2
Heurística GRASP

A metaheurística GRASP é um processo multipartida para obter soluções aproximadas, eventualmente ótimas, para problemas de otimização combinatória computacionalmente difíceis. Suas iterações consistem, basicamente, de duas fases: construção e busca local. A primeira fase é responsável pela construção de uma solução viável para o problema. Vizinhanças da solução construída são sucessivamente investigadas até que um ótimo local seja encontrado na segunda fase. A melhor solução encontrada ao longo de todas as iterações é fornecida como resultado.

O objetivo deste capítulo é discutir os componentes básicos desta metaheurística, descrever algumas técnicas de melhoria de soluções baseadas em memória e apresentar uma heurística GRASP para 2PNDP. Na Seção 2.1, os componentes básicos da metaheurística GRASP são discutidos. Na Seção 2.2, é realizado um levantamento de mecanismos aprimorados para a construção de soluções e para a aceleração da busca local, de técnicas bem sucedidas de implementação e de estratégias de intensificação de soluções. Na Seção 2.3, a heurística GRASP para 2PNDP é apresentada. Finalmente, na Seção 2.4, algumas considerações finais sobre GRASP são feitas.

2.1
Introdução

Considera-se neste trabalho um problema de otimização combinatória definido por um conjunto base $E = \{1, \ldots, n\}$, um conjunto de soluções viáveis $F \subseteq 2^E$ e uma função objetivo $f : 2^E \rightarrow R$ tal que $f(S) = \sum_{e \in S} c(e)$, $\forall S \in 2^E$, onde $c(e)$ é o custo associado à inclusão do elemento $e \in E$ na solução S. Na versão de minimização, procura-se uma solução ótima $S^* \in F$ tal que $f(S^*) \leq f(S), \forall S \in F$. O conjunto E, o vetor de custos c e o conjunto de soluções viáveis F são definidos para cada problema específico. Por exemplo, no caso de 2PNDP, o conjunto E é formado pelas arestas do grafo, $c(e)$ é o custo de cada aresta $e \in E$ e F é formado por todos

os subconjuntos de arestas que contenham um 2-caminho para cada par pertencente ao conjunto de pares origem-destino.

Metaheurísticas são procedimentos genéricos para a solução aproximada de problemas de otimização combinatória computacionalmente difíceis. Entre elas, podem ser citadas busca tabu, *simulated annealing*, GRASP, algoritmos genéticos, *scatter search*, VNS, colônias de formigas e outras. Cada uma delas está baseada em um paradigma distinto e oferece diferentes mecanismos que permitem escapar de ótimos locais, contrariamente aos algoritmos construtivos ou de melhoria. O processo de busca de uma boa solução aproximada consiste em aplicar a cada passo heurísticas subordinadas, projetadas para cada problema específico. A adaptação ("instanciação") de uma determinada metaheurística para um certo problema fornece uma heurística para este problema.

A metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*) é um procedimento multipartida [30, 31, 84, 86], no qual cada iteração consiste de duas fases: construção e busca local. A fase de construção é responsável por construir uma solução viável, cuja vizinhança é investigada até que um ótimo local seja encontrado durante a fase de busca local. A melhor solução obtida ao longo de todas as iterações é fornecida como resultado. O pseudo-código da Figura 2.1 ilustra os principais blocos de um procedimento GRASP para minimização, no qual são realizadas `MaxIter` iterações e o valor `Semente` é usado como a semente inicial para o gerador de números pseudo-aleatórios.

A Figura 2.2 ilustra o pseudo-código da fase de construção. A cada iteração desta fase, o conjunto de elementos candidatos é formado por todos os elementos que ainda não foram incorporados à solução parcial em construção e que podem sê-lo sem destruir sua viabilidade. A seleção do próximo elemento a ser incorporado é determinada pela avaliação de todos os elementos candidatos de acordo com uma função gulosa. Esta função de avaliação representa o aumento incremental na função objetivo devido à incorporação deste elemento na solução em construção. A avaliação dos elementos segundo esta função leva à criação de uma lista restrita de candidatos (LRC) formada pelos melhores elementos, isto é, aqueles cuja incorporação à solução parcial corrente resulta nos menores custos incrementais (aspecto guloso do algoritmo). O elemento a ser incorporado à solução parcial é selecionado aleatoriamente dentre aqueles da LRC (aspecto probabilístico do algoritmo). Uma vez que o elemento selecionado foi incorporado à solução parcial, a lista de candidatos é atualizada e os custos incrementais são reavaliados (aspecto adaptativo do algoritmo). Esta

estratégia é similar à heurística semi-gulosa de Hart e Shogan [49], que também é uma abordagem multipartida baseada em construções aleatórias, mas sem a etapa de busca local.

A vizinhança N_i de uma solução x pode ser definida como o conjunto de soluções que diferem de x por i atributos. Por exemplo, no caso de 2PNDP, uma solução x' pertence a vizinhança N_1 de uma solução x se ela possui somente um 2-caminho roteado de maneira diferente da de x.

As soluções geradas por uma construção gulosa randomizada não são necessariamente ótimas, mesmo em relação a vizinhanças simples (isto é, N_1). A fase de busca local procura melhorar a solução construída. Algoritmos de busca local funcionam de maneira iterativa, substituindo sucessivamente a solução corrente por outra melhor em sua vizinhança. A busca local termina quando nenhuma solução aprimorante é encontrada na vizinhança da solução corrente. O pseudo-código de um algoritmo básico de busca local iniciado a partir da solução x construída na primeira fase e usando uma vizinhança N é apresentado na Figura 2.3.

```
procedimento GRASP;
1  f* ← ∞; x* ← ∅;
2  Ler os dados do problema;
3  para k = 1,...,MaxIter faça
4      Construir uma solução randomizada x (fase de construção);
5      Encontrar y aplicando busca local a x (fase de busca local);
6      se f(y) < f* então x* ← y; f* ← f(y); fim-se;
7  fim-para;
8  retornar x*;
fim GRASP;
```

Figura 2.1: Pseudo-código da versão básica da metaheurística GRASP.

2.2 Principais Componentes

Heurísticas GRASP para diversos problemas de otimização de diferentes áreas são encontradas na literatura. Um extenso levantamento bibliográfico é apresentado em [35], incluindo artigos sobre aspectos algorítmicos e aplicações. Nesta seção, apresenta-se um sumário de alguns aspectos importantes no projeto de uma heurística GRASP, tais como um estudo sobre a construção da lista restrita de candidatos (LRC), os mecanismos alternativos para a construção de soluções, as estratégias de intensificação

```
procedimento GULOSO_RANDOMIZADO;
1  x ← ∅;
2  Avaliar os custos incrementais dos elementos candidatos;
3  Construir a lista restrita de candidatos LRC;
4  enquanto x não for uma solução completa faça
5      Selecionar aleatoriamente um elemento s dentre aqueles da LRC;
6      x ← x ∪ {s};
7      Atualizar a LRC;
8      Reavaliar os custos incrementais dos elementos da LRC;
9  fim-enquanto;
10 retornar x;
fim GULOSO_RANDOMIZADO;
```

Figura 2.2: Pseudo-código da fase de construção.

```
procedimento BUSCA_LOCAL;
1  enquanto x não for uma solução localmente otima faça
2      Obter x' ∈ N(x) tal que f(x') < f(x);
3      x ← x';
4  fim-enquanto;
5  retornar x;
fim BUSCA_LOCAL;
```

Figura 2.3: Pseudo-código da fase de busca local.

por reconexão por caminhos, algumas extensões do procedimento GRASP básico e algumas hibridizações com outras metaheurísticas.

2.2.1 Construção da Lista Restrita de Candidatos

Segundo [3, 84], existem duas estratégias básicas usadas para construir a LRC. A primeira delas é um esquema baseado 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 um esquema baseado no *valor* associado a cada elemento candidato. Seja $c'(e)$ o custo incremental associado à incorporação do elemento $e \in E$ na solução em construção. Eventualmente, $c'(e) = c(e)$. Sejam $C^{max} = \max\{c'(e)|e \in C\}$, $C^{min} = \min\{c'(e)|e \in C\}$ e um parâmetro $\alpha \in [0, 1]$. Em um problema de minimização, uma LRC baseada em valores será determinada por $LRC = \{e \in C|c'(e) \leq C^{min} + \alpha(C^{max} - C^{min})\}$. O caso $\alpha = 0$ corresponde ao algoritmo guloso puro, enquanto que, para $\alpha = 1$, são construídas soluções aleatórias. Analogamente, em um esquema baseado em *cardinalidade*, o

caso $k = 1$ corresponde a um algoritmo puramente guloso. Já para $k = |C|$, são construídas soluções aleatórias. Assim, os parâmetros α e k determinam se a fase construtiva da metaheurística GRASP será mais semelhante a um algoritmo de construção puramente guloso ou a um algoritmo aleatório. O pseudo-código da Figura 2.4 é um refinamento do algoritmo descrito na Figura 2.2. Ele mostra que o parâmetro α controla a intensidade dos aspectos guloso e randomizado do algoritmo.

De acordo com [72], a maioria das implementações de GRASP encontradas na literatura usam algum tipo de LRC baseada no valor da função gulosa dos elementos candidatos. As primeiras implementações deste método utilizavam valores fixos para α, que eram determinados através de experimentos computacionais. Para cada problema tratado, ou até mesmo para cada classe de instâncias, um valor distinto de α era estabelecido. Porém, segundo [66], a fase construtiva de um algoritmo GRASP com α fixo pode nunca gerar uma solução que leve a um ótimo global. Para evitar este problema, Prais e Ribeiro [76] propuseram o uso de α gerado aleatoriamente no intervalo $[0, 1]$ em cada iteração do algoritmo.

A metaheurística GRASP pode ser vista como uma técnica repetitiva de amostragem no espaço de busca [84]. Cada iteração produz uma solução que é uma amostra de uma distribuição desconhecida. A média e a variância do valor das soluções obtidas ao longo das `MaxIter` iterações dependem da natureza restritiva da LRC. Por exemplo, se a LRC for restrita a um único elemento, então a mesma solução será produzida ao longo de todas as iterações. A variância da distribuição será nula e a média será igual ao valor da solução gulosa. Se a LRC possuir mais de um elemento, então muitas soluções diferentes serão produzidas e a variância será maior. Como o aspecto guloso desempenha um papel importante neste caso, o valor médio das soluções será pior. Entretanto, a melhor solução encontrada será superior à solução obtida pelo algoritmo puramente guloso.

Resende e Ribeiro [84] estudaram o comportamento de um algoritmo GRASP em relação à diversidade de soluções, à qualidade de soluções e ao tempo computacional, em função do parâmetro α, que restringe a LRC. Eles concluíram que, quanto maior a variância dos valores das soluções encontradas durante a fase de contrução, maior será a variância das soluções obtidas após a busca local. A probabilidade de um algoritmo GRASP encontrar uma solução ótima é maior quando a variância das soluções construídas é grande. Porém, neste caso, como a diferença entre o valor médio das soluções construídas e o valor da melhor solução é grande, a busca local precisará, em média, de um maior tempo computacional.

Conseqüentemente, o tempo computacional do algoritmo será maior quando a variância dos custos das soluções produzidas na fase construtiva for alta. Concluiu-se, então, que a escolha apropriada do parâmtero α é crítica para ser alcançado um bom compromisso entre o tempo de processamento e a qualidade da solução.

```
procedimento GULOSO_RANDOMIZADO;
1   x ← ∅;
2   Inicializar o conjunto de elementos candidatos C ← E;
3   Avaliar os custos incrementais c'(e) para cada elemento e ← C;
4   enquanto C ≠ ∅ faça
5       C^min ← min{c'(e)|e ∈ C};
6       C^max ← max{c'(e)|e ∈ C};
7       LRC ← {e ∈ C|c'(e) ≤ C^min + α(C^max − C^min)};
8       Selecionar aleatoriamente um elemento s dentre aqueles da LRC;
9       x ← x ∪ {s};
10      Atualizar o conjunto C de elementos candidatos;
11      Reavaliar o custo incremental c'(e) de cada elemento e ← C;
12  fim-enquanto;
13  retornar x;
fim GULOSO_RANDOMIZADO;
```

Figura 2.4: Refinamento do pseudo-código da fase de construção.

2.2.2 Mecanismos Alternativos Usados na Fase Construtiva

Segundo [84], uma possível deficiência do procedimento GRASP padrão, tal como foi originalmente proposto, é a independência de suas iterações. Ou seja, implementações diretas desta metaheurística não utilizam as informações obtidas ao longo das iterações precedentes: o algoritmo básico descarta as informações sobre qualquer solução encontrada anteriormente que não melhorava a melhor solução conhecida. Entretanto, informações provenientes de boas soluções podem ser usadas para implementar procedimentos alternativos baseados na memória da busca. Tais informações podem influenciar a fase de construção, modificando as probabilidades de seleção associadas a cada elemento da lista restrita de candidatos. Nesta seção, são analisadas diversas extensões e técnicas alternativas para a fase de construção da metaheurística GRASP. Entre elas, pode-se citar: GRASP reativo, perturbações dos custos, funções de tendência, memória e aprendizado e, finalmente, busca local sobre soluções parcialmente construídas.

2.2.2.1 GRASP Reativo

Uma primeira estratégia possível para incorporar mecanismos de aprendizado na fase de construção é o procedimento GRASP reativo [75]. Neste caso, o valor do parâmetro α que define a LRC não é fixo. Em vez disto, este valor é selecionado a cada iteração dentre os elementos de uma distribuição discreta de valores possíveis. Esta seleção é guiada pelos valores das soluções encontradas ao longo das iterações precedentes. Uma maneira possível de alcançar este efeito consiste em usar a regra proposta por Prais e Ribeiro [73, 74, 75, 76]. Seja $\Psi = \{\alpha_1, \ldots, \alpha_m\}$ o conjunto de valores possíveis para α. As probabilidades associadas à escolha de cada valor são inicializadas com $p_i = 1/m, i = 1, \ldots, m$. Além disto, sejam z^* a melhor solução já encontrada e A_i o valor médio de todas as soluções encontradas utilizando-se $\alpha = \alpha_i, i = 1, \ldots, m$. As probabilidades de seleção são periodicamente reavaliadas como $p_i = q_i / \sum_{j=1}^{m} q_j$, onde $q_i = z^*/A_i$ para $i = 1, \ldots, m$. O valor de q_i será maior para os valores $\alpha = \alpha_i$ que levam às melhores soluções. Maiores valores de q_i correspondem a valores mais apropriados para o parâmetro α. As probabilidades associadas aos valores mais apropriados irão aumentar quando elas forem reavaliadas.

O enfoque reativo leva a melhorias sensíveis em relação ao procedimento básico, em termos da robustez do algoritmo e da qualidade das soluções, devido à maior diversidade destas últimas e à menor dependência do ajuste de parâmetros. Além das aplicações descritas em [73, 74, 75, 76], esta abordagem foi usada no planejamento de sistemas de transmissão de potência [16] e no problema de localização capacitado [27].

2.2.2.2 Perturbação dos Custos

A idéia de introduzir ruído nos custos originais é similar àquela do método de ruídos ("noising method") proposto por Charon e Hudry [20, 21]. Ela adiciona mais flexibilidade no projeto de algoritmos e pode ser mais eficaz do que a construção gulosa randomizada do procedimento GRASP básico, em circunstâncias onde os algoritmos de construção não são muito sensíveis à randomização. Este é o caso, por exemplo, da heurística de Takahashi e Matsuyama [97], usada como uma das principais componentes da fase de construção da heurística GRASP híbrida proposta por Ribeiro et al. [91] para o problema de Steiner em grafos, que é a mais eficaz atualmente disponível para este problema na literatura. Outra situação

em que perturbações dos custos podem alcançar bons resultados ocorre quando não existe um algoritmo apropriado para a fase de construção que possa ser facilmente randomizado. Isto acontece, por exemplo, no caso do procedimento GRASP híbrido proposto por Canuto et al. [19] para o problema da árvore de Steiner com prêmios, que se utiliza do algoritmo primal-dual de Goemans e Williamson [44] para construir soluções iniciais usando custos perturbados.

No caso do procedimento GRASP híbrido para o problema de Steiner com prêmios [18, 19], uma nova solução é construída a cada iteração, usando prêmios que são atualizados por uma função de perturbação de acordo com a estrutura da solução corrente. Dois esquemas diferentes de perturbação de prêmios são empregados:

- *perturbação por eliminações*: para forçar a diversificação da busca, o algoritmo primal-dual usado na fase de construção é orientado para construir uma nova solução sem alguns dos nós da solução construída na iteração precedente. Isto é realizado reduzindo-se temporariamente a zero os prêmios associados a alguns nós persistentes, que compunham a última solução construída e que nela permaneceram ao final da fase de busca local. Um parâmetro controla a fração dos nós persistentes cujos prêmios são temporariamente fixados em zero.
- *perturbação por modificação dos prêmios*: para forçar a diversificação da busca, o algoritmo primal-dual usado na fase de construção é orientado para introduzir um ruído nos prêmios de maneira a modificar a função objetivo. Para cada nó i, um fator de perturbação $\beta(i)$ é gerado aleatoriamente no intervalo $[1-a, 1+a]$, onde a é um parâmetro de implementação. O prêmio associado ao nó i é temporariamente modificado para $\bar{\pi}(i) = \pi(i) \cdot \beta(i)$, onde $\pi(i)$ é o seu valor original.

Na abordagem GRASP proposta por Ribeiro et al. [91] para o problema de Steiner, são introduzidos métodos de perturbação de custos que incorporam os mecanismos de intensificação e diversificação, originalmente propostos no contexto da busca tabu. São três estratégias de perturbação de custos, chamadas D, I e U. Os pesos originais, isto é, sem perturbação, são usados nas três primeiras iterações do algoritmo, combinados com três heurísticas de construção. As três estratégias de perturbação de custos são usadas de forma cíclica nas iterações seguintes. Seja w_e o peso da aresta e. A cada iteração i, o peso modificado w_e^i de cada aresta e é selecionado aleatoriamente de uma distribuição uniforme, entre w_e e $r_i(e) \cdot w_e$, onde o coeficiente $r_i(e)$ depende da estratégia de perturbação de custos aplicada na

iteração i. Seja $t_{i-1}(e)$ o número de ótimos locais onde a aresta e aparece após $i-1$ iterações do algoritmo GRASP. As estratégias de perturbação de custos D, I e U são descritas a seguir:

1. D é uma estratégia de diversificação, com $r_i(e) = 1.25 + 0.75 \cdot t_{i-1}(e)/(i-1)$. Assim, valores elevados de $r_i(e)$ são atribuídos a arestas que aparecem mais freqüentemente nos ótimos locais encontrados anteriormente.

2. I é uma estratégia de intensificação, com $r_i(e) = 2 - 0.75 \cdot t_{i-1}(e)/(i-1)$. Assim, arestas que aparecem poucas vezes nos ótimos locais encontrados anteriormente são fortemente penalizadas.

3. U é uma estratégia de penalização uniforme, independente de informações de freqüência, onde $r_i(e) = 2$.

A alternância entre os métodos de intensificação (I) e diversificação (D) caracteriza uma abordagem de oscilação estratégica [40]. O algoritmo GRASP com perturbações de custo e reconexão por caminhos está entre as melhores heurísticas disponíveis para o problem de Steiner em grafos.

2.2.2.3 Funções de Tendência

Na fase de construção do procedimento GRASP básico, o próximo elemento a ser inserido na solução parcial corrente é escolhido aleatoriamente, dentre aqueles da lista restrita de candidatos. Os elementos da LRC recebem probabilidades iguais de serem escolhidos. Entretanto, outras distribuições de probabilidade poderiam ser usadas para orientar a seleção em favor de determinados candidatos. Outro mecanismo de construção foi sugerido por Bresina [17], que propôs uma família de distribuições de probabilidade. Elas são baseadas na posição $r(\sigma)$ atribuída a cada elemento candidato σ, de acordo com seu valor da função gulosa. Diversas funções de tendência são introduzidas, incluindo:

- tendência aleatória: $bias(r) = 1$;
- tendência linear: $bias(r) = 1/r$;
- tendência logarítmica: $bias(r) = log^{-1}(r+1)$;
- tendência exponencial: $bias(r) = e^{-r}$; e
- tendência polinomial de ordem n: $bias(r) = r^{-n}$.

Sejam $r(\sigma)$ a posição do elemento σ e $bias(r(\sigma))$ uma das funções de tendência definidas acima. Uma vez que estes valores tenham sido avaliados para todos elementos da LRC, a probabilidade $\pi(\sigma)$ de selecionar o elemento σ é dada por:

$$\pi(\sigma) = bias(r(\sigma)) / \sum_{\sigma' \in LRC} bias(r(\sigma))$$

A avaliação destas funções de tendência pode ser restrita aos elementos da LRC. O procedimento de seleção de Bresina com uma função de tendência linear foi usado em [15], restrito aos elementos da LRC. Note-se ainda que o procedimento GRASP padrão usa uma função de tendência aleatória.

2.2.2.4 Memória e Aprendizado

Fleurent e Glover [36] observaram que o procedimento GRASP básico não usa uma memória de longo prazo (isto é, informações coletadas de iterações prévias) e propuseram um esquema para tratar esta questão em heurísticas do tipo multipartida.

O esquema proposto por estes autores mantém um conjunto de soluções de elite para ser usado na fase de construção. Para tornar-se uma solução de elite, uma solução deve ser melhor do que todas as soluções presentes neste conjunto, ou melhor do que a pior das soluções no conjunto mas suficientemente diferente de todas elas (por exemplo, pode-se contar o número de componentes idênticas em duas soluções e fixar-se um limiar de rejeição).

Uma variável *fortemente determinada* satisfaz a propriedade de que seu valor não pode ser modificado, sem que o valor da função objetivo seja comprometido ou que muitas outras variáveis tenham seus valores significativamente afetados. Uma variável *consistente* assume o mesmo valor numa grande porção do conjunto de soluções de elite. Seja $I(e)$ uma medida das características de forte determinação e de consistência de um elemento de solução $e \in E$. $I(e)$ torna-se maior a medida que o elemento e aparece mais freqüentemente no conjunto de soluções de elite. Esta função de intensidade $I(e)$ é usada na fase de construção da maneira descrita a seguir. Observa-se que $c'(e)$ é a função gulosa, isto é, representa o custo incremental associado à incorporação do elemento $e \in E$ à solução em construção. Seja ainda $K(e) = F(c'(e), I(e))$ uma função composta definida a partir das funções gulosa e de intensificação. Por exemplo, $K(e) = \lambda \cdot c'(e) + I(e)$.

O esquema de intensificação privilegia a seleção de elementos $e \in E$ da LRC com altos valores de $K(e)$, através da fixação de sua probabilidade de seleção em $p(e) = K(e)/\sum_{s \in LRC} K(s)$.

A função $K(e)$ pode variar ao longo das iterações com o valor de λ, que pode ser inicializado com um valor alto que diminui quando é disparada uma etapa de diversificação. Procedimentos para a atualização do valor de λ são descritos por Fleurent e Glover [36] e por Binato et al. [15]. Os resultados obtidos são melhores do que aqueles alcançados sem o uso deste tipo de memória.

2.2.2.5 Princípio da Otimalidade Progressiva

O princípio da otimalidade progressiva (POP) baseia-se na idéia de que "boas soluções em um determinado nível encontram-se provavelmente próximas a boas soluções em um nível adjacente" [41]. Fleurent e Glover [36] forneceram uma interpretação deste princípio no contexto da metaheurística GRASP. Eles sugeriram que imperfeições introduzidas durante passos anteriores na etapa de construção podem ser corrigidas aplicando-se busca local durante esta primeira fase, e não apenas após seu término.

Devido a considerações de eficiência, uma implementação prática deste princípio em conjunto com GRASP consiste em aplicar uma busca local durante alguns instantes específicos da fase de construção, e não a cada uma de suas iterações. Na aplicação descrita por Binato et al. [15], a busca local é aplicada em duas ocasiões, após a efetivação de 40% e 80% dos movimentos de construção, além de uma outra vez ao final da fase de construção. Nestas duas aplicações [15, 36], o uso do POP como uma alternativa para corrigir imperfeições de soluções parciais permitiu melhorar a qualidade das soluções encontradas.

2.2.3 Reconexão por Caminhos

Uma das maneiras de se melhorar a qualidade das soluções obtidas pelo GRASP básico é o uso de reconexão por caminhos. A técnica de reconexão por caminhos foi proposta, originalmente, por Glover [39] como uma estratégia de intensificação, explorando trajetórias que conectavam soluções de elite obtidas por busca tabu ou *scatter search* [40, 41, 42]. Neste contexto, na busca por melhores soluções são gerados e explorados caminhos

no espaço de soluções partindo de uma ou mais soluções de elite e levando a outras soluções de elite. Isto é alcançado selecionando-se movimentos que introduzem atributos das soluções guia na solução corrente. Esta técnica pode ser vista como uma estratégia que busca incorporar atributos das soluções de boa qualidade, favorecendo a seleção de movimentos que os contenham.

O uso de reconexão por caminhos em um procedimento GRASP como uma estratégia de intensificação aplicada a cada ótimo local foi primeiramente proposto por Laguna e Martí [55]. Esta primeira utilização foi seguida por diversas extensões, melhorias e aplicações bem sucedidas [3, 4, 19, 85, 91]. Estratégias de implementação são investigadas com detalhes por Resende e Ribeiro [84]. As duas estrátegias básicas de aplicação de reconexão por caminhos em procedimentos GRASP são as seguintes:

- reconexão por caminhos aplicada como uma estratégia de pós-otimização entre todos os pares de soluções de elite;
- reconexão por caminhos aplicada como uma estratégia de intensificação a cada ótimo local obtido após a fase de busca local.

A aplicação da técnica de reconexão por caminhos como uma estratégia de intensificação a cada ótimo local é mais eficaz do que empregá-la como um procedimento de pós-otimização. Neste contexto, a reconexão por caminhos é aplicada a pares (x_1, x_2) de soluções, onde x_1 é uma solução localmente ótima obtida após o uso do procedimento de busca local e x_2 é uma solução selecionada aleatoriamente de um conjunto formado por um número limitado, `MaxElite`, de soluções de elite encontradas ao longo da execução do GRASP. Este conjunto está, originalmente, vazio. Cada solução obtida pela busca local é considerada como uma candidata a ser inserida no conjunto de elite, se ela é diferente de todas as outras soluções que estão atualmente neste conjunto. Se o conjunto de elite já possui `MaxElite` soluções e a candidata é melhor que a pior solução existente no conjunto, então a primeira substitui a última. Se o conjunto de elite ainda não está completo, a candidata é simplesmente inserida.

O algoritmo inicia computando a diferença simétrica $\triangle(x_1, x_2)$ entre x_1 e x_2, resultando no conjunto de movimentos que devem ser aplicados a uma delas (a solução inicial) para alcançar a outra (a solução guia). A partir da solução inicial, o melhor movimento ainda não executado de $\triangle(x_1, x_2)$ é aplicado à solução corrente, até que a solução guia seja atingida. A melhor solução encontrada ao longo desta trajetória é considerada como candidata à inserção no conjunto de elite e a melhor solução globalmente já encontrada

é atualizada. Diversas alternativas têm sido consideradas e combinadas em implementações recentes:

- não aplicar reconexão por caminhos a cada iteração GRASP, mas sim apenas periodicamente;
- explorar duas trajetórias potencialmente diferentes, usando x_1 como solução inicial e depois x_2 no mesmo papel;
- explorar apenas uma trajetória, usando ou x_1 ou x_2 como solução inicial; e
- não percorrer a trajetória completa de x_1 até x_2, mas sim apenas parte dela (reconexão por caminhos truncada).

Estas alternativas envolvem um compromisso entre tempo de processamento e qualidade da solução. Ribeiro et al. [91] observaram que a exploração de duas trajetórias potencialmente diferentes para cada par (x_1, x_2) consome, aproximadamente, o dobro do tempo de processamento necessário para explorar apenas uma delas, com um ganho marginal muito pequeno em termos de qualidade de solução. Também observaram que, se apenas uma trajetória deve ser investigada, melhores soluções são obtidas quando a reconexão por caminhos usa como solução inicial a melhor solução dentre x_1 e x_2. Como a vizinhança da solução inicial é explorada com muito maior profundidade do que aquela da solução guia, usar como solução inicial a melhor dentre x_1 e x_2 oferece mais chances ao algoritmo de investigar mais detalhadamente a vizinhança da solução mais promissora. Pela mesma razão, as melhores soluções são normalmente encontradas próximas da solução inicial, permitindo que o procedimento de reconexão por caminhos seja interrompido após algumas iterações, antes da solução guia ser alcançada.

Resultados computacionais ilustrando os ganhos e compromissos entre estas estratégias para o problema de roteamento de circuitos virtuais permanentes em serviços de $f$rame-relay foram relatados por Resende e Ribeiro [85]. Os resultados mostraram que a reconexão por caminhos permitiu obter soluções de qualidade superior àquelas obtidas pelo algoritmo GRASP básico. Resende e Ribeiro [85] puderam, então, concluir que a técnica de reconexão por caminhos é uma estratégia bastante eficaz para introduzir uma componente de memória em procedimentos do tipo GRASP, levando a implementações muito robustas. Esta conclusão pode ser ainda reforçada pelos resultados obtidos pela heurística híbrida combinando GRASP e reconexão por caminhos proposta em [91] para o problema de Steiner em grafos. Esta heurística encontrou melhores soluções do que aquelas então

conhecidas para 33 dos 41 problemas em aberto da série i640 do repositório SteinLib [102].

No restante desta tese, será considerada, apenas, a aplicação da técnica de reconexão por caminhos como uma estratégia de intensificação a cada ótimo local obtido após a fase de busca local. A Figura 2.5 ilustra o pseudo-código genérico contendo os principais blocos do procedimento GRASP com intensificação por reconexão por caminhos para um problema de minimização.

```
procedimento GRASP+RC;
1   f* ← ∞; Pool ← ∅; x* ← ∅;
2   Ler os dados do problema;
3   para k = 1, ..., MaxIter faça
4       Construir uma solução randomizada x (fase de construção);
5       Encontrar y aplicando busca local a x (fase de busca local);
6       se y deve pertencer ao conjunto de soluções de elite então
7           Inserir y no Pool;
8       Escolher aleatoriamente z ∈ Pool (z ≠ y) com probabilidade
        uniforme;
9       Atribuir a y' a melhor solução obtida aplicando reconexão de
        caminhos ao par (z, y);
10      se y' deve pertencer ao conjunto de soluções de elite então
11          Inserir y' no Pool;
12      se f(y') < f* então x* ← y'; f* ← f(y'); fim-se;
13  fim-para;
14  retornar x*;
fim GRASP+RC;
```

Figura 2.5: Pseudo-código da heurística GRASP com intensificação por reconexão por caminhos para um problema de minimização

A Figura 2.6 ilustra o procedimento de reconexão por caminhos para um problema de minimização baseado em abordagens encontradas na literatura. O algoritmo de reconexão por caminhos unidirecional inicia determinando o conjunto de movimentos $\Delta(z, y)$ que será aplicado a z (solução inicial) até chegar a y (solução guia) (linha 3). Cada iteração do procedimento de reconexão por caminhos unidirecional possui os quatro passos seguintes:

- aplicar à solução corrente $\bar{y}$ o melhor movimento do conjunto de movimentos (linha 5), obtendo a solução y'';
- excluir o melhor movimento do conjunto de movimentos ainda possíveis (linha 6);
- atualizar a solução corrente (linha 7); e

– testar se a solução corrente, $\bar{y}$, é melhor que a melhor solução, y', encontrada ao longo da trajetória aplicada a z para chegar a y. Em caso afirmativo, atribui-se $\bar{y}$ a y' (linha 8). No início do método de reconexão por caminhos, atribui-se a y' a melhor solução dentre z e y (linha 2).

Quando a solução corrente chega a y, o algoritmo de reconexão por caminhos pára (linha 4) e retorna a solução y' (linha 10).

```
procedimento RC;
1   ȳ ← z;
2   Atribuir a y' a melhor solução dentre z e y;
3   Calcular o conjunto de movimentos possiveis Δ(z, y);
4   enquanto |Δ(z, y)| ≠ 0 faça
5       Atribuir a y'' a melhor solução obtida aplicando o melhor
        movimento de Δ(z, y) a ȳ;
6       Excluir de Δ(z, y) este movimento;
7       ȳ ← y'';
8       se f(ȳ) < f(y') então y' ← ȳ;
9   fim-enquanto;
10  retornar y';
fim RC;
```

Figura 2.6: Pseudo-código do procedimento de reconexão por caminhos para um problema de minimização baseado em estratégias encontradas na literatura.

2.2.4 Extensões

Nesta seção são discutidas outras extensões, tais como melhorias da fase de busca local, implementações híbridas de GRASP com outras técnicas e uma tentativa de evitar que a heurística GRASP com intensificação por reconexão por caminhos fique por muito sem inserir uma solução no conjunto de soluções de elite.

2.2.4.1 Filtragem

O uso de tabelas de *hashing* para evitar ciclagem em conjunto com busca tabu foi originalmente proposto por Woodruff e Zemel [103]. Um enfoque similar foi explorado posteriormente por Ribeiro et al. [88] em seu algoritmo de busca tabu para otimização de consultas em bancos de dados

relacionais. No contexto de implementações de metaheurística GRASP, tabelas de *hashing* foram utilizadas primeiramente por Martins et al. [61] em sua heurística para o problema de Steiner em grafos baseada no uso de múltiplas vizinhanças na etapa de busca local, para evitar a aplicação da busca local a soluções já visitadas em iterações anteriores.

Outras estratégias de filtragem têm sido usadas para acelerar as iterações de algoritmos GRASP, como descrito por exemplo em [32, 61, 73, 75]. Nestes casos, a busca local não é aplicada a todas as soluções construídas na primeira fase, mas sim apenas a algumas soluções promissoras que não foram visitadas anteriormente, definidas por um parâmetro de limiar em relação ao valor da melhor solução já encontrada. Os resultados apresentados em [73, 75] mostram que este tipo de estratégia de filtragem permite acelerar sobremaneira a heurística GRASP, com uma perda mínima em termos da qualidade das soluções encontradas.

2.2.4.2 Busca Local

Quase todo o esforço de randomização no procedimento GRASP básico ocorre na fase de construção. A busca local é interrompida após encontrar o primeiro ótimo local. Por outro lado, estratégias tais como VNS (*Variable Neighborhood Search*) e VND (*Variable Neighborhood Descent*), propostas por Hansen e Mladenović [48, 65], baseiam-se quase inteiramente na randomização da busca local para escapar de ótimos locais. Com relação a esta questão, GRASP e estratégias baseadas em vizinhanças variáveis podem ser consideradas como complementares e potencialmente capazes de conduzir a métodos híbridos bastante eficazes.

Uma primeira tentativa de aplicação desta idéia foi desenvolvida por Martins et al. [59, 61]. A fase de construção de sua heurística híbrida para o problema de Steiner em grafos segue a estratégia gulosa randomizada do GRASP, enquanto que a fase de busca local usa duas estruturas de vizinhança distintas como em uma estratégia do tipo VND. A segunda vizinhança (definida pela eliminação de caminhos-chave), caracterizada por uma maior complexidade de exploração, só é investigada após ter sido encontrado um ótimo local em relação à primeira vizinhança (definida pela inserção e pela eliminação de vértices de Steiner). Esta heurística foi posteriormente melhorada por Ribeiro et al. [91], sendo que uma das principais componentes deste novo algoritmo é outra estratégia para a exploração de diferentes vizinhanças.

Andreatta e Ribeiro [8, 9] desenvolveram uma heurística GRASP para o problema de inferência de árvores filogenéticas sob o critério de parcimônia, usando uma estratégia de busca local do tipo VND com três estruturas de vizinhança diferentes. As soluções obtidas são melhores do que aquelas produzidas por busca tabu e estão sistematicamente dentro de um limite de 1% do valor da solução conhecida.

Festa et al. [33, 34] também estudaram diferentes variantes e combinações de GRASP, VNS e reconexão por caminhos para o problema do corte de peso máximo (MAX-CUT), melhorando as melhores soluções até então conhecidas para diversas instâncias em aberto na literatura.

Ainda utilizando a idéia de melhorar a fase de busca local através de uma estratégia mais elaborada, Ribeiro e Souza [90] desenvolveram uma heurística híbrida para o problema da árvore geradora de peso mínimo com restrições sobre o grau de cada nó, combinando as estratégias VNS e VND. Um dos resultados obtidos por esta heurística híbrida foi a melhoria das melhores soluções até então conhecidas para duas das instâncias do conjunto STR (associadas a grafos com 70 e 100 nós).

A fase de busca local procura melhorar as soluções geradas na fase de construção. Por ser uma estratégia desprovida de memória em sua versão original, o procedimento GRASP básico não se utiliza das informações coletadas ao longo da busca. Além disto, freqüentemente, a fase de busca local utiliza muito tempo de processamento e repetidamente pára no mesmo ótimo local em diferentes iterações, eventualmente deixando escapar melhores soluções que não são alcançadas mesmo estando muito próximas deste ótimo local. Para melhorar a eficácia da fase de busca local, Souza et al. [95] propuseram adicionar a esta fase uma memória de muito curto prazo, caracterizando uma busca local estendida. Contudo, em vez de manter uma lista de atributos proibidos, como no contexto da busca tabu [41], armazena-se nesta memória as últimas `TabuTenure` soluções visitadas. O algoritmo de busca local sempre se desloca para o melhor vizinho que não faz parte desta lista, independentemente de ser uma solução aprimorante ou não. A busca é interrompida quando a melhor solução global não é melhorada após uma seqüência de `TabuTenure` iterações da busca local. Um valor pequeno é usado para `TabuTenure`, a fim de manter os tempos de processamento reduzidos.

A heurística GRASP com reconexão por caminhos usando busca local estendida para o problema da árvore geradora de peso mínimo capacitada [95] foi executada em um processador Celeron 800 MHz com 128 Mbytes de memória. Observou-se que esta estratégia encontrou melhores soluções que o

procedimento GRASP básico em todas as 12 maiores instâncias encontradas na literatura, associadas a grafos com 120 e 160 nós.

2.2.4.3 Hibridizações

A metaheurística GRASP também tem sido usada em conjunto com algoritmos genéticos. Em geral, a estratégia gulosa randomizada empregada na fase de construção de um procedimento GRASP é utilizada para gerar a população inicial de soluções para um algoritmo genético. Pode-se citar, por exemplo, o algoritmo genético de Ahuja et al. [2] para o problema quadrático de alocação, que se utiliza da heurística GRASP proposta em [57] para criar a população inicial. Um enfoque similar foi usado por Armony et al. [10], com a população inicial formada tanto por soluções geradas aleatoriamente, como por outras geradas por um algoritmo GRASP.

A hibridização da metaheurística GRASP com busca tabu foi estudada originalmente por Laguna e González-Velarde [54]. Delmaire et al. [27] desenvolveram dois enfoques distintos. No primeiro, uma heurística GRASP é aplicada como uma poderosa estratégia de diversificação no contexto de um procedimento de busca tabu. O segundo enfoque é uma implementação do algoritmo GRASP reativo apresentado na Seção 2.2.2.1, no qual a fase de busca local é reforçada por busca tabu. Resultados relatados para o problema de localização capacitado mostram que os enfoques híbridos têm melhor desempenho que cada um dos métodos empregados isoladamente. Heurísticas a dois estágios foram propostas em [1] para um problema de projeto de *layout*. GRASP/TS aplica uma heurística GRASP para encontrar o *layout* inicial e, posteriormente, busca tabu para refiná-lo.

2.2.4.4 Tempo de Corte de Execução

Freqüentemente, a heurística GRASP com intensificação por reconexão por caminhos repetidamente não consegue inserir uma solução no conjunto de soluções de elite em sucessivas iterações. Uma tentativa simples de evitar a computação de soluções que, por muito tempo, não melhoram o conjunto de elite é acrescentar o conceito de tempo de corte de execução a esta heurística: se o procedimento GRASP não inserir uma solução no conjunto de soluções de elite após um número limitado `MaxCorte` de soluções depois da última inserção, este procedimento executa as iterações restantes

a partir do valor inicial de `Semente`, mantendo inalterado o conjunto de soluções de elite.

A Figura 2.7 ilustra como o conceito de corte pode ser aplicado ao GRASP com intensificação por reconexão por caminhos. O algoritmo é desenvolvido a partir daquele descrito na Figura 2.5. As diferenças entre os dois métodos são apresentadas a seguir. Na linha 1, a variável que registra a quantidade de soluções sem melhoria no conjunto de elite, chamada *iteracoes_corte*, é inicializada com zero. Toda vez que uma solução puder pertencer ao conjunto de soluções de elite, a variável *iteracoes_corte* é reinicializada com zero. Caso contrário, ela é incrementada.

Se a solução encontrada pelo procedimento de busca local puder fazer parte do conjunto de soluções de elite (linha 6), esta solução é inserida no conjunto e *iteracoes_corte* é reinicializada com zero (linha 7). Senão, *iteracoes_corte* é incrementada (linhas 8 e 9).

Se a solução obtida pelo procedimento de reconexão por caminhos puder pertencer ao conjunto de soluções de elite (linha 13), esta solução é inserida no conjunto e *iteracoes_corte* é reinicializada com zero (linha 14). Senão, *iteracoes_corte* é incrementada (linhas 15 e 16).

Se a variável *iteracoes_corte* indicar que a quantidade de soluções geradas sem melhorar o conjunto de elite atingiu o valor limite `MaxCorte` (linha 19), o procedimento GRASP executa as suas iterações restantes a partir do valor inicial de `Semente`, reinicializando a variável *iteracoes_corte* com zero (linhas 20 e 21).

2.3 GRASP para 2PNDP

Nesta seção são discutidos os componentes básicos da heurística GRASP para 2PNDP, dentre eles um procedimento responsável pela fase de construção de soluções viáveis e dois procedimentos de busca local. Um destes dois últimos utiliza uma memória de muito curto prazo, como foi descrito na Seção 2.2.4.2. Além disto, a abordagem de reconexão por caminhos unidirecional é aplicada ao 2PNDP e uma nova técnica mista de reconexão por caminhos é proposta.

```
procedimento GRASP+RC+TC;
1   f* ← ∞; Pool ← ∅; x* ← ∅; iteracoes_corte ← 0; Semente' ← Semente;
2   Ler os dados do problema;
3   para k = 1, ..., MaxIter faça
4       Construir uma solução randomizada x (fase de construção);
5       Encontrar y aplicando busca local a x (fase de busca local);
6       se y deve pertencer ao conjunto de soluções de elite então
7           Inserir y no Pool; iteracoes_corte ← 0;
8       senão
9           iteracoes_corte ← iteracoes_corte + 1;
10      fim-se;
11      Escolher aleatoriamente z ∈ Pool (z ≠ y) com probabilidade
        uniforme;
12      Atribuir a y' a melhor solução obtida aplicando reconexão de
        caminhos ao par (z, y);
13      se y' deve pertencer ao conjunto de soluções de elite então
14          Inserir y' no Pool; iteracoes_corte ← 0;
15      senão
16          iteracoes_corte ← iteracoes_corte + 1;
17      fim-se;
18      se f(y') < f* então x* ← y'; f* ← f(y'); fim-se;
19      se iteracoes_corte ≥ MaxCorte então
20          Atribuir a Semente o valor de Semente';
21          iteracoes_corte ← 0;
22      fim-se;
23  fim-para;
24  retornar x*;
fim GRASP+RC+TC;
```

Figura 2.7: Pseudo-código da heurística GRASP com intensificação por reconexão por caminhos com tempo de corte de execução para um problema de minimização

2.3.1 Fase de Construção

Em cada iteração, o algoritmo guloso constrói um 2-caminho para um par origem-destino pertencente ao conjunto de demandas. Este algoritmo não tem qualquer semelhança com o algoritmo guloso descrito em [25], que resolve, aproximadamente, o dual da relaxação linear de um modelo de programação inteira.

A Figura 2.8 ilustra a fase de construção para 2PNDP. A construção de uma solução começa pela inicialização dos pesos modificados das arestas com o valor dos pesos das arestas originais (linha 1). O uso de pesos modificados é uma maneira de informar ao algoritmo que uma determinada aresta já está sendo usada na construção da solução. Toda vez que uma aresta já

fizer parte da solução que está sendo construída, o seu peso modificado é reinicializado com zero.

Cada iteração do algoritmo construtivo faz, inicialmente, a seleção aleatória de um par origem-destino pertencente ao conjunto de demandas (linha 3). O menor 2-caminho entre as extremidades deste par é calculado (linha 4), os pesos modificados das arestas que fazem parte deste 2-caminho são reinicializados com zero (linha 5), o par é removido do conjunto de demandas (linha 6) e este 2-caminho é inserido na solução (linha 7). Se todo o conjunto de demandas já tiver sido tratado, o algoritmo pára (linha 2). Senão, uma nova iteração é realizada. Portanto, o algoritmo responsável por construir uma solução gulosa pára quando 2-caminhos tenham sido calculados para todos os pares origem-destino do conjunto de demandas, gerando uma solução viável.

O laço das linhas 2 a 8 é executado p vezes, e o cálculo do menor 2-caminho tem complexidade $O(n)$, onde $p = |D|$ e $n = |V|$. Como os demais passos do algoritmo são $O(1)$, a complexidade do algoritmo da Figura 2.8 é $O(pn)$.

```
procedimento GULOSO;
1   x ← ∅; c' ← c;
2   enquanto D ≠ ∅ faça
3       Selecionar aleatoriamente um par (a, b) pertencente a D;
4       Calcular o menor 2-caminho entre (a, b) usando os pesos
        modificados c';
5       Reinicializar com zero os pesos modificados das arestas do 2-caminho
        calculado;
6       Remover (a, b) de D;
7       Inserir em x o 2-caminho calculado;
8   fim-enquanto;
9   retornar x;
fim GULOSO;
```

Figura 2.8: Pseudo-código da fase de construção para 2PNDP.

2.3.2 Fase de Busca Local

Como foi discutido no início deste capítulo, a fase de busca local tenta melhorar cada solução construída na fase de construção. Nesta seção são discutidas duas abordagens de busca local desenvolvidas para 2PNDP: busca local por ordem circular e busca local tabu. Esta última utiliza um mecanismo de memória de muito curto prazo.

2.3.2.1 Busca Local por Ordem Circular

A Figura 2.9 descreve o procedimento de busca local por ordem circular. No caso do 2PNDP, cada solução pode ser vista como um conjunto de 2-caminhos, um para cada par origem-destino pertencente ao conjunto de demandas. No início do algoritmo, uma permutação aleatória dos pares do conjunto de demandas é gerada (linha 2). Este passo é realizado para que se introduza diversidade nesta busca, de modo que diferentes aplicações deste algoritmo possam levar a diferentes ótimos locais.

Cada iteração deste algoritmo faz, inicialmente, a escolha do próximo par origem-destino em D (linha 4), de acordo com a permutação previamente gerada. Procura-se então melhorar este 2-caminho, substituindo-o por outro 2-caminho de menor custo. Para realizar este movimento, deve-se fazer os seguintes passos:

- reinicializar com zero os pesos modificados das arestas usadas pelos 2-caminhos associados aos demais pares origem-destino em D (linha 5);
- calcular o menor 2-caminho entre as extermidades do par selecionado (linha 6); e
- atualizar a solução corrente, se ela for melhorada pelo novo 2-caminho (linhas 7 e 8).

Se $|D|$ caminhos tiverem sido investigados sem melhorar a solução atual, o algoritmo pára (linha 3). Senão, uma nova iteração é realizada. A variável *sem_mudanca* é responsável por controlar o número de 2-caminhos que foram tratados sem melhorar a solução atual. Toda vez que um 2-caminho melhora a solução atual, esta variável é reinicializada com zero (linha 9). Senão, esta variável é incrementada (linha 11).

Os passos de redução a zero (linha 5) e de cálculo de menor caminho (linha 6) têm complexidade $O(n)$. O passo de geração de uma permutação aleatória têm complexidade $O(p)$. Os demais passos têm complexidade $O(1)$. Logo, a complexidade de cada iteração da busca local por ordem circular é $O(n)$.

```
procedimento BL_ORDEM_CIRCULAR;
1   x' ← x; c' ← c; sem_mudanca ← 0;
2   Gerar uma permutação aleatoria dos pares de D;
3   enquanto sem_mudanca ≤ |D| faça
4       Escolher o proximo par origem-destino (a, b) em D;
5       Reinicializar com zero os pesos modificados das arestas dos
        2-caminhos associados aos demais pares origem-destino em D;
6       Calcular o menor 2-caminho entre (a, b) usando os pesos
        modificados c';
7       Obter x' substituindo o 2-caminho anterior em x pelo novo;
8       se f(x') < f(x) então
9           x ← x'; sem_mudanca ← 0;
10      senão
11          sem_mudanca ← sem_mudanca + 1;
12      fim-se;
13  fim-enquanto;
14  retornar x;
fim BL_ORDEM_CIRCULAR;
```

Figura 2.9: Pseudo-código da busca local por ordem circular para 2PNDP.

2.3.2.2 Busca Local Tabu

Conforme discutido anteriormente, o algoritmo GRASP básico não se utiliza das informações coletadas ao longo da busca. Freqüentemente, a fase de busca local gasta muito tempo de processamento e, repetidamente, pára no mesmo ótimo local em diferentes iterações. Para melhorar a eficácia desta fase, foi desenvolvido um método de busca local que usa as idéias de busca local estendida [95]. O procedimento mantém uma lista das últimas `TabuTenure` soluções proibidas e sempre se desloca para o melhor vizinho que não faz parte desta lista, independentemente de ser uma solução aprimorante ou não. A busca é interrompida quando a melhor solução global não é melhorada após uma seqüência de `TabuTenure` iterações da busca local.

A Figura 2.10 ilustra o procedimento de busca local tabu para 2PNDP. No início deste algoritmo, a lista das soluções proibidas, chamada *Lista_Tabu*, está vazia e a variável que conta o número de iterações sem mudanças na melhor solução encontrada ao longo da busca, chamada *sem_mudanca_global*, é inicializada com zero (linha 1).

Similar à discussão do procedimento de busca local de ordem circular realizada na Seção 2.3.2.1, cada iteração deste algoritmo faz, inicialmente, a escolha do próximo par origem-destino em D (linha 5). Procura-se então melhorar este 2-caminho, substituindo-o por outro 2-caminho de menor

custo. Os movimentos de melhoria de um 2-caminho são os mesmos do algoritmo descrito na Figura 2.3.2.1.

Se a solução encontrada não pertence à lista de soluções proibidas (linha 9), verifica-se se ela é melhor que a solução corrente (linha 10). Se for, armazena-se esta solução como a solução corrente e atribui-se à variável *melhor_local* uma marca que indique que o algoritmo encontrou uma solução aprimorante (linha 11). Se esta solução não for melhor que a solução corrente (linha 12), testa-se se ela é melhor que a melhor solução não aprimorante. Em caso afirmativo, armazena-se esta solução como a melhor solução não aprimorante (linhas 13 a 15). Se a solução encontrada pertence à lista de soluções proibidas, esta solução é descartada (linhas 16 a 18).

Se a variável *melhor_local* indicar que o algoritmo encontrou uma solução aprimorante (linha 20), a variável *sem_mudanca_global* é reinicializada com zero (linha 21), a lista de soluções proibidas é esvaziada (linha 22) e verifica-se se esta solução é melhor que a melhor solução encontrada ao longo da busca (linha 23). Em caso afirmativo, atualiza-se esta última (linhas 24 e 25).

Se a variável *melhor_local* indicar que o algoritmo não encontrou uma solução aprimorante (linha 26), atribui-se a solução corrente à melhor solução não aprimorante e incrementa-se a variável *sem_mudanca_global* (linha 27). Além disto, esta solução passa a pertencer à lista de soluções proibidas (linhas 28 e 29).

Se `TabuTenure` iterações desta busca tiverem sido realizadas sem melhorar a melhor solução encontrada, o algoritmo pára (linha 2). Senão, uma nova iteração é realizada.

O laço das linhas 4 a 19 é executado, no pior caso, p vezes. Como todos os passos deste algoritmo têm complexidade $O(1)$, exceto os passos citados nas Seções 2.3.1 e 2.3.2.1 que têm complexidade $O(n)$, a complexidade de cada iteração da busca local tabu é $O(pn)$.

2.3.3 Reconexão por Caminhos

Nesta seção o procedimento de reconexão por caminhos unidirecional (Seção 2.2.3) é especializado para 2PNDP. De acordo com a descrição da Figura 2.6, este procedimento inicia determinando todos os pares origem-destino que são roteados de maneira diferente entre as soluções z e y, gerando um conjunto de movimentos $\Delta(z, y)$ (linha 3). Este passo tem complexidade $O(p)$, onde $p = |D|$, porque p 2-caminhos são testados. Cada movimento é

```
procedimento BL_TABU;
1   f* ← ∞; x* ← x; c' ← c; sem_mudanca_global ← 0; Lista_Tabu ← ∅;
2   enquanto sem_mudanca_global ≤ TabuTenure faça
3       i ← 1; x' ← x; melhor_local ← falso; g ← ∞; x_pior ← ∞;
4       enquanto i ≤ |D| e melhor_local = falso faça
5           Escolher o proximo par origem-destino (a, b) em D;
6           Reinicializar com zero os pesos modificados das arestas dos
            2-caminhos associados aos demais pares origem-destino em D;
7           Calcular o menor 2-caminho entre (a, b) usando os pesos
            modificados c';
8           Obter x' substituindo o 2-caminho anterior em x pelo novo;
9           se x' ∉ Lista_Tabu então
10              se f(x') < f(x) então
11                  x ← x'; melhor_local ← verdadeiro;
12              senão
13                  i ← i + 1;
14                  se f(x') < g então x_pior ← x'; g ← f(x'); fim-se;
15              fim-se;
16          senão
17              i ← i + 1;
18          fim-se;
19      fim-enquanto;
20      se melhor_local = verdadeiro então
21          sem_mudanca_global ← 0;
22          Lista_Tabu ← ∅;
23          se f(x) < f* então
24              f* ← f(x); x* ← x;
25          fim-se;
26      senão
27          x ← x_pior; sem_mudanca_global ← sem_mudanca_global + 1;
28          Lista_Tabu ← Lista_Tabu ∪ {x};
29      fim-se;
30  fim-enquanto;
31  retornar x*;
fim BL_TABU;
```

Figura 2.10: Pseudo-código da busca local tabu para 2PNDP.

caracterizado por um par de 2-caminhos, um a ser inserido e outro a ser eliminado da solução corrente.

O passo de aplicação do melhor movimento (linha 5) tem complexidade $O(p)$, porque todos os 2-caminhos que são diferentes entre eles precisam ser verificados, para que se descubra qual é o melhor movimento que pode ser realizado. Os passos de exclusão do melhor movimento do conjunto de movimentos (linha 6), de atualização da solução corrente (linha 7) e de verificação da solução corrente (linha 8) têm complexidade $O(1)$.

O laço das linhas 4 a 9 é executado, no pior caso, p vezes. Como os demais passos têm complexidade $O(1)$, a complexidade do algoritmo da Figura 2.6 é $O(p^2)$.

Conforme discutido na Seção 2.2.3, o algoritmo de reconexão por caminhos unidirecional calcula o conjunto de movimentos $\Delta(z, y)$ que será aplicado a z (solução inicial) até chegar a y (solução guia). Esta técnica de reconexão por caminhos possui uma única trajetória de investigação de vizinhanças. Pode ser interessante procurar novas soluções usando mais de uma trajetória. A proposta da técnica de reconexão por caminhos mista é examinar duas trajetórias distintas de investigação de vizinhanças.

A Figura 2.11 ilustra o método misto de reconexão por caminhos para 2PNDP. O algoritmo inicia determinando todos os pares origem-destino que são roteados de maneira diferente entre as soluções z e y, gerando um conjunto de movimentos $\Delta(z, y)$ (linha 3) que será aplicado a z e a y até chegar a uma solução comum entre as duas trajetórias de investigação de vizinhanças. Neste algoritmo, z é uma solução selecionada aleatoriamente de um conjunto formado por um número limitado de soluções de elite encontradas anteriormente, enquanto que y é uma solução localmente ótima obtida após o uso do procedimento de busca local.

Duas soluções correntes, $\bar{z}$ e $\bar{y}$ são inicializadas com as soluções z e y, respectivamente (linha 1). Inicializa-se a variável que armazena a melhor solução encontrada, y', como a melhor solução dentre z e y (linha 2). De forma similar ao algoritmo que foi discutido na Seção 2.2.3, cada movimento é caracterizado por um par de 2-caminhos, um a ser inserido e outro a ser eliminado da solução corrente.

Se a cardinalidade do conjunto de movimentos $\Delta(z, y)$ for par (linha 5), aplica-se à solução corrente $\bar{z}$ o melhor movimento do conjunto de movimentos, obtendo a solução z'' (linha 6). Atualiza-se a solução corrente $\bar{z}$ (linha 7) e verifica-se se $\bar{z}$ é melhor que y'. Se for, atribui-se $\bar{z}$ a y' (linha 8). Se o conjunto de movimentos $\Delta(z, y)$ for ímpar (linha 9), aplica-se à solução corrente $\bar{y}$ o melhor movimento do conjunto de movimentos,

obtendo a solução y'' (linha 10). Atualiza-se a solução corrente $\bar{y}$ (linha 11) e verifica-se se $\bar{y}$ é melhor que y'. Se for, atribui-se $\bar{y}$ a y' (linha 12). O melhor movimento do conjunto de movimentos é excluído (linha 14). Se o conjunto de movimento estiver vazio, o algoritmo de reconexão por caminhos pára (linha 4) e retorna a solução y'. Senão, uma nova iteração é realizada.

Os passos de aplicação do melhor movimento das linhas 6 e 9 têm complexidade $O(p)$. O laço das linhas 4 a 15 é executado, no pior caso, p vezes. Como os demais passos têm complexidade $O(1)$, a complexidade do algoritmo da Figura 2.11 é $O(p^2)$.

```
procedimento RC_MISTO;
1   ȳ ← y; z̄ ← z;
2   Atribuir a y' a melhor solução dentre z e y;
3   Calcular o conjunto de movimentos possiveis Δ(z, y);
4   enquanto |Δ(z, y)| ≠ 0 faça
5       se |Δ(z, y)| for par então
6           Atribuir a z'' a melhor solução obtida aplicando o melhor
            movimento de Δ(z, y) a z̄;
7           z̄ ← z'';
8           se f(z̄) < f(y') então y' ← z̄;
9       senão
10          Atribuir a y'' a melhor solução obtida aplicando o melhor
            movimento de Δ(z, y) a ȳ;
11          ȳ ← y'';
12          se f(ȳ) < f(y') então y' ← ȳ;
13      fim-se;
14      Excluir de Δ(z, y) o movimento realizado;
15  fim-enquanto;
16  retornar y';
fim RC_MISTO;
```

Figura 2.11: Pseudo-código do método misto de reconexão por caminhos para 2PNDP.

2.4 Considerações Finais

Desenvolvimentos recentes apresentados na Seção 2.2 mostraram que diferentes extensões do procedimento básico permitem melhorar sobremaneira a qualidade das soluções produzidas por heurísticas GRASP. No caso do 2PNDP, foram estudadas a busca local tabu, que incorpora um mecanismo de memória à fase de busca local e a técnica de reconexão por caminhos, que permite a implementação de estratégias de intensificação

baseadas na memória de soluções de elite. Também foi proposta uma nova estratégia mista de reconexão por caminhos e foi discutida uma tentativa simples de impedir a estagnação da qualidade das soluções do conjunto de elite do algoritmo GRASP com intensificação por reconexão por caminhos.

As variantes de GRASP com reconexão por caminhos para 2PNDP serão avaliadas e comparadas no próximo capítulo. A melhor variante de GRASP com reconexão por caminhos será comparada com o algoritmo guloso da literatura [25].