6
Buscas locais e heurísticas baseadas em metaheurísticas para AGMD

Neste capítulo são propostas quatro tipos de vizinhanças e duas heurísticas baseadas em metaheurísticas para AGMD. As vizinhanças são denominadas de "1-opt", "2-opt", "adoção de sub-árvore" e "substituição de caminho" e são apresentadas juntamente com as buscas locais que as exploram. A primeira heurística desenvolvida é do tipo GRASP Reativo, usando na fase de busca local um procedimento parecido com VND. O segundo algoritmo proposto é uma hibridização que utiliza fundamentos de GRASP, ILS e a busca local utilizada na primeira heurística baseada em metaheurística desenvolvida.

6.1
Vizinhanças e buscas locais

A vizinhança de uma solução é um conjunto de soluções que podem ser alcançadas a partir de pequenas modificações realizadas na solução considerada. Modificações são comumente chamadas de movimentos. Os procedimentos que exploram sistematicamente uma dada vizinhança, no espaço de soluções de um problema, são denominados de buscas locais. Existem buscas que trocam de solução tão logo uma solução aprimorante seja encontrada e outras que avaliam todos os vizinhos e somente então substituem a solução corrente pela mais aprimorante.

Quatro vizinhanças foram desenvolvidas e são apresentadas a seguir. As buscas locais que as exploram são identificadas pelo mesmo nome da vizinhança. Essas buscas são avaliadas em termos da complexidade de se examinar todos os vizinhos em uma passagem da busca local e da complexidade de atualização das estruturas de dados.

6.1.1
Vizinhança e busca local 1-opt

Soluções na vizinhança 1-opt são obtidas através da substituição de uma aresta de uma árvore viável por uma outra inicialmente fora da solução, atendendo determinados critérios. Esses critérios, especificados a seguir, determinam como uma nova aresta entra na solução. A retirada de uma aresta de

uma solução quebra a árvore inicial em duas sub-árvores. A que contém a raiz da árvore inicial é chamada nesta tese de árvore principal e a outra de árvore secundária. As arestas candidatas a reconectarem essas duas sub-árvores são apenas aquelas que possuem uma extremidade como raiz da árvore secundária e a outra pode ser qualquer nó da árvore principal. Dentre todas as arestas candidatas a substituir a aresta eliminada, escolhe-se aquela que gera a maior redução no valor do custo da solução e cuja inclusão mantém a viabilidade da solução.

Um exemplo de uma migração de uma solução para uma outra vizinha é dado na Figura 6.1.

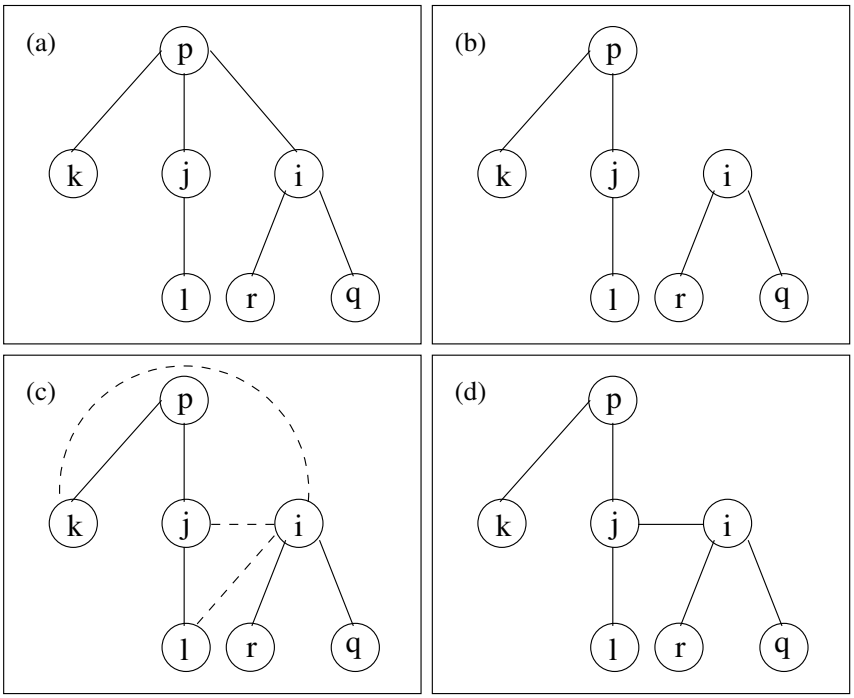

Figura 6.1: Exemplo de migração para uma solução vizinha utilizando 1-opt.

Na Figura 6.1-(a) encontra-se uma solução viável inicial. Pretende-se obter a melhor solução vizinha a partir da eliminação da aresta $[p, i]$. A remoção da aresta $[p, i]$, como ilustrado na Figura 6.1-(b), gera duas sub-árvores. A primeira, com raiz no nó p, é a sub-árvore principal. A segunda, chamada de sub-árvore secundária, é aquela com raiz no nó i. As arestas candidatas a integrarem a solução estão identificadas por linhas pontilhadas na Figura 6.1-(c). Considera-se que a aresta $[j, i]$ possui o menor custo dentre todas as arestas candidatas a entrarem na solução e que mantém a viabilidade da solução. Então, $[j, i]$ entra na solução como ilustrado na Figura 6.1-(d).

Na busca local desenvolvida para explorar essa vizinhança, tenta-se eliminar cada uma das arestas da solução corrente uma vez. Após a eliminação

de uma aresta, é realizada uma busca com o objetivo de encontrar uma aresta fora da solução de custo menor do que o da aresta eliminada e que, uma vez inserida na solução, mantenha a sua viabilidade. Se for encontrada alguma aresta que satisfaça estas condições, então esta aresta entra na solução. Caso contrário, retorna-se à solução anterior e passa-se à eliminação de uma outra aresta.

Quatro estruturas de dados principais são utilizadas na busca 1-opt. A primeira estrutura, chamada de "árvore", é um vetor que contém o pai de cada nó na árvore. O pai de um nó i é o primeiro nó no caminho de i até a raiz. As estruturas chamadas de "filhos" são listas contendo os filhos de cada nó na árvore. A estrutura chamada de "profundidade" corresponde a um vetor que guarda o nível de profundidade de cada nó em relação a seu pai na árvore. A quarta estrutura, chamada de "altura", é um vetor que guarda a quantidade de níveis abaixo de cada nó. Essas estruturas de dados são utilizadas em todas as buscas locais desenvolvidas.

Para testar a viabilidade de um movimento na busca 1-opt, é averiguado se a profundidade do nó candidato a ser pai da sub-árvore secundária adicionada de um mais a altura da sub-árvore secundária é menor ou igual a $D/2$. Se for, então a aresta considerada pode entrar na solução.

O tamanho da vizinhança 1-opt é $O(|V|^2)$. Isto porque, uma árvore possui $O(|V|)$ arestas que podem ser removidas. Para testar a melhor posição para a reconexão da sub-árvore secundária, $O(|V|)$ testes são realizados no pior caso. Se um movimento for realizado, a atualização das estruturas de dados é realizada em tempo $O(|V|)$. Isto porque, para atualizar as estruturas de dados árvore, profundidade e altura após um movimento da busca 1-opt consome-se tempos $O(1)$, $O(|V|)$ e $O(|V|)$, respectivamente. Na inclusão e na eliminação de um nó de uma lista de filhos consome-se no pior caso $O(1)$ e $O(|V|)$, respectivamente. A inclusão de um novo filho é realizada ao final da lista. A exclusão de um nó implica na pesquisa do mesmo na estrutura, para que possa ser removido. Portanto, a avalição de todos os vizinhos em uma passagem da busca local 1-opt tem complexidade $O(|V|^2)$.

6.1.2 Vizinhança e busca local 2-opt

Soluções na vizinhança 2-opt são obtidas a partir da substituição de duas arestas de uma árvore viável por duas outras inicialmente fora da solução, atendendo determinados critérios. Esses critérios especificam como duas arestas são removidas e duas outras são incluídas na solução. A eliminação das duas arestas ocorre em duas etapas. Na primeira etapa, uma aresta é eliminada da solução, gerando as sub-árvores principal e secundária, como na

busca 1-opt. A segunda etapa consiste na eliminação de uma outra aresta da árvore principal. Ao final desse processo, obtém-se uma árvore principal e duas sub-árvores secundárias. As duas sub-árvores secundárias são reconectadas por suas raízes, formando uma única e nova sub-árvore secundária. Para ser a raiz desta nova sub-árvore secundária, escolhe-se dentre as duas raízes das sub-árvores secundárias iniciais, aquela que se conecta com menor custo a algum nó da árvore principal. A reconexão da nova sub-árvore secundária com a principal é realizada tomando-se a aresta que leva à maior redução no valor do custo da solução e não viola a restrição de diâmetro.

Um exemplo para obtenção de uma solução vizinha é dado na Figura 6.2. Na Figura 6.2-(a) encontra-se uma solução viável inicial. Pretende-se obter uma solução vizinha aprimorante a partir da eliminação das arestas $[p, i]$ e $[q, j]$. Na primeira etapa de eliminação, a aresta $[p, i]$ é retirada da solução como ilustrado na Figura 6.2-(b). A árvore com raiz em r é a árvore principal e a primeira sub-árvore secundária é aquela com raiz em i. Na segunda etapa, a aresta $[q, j]$ é removida da solução, como ilustrado na Figura 6.2-(c). A segunda sub-árvore secundária foi obtida e tem raiz em j. Em seguida, inicia-se a etapa de reconexão das árvores. Primeiramente, as duas sub-árvores secundárias são conectadas por suas raízes, como apresentado na Figura 6.2-(d). As arestas candidatas a entrarem na solução estão identificadas por linhas pontilhadas nas Figuras 6.1-(e) e 6.1-(f). Se a aresta $[j, k]$ é aquela que mais aprimora a solução corrente e mantém a viabilidade da solução, então $[j, k]$ entra na solução como ilustrado na Figura 6.1-(g).

A busca local que explora esta vizinhança examina a remoção de arestas como descrito acima. Após a obtenção das três sub-árvores, as sub-árvores secundárias são conectadas por suas respectivas raízes. Em seguida, é realizada uma busca objetivando encontrar uma aresta fora da solução para reconectar a nova sub-árvore secundária à árvore principal. A soma dos custos das novas arestas candidatas a integrarem a solução deve ser menor do que a soma dos custos das arestas eliminadas. Além disto, a reconexão das sub-árvores deve ser realizada de modo que a solução permaneça viável. Se forem encontradas tais arestas, então elas entram na solução. Caso contrário, a segunda sub-árvore secundária é reconectada na sua posição inicial. Em seguida, elimina-se outra aresta da árvore principal para obter-se uma nova sub-árvore secundária. Este procedimento é repetido para todas as arestas da árvore principal. Se nenhum movimento aprimorante for encontrado, retorna-se à solução inicial e o processo de eliminação de arestas é reiniciado. A busca local é interrompida tão logo seja encontrado um movimento aprimorante.

Dois testes são realizados para averiguar se um movimento é viável. Estes testes envolvem a profundidade do nó candidato a ser pai da nova sub-árvore

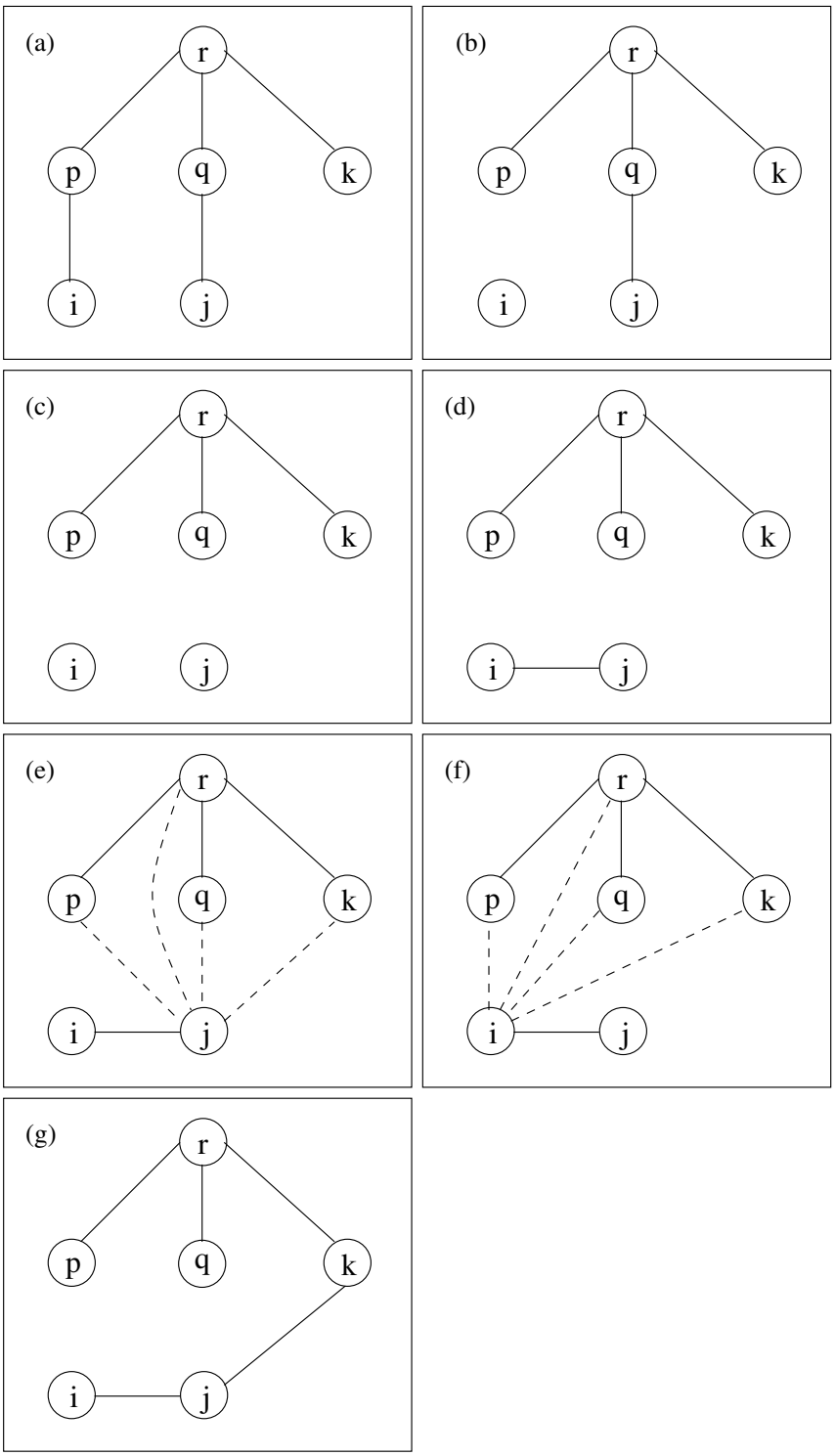

Figura 6.2: Exemplo de migração para uma solução vizinha utilizando 2-opt.

secundária e a altura das antigas sub-árvores secundárias. A sub-árvore secundária, que poderá ser conectada imediatamente um nível abaixo do nó candidato a ser pai da nova sub-árvore, terá seu nível de profundidade igual ao de seu pai mais um. Então, o primeiro teste de viabilidade consiste em tomar

essa nova profundidade e somá-la à altura da sub-árvore secundária considerada. Para que a solução permaneça viável, este resultado deve ser menor ou igual a $D/2$. O segundo teste é realizado com a outra sub-árvore secundária. Esta sub-árvore fica dois níveis de profundidade abaixo do nó candidato a ser pai da nova sub-árvore. Conseqüentemente, a nova profundidade de seus nós é incrementada em dois. O segundo teste consiste em averiguar se a nova profundidade mais a altura da sub-árvore considerada é menor ou igual a $D/2$. Somente se os dois testes forem satisfeitos é que o movimento pode ser realizado. Por exemplo, para inclusão da aresta $[j, k]$ na Figura 6.2-(g), o primeiro teste consiste em averiguar se a profundidade do nó k adicionada a um mais a altura da sub-árvore com raiz em j é menor ou igual a $D/2$. O segundo teste consiste em verificar se a profundidade do nó k adicionada a dois mais a altura da sub-árvore com raiz no nó i é menor ou igual a $D/2$.

Existem $O\left(C^2_{|V|}\right)$ possibilidades de remoção de pares de arestas e $O(|V|)$ possibilidades para reconectar as sub-árvores. As estruturas de dados são atualizadas no pior caso em $O(|V|)$. Portanto, o custo de se avaliar todos os movimentos em uma passagem desta busca local é $O(|V|^3)$.

6.1.3 Vizinhança e busca local adoção de sub-árvore

Cada solução na vizinhança adoção de sub-árvore é obtida transferindo-se todos os filhos da raiz de uma sub-árvore para um de seus filhos. Após a transferência dos filhos, a antiga raiz da sub-árvore torna-se filho da nova raiz. As sub-árvores consideradas são apenas aquelas com raiz em um nó que possui grau de saída maior ou igual a dois. Neste tipo de movimento não existe possibilidade de violação de diâmetro. O único nó que desce um nível na árvore é a antiga raiz da sub-árvore. Entretanto, este nó torna-se uma folha da árvore no mesmo nível de profundidade de seus antigos filhos.

Exemplos de soluções vizinhas nesta vizinhança são dados na Figura 6.3. A Figura 6.3-(a) apresenta uma árvore viável inicial. A sub-árvore a ser avaliada é aquela com raiz em p. Então, existem três possibilidades neste caso, o nó i, j ou k pode herdar como filhos todos os antigos filhos de p. As árvores que representam estas possibilidades estão ilustradas respectivamente nas Figuras 6.3-(b), (c) e (d). Essas sub-árvores são consideradas soluções vizinhas da árvore inicial.

Na busca local desenvolvida para explorar essa vizinhança, avalia-se cada sub-árvore cuja raiz tenha grau de saída maior ou igual a dois. Para cada filho da raiz da sub-árvore considerada, é averiguado se realizar um movimento como descrito acima aprimora a solução. Dentre todas as possibilidades de movimentos em uma sub-árvore, realiza-se aquele que corresponde ao mais

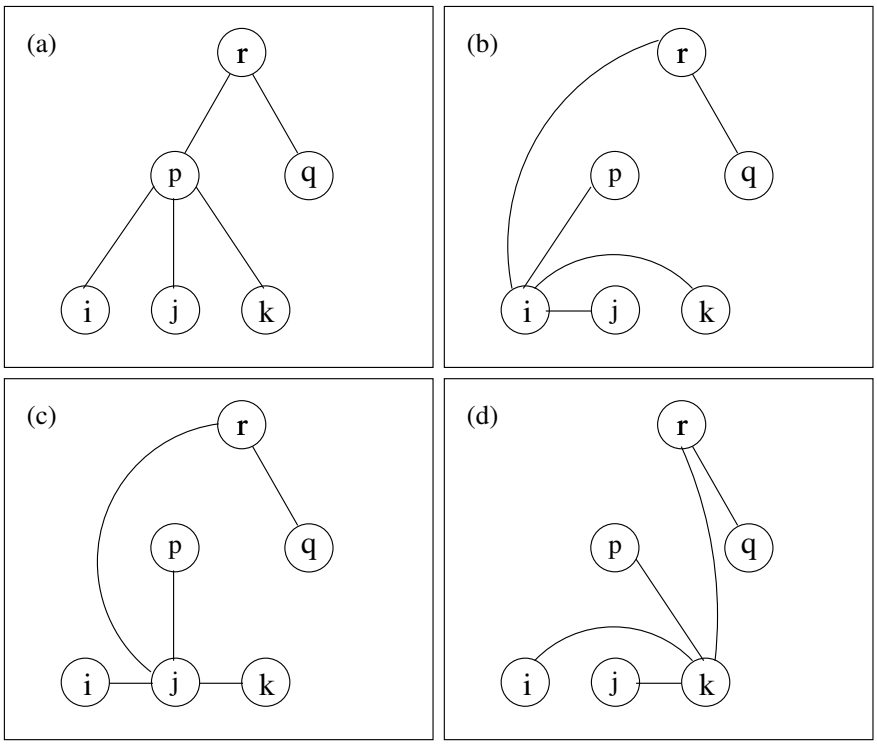

Figura 6.3: Exemplo de soluções vizinhas na vizinhança adoção de sub-árvore.

aprimorante.

Existem no máximo $O(|V|)$ nós que podem participar deste movimento e a variação de custo é calculada em $O(|V|)$. Para atualizar as estruturas de dados consome-se no pior caso $O(|V|)$. Portanto, o custo de se avaliar todos os movimentos em uma passagem desta busca local é no pior caso $O(|V|^2)$.

6.1.4 Vizinhança e busca local substituição de caminho

Soluções nesta vizinhança são obtidas a partir da substituição das duas arestas no caminho que se inicia no pai de uma sub-árvore até um dos filhos da raiz dessa sub-árvore. Este filho necessariamente participa do movimento. As sub-árvores consideradas são apenas aquelas com raiz em um nó que tem grau de saída maior ou igual a dois. Um outro filho da sub-árvore é tomado para participar do movimento. O novo caminho é formado por uma aresta que conecta os dois filhos considerados e outra que conecta um desses nós ao pai da sub-árvore. A antiga raiz da sub-árvore torna-se filho da nova raiz desta sub-árvore. Movimentos como este são realizados se a soma dos custos das arestas que entram na solução for menor do que a soma dos custos das arestas que saem da solução.

A Figura 6.4 apresenta um exemplo do esquema citado acima. Na

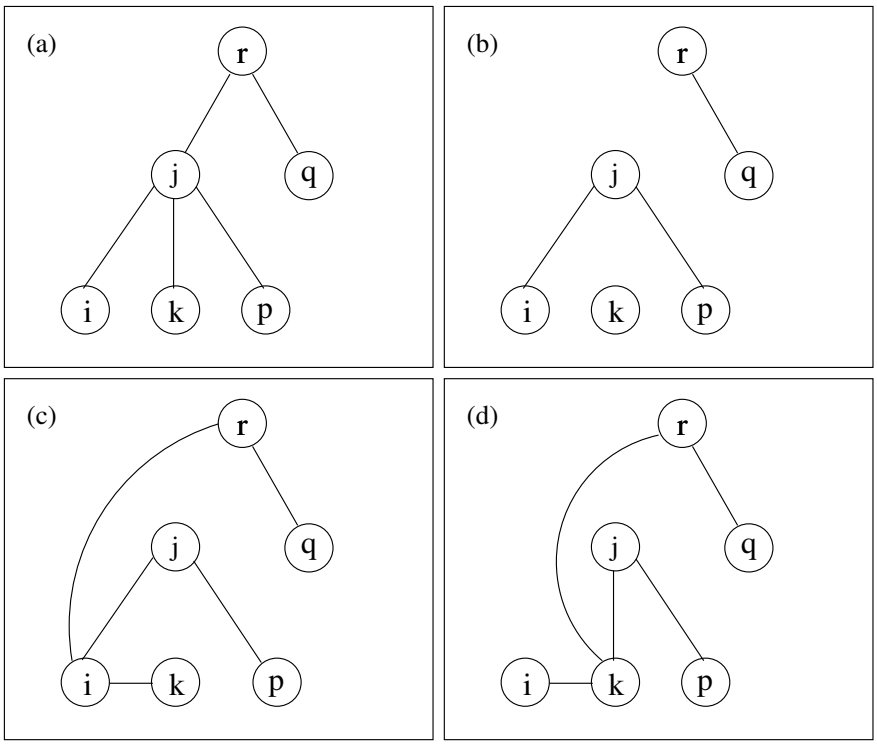

Figura 6.4: Exemplo de movimento da vizinhança substituição de caminho.

Figura 6.4-(a) encontra-se uma árvore viável inicial. A sub-árvore considerada para realização do movimento tem raiz no nó j. O caminho a ser substituído inicia-se no nó r, passa pelo nó j e termina no nó k. Na Figura 6.4-(b) as arestas neste caminho são eliminadas da solução. O primeiro filho da sub-árvore que participará necessariamente do movimento é o nó k. Considera-se que o segundo filho escolhido seja o nó i. No primeiro caso, ilustrado na Figura 6.4-(c), é testado se o novo caminho que se inicia no nó r, passa pelo nó i e termina no nó k aprimora a solução (a antiga raiz j da sub-árvore torna-se filho de i). No segundo caso, apresentado na Figura 6.4-(d), é averiguado se o novo caminho que se inicia no nó r, passa pelo nó k e termina em i aprimora a solução (a antiga raiz j da sub-árvore torna-se filho de k). Escolhe-se dentre essas duas soluções, aquela que gera a maior redução no custo em relação à solução corrente.

Neste tipo de movimento é preciso realizar apenas um teste de viabilidade envolvendo a antiga raiz da sub-árvore, já que este nó desce um nível de profundidade na árvore. Conseqüentemente, pode vir a aumentar o diâmetro da árvore. Quanto aos outros nós que participam do movimento, um sobe um nível de profundidade, enquanto que o outro permanece no mesmo nível de profundidade. A partir do exemplo da Figura 6.4-(c), pode-se observar que não é necessário realizar testes de violação de diâmetro para as sub-árvores com

raízes nos nós k e i. A sub-árvore com raiz no nó k permanece no mesmo nível de profundidade e a sub-árvore com raiz em i sobe um nível de profundidade na árvore. Um teste de viabilidade deve ser realizado na sub-árvore com raiz em j. Para que a solução permaneça viável, a nova profundidade do nó j mais a altura da sub-árvore com raiz em j deve ser menor ou igual a $D/2$. Satisfeita esta condição, o movimento pode ser realizado.

Existem $O\left(C^2_{|V|}\right)$ movimentos a serem avaliados deste tipo. O pior caso ocorre quando um nó tem um pai e todos os outros nós da árvore são seus filhos. As estruturas de dados são atualizadas no pior caso em $O(|V|)$. Portanto, o custo de se avaliar todos os movimentos em uma passagem desta busca local é no pior caso é $O(|V|^2)$.

6.2 Otimizações realizadas nas buscas locais

Algumas otimizações nas buscas locais foram realizadas objetivando reduzir a quantidade de testes a serem realizados.

Nas buscas locais 1-opt e 2-opt procura-se conectar sub-árvores à árvore principal. Testes para encontrar o nó na árvore principal para participar desta reconexão podem ser economizados. Se um movimento aplicado à raiz de uma sub-árvore não pode ser realizado porque viola o diâmetro, então nenhuma conexão com um nó desta sub-árvore precisa ser avaliada. Isto porque os nós desta sub-árvore estão em níveis de profundidade mais baixos do que o da sua raiz. Conseqüentemente, também violariam a restrição de diâmetro.

Na busca local adoção de sub-árvore, para calcular o custo de uma nova sub-árvore em um movimento, consome-me no pior caso $O(|V|)$, como citado anteriormente. A otimização inserida nesta operação consiste em contabilizar os custos das arestas somente enquanto o somatório desses custos não excede o custo da antiga sub-árvore.

6.3 Heurísticas baseadas em metaheurísticas

Heurísticas baseadas em metaheurísticas têm sido amplamente utilizadas para obter soluções de problemas de otimização combinatória NP-díficeis [46]. Neste tipo de estratégia, boas soluções (não necessariamente ótimas) são geradas, comumente, em um tempo computacional bastante inferior ao requerido para realizar uma enumeração completa. *Simulated anneling*, busca *tabu*, GRASP, VNS, algoritmos genéticos, times assíncronos e colônia de formigas são alguns exemplos de metaheurísticas presentes na literatura.

Nesta tese, duas heurísticas baseadas em metaheurísticas foram desenvolvidas. A primeira é do tipo GRASP Reativo e a segunda uma heurística

híbrida (assim chamada porque contém características de mais de uma metaheurística). A seguir, os fundamentos básicos das metaheurísticas utilizadas são apresentados.

A metaheurística GRASP (*Greedy Randomized Adaptative Search Procedures*) foi proposta em [22]. GRASP é um processo iterativo, onde cada iteração é independente das demais e consiste de duas fases: a construtiva e a de busca local. A melhor solução global é guardada como resultado.

O Algoritmo 9 ilustra um procedimento GRASP padrão para um problema de minimização. O procedimento retorna um ótimo local S^*. A variável auxiliar $custo^*$ guarda o custo da melhor solução conhecida.

```
   Algoritmo: GRASP_padrão()
   Entrada: critério de parada, semente
   Saída: S*
 1 custo* ← ∞;
 2 enquanto (critério de parada não satisfeito) faça
 3 |   S ← Construção(semente);
 4 |   S' ← Busca_local(S);
 5 |   se (custo(S') < custo*) então
 6 |   |   S* ← S';
 7 |   |   custo* ← custo(S*);
 8 |   fim
 9 fim
10 retorna S*;
```

Algoritmo 9: Pseudo-código GRASP padrão para um problema de minimização.

Na linha 1, a variável $custo^*$ é inicializada. O laço das linhas de 2 a 9 é repetido até que o *critério de parada* seja satisfeito. Na linha 3, uma solução S é construída através de um algoritmo guloso aleatorizado. Na linha 4, uma busca local é aplicada à solução S, obtendo-se uma solução S'. Na linha 5, é averiguado se o custo da solução S' é menor do que $custo^*$. Se for, a solução S^* e a variável $custo^*$ são atualizadas respectivamente nas linhas 6 e 7. O ótimo local S^* é retornado na linha 10.

GRASP tem sido utilizado juntamente com outras técnicas existentes na literatura, por exemplo memórias de curto e longo prazos [59]. GRASP Reativo foi proposto em [55] e inclui um mecanismo de aprendizagem na fase construtiva. Os detalhes desta estratégia foram apresentados na Seção 5.1.

O segundo algoritmo baseado em metaheurísticas proposto nesta tese utiliza os fundamentos da metaheurística ILS (*Iterated Local Search*), além das idéias provenientes das estratégias GRASP e VND. ILS contrói soluções iterativamente através da aplicação de perturbações e busca local a uma solução

viável. A qualidade das soluções depende da solução inicial, da perturbação, da busca local e do critério de aceitação usados [48]. O critério de aceitação determina quando uma solução substitui a solução corrente.

O Algoritmo 10 ilustra um procedimento ILS padrão.

```
    Algoritmo: ILS_padrão()
    Entrada: critério de parada, semente
    Saída: S*
 1  S ← Gera_solução();
 2  S ← Busca_local(S);
 3  custo* ← custo(S);
 4  enquanto (critério de parada não satisfeito) faça
 5  |  S' ← Perturbação(S);
 6  |  S' ← Busca_local(S');
 7  |  S ← Critério_de_aceitação(S, S');
 8  |  se (custo(S') < custo*) então
 9  |  |  S* ← S';
10  |  |  custo* ← custo(S*);
11  |  fim
12  fim
13  retorna S*;
```

Algoritmo 10: Pseudo-código ILS padrão para um problema de minimização.

Na linha 1, uma solução inicial S é gerada. Em seguida, na linha 2, uma busca local é aplicada à solução S. A variável $custo^*$, inicializada na linha 3, guarda o valor da melhor solução. O laço das linhas de 4 a 12 é repetido até que o *critério de parada* seja satisfeito. Uma perturbação é aplicada à solução S na linha 5, obtendo-se S'. Posteriormente, na linha 6, uma busca local tenta aprimorar a solução S'. Se o ótimo local S' satisfaz o critério de aceitação, então ocorre uma migração para esta solução na linha 7. Na linha 8, é averiguado se o custo de S' é melhor do que $custo^*$. Se for, então a solução S^* e a variável $custo^*$ são atualizadas respectivamente nas linhas 9 e 10. O procedimento retorna S^* na linha 13.

As buscas locais apresentadas na Seção 6.1 são exploradas em um procedimento parecido com *Variable Neighborhood Descent* (VND). Procedimentos do tipo VND consistem em explorar completamente uma vizinhança até que um ótimo local seja encontrado. Em seguida, passa-se para uma outra vizinhança. Se uma solução melhor do que a melhor conhecida for encontrada, em alguma das vizinhanças disponíveis, então retorna-se para a primeira vizinhança [36, 37].

6.4 Um algoritmo GRASP para AGMD

O procedimento do tipo GRASP desenvolvido para AGMD utiliza na fase construtiva a heurística OTT-M2. Esta heurística não garante a obtenção de soluções iniciais viáveis para grafos esparsos. Para tratar este problema, as arestas que não existem neste tipo de instâncias foram introduzidas com custo M muito elevado (chamadas de falsas arestas). A solução inicial é gerada como se o grafo fosse completo. Na fase de busca local, um procedimento parecido com VND é aplicado para explorar as buscas locais adoção de sub-árvore, 1-opt, substituição de caminho e 2-opt, nesta ordem.

O pseudo-código da heurística GRASP desenvolvida encontra-se ilustrado no Algoritmo 11. Este procedimento é executado até que o critério de parada estabelecido seja atingido. O tempo de processamento ou a quantidade de iterações são os critérios que podem ser utilizados. O que determinará o critério de parada a ser utilizado é o objetivo que se pretende na execução do algoritmo. Um filtro é utilizado para controlar a quantidade de falsas arestas autorizadas em uma solução inicial. Se a solução gerada na fase construtiva possui mais de 1/3 de falsas arestas entre as $|V|-1$ arestas da árvore, então a solução é descartada. O procedimento retorna um ótimo local S^*.

```
    Algoritmo: GRASP_AGMD()
    Entrada: G=(V,E), critério de parada, semente
    Saída: S*
 1  filtro ← [(|V| − 1)/3] · M;
 2  custo* ← ∞;
 3  enquanto (critério de parada não satisfeito) faça
 4  |  repita
 5  |  |  S ← OTT-M2(semente);
 6  |  até (custo(S) ≤ filtro);
 7  |  S' ← Busca_local(S);
 8  |  se (custo(S') < custo*) então
 9  |  |  S* ← S';
10  |  |  custo* ← custo(S*);
11  |  fim
12  fim
13  retorna S*;
```

Algoritmo 11: Procedimento GRASP para AGMD.

A variável $filtro$ é inicializada na linha 1. O valor de $[(|V|-1)/3] \cdot M$ atribuído à variável $filtro$ indica o custo relativo a 1/3 de falsas arestas em uma árvore. Na linha 2, a variável $custo^*$ é inicializada. O laço das linhas de 3 a 12 é repetido até que o o critério de parada estabelecido seja alcançado. O

laço das linhas de 4 a 6 é repetido até que uma solução com custo menor do que o valor $filtro$ seja produzida. Embora, teoricamente, este laço possa ser infinito, os testes realizados e apresentados no Capítulo 7 indicam que este tipo de situação, na prática, não ocorre com as instâncias esparsas utilizadas nesta tese. Na linha 5, uma solução é construída utilizando a heurística OTT-M2, sendo armazenada em S. Na linha 7, uma busca local é aplicada à solução S. O pseudo-código do procedimento de busca local utilizado é apresentado no Algoritmo 12. Se o custo de S' é menor do que $custo^*$ (linha 8), então a solução S^* e a variável $custo^*$ são atualizadas respectivamente nas linhas 9 e 10. A solução S^* é retornada na linha 13.

O procedimento de busca local desenvolvido é ilustrado no pseudo-código do Algoritmo 12 e difere do VND porque não se retorna para a primeira vizinhança quando é encontrada uma solução melhor do que a melhor conhecida.


```
    Algoritmo: Busca_local()
    Entrada: G = (V, E), S
    Saída: S*
 1  custo* ← ∞;
 2  repita
 3  |  repita
 4  |  |  mudou ← falso;
 5  |  |  S ← adoção de sub-árvore(S) ;
 6  |  |  S ← 1-opt(S);
 7  |  |  S ← substituição de caminho(S);
 8  |  |  se (custo(S) < custo*) então
 9  |  |  |  S* ← S;
10  |  |  |  custo* ← custo(S*);
11  |  |  |  mudou ← verdadeiro;
12  |  |  fim
13  |  até (mudou = falso);
14  |  S ← 2-opt(S);
15  |  se (custo(S) < custo*) então
16  |  |  S* ← S;
17  |  |  custo* ← custo(S*);
18  |  |  mudou ← verdadeiro;
19  |  fim
20  até (mudou = falso);
21  retorna S*;
```

Algoritmo 12: Procedimento de busca local para AGMD.

Este procedimento recebe como parâmetros uma solução S e retorna um ótimo local S^*. A variável *mudou* é *verdadeira* quando uma melhoria na solução corrente é obtida após a aplicação das buscas locais.

Na linha 1, a variável $custo^*$ é inicializada. O laço das linhas de 2 a 20 é executado até que as buscas não consigam mais sair do ótimo local corrente. Na linha 3, a variável *mudou* é inicializada. O laço das linhas de 4 a 13 é interrompido quando nenhum aprimoramento é obtido para a solução utilizando-se as buscas locais adoção de sub-árvore, 1-opt e substituição de caminho. As buscas locais adoção de sub-árvore, 1-opt e substituição de caminho são aplicadas à solução S respectivamente nas linhas 5 a 7. Na linha 8, é averiguado se o custo da solução S é menor do que $custo^*$. Se for, a solução S^* e as variáveis $custo^*$ e *mudou* são atualizadas respectivamente nas linhas 9 a 11. A busca local 2-opt é então aplicada à solução S na linha 14. Se o custo de S for menor do que $custo^*$ (linha 15), então a solução S^* e as variáveis $custo^*$ e *mudou* são atualizadas respectivamente nas linhas 16 a 18. A solução S^* é retornada na linha 21.

Optou-se por utilizar a busca local 2-opt após a aplicação das buscas adoção de sub-árvore, 1-opt e substituição de caminho porque, nos experimentos realizados, observou-se que a busca local 2-opt consegue sair dos ótimos locais provenientes daquelas buscas.

6.5 Um algoritmo híbrido para AGMD

O algoritmo híbrido desenvolvido é baseado nas metaheurísticas GRASP e ILS. A heurística OTT-M2 é utilizada para a geração de soluções iniciais e a busca local é aquela apresentada no Algoritmo 12. Dois tipos de perturbações são propostas nesta tese.

Na primeira perturbação, um nó filho do nó central da árvore (caso par) é considerado o novo centro da árvore. No caso ímpar, um dos nós filhos de uma das extremidades da aresta central torna-se uma extremidade da aresta central. Esta perturbação é chamada de deslocamento de centro para um vizinho (DCV).

A Figura 6.5 ilustra um exemplo de movimento DCV. Neste caso, o diâmetro da árvore deve ser menor ou igual a quatro. A árvore inicial é apresentada na Figura 6.5-(a) e l é o nó central. Os níveis de profundidade de cada nó em relação ao nó central l encontram-se indicados por números próximos ao respectivo nó. O nó j torna-se o centro da árvore como apresentado na Figura 6.5-(b). Ao realizar este deslocamento, os níveis de profundidade de todos os nós são atualizados. Os nós cuja profundidade é maior que $D/2$ são retirados da solução e inseridos novamente para que a solução torne-se viável com o novo nó central j. Por exemplo, os nós u, p, v passaram a ter profundidades maiores que $D/2$. Neste caso, as arestas com extremidades nesses nós são eliminadas da solução, como ilustrado na Figura 6.5-(c).

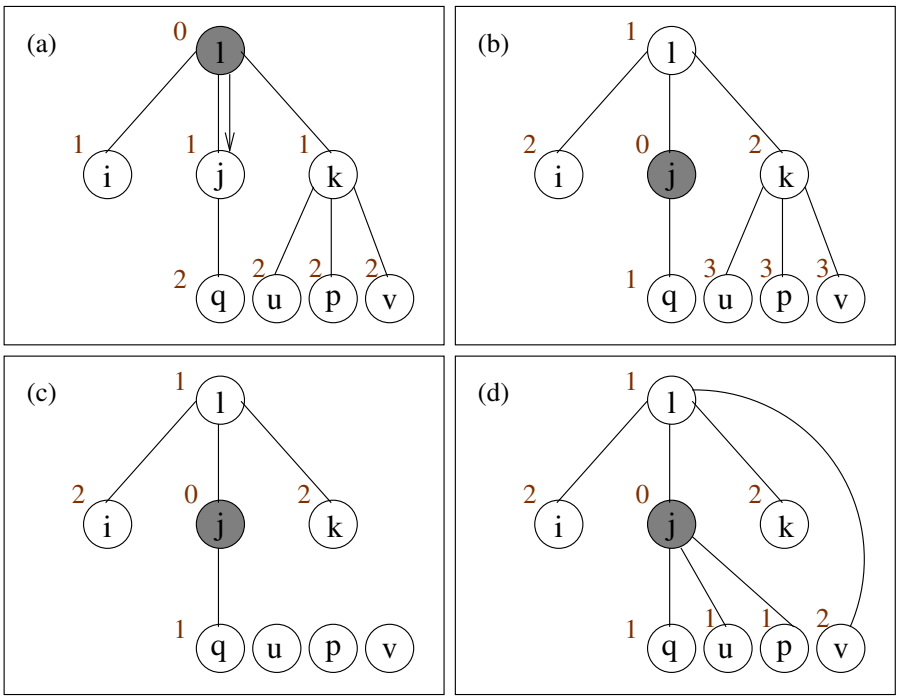

Figura 6.5: Exemplo da perturbação DCV.

Posteriormente, esses nós devem ser conectados a custo mínimo, de modo que a solução permaneça viável. Considera-se que as arestas $[j, u]$, $[j, p]$ e $[l, v]$ satisfazem estas condições. Então, estas arestas passam a compor a solução como ilustrado na Figura 6.5-(d). Quando não ocorre nenhuma violação de diâmetro, ainda assim este tipo de movimento pode provocar uma mudança considerável na solução. Isto porque a alteração no nível de profundidade dos nós tem por conseqüência a liberação para inclusão de diversas arestas que não podiam entrar na solução anteriormente, porque provocariam violação da restrição de diâmetro.

No segundo tipo de perturbação, um nó é escolhido aleatoriamente para ser o novo centro da árvore no caso par (resp. substituir uma das extremidades da aresta central no caso ímpar). Todos os filhos do antigo nó central no caso par (resp. de uma das extremidades da antiga aresta central no caso ímpar) são herdados pelo novo nó. O antigo nó central é reconectado à árvore a custo mínimo, de modo que a solução permaneça viável. Esta perturbação é chamada de substituição aleatória de raiz (SAR).

A Figura 6.6 ilustra um exemplo da SAR. A árvore inicial é apresentada na Figura 6.5-(a), cujo nó central é l. O nó p foi escolhido aleatoriamente para tornar-se o novo centro da árvore. Na Figura 6.6-(b), o nó p herda como filhos todos os antigos filhos de l. Em seguida, o nó l é reconectado à árvore, por exemplo, como ilustrado na Figura 6.6-(c).

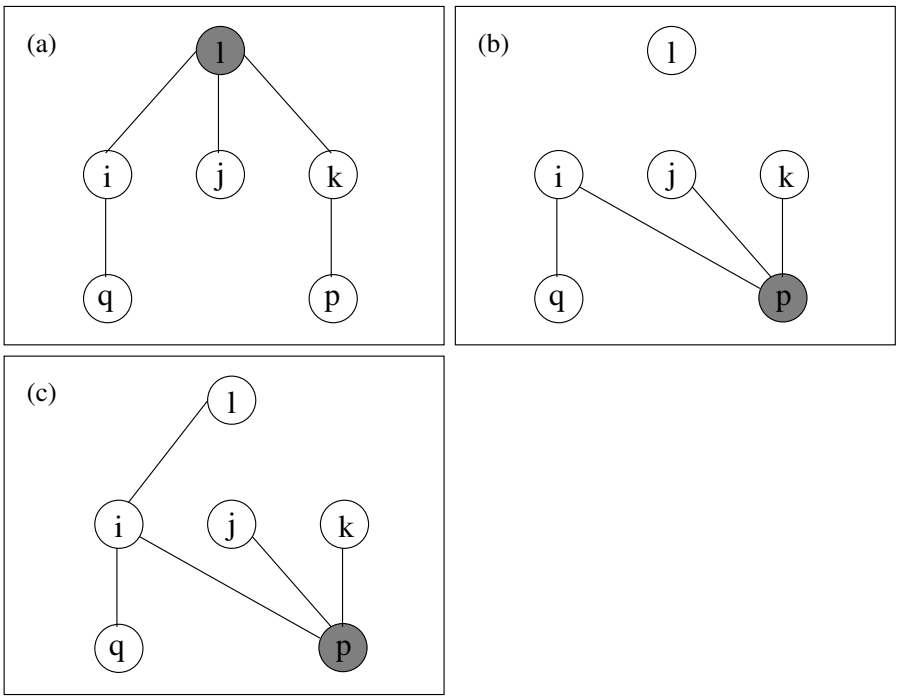

Figura 6.6: Exemplo da perturbação SAR.

No algoritmo híbrido desenvolvido, uma solução inicial é gerada utilizando-se a heurística OTT-M2. Em seguida, a busca local apresentada no Algoritmo 12 é aplicada à solução inicial gerada. As perturbações DCV e SAR são aplicadas alternadamente ao longo das iterações à solução proveniente da busca local. Em seguida, a busca local é aplicada novamente. Se a nova solução for melhor do que a corrente, então a solução é aceita e assim sucessivamente. Para interromper a aplicação da perturbação e da busca local, estabeleceu-se um limite para parar este processo e gerar uma nova solução inicial utilizando-se a heurística OTT-M2. Se a perturbação a ser aplicada é DCV, então o limite máximo de perturbações a serem aplicadas deste tipo em uma solução é igual ao grau de saída do antigo nó central no caso par ou da extremidade da aresta central a ser substituída no caso ímpar. Quando a perturbação é do tipo SAR, o limite máximo de perturbações a serem aplicadas em uma solução é $|V|/3$.

Dois filtros são utilizados neste algoritmo. O primeiro filtro controla a quantidade de falsas arestas, como no algoritmo do tipo GRASP descrito anteriormente. O segundo filtro controla a qualidade das soluções provenientes de uma perturbação a serem exploradas pela busca local. O objetivo deste filtro é especificar em quais soluções pretende-se consumir tempo de pesquisa. Uma solução gerada a partir de uma perturbação só é submetida à busca local se seu custo encontra-se a no máximo 15% do valor da melhor solução conhecida. A medida que a pesquisa evolui e torna-se mais difícil aprimorar a

melhor solução conhecida, então este filtro é relaxado. Sempre que o algoritmo permanece uma determinada quantidade de iterações sem aprimorar a melhor solução conhecida, então o filtro é incrementado em 2%.

O pseudo-código do algoritmo híbrido desenvolvido é apresentado no Algoritmo 13. Este procedimento é executado até que um *critério de parada* estabelecido seja atingido, retornando uma solução S^*. Dois critérios de parada podem ser utilizados: o tempo ou a quantidade de iterações. A variável auxiliar $custo^*$ é inicializada na linha 1. A variável *tipo*, inicializada na linha 2, identifica qual a perturbação a ser executada em uma dada iteração. As variáveis $filtro$ e $filtro\_bl$ são inicializadas respectivamente nas linhas 3 e 4.

O laço das linhas de 5 a 43 é executado até que o *critério de parada* estabelecido seja alcançado. No laço das linhas de 6 a 8, uma solução é gerada utilizando a heurística OTT-M2. Se o custo da solução gerada na linha 7 é maior que o valor de $filtro$, então uma outra solução S é gerada. Como explicado anteriormente, este laço pode teoricamente ser infinito. Entretanto, na prática, este tipo de situação não ocorreu com as instâncias esparsas utilizadas nesta tese. Na linha 9, a busca local apresentada no Algoritmo 12 é aplicada à solução S. Se o custo de S é menor do que $custo^*$ (linha 10), então a solução S^* e a variável $custo^*$ são atualizadas respectivamente nas linhas 11 e 12. A variável *contador* contabiliza a quantidade de vezes que uma solução é produzida pelo algoritmo sem aprimorar a melhor solução conhecida. Se uma solução melhor do que a melhor conhecida é encontrada, então *contador* é atualizado com valor zero na linha 13. Caso contrário, *contador* é incrementado na linha 15.

Na linha 17, é estabelecida a quantidade de vezes que será aplicada um determinado tipo de perturbação. Vale ressaltar que se perturbação a ser aplicada é do tipo DCV, então o limite é atualizado sempre que ocorre a aceitação de uma nova solução proveniente da perturbação. Como o critério de aceitação é a melhoria no custo da solução corrente e esta solução não pode ser aprimorada indefinidamente, então este fato limita indiretamente a quantidade de vezes que este tipo de perturbação é aplicada.

O laço das linhas de 18 a 36 é repetido enquanto $maxPert$ for maior que zero. Uma perturbação é aplicada à solução S na linha 19 e a solução modificada é armazenada em S'. O desvio relativo $(custo(S') - custo^*)/custo^*$ é calculado a partir dos custos da melhor solução conhecida e da solução S' na linha 20. Para que a busca local seja aplicada à solução proveniente da perturbação, *desvio* deve ser menor ou igual ao valor de $filtro\_bl$ (linha 21). Se for, a busca local apresentada no Algoritmo 12 é aplicada à solução S' na linha 22. Se S' é melhor do que S (linha 23), então a solução S' é aceita como solução corrente e S é atualizada na linha 24. Na linha 25, é averiguado se a

```
Algoritmo: Híbrido_AGMD()
Entrada: G=(V,E), critério de parada, semente
Saída: S*
custo* ← ∞;
tipo ← DCV;
filtro ← [(|V| − 1)/3] · M;
filtro_bl ← 0,15;
enquanto (critério de parada não satisfeito) faça
    repita
        S ← OTT-M2(semente);
    até (custo(S) ≤ filtro);
    S ← Busca_local(S);
    se (custo(S) < custo*) então
        S* ← S;
        custo* ← custo(S*);
        contador ← 0;
    senão
        contador ← contador + 1;
    fim
    inicializar maxPert;
    enquanto (maxPert > 0) faça
        S' ← Perturbação(tipo, S);
        desvio ← (custo(S') − custo*)/custo*;
        se (desvio ≤ filtro_bl) então
            S' ← Busca_local(S');
            se (custo(S') < custo(S)) então
                S ← S';
                se (custo(S') < custo*) então
                    S* ← S';
                    custo* ← custo(S*);
                    contador ← 0;
                senão
                    contador ← contador + 1;
                fim
                atualizar maxPert;
            fim
        fim
        maxPert ← maxPert − 1;
    fim
    se (contador ≥ 100) então
        filtro_bl ← filtro_bl + 0,02;
        contador ← 0;
    fim
    se (tipo = DCV ) então tipo ← SAR;
    senão tipo ← DCV;
fim
retorna S*;
```

Algoritmo 13: Heurística híbrida para AGMD.

nova solução S' é melhor do que a melhor conhecida. Se for, a solução S^* e as variáveis $custo^*$ e *contador* são atualizadas respectivamente nas linhas 26 a 28. Senão (linha 29), *contador* é incrementado na linha 30. Se uma nova solução foi aceita e a perturbação aplicada na iteração é a DCV, então a quantidade máxima de perturbações $maxPert$ é atualizada na linhas 32. A variável $maxPert$ é decrementada na linha 35.

Nas linhas 37 a 40 é averiguado se o algoritmo gerou mais de 100 soluções sem aprimorar a melhor solução conhecida. Neste caso, o segundo filtro é relaxado na linha 38 e *contador* atualizado na linha 39. A variável *tipo* é atualizada nas linhas 41 e 42. Como as perturbações são executadas alternadamente, se em uma dada iteração *tipo* é igual a DCV, então na próxima iteração a perturbação SAR será utilizada (linha 41). Caso contrário, a perturbação a ser executada é a DCV (linha 42). O procedimento retorna um ótimo local S^* na linha 44.

Os experimentos computacionais realizados com os algoritmos do tipo GRASP e híbrido desenvolvidos para AGMD são relatados no Capítulo 7.