	938,0	90197,0	400,3	98,5	250,0	250,0	90751,0	150,3	97,9	333,0	333,0	90485,0	210,5	98,2
nug18	1930	975,2	1932,0	362,4	99,9	293,3	291,3	1938,8	163,0	99,5	213,6	212,8	1940,0	78,5	99,5	257,8	256,5	1939,0	132,2	99,5
nug25	3744	998,8	3746,2	465,7	99,9	897,3	896,3	3746,8	386,6	99,9	250,0	250,0	3751,0	127,1	99,8	333,0	332,9	3750,0	172,3	99,8
nug30	6124	1000,0	6155,4	402,4	99,5	938,0	938,0	6155,4	402,4	99,5	250,0	250,0	6182,4	131,5	99,0	333,0	333,0	6173,0	158,0	99,2
rou20	725522	999,1	727457,4	644,0	99,7	923,8	922,9	727603,0	552,6	99,7	250,0	249,9	730389,6	126,7	99,3	333,0	332,8	729807,8	136,2	99,4
sko42	15812	1000,0	15921,6	483,0	99,3	938,0	938,0	15922,4	389,2	99,3	250,0	250,0	15944,2	120,9	99,2	333,0	333,0	15934,2	168,0	99,2
ste36a	9526	1000,0	9730,2	586,2	97,9	938,0	938,0	9731,6	501,3	97,8	250,0	250,0	9801,4	94,2	97,1	333,0	333,0	9785,8	147,3	97,3
tai25a	1167256	1000,0	1186543,0	424,7	98,3	938,0	938,0	1188718,4	290,3	98,2	250,0	250,0	1191793,8	94,7	97,9	333,0	333,0	1190592,2	128,4	98,0
tai30a	1818146	1000,0	1847617,4	497,3	98,4	938,0	938,0	1847896,6	451,8	98,4	250,0	250,0	1191793,8	94,7	65,5	333,0	333,0	1855483,4	164,3	97,9
tho30	149936	1000,0	150764,4	523,1	99,4	938,0	938,0	150764,4	523,1	99,4	250,0	250,0	151128,6	106,7	99,2	333,0	333,0	151010,8	154,1	99,3
wil50	48816	1000,0	48947,8	601,1	99,7	938,0	938,0	48947,8	601,1	99,7	250,0	250,0	49045,2	128,3	99,5	333,0	333,0	49019,8	227,0	99,6
média		939,6		434,6	97,9	757,4	756,9		361,2	97,3	227,0	224,4		99,1	94,7	301,8	296,5		138,8	96,7

7 Conclusão

O principal objetivo deste trabalho foi apresentar, implementar e testar o *framework* GRASP_BSR usado para incorporar critérios de parada baseados em estatística bayesiana em heurísticas baseadas na metaheurística GRASP. Para avaliar a eficiência do GRASP_BSR, foram feitos estudos comparativos entre os programas sob o *framework* GRASP_RPB e os programas originais com parada determinada por um número fixo de iterações. Foram avaliados três critérios de parada apresentados por Boender e Rinnooy Kan (detalhes na Seção 4.3) mais o critério de parada baseado em um número fixo de iterações em cinco heurísticas baseadas na metaheurística GRASP (detalhes no Capítulo 5). Além dos parâmetros específicos de cada heurística, o *framework* possui dois parâmetros: o critério de parada identificado por L_j e o parâmetro de controle c usado no critério de parada para calcular o limite superior de iterações n_j^*. Para quatro heurísticas e para cada um dos três critérios de parada foram avaliadas cinco instâncias em 10 execuções. Para a heurística do problema quadrático de atribuição foram avaliadas dezoito instâncias com os três critérios de parada, em 10 execuções. Como resultado, o *framework* fornece, além da solução, o número de iterações executadas *n* (tempo de processamento), o número de ótimos locais visitados *w* e o número da iteração *mi* onde foi encontrada a melhor solução. Três medidas de qualidade foram consideradas: o tempo de processamento, a qualidade da solução medida como a diferença relativa entre a qualidade da solução obtida usando o *framework* e a qualidade da solução usando a melhor solução conhecida, e a medida gráfica que relaciona tempo de processamento e qualidade (tempo-qualidade). Resultados computacionais mostraram que, para um conjunto de 6.080 experimentos, o uso de

critérios de parada bayesiano é uma boa estratégia. Em particular, os critérios de parada L_3 e L_4 oferecem a melhor relação "tempo-qualidade".

Uma desvantagem do uso dos critérios de parada estudadas nesta dissertação é o custo adicional de $O(n^*m)$, onde m é o tamanho do problema e n_j^* o limite superior do número de iterações, que o *framework* agrega ao custo de uma iteração da heurística original. Por outro lado, do ponto de vista da robustez do algoritmo, o uso dos critérios se mostra muito interessante pois o critério de parada, por ser dinâmico, faz com que o número efetivo n de iterações realizadas varie em função da dificuldade do problema e da instância, o que pôde ser verificado nos experimentos.

Assim, conclui-se que as duas principais contribuições desta dissertação foram: implementação e teste do *framework* `GRASP_BSR` e a avaliação de que o uso de critérios de parada baseados em teoria bayesiana produz algoritmos mais robustos quando comparados à suas respectivas versões originais.

Como proposta de trabalho futuro pretende-se fazer uma análise experimental do custo adicional que o *framework* embute na heurística original. Além disso, pretende-se estudar a extensão do critério de Boender e Rinnooy Kan proposto em [19], que considera também o valor da função objetivo dos ótimos locais como informação de entrada na definição do critério de parada, em heurísticas baseadas na metaheurística GRASP. Outra proposta de estudo seria incorporar o parâmetro c como mais uma variável no modelo estatístico, a fim de deixar a própria heurística encontrar o melhor valor para o parâmetro. Por último, deseja-se estudar critérios de parada bayesianos aplicados a outros métodos com múltiplas inicializações, como algoritmos genéticos.

Referências Bibliográficas

[1] BOENDER, C., RINNOOY KAN, A., "Bayesian Stopping Rules for Multistart Global Optimization Methods", *Mathematical Programming*, v. 37, pp. 59–80, 1987.

[2] HOOS, H., STÜTZLE, T., *Stochastic Local Search - Foundations and Applications*. 2 ed. San Francisco, Elsevier, 2005.

[3] RARDIN, R., UZSOY, R., "Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial", *Journal of Heuristics*, v. 7, pp. 261–304, 2001.

[4] FEO, T.A., RESENDE, M.G.C., "A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem", *Operations Research Letters*, v. 8, pp. 67–71, 1989.

[5] FEO, T.A., RESENDE, M.G.C., "Greedy Randomized Adaptive Search Procedures", *Journal of Global Optimization*, v. 6, pp. 109–133, 1995.

[6] RESENDE, M.G.C., RIBEIRO, C.C., "Greedy Randomized Adaptive Search Procedures". In: Kochenberger, G., Glover, F. (eds), *Handbook of Metaheuristics*, pp. 219–249, Boston, Kluwer, 2003.

[7] RESENDE, M.G.C., RIBEIRO, C.C., "Greedy Randomized Adaptive Search Procedures - Advances and Applications". In: Potvin, J.Y., Gendreau, M. (eds), *Handbook of Metaheuristics*, Springer, 2009.

[8] AIEX, R.M., *Uma Investigação Experimental da Distribuição de Probabilidade do Tempo de Solução em Heurísticas GRASP e sua Aplicação na Análise de*

Implementações Paralelas. Tese de D.Sc, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, 2002.

[9] RESENDE, M.G.C., PITSOULIS, L.S., PARDALOS, P.M., "Fortran Subroutines for Computing Approximate Solutions of Weighted MAX-SAT Problems Using GRASP", *Discrete Applied Mathematics*, v. 100, pp. 95–113, 1999.

[10] PRAIS, M., RIBEIRO, C.C., "Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment Informs", *Journal on Computing*, v. 12, pp. 164–176, 2000.

[11] ORSENIGO, C., VERCELLIS, C., "Bayesian Stopping Rules for Greedy Randomized Procedure", *Journal of Global Optimization*, v. 36, pp. 365–377, 2006.

[12] HART, W., "Sequential Stopping Rules for Random Optimization Methods with Applications to Multistart Local Search", *SIAM Journal of Optimization*, v. 9, pp. 270–290, 1998.

[13] AIEX, R.M., RESENDE, M.G.C., RIBEIRO, C.C., "Probability Distribution of Solution Time in GRASP: An Experimental Investigation", *Journal of Heuristics*, v. 8, pp. 343–373, 2002.

[14] BARTKUTÉ, V., FELINSKAS, G., SAKALAUSKAS, L., "Optimality Testing in Stochastic and Heuristic Algorithms", *Technological and Economic Development of Economy*, v. 12, pp. 4–10, 2006.

[15] BARTKUTÉ, V., FELINSKAS, G., SAKALAUSKAS, L., "Application of Stochastic Approximation in Technical Design". In: Bogle, I.D.L., Zilinskas, J. (eds), *Computer Aided Methods in Optimal Design and Operations*, pp. 29–38, Singapore, World Scientific, 2006.

[16] BARTKUTÉ, V., SAKALAUSKAS, L., "Simultaneous Perturbation Stochastic Approximation of Nonsmooth Functions", *European Journal of Operational Research*, v. 181, pp. 1174–1188, 2007.

[17] GRIGAITIS, D., BARTKUTÉ, V., SAKALAUSKAS, L., "An Optimization of System for Automatic Recognition of Ischemic Stroke Areas in Computed Tomography Images", *Informatica*, v. 18, pp. 603–614, 2007.

[18] BARTKUTÉ, V., SAKALAUSKAS, L., "Statistical Inferences for Termination of Markov Type Random Search Algorithms", *Journal of Optimization Theory and Applications*, publicado on-line em http://dx.doi.org/10.1007/s10957-008-9502-3, 2009.

[19] BOENDER, C., RINNOOY KAN, A., "On When to Stop Sampling for the Maximum", *Journal of Global Optimization*, v. 1, pp. 331–340, 1991.

[20] DE GROOT, M., *Optimal Statistical Decisions*. 1 ed. New York, McGraw-Hill, 1970.

[21] FEO, T.A., RESENDE, M.G.C., SMITH, S.H., "A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set", *Operations Research*, v. 42, pp. 860–878, 1994.

[22] RESENDE, M.G.C., PARDALOS, P.M., LI, Y., "Algorithm 754: Fortran Subroutines for Approximate Solution of Dense Quadratic Assignment Problems Using GRASP", *ACM Transactions on Mathematical Software*, v. 22, pp. 104–118, 1996.

[23] RIBEIRO, C.C., RESENDE, M.G.C., "Algorithm 797: Fortran Subroutines for Approximate Solution of Graph Planarization Problems Using GRASP", *ACM Transactions on Mathematical*, v. 25, pp. 341–352, 1999.

[24] RESENDE, M.G.C., FEO, T.A., SMITH, S.H., "Algorithm 787: Fortran Subroutines for Approximate Solution of Maximum Independent Set Problems Using GRASP", *ACM Transactions on Mathematical Software*, v. 24, pp. 386–394, 1998.

[25] BURKARD, R., KARISCH, S., RENDL, F., "QAPLIB: A Quadratic Assignment Problem Library", *European Journal of Operations Research*, v. 55, pp. 115–119, 1991.

[26] SCHRAGE, L., "A More Portable Fortran Random Number Generator", *ACM Transactions on Mathematical Software*, v. 5, pp. 132–138, 1979.

[27] JOHNSON, D., "A Theoretician's Guide to the Experimental Analysis of Algorithms". In: Goldwasser, M.H., Johnson, D.S., McGeoch, C.C. (eds), *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, pp. 215–250, Providence, American Mathematical Society, 2002.

[28] BOLLOBAS, B., *Random Graphs*. 2 ed. Cambridge, Cambridge University Press, 2001.

[29] CIMIKOWSKI, R., "An Analysis of Heuristics for the Maximum Planar Subgraph Problem". In: *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, pp. 322–331, Jan, 1995.

[30] GOLDSCHMIDT, O., TAKVORIAN, A., "An Eficient Graph Planarization Two-Phase Heuristic", *Networks*, v. 24, pp. 69–73, 1994.

[31] FULKERSON, D., NEMHAUSER, G., TROTTER, L., "Two Computationally Difficult Set Covering Problems that Arise in Computing the 1 Width of Incidence Matrices of Steiner Triple Systems", *Mathematical Programming Study*, v. 2, pp. 72–81, 1974.

[32] BUSSAB, W., MORETTIN, P. *Estatística Básica*. 5 ed. São Paulo, Saraiva, 2004.

[33] FREUND, J. *Estatística Aplicada: Economia, Administração e Contabilidade*. 11 ed. São Paulo, Bookman, 2006.

[34] RAMAKRISHNAN, K., "Private Communication", 1987.