3
Algoritmo de Busca Local

Um algoritmo de busca local define, para cada solução, uma vizinhança composta por um conjunto de soluções com características "muito próximas". Dada uma solução corrente, uma das formas de implementar um algoritmo de busca local é percorrer a vizinhança dessa solução em busca de outra com valor menor (para um problema de minimização). Se tal solução vizinha for encontrada, torna-se a nova solução corrente e o algoritmo continua. Caso contrário, a solução corrente é um ótimo local em relação à vizinhança adotada.

A abordagem utilizada no algoritmo de busca local decompõe o problema das p-medianas conectadas em dois sub-problemas: localizar p facilidades nos vértices do grafo, com os usuários servidos pela facilidade aberta mais próxima (SP-1) e conectar as p facilidades abertas por uma árvore de Steiner (SP-2). Para resolver SP-1 e SP-2, respectivamente, utiliza-se o algoritmo de busca local do problema das p-medianas, cuja implementação foi proposta por Resende e Werneck [47] e a heurística construtiva gulosa de Takahashi e Matsuyama (Prim) [57].

O algoritmo de busca local básico é inicializado através de uma heurística construtiva. A vizinhança é definida como o conjunto de soluções alcançadas pela troca de uma facilidade aberta (presente na solução) por outra fechada (que não faz parte da solução). Em cada iteração, o algoritmo procura trocar uma facilidade pertencente à solução por outra que não pertence à solução, avaliando essa troca em termos do custo de atendimento dos usuários. As facilidades abertas são conectadas por uma árvore de Steiner (custo de interconexão das facilidades). Caso ocorra uma diminuição no valor da função objetivo, efetua-se a troca e tem-se uma nova solução corrente. O algoritmo continua até que nenhum melhoramento aconteça, permanecendo assim em um ótimo local.

Porém, conectar as p facilidades abertas através de uma árvore de Steiner a cada nova troca é uma rotina extremamente custosa computacionalmente. Para acelerar o algoritmo, propõe-se uma estratégia híbrida que concatena duas variantes da busca local básica, incorporando idéias como a estratégia de busca por melhoria iterativa, circularidade e um teste simples com o objetivo de descartar vizinhos, evitando-se a execução da heurística de Steiner a cada nova troca.

Como no problema das p-medianas conectadas cada vértice é simultaneamente uma potencial localização de facilidade e um usuário, tem-se que $m = n$, onde m representa o número de possíveis localizações de facilidades e n representa o número de usuários. Quando necessário, a distinção será mantida.

Esse capítulo está organizado como se segue: a Seção 3.1 apresenta a heurística construtiva utilizada como solução inicial nos algoritmos de busca local. A Seção 3.2 descreve o algoritmo básico de busca local para o problema das p-medianas conectadas. A Seção 3.3 mostra as descrições das três versões da busca local básica e o estudo computacional comparando essas versões com o algoritmo básico. A Seção 3.4 apresenta a estratégia híbrida (busca local concatenada) e os resultados computacionais em relação às demais buscas locais. Por último, a Seção 3.5 apresenta as considerações finais apontando a melhor busca local para o problema das p-medianas conectadas em termos de qualidade das soluções encontradas e dos tempos de processamento obtidos nas instâncias testadas.

3.1 Solução Inicial dos Algoritmos de Busca Local

Heurísticas construtivas são algoritmos que criam soluções viáveis. Podem ser utilizadas de maneira isolada, contudo, são freqüentemente utilizadas em métodos mais elaborados como algoritmos de busca local, metaheurísticas e algoritmos exatos.

Para a geração das soluções iniciais dos algoritmos de busca local, o sub-problema SP-1 será resolvido utilizando-se a heurística Sample Greedy, uma variante da heurística construtiva clássica para o problema das p-medianas e o sub-problema SP-2 será resolvido utilizando-se a heurística Prim, uma extensão direta do algoritmo exato para o problema da árvore geradora mínima adaptada para o problema de Steiner em grafos. Essas heurísticas foram escolhidas devido aos bons resultados alcançados nos problemas das p-medianas e no problema de Steiner [48, 61]. Assim, pretende-se resolver cada um dos sub-problemas separadamente, combinando-se as suas soluções.

A Figura 3.1 descreve o algoritmo, onde $C(f)$ representa o acréscimo obtido no valor da função objetivo relativo ao custo de atendimento dos usuários ao adicionar a facilidade $f \in V$ à solução sendo construída. A função recebe como parâmetros a *semente* usada para inicializar o gerador de números aleatórios, o número de vértices do grafo e o número de facilidades a serem instaladas.

A heurística construtiva clássica para o problema das p-medianas constrói uma solução iterativamente, selecionando as facilidades seqüencialmente, até que p facilidades tenham sido escolhidas, minimizando o aumento na função objetivo em cada passo. A heurística Sample Greedy (linhas 1 a 16) seleciona

Função SampleGreedy_Prim (*semente*, n, p)
1. $S \leftarrow \emptyset$;
2. Inicialize o conjunto V das possíveis localizações de facilidades;
3. $q \leftarrow \lceil \log_2(n/p) \rceil$;
4. **Enquanto** $(|S| \neq p)$ **Faça**
5. $\quad k \leftarrow 0, Q \leftarrow \emptyset, V_{aux} = V$;
6. $\quad$ **Enquanto** $(k \neq q)$ **Faça**
7. $\quad\quad$ Selecione f aleatoriamente de V_{aux};
8. $\quad\quad$ Avalie o acréscimo $C(f)$;
9. $\quad\quad$ $Q \leftarrow Q \cup \{f\}$;
10. $\quad\quad$ $V_{aux} \leftarrow V_{aux} \setminus \{f\}$;
11. $\quad\quad$ $k \leftarrow k + 1$;
12. $\quad$ **Fim-Enquanto**
13. $\quad$ $f \leftarrow \text{argmin}\{C(f), f \in Q\}$;
14. $\quad$ $S \leftarrow S \cup \{f\}$;
15. $\quad$ $V \leftarrow V \setminus \{f\}$;
16. **Fim-Enquanto**
17. Inicie com uma sub-árvore $T_1(S)$ consistindo de um terminal $f \in S$ escolhido aleatoriamente;
18. $k \leftarrow 1, X_k \leftarrow X_k \cup \{f\}$;
19. **Enquanto** $(k < p)$ **Faça**
20. $\quad$ Determine um terminal $f \in S \setminus X_k$ mais próximo de $T_k(S)$; Construa uma árvore $T_{k+1}(S)$ adicionando o caminho mínimo entre f e $T_k(S)$;
21. $\quad$ $k \leftarrow k + 1, X_k \leftarrow X_k \cup \{f\}$;
22. **Fim-Enquanto**
23. $T(S) \leftarrow T_p(S)$;
24. **Retorne** S;

Fim

Figura 3.1: Algoritmo que gera as soluções iniciais das buscas locais.

aleatoriamente, em cada uma das p iterações, $q < n$ possíveis localizações de facilidades (linhas 5 a 12), escolhendo aquela que minimiza o aumento na função objetivo relativo ao custo de atendimento dos usuários (linha 13). A idéia é fazer q suficientemente pequeno para reduzir o tempo de execução em comparação ao algoritmo puramente guloso, mantendo um certo grau de aleatorização. Como em [48], utiliza-se $q = \lceil \log_2(n/p) \rceil$ (linha 3). A complexidade de pior caso da heurística Sample Greedy para o problema das p-medianas é $O(m + pqn)$ [48]. No problema das p-medianas conectadas tem-se $m = n$, então a complexidade de pior caso da heurística que resolve SP-1 (linhas 1 a 16) é $O(pqn)$.

A heurística Prim (linhas 17 a 23) é um dos algoritmos mais eficientes e mais utilizados na literatura para o cálculo de soluções aproximadas gulosas [52]. Inicia-se o algoritmo com uma sub-árvore $T_1(S)$ e um conjunto X contendo um terminal qualquer (linhas 17 e 18). Em cada uma das $p - 1$ iterações

(linhas 19 a 22), o algoritmo determina qual terminal $f \in S \backslash X$ possui distância mínima à sub-árvore corrente T_k, adicionando à mesma o terminal e os vértices e arestas no caminho entre ele e a sub-árvore (linha 20). Na linha 21 atualiza-se o número de iterações e o conjunto X_k. Ao final das iterações, o algoritmo conectará os p vértices terminais e possíveis não-terminais através de uma árvore de Steiner (linha 23). A complexidade de pior caso da heurística Prim depende da estrutura de dados utilizada. Utilizando-se um *heap* de Fibonacci (resp. *heap* binário), a complexidade é $O(p(|E|+n\log n))$ (resp. $O(p\,|E|\log n)$), onde $|E|$ corresponde ao número de arestas do grafo. Não utilizando um *heap*, a complexidade é $O(pn^2)$. Na tese foi utilizada a estrutura de dados *heap* binário. Essa implementação é bastante eficiente e, na prática, o fator p da complexidade de pior caso da heurística não se manifesta [61].

Por último, na linha 24, o algoritmo retorna uma solução definida pelo conjunto S formado pelas p facilidades abertas e pelo conjunto $T(S)$ de arestas obtidas quando se conecta estas facilidades por uma árvore de Steiner.

A complexidade de pior caso do algoritmo da Figura 3.1 é $O(p\,|E|\log n)$, dominada pela complexidade da heurística Prim que resolve o sub-problema SP2 (linhas 17 a 23).

3.2 Algoritmo Básico de Busca Local

O algoritmo básico de busca local para o problema das p-medianas conectadas utiliza o algoritmo de busca local clássico para o problema das p-medianas, juntamente com a heurística Prim.

O algoritmo de busca local para o problema das p-medianas, proposto por Teitz e Bart [58], é baseado na troca de facilidades. Dada uma solução composta por p facilidades, uma iteração do algoritmo determina, para cada facilidade que não está presente na solução, qual facilidade presente na solução (se houver) melhora ao máximo o valor da função objetivo se ambas forem trocadas. Se tal movimento existe, uma nova iteração será realizada a partir desta nova solução. Caso contrário, o algoritmo termina com um mínimo local sendo encontrado. A complexidade de pior caso do algoritmo é $O(pmn)$ por iteração.

Whitaker [62] descreveu uma implementação eficiente desse método, denominado de *Fast Interchange*, com complexidade $O(mn)$ por iteração. Resende e Werneck [47] propuseram uma implementação alternativa utilizando as mesmas operações básicas usadas por Whitaker em uma ordem diferente e com a mesma complexidade de pior caso. Apesar de requerer espaço de memória superior à heurística de Whitaker, acelerações significativas foram obtidas na prática em termos de tempo de processamento de até três ordens de magnitude nas instâncias testadas em relação à *Fast Interchange* [47].

O algoritmo de busca local proposto por Resende e Werneck foi modificado para se adaptar ao problema das p-medianas conectadas, onde as operações relacionadas à localização das p facilidades e à associação dos usuários às facilidades abertas mais próximas são realizadas pelo algoritmo de busca local para o problema das p-medianas [47], sendo que, a cada troca, as p facilidades abertas são conectadas via heurística de Prim.

3.2.1 Descrição do Algoritmo

A primeira e a segunda facilidade mais próxima do usuário $u \in V$ serão denotadas por ϕ_1^u e ϕ_2^u, respectivamente; d_1^u e d_2^u representam a distância do usuário u às facilidades ϕ_1^u e ϕ_2^u, respectivamente; a solução corrente é definida pelo conjunto S formado pelas p facilidades abertas e pelo conjunto $T(S)$ de arestas obtidas quando se conecta estas facilidades por uma árvore de Steiner através da heurística Prim; $F(S)$ é o custo da solução S e $F_T(S)$ é o custo da árvore de Steiner calculada acima. Representa-se por $i \notin S$ (resp. $r \in S$) uma facilidade candidata a inserção (resp. remoção) no (resp. do) conjunto de p facilidades abertas; uma solução vizinha $S^{'}$ da solução corrente S é obtida trocando-se uma facilidade candidata a remoção $r \in S$ por uma facilidade candidata a inserção $i \notin S$.

A descrição do algoritmo de busca local será baseada na descrição do algoritmo para o problema das p-medianas proposto em [47], realçando-se, quando necessário, as adaptações realizadas. Com o objetivo de acelerar passos posteriores do algoritmo, resultados parcias obtidos são armazenados em estruturas de dados auxiliares (*ganho*, *perda* e *extra*).

Dada uma solução S, para cada facilidade i candidata a inserção, define-se $ganho(i)$ como o total economizado quando i é adicionada a solução, não ocorrendo remoção de facilidade alguma. A economia obtida ao associar cada usuário a i cuja facilidade mais próxima estava mais distante do que i é dada por

$$ganho(i) = \sum_{u \in V} \max\{0, d_1^u - d_{ui}\}. \tag{3-1}$$

Similarmente, para cada facilidade r candidata a remoção, define-se $perda(r)$ como o aumento no valor da solução, resultante da remoção de r de S, não ocorrendo inserção de facilidade alguma. Este é o custo de transferir cada usuário associado a r à sua segunda facilidade mais próxima

$$perda(r) = \sum_{u:\phi_1^u = r} (d_2^u - d_1^u). \tag{3-2}$$

O cálculo de $perda(r)$ assume que o usuário u sempre será transferido da facilidade candidata a remoção r para a sua segunda facilidade mais próxima ϕ_2^u, sem levar em consideração a entrada da facilidade candidata a inserção i.

Porém, ocorrerá situações em que o usuário u estará mais próximo de i do que de ϕ_2^u. Assim, algumas parcelas no cálculo de $perda(r)$ precisam ser corrigidas. A estrutura de dados $extra(i, r)$ é responsável por essas correções. Para a sua definição, observa-se que, para cada usuário u, tem-se os seguintes casos:

1. $\phi_1^u \neq r$ - quando a facilidade mais próxima do usuário u não é r, pode-se eventualmente economizar, associando-se o usuário à facilidade i candidata a inserção. O valor economizado já está sendo embutido em $ganho(i)$.

2. $\phi_1^u = r$ - quando a facilidade mais próxima do usuário u é r, existem três possibilidades:

   (a) $d_1^u \leq d_2^u \leq d_{ui}$ - Como a distância do usuário u a i é maior do que a distância do usuário u a ϕ_2^u e, conseqüentemente a ϕ_1^u, o usuário deve ser transferido de r para ϕ_2^u, o que já está contabilizado no cálculo de $perda(r)$.

   (b) $d_1^u \leq d_{ui} < d_2^u$ - Como a distância do usuário u a i é menor do que a distância do usuário u a ϕ_2^u e maior ou igual a distância do usuário u a ϕ_1^u, o usuário deve ser transferido de r para i. Porém, o cálculo de $perda(r)$ assume que o usuário foi transferido de r para ϕ_2^u e deve ser corrigido pela parcela $d_2^u - d_{ui}$.

   (c) $d_{ui} < d_1^u \leq d_2^u$ - Como a distância do usuário u a i é menor do que a distância do usuário u a ϕ_1^u e, conseqüentemente a ϕ_2^u, o usuário u deve ser transferido de r para i, hipótese já contabilizada no cálculo de $ganho(i) = d_1^u - d_{ui}$. Porém, o cálculo de $perda(r)$ assume que o usuário foi transferido de r para ϕ_2^u e deve ser corrigido pela parcela $d_2^u - d_1^u$.

A função $extra(i, r)$ calcula as parcelas associadas aos itens (2b) e (2c) acima, fazendo as correções necessárias, e pode ser expressa como em [47]:

$$\begin{aligned} extra(i,r) = & \sum_{u:(\phi_1^u = r)\ e\ (d_1^u \leq d_{ui} < d_2^u)} (d_2^u - d_{ui}) + \\ & \sum_{u:(\phi_1^u = r)\ e\ (d_{ui} < d_1^u \leq d_2^u)} (d_2^u - d_{ur}) = \\ & \sum_{u:(\phi_1^u = r)\ e\ (d_{ui} < d_2^u)} (d_2^u - max\{d_{ui}, d_{ur}\}). \end{aligned} \tag{3-3}$$

A economia obtida no problema das p-medianas ao se trocar a facilidade i pela facilidade r pode ser dada pela seguinte equação [47]:

$$lucroP(i,r) = ganho(i) - perda(r) + extra(i,r). \tag{3-4}$$

Quando uma facilidade aberta r é substituída por uma nova facilidade i, os valores nas estruturas *ganho*, *perda*, *extra*, ϕ_1^u e ϕ_2^u ficam desatualizados. Uma maneira direta para a atualização dessas estruturas para a próxima iteração da busca local é recalcular ϕ_1^u e ϕ_2^u para cada usuário u, reinicializar todas as entradas das estruturas auxiliares e então atualizar *ganho*, *perda* e *extra*. Uma desvantagem dessa atualização é que nenhuma informação reunida em iterações anteriores é utilizada em iterações subseqüentes do algoritmo, podendo ocorrer repetição de cálculos.

Pode-se, porém, acelerar a busca local, pois as ações realizadas para atualizar as estruturas auxiliares dependem exclusivamente de u, ϕ_1^u e ϕ_2^u. Portanto, se ϕ_1^u e ϕ_2^u não mudam de uma iteração para outra, não existe a necessidade de atualizar *ganho*, *perda* e *extra* novamente para o usuário u.

Um usuário u é considerado afetado, se, após a troca entre i e r, uma das seguintes condições é satisfeita:

1. r é a primeira ou segunda facilidade mais próxima do usuário u;
2. a facilidade i está mais próxima do usuário u do que a sua segunda facilidade mais próxima ϕ_2^u.

As estruturas *ganho*, *perda* e *extra* só precisam ser atualizadas para os usuários afetados. Se o número de usuários afetados é pequeno, ganhos significativos em termos de tempo de processamento podem ser obtidos.

A Figura 3.2 mostra o algoritmo de busca local para o problema das p-medianas conectadas que recebe como parâmetros uma solução inicial S_0 gerada pela heurística construtiva apresentada na Seção 3.1 e a primeira e segunda facilidade mais próximas de cada usuário u, ϕ_1^u e ϕ_2^u respectivamente.

A linha 2 do algoritmo consiste em construir, para cada usuário u, uma lista com todas as facilidades ordenadas pela distância ao usuário. Possui como objetivo reduzir o tempo gasto na atualização das estruturas *ganho*, *perda* e *extra*, como será visto na explicação das funções *Atualiza_Estruturas* e *Desfaz_Atualiza_Estruturas*. Esse passo é executado uma única vez, mesmo quando a busca local for utilizada diversas vezes dentro de outro algoritmo, como, por exemplo, uma metaheurística [47]. Na linha 3, o conjunto de usuários afetados V_a é inicializado com todos os usuários. Na linha 4, as estruturas auxiliares *ganho*, *perda* e *extra* são inicializadas. O laço das linhas 6 a 27 é executado enquanto existir uma solução vizinha melhor do que a solução corrente.

O laço das linhas 7 a 9 é responsável pela atualização eficiente das estruturas auxiliares. Pelas Equações (3-1), (3-2) e (3-3), nota-se que, cada parcela calculada em *ganho*, *perda* e *extra* é uma soma percorrendo um subconjunto de usuários. Assim, a contribuição de cada cliente pode ser calculada independentemente. A função *Atualiza_Estruturas*, executada somente para os usuários

Algoritmo `Busca_Local_p-Medianas_Conectadas`(S_0, ϕ_1, ϕ_2)
1. $S \leftarrow S_0$;
2. Para cada $u \in V$, ordenar as facilidades de acordo com a distância ao usuário;
3. $V_a \leftarrow V$;
4. $ganho(i) \leftarrow 0, perda(r) \leftarrow 0, extra(i,r) \leftarrow 0, \forall\ i \notin S, \forall\ r \in S$;
5. $Melhoria \leftarrow Verdadeiro$;
6. **Enquanto** $(Melhoria = Verdadeiro)$ **Faça**
7. **Para Todo** $(u \in V_a)$ **Faça**
8. *Atualiza_Estruturas(ganho, perda, extra, u,* ϕ_1^u, ϕ_2^u*)*;
9. **Fim-Para-Todo**
10. $(i, r, lucro(i,r)) \leftarrow$ *Primeiro_Aprimorante(S, ganho, perda, extra)*;
11. **Se** $(lucro(i,r) \leq 0)$ **Então**
12. $Melhoria \leftarrow Falso$;
13. Vá para a linha 27;
14. **Fim-Se**
15. $V_a \leftarrow \emptyset$;
16. **Para Todo** $(u \in V)$ **Faça**
17. **Se** $(\phi_1^u = r$ **ou** $\phi_2^u = r$ **ou** $d_{ui} < d_2^u)$ **Então**
18. $V_a \leftarrow V_a \cup \{u\}$;
19. **Fim-Se**
20. **Fim-Para-Todo**
21. **Para Todo** $(u \in V_a)$ **Faça**
22. *Desfaz_Atualiza_Estruturas(ganho, perda, extra, u,* ϕ_1^u, ϕ_2^u*)*;
23. **Fim-Para-Todo**
24. $S \leftarrow S \backslash \{r\}$;
25. $S \leftarrow S \cup \{i\}$;
26. Atualizar a primeira e segunda facilidade mais próxima de cada usuário $u \in V_a$;
27. **Fim-Enquanto**
28. **Retorne** S;

Fim

Figura 3.2: Algoritmo de busca local para o problema das p-medianas conectadas.

afetados é responsável por essa atualização e seu pseudo-código será explicado posteriormente. A função *Primeiro_Aprimorante* (linha 10) utiliza a estratégia de melhoria iterativa para acelerar os passos da busca local, selecionando, quando houver, a primeira solução aprimorante encontrada na vizinhança da solução corrente. A função *Primeiro_Aprimorante* será explicada com mais detalhes posteriormente. Quando não existir uma solução vizinha melhor do que a solução corrente (caso em que $lucro(i, r) \leq 0$), o algoritmo é finalizado nas linhas 11 a 14 e o ótimo local é retornado na linha 28. Caso contrário, ocorre a atualização do conjunto de usuários afetados nas linhas 15 a 20, conforme descrito anteriormente. As informações presentes em *ganho*, *perda* e *extra* encontram-se desatualizadas em relação à troca ocorrida. Para que uma nova iteração seja realizada, a contribuição da iteração anterior de cada usuário u deve ser subtraída antes que uma nova adição seja feita pela função *Atualiza_Estruturas*. A função *Desfaz_Atualiza_Estruturas*, executada somente para os usuários afetados (laço das linhas 21 a 23), é responsável por essa operação e é similar à função *Atualiza_Estruturas*. As linhas 24 e 25, respectivamente, são responsáveis pela remoção e inserção das facilidades r e i no conjunto das p facilidades abertas na solução corrente. Na linha 26, ocorre a atualização da primeira e da segunda facilidade mais próxima de cada usuário afetado, levando-se em consideração a troca efetuada.

Funções Atualiza_Estruturas e Desfaz_Atualiza_Estruturas

A Figura 3.3 apresenta o pseudo-código da função *Atualiza_Estruturas* que recebe como parâmetros as estruturas auxiliares *ganho*, *perda* e *extra*, além do usuário e de suas respectivas facilidades mais próximas. A expressão $a \overset{+}{\leftarrow} b$ significa que o valor de a é incrementado com b unidades. A função *Desfaz_Atualiza_Estruturas* é similar a essa função, com $a \overset{+}{\leftarrow} b$ sendo substituída por $a \overset{-}{\leftarrow} b$.

Função *Atualiza_Estruturas(ganho, perda, extra, u, ϕ_1^u, ϕ_2^u)*
1. $r \leftarrow \phi_1^u$;
2. $perda(r) \overset{+}{\leftarrow} (d_2^u - d_1^u)$;
3. **Para Todo** $(i \notin S)$ **Faça**
4. **Se** $(d_{ui} < d_2^u)$ **Então**
5. $ganho(i) \overset{+}{\leftarrow} \max\{0, d_1^u - d_{ui}\}$;
6. $extra(i, r) \overset{+}{\leftarrow} d_2^u - \max\{d_{ui}, d_{ur}\}$;
7. **Fim-Se**
8. **Fim-Para-Todo**

Fim

Figura 3.3: Função que atualiza as estruturas *ganho*, *perda* e *extra*.

Na linha 1 obtém-se a facilidade mais próxima do usuário u. A con-

tribuição individual de cada usuário para a estrutura *perda* é calculada na linha 2. O laço das linhas 3 a 8 percorre as $(n - p)$ facilidades que não estão presentes na solução. A contribuição individual de cada usuário para as estrututras *ganho* e *extra* só ocorrem nas linhas 5 e 6, respectivamente, se a distância do usuário u à facilidade i não é maior do que d_2^u (teste da linha 4).

Com o objetivo de tentar reduzir o tempo gasto em cada chamada às funções *Atualiza_Estruturas* e *Desfaz_Atualiza_Estruturas*, utiliza-se a lista com todas as facilidades ordenadas pela distância ao usuário u obtida na linha 2 do algoritmo de busca local (Figura 3.2). Dessa maneira, ao invés de percorrer todas as facilidades que não estão presentes na solução na linha 3 (Figura 3.3), percorre-se o conjunto de facilidades $i \notin S$ ordenadas pela distância ao usuário afetado até que a distância de u a i seja maior ou igual a distância de u a ϕ_2^u ($d_{ui} \geq d_2^u$), que é potencialmente um subconjunto pequeno de facilidades em relação a $(n - p)$.

Função Primeiro_Aprimorante

Resende e Werneck [47] utilizaram a estratégia de busca de descida mais rápida no problema das p-medianas, calculando-se o lucro obtido em cada troca entre uma facilidade pertencente e outra não pertencente à solução corrente através da equação (3-4). Utiliza-se a estratégia de melhoria iterativa, sendo a função *Primeiro_Aprimorante* responsável por encontrar o primeiro vizinho que melhore o valor da função objetivo ao se trocar a facilidade r pela facilidade i, calculando-se o lucro obtido no problema das p-medianas (equação (3-4)) e o lucro obtido no problema de Steiner (diferença no custo entre a árvore de Steiner corrente e a árvore de Steiner obtida na troca de r por i).

A Figura 3.4 apresenta a função que recebe como parâmetros a solução corrente composta pelo conjunto S de p facilidades abertas e pela árvore de Steiner $T(S)$ correspondente, além das estruturas *ganho*, *perda* e *extra*.

A função é composta por dois laços aninhados. O externo (linhas 2 a 11) percorre as facilidades que não estão na solução corrente e o interno (linhas 3 a 10) percorre as facilidades pertencentes à solução corrente. Ambos são executados até que seja encontrado o primeiro vizinho aprimorante ou então até que nenhum melhoramento seja possível. Se uma solução vizinha melhor do que a solução corrente é encontrada ($lucro(i, r) > 0$ na linha 7), retorna-se, na linha 12, as facilidades candidatas a inserção e remoção i e r, respectivamente, e o lucro total obtido. Caso contrário, se $lucro(i, r) \leq 0$, retorna-se na linha 12, as facilidades i e r da última avaliação realizada e seu respectivo lucro.

Para acelerar a busca, a investigação da vizinhança inicia-se do vértice seguinte àquele que obteve o movimento aprimorante na iteração prévia, tanto no conjunto dos candidatos a inserção quanto no conjunto dos candidatos a remoção. A circularidade foi utilizada, por exemplo, para acelerar os passos

```
Função Primeiro_Aprimorante(S, ganho, perda, extra)
1.  Melhoria ← Falso;
2.  Circ Para Todo (i ∉ S) e (Melhoria = Falso) Faça
3.      Circ Para Todo (r ∈ S) e (Melhoria = Falso) Faça
4.          lucroP(i, r) ← ganho(i) − perda(r) + extra(i, r);
5.          lucroS(i, r) ← F_T(S) − F_T(S');
6.          lucro(i, r) ← lucroP(i, r) + lucroS(i, r);
7.          Se (lucro(i, r) > 0) Então
8.              Melhoria ← Verdadeiro;
9.          Fim-Se
10.     Fim-Circ-Para-Todo
11. Fim-Circ-Para-Todo
12. Retorne i, r, lucro(i, r);
Fim
```

Figura 3.4: Função que determina o primeiro vizinho aprimorante na busca local básica.

do algoritmo de busca local em [10]. O lucro obtido ao se trocar r por i para o problema das p-medianas é calculado na linha 4. O lucro obtido para o problema de Steiner é calculado na linha 5 como a diferença entre o custo da árvore de Steiner corrente ($F_T(S)$) e o custo da árvore de Steiner obtida ao se trocar r por i executando-se Prim ($F_T(S')$). O lucro total obtido é calculado na linha 6 como a soma dos lucros obtidos com o problema das p-medianas e com o problema de Steiner.

Complexidade do Algoritmo de Busca Local

A linha 2 do algoritmo possui complexidade $O(n^2 \log n)$ desde que, no máximo n usuários terão suas n facilidades ordenadas com complexidade $O(n \log n)$. A inicialização do conjunto de usuários afetados (linha 3) possui complexidade $O(n)$. A inicialização das estruturas de dados auxiliares (linha 4) possui complexidade $O(pn)$, pois *extra* possui $p \times (n-p)$ entradas. As funções *Atualiza_Estruturas* e *Desfaz_Atualiza_Estruturas* (linhas 8 e 22, respectivamente) possuem complexidade $O(n)$, pois apresentam um laço percorrendo as $(n-p)$ facilidades que não estão presentes na solução. Levando-se em consideração que no máximo n chamadas são executadas para cada uma dessas funções, a complexidade das linhas 7 e 8 assim como das linhas 21 e 22 é $O(n^2)$. A complexidade da função *Primeiro_Aprimorante* é $O(p^2 |E| n \log n)$, pois, no pior caso, $p \times (n-p) = O(pn)$ trocas serão avaliadas, onde, em cada troca ocorre a execução da heurística Prim com complexidade $O(p |E| \log n)$, utilizando-se a estrutura de dados *heap* binário [61]. A atualização do conjunto de usuários afetados (linhas 15 a 20) possui complexidade $O(n)$. A remoção (resp. inserção) de uma facilidade do (resp. no) conjunto de p facilidades abertas (linhas 24 e 25, respectivamente) possui complexidade $O(1)$. Por último,

a linha 26 possui complexidade $O(pn)$, pois para atualizar a primeira e segunda facilidade mais próxima de cada usuário afetado u, deve-se percorrer o conjunto das p facilidades abertas pertencentes à solução corrente.

A complexidade de pior caso de cada iteração do algoritmo de busca local (laço das linhas 6 a 27) é dominada pela complexidade da função *Primeiro_Aprimorante* que é $O(p^2 \left|E\right| n \log n)$.

3.3 Otimizando o Algoritmo Básico de Busca Local

O algoritmo básico de busca local demanda um alto custo computacional, principalmente na função *Primeiro_Aprimorante*, pois, em cada troca, uma árvore de Steiner deve ser recalculada quando troca-se a facilidade candidata a remoção r pela facilidade candidata a inserção i. Pode-se, porém, utilizar um simples teste com o objetivo de filtrar possíveis bons vizinhos.

Seja a solução corrente definida pelo conjunto S das p facilidades abertas e pela árvore de Steiner $T(S)$ correspondente, com seu respectivo custo $F_T(S)$. O teste é iniciado pela retirada de um vértice $r \in S$. Em seguida, calcula-se o custo da árvore de Steiner sem o vértice r, isto é, executa-se a heurística Prim com $p-1$ facilidades. Seja $F_T(S\backslash\{r\})$ o custo dessa árvore. Define-se, agora, a possível melhoria de Steiner (pms) como a diferença entre o custo da árvore de Steiner corrente e o custo da árvore de Steiner sem o vértice r, isto é, $pms = F_T(S) - F_T(S\backslash\{r\})$. Seja i uma facilidade qualquer candidata a inserção. O valor *lucroP* (i, r) representa o lucro obtido ao se trocar r por i no problema das p-medianas. Pode-se então definir o seguinte teste com o objetivo de selecionar possíveis bons vizinhos: se $lucroP(i, r) + pms \leq 0$, descarta-se a facilidade candidata a inserção i; caso contrário, i é um vizinho que deve ser investigado.

Três diferentes versões da busca local básica foram propostas incorporando esse teste na função *Primeiro_Aprimorante* e serão descritas a seguir.

3.3.1 Busca Local com Teste

A Figura 3.5 mostra a modificação realizada na função *Primeiro_Aprimorante*, incorporando o teste descrito acima. Por isso, essa otimização será denominada de busca local com teste. Essa função recebe como parâmetros a solução corrente composta pelo conjunto S de p facilidades abertas e pela árvore de Steiner $T(S)$ correspondente, além das estruturas *ganho*, *perda* e *extra*.

A função possui dois laços aninhados. O externo (linhas 2 a 18) percorre as facilidades pertencentes a solução corrente e o interno (linhas 8 a 17) percorre as facilidades que não estão na solução corrente. Ambos são executa-

```
Função Primeiro_Aprimorante_Teste(S, ganho, perda, extra)
1.  Melhoria ← Falso;
2.  Circ Para Todo (r ∈ S) e (Melhoria = Falso) Faça
3.      pms ← F_T(S) − F_T(S\{r});
4.      Se (pms < 0) Então
5.          T(S\{r}) ← T(S);
6.          pms ← 0;
7.      Fim-Se
8.      Circ Para Todo (i ∉ S) e (Melhoria = Falso) Faça
9.          lucroP(i, r) ← ganho(i) − perda(r) + extra(i, r);
10.         Se (lucroP(i, r) + pms > 0) Então
11.             lucroS(i, r) ← F_T(S) − F_T(S');
12.             lucro(i, r) ← lucroP(i, r) + lucroS(i, r);
13.             Se (lucro(i, r) > 0) Então
14.                 Melhoria ← Verdadeiro;
15.             Fim-Se
16.         Fim-Se
17.     Fim-Circ-Para-Todo
18. Fim-Circ-Para-Todo
19. Retorne i, r, lucro(i, r);
Fim
```

Figura 3.5: Função que determina o primeiro vizinho aprimorante na busca local com teste.

dos até que seja encontrado o primeiro vizinho aprimorante ou então até que nenhum melhoramento seja possível.

Na linha 3 calcula-se o valor da possível melhoria de Steiner como a diferença entre o custo da árvore de Steiner corrente e o custo da árvore de Steiner obtida pela execução de Prim sem o vértice r. A abordagem utilizada para a construção da árvore de Steiner é heurística e, por isso, pode acontecer que, ao se retirar um determinado vértice r da árvore corrente, o custo da árvore resultante seja pior do que o custo da árvore corrente. Assim, *pms* terá um valor negativo. Caso isso aconteça (linha 4), a árvore $T(S\backslash\{r\})$ será a árvore corrente $T(S)$, passando r de vértice terminal a vértice de Steiner na árvore e fazendo *pms* igual a 0 (linhas 5 e 6).

O lucro obtido no problema das p-medianas é calculado na linha 9 e o teste que otimiza a busca local é realizado na linha 10. Se a facilidade candidata a inserção passa no teste, executa-se Prim com p facilidades trocando-se r por i e verifica-se o lucro de Steiner ao realizar essa troca em relação ao custo da árvore corrente (linha 11). Na linha 12 calcula-se o lucro total obtido. Se i não passa no teste, avalia-se a próxima facilidade candidata a inserção. Se $lucro(i, r) > 0$ (linha 13), encontra-se o primeiro vizinho aprimorante na vizinhança da solução corrente. Retorna-se na linha 19 as facilidades candidatas a inserção e remoção i e r, respectivamente, e o lucro total obtido.

Caso contrário, se $lucro(i, r) \leq 0$, retorna-se na linha 19, as facilidades i e r da última avaliação realizada e seu respectivo lucro.

3.3.2 Busca Local pelas Bordas

Intuitivamente, facilidades próximas à árvore de Steiner podem ser melhores candidatas a inserção do que facilidades mais distantes. A modificação realizada na função *Primeiro_Aprimorante* incorpora o teste definido anteriormente e pode ser assim explicada, em linhas gerais: se i não passa no teste, avalia-se a próxima facilidade candidata; se i passa no teste, verifica-se se a mesma é adjacente à árvore de Steiner $T(S\backslash\{r\})$. Caso i seja adjacente a $T(S\backslash\{r\})$, executa-se Prim com p facilidades trocando-se r por i; caso contrário avalia-se a próxima facilidade candidata. Diz-que a facilidade i é adjacente à árvore de Steiner $T(S\backslash\{r\})$ se possui, na sua lista de adjacências, algum nó pertencente à essa árvore. Por priorizar facilidades adjacentes à árvore, a otimização realizada será denominada de busca local pelas bordas.

A Figura 3.6 mostra a modificação realizada na função *Primeiro_Aprimorante* que recebe como parâmetros a solução corrente composta pelo conjunto S de p facilidades abertas e pela árvore de Steiner $T(S)$ correspondente, além das estruturas *ganho*, *perda* e *extra*.

A função possui dois laços aninhados. O externo (linhas 2 a 35) percorre as facilidades pertencentes a solução corrente e o interno (linhas 8 a 34) percorre as facilidades que não estão na solução corrente. Ambos são executados até que seja encontrado o primeiro vizinho aprimorante ou então até que nenhum melhoramento seja possível.

Como na seção anterior, calcula-se *pms* na linha 3, o lucro das p-medianas na linha 9 e executa-se o teste que otimiza a busca local na linha 10. Se i não passa no teste, avalia-se a próxima facilidade candidata a inserção; caso contrário, verifica-se se i pertence ou não à árvore $T(S\backslash\{r\})$; se a facilidade candidata não pertence à árvore (linha 11), o próximo passo é verificar se a facilidade é adjacente à mesma. Para isso basta que a facilidade i possua, na sua lista de adjacências ($Adj(i)$), um nó pertencente a $T(S\backslash\{r\})$ (linhas 12 a 17). Em caso afirmativo, executa-se Prim com p facilidades e calcula-se o lucro de Steiner (linha 19) e o lucro total obtido (linha 20); caso contrário, avalia-se a próxima facilidade candidata i. Caso i pertença à árvore $T(S\backslash\{r\})$, isto é, i é um vértice de Steiner nessa árvore, não é necessário executar Prim com p facilidades e o custo da árvore será igual ao custo de $T(S\backslash\{r\})$; assim, o lucro para o problema de Steiner é igual ao valor *pms* (linha 27). Se $lucro(i, r) > 0$ (linha 21 ou 29), encontra-se o primeiro vizinho aprimorante na vizinhança da solução corrente, retornando-se, na linha 36, as facilidades candidatas a inserção e remoção i e r, respectivamente, e o lucro total obtido. Caso contrário,

```
Função Primeiro_Aprimorante_B(S, ganho, perda, extra)
1.  Melhoria ← Falso;
2.  Circ Para Todo (r ∈ S) e (Melhoria = Falso) Faça
3.      pms ← F_T(S) − F_T(S\{r});
4.      Se (pms < 0) Então
5.          T(S\{r}) ← T(S);
6.          pms ← 0;
7.      Fim-Se
8.      Circ Para Todo (i ∉ S) e (Melhoria = Falso) Faça
9.          lucroP(i, r) ← ganho(i) − perda(r) + extra(i, r);
10.         Se (lucroP(i, r) + pms > 0) Então
11.             Se (i ∉ T(S\{r})) Então
12.                 Adj_Arvore ← Falso;
13.                 Para Todo (f ∈ Adj(i)) Faça
14.                     Se (f ∈ T(S\{r})) Então
15.                         Adj_Arvore ← Verdadeiro;
16.                     Fim-Se
17.                 Fim-Para-Todo
18.                 Se (Adj_Arvore = Verdadeiro) Então
19.                     lucroS(i, r) ← F_T(S) − F_T(S′);
20.                     lucro(i, r) ← lucroP(i, r) + lucroS(i, r);
21.                     Se (lucro(i, r) > 0) Então
22.                         Melhoria ← Verdadeiro;
23.                     Fim-Se
24.                 Fim-Se
25.             Fim-Se
26.             Senão
27.                 lucroS(i, r) ← pms;
28.                 lucro(i, r) ← lucroP(i, r) + lucroS(i, r);
29.                 Se (lucro(i, r) > 0) Então
30.                     Melhoria ← Verdadeiro;
31.                 Fim-Se
32.             Fim-Senão
33.         Fim-Se
34.     Fim-Circ-Para-Todo
35. Fim-Circ-Para-Todo
36. Retorne i, r, lucro(i, r);
Fim
```

Figura 3.6: Função que determina o primeiro vizinho aprimorante na busca local pelas bordas.

se $lucro(i, r) \leq 0$, retorna-se na linha 36, as facilidades i e r da última avaliação realizada e seu respectivo lucro.

3.3.3
Busca Local pelas Bordas com Teste da Menor Aresta

Essa modificação também prioriza facilidades que são adjacentes à árvore de Steiner $T(S\backslash\{r\})$. Adicionalmente, para a execução de Prim com p facilidades trocando-se r por i, o seguinte teste deve ser satisfeito: $MA - pms - lucroP(i, r) < 0$, onde MA é a aresta de menor custo que liga a facilidade i a $T(S\backslash\{r\})$. O valor $MA - pms < 0$ indica que a facilidade candidata i pode melhorar o valor da função objetivo referente ao problema de Steiner. Já o valor $lucroP(i, r) > 0$ indica que uma possível troca entre r e i melhora o valor da função objetivo referente ao problema das p-medianas. Assim, o teste $MA - pms - lucroP(i, r) < 0$ indica que a facilidade i é um possível bom vizinho a ser investigado.

A Figura 3.7 mostra a modificação realizada na função *Primeiro_Aprimorante* que recebe como parâmetros a solução corrente composta pelo conjunto S de p facilidades abertas e pela árvore de Steiner $T(S)$ correspondente, além das estruturas *ganho*, *perda* e *extra*.

A função possui dois laços aninhados. O externo (linhas 2 a 35) percorre as facilidades pertencentes a solução corrente e o interno (linhas 8 a 34) percorre as facilidades que não estão na solução corrente. Ambos são executados até que seja encontrado o primeiro vizinho aprimorante ou então até que nenhum melhoramento seja possível.

Como nas duas seções anteriores, calcula-se *pms* na linha 3, o lucro das p-medianas na linha 9 e executa-se o teste que otimiza a busca local na linha 10. Se i não passa no teste, avalia-se a próxima facilidade candidata a inserção. Caso contrário, testa-se se i pertence ou não à árvore $T(S\backslash\{r\})$ (linha 11). Se a facilidade candidata i não pertence à árvore, verifica-se a aresta de menor custo que liga a facilidade à árvore. Para isso, percorre-se a lista de adjacências de i ($Adj(i)$), procurando algum nó pertencente a $T(S\backslash\{r\})$. Em caso afirmativo, armazena-se a aresta de menor custo (linhas 12 a 17). Se o teste $MA - pms - lucroP(i, r) < 0$ for verdadeiro (linha 18), executa-se Prim com p facilidades e calcula-se o lucro de Steiner (linha 19) e o lucro total obtido (linha 20). Caso contrário, avalia-se a próxima facilidade i. Se não existe aresta que liga i à árvore ($MA = \infty$), então i não é adjacente à árvore e passa-se para a próxima facilidade candidata. Caso i pertença à árvore $T(S\backslash\{r\})$, não é necessário executar Prim e o custo da árvore será igual ao custo de $T(S\backslash\{r\})$. Assim, o lucro para o problema de Steiner é igual ao valor *pms* (linha 27). Se $lucro(i, r) > 0$ (linha 21 ou 29), encontra-se o primeiro vizinho aprimorante na vizinhança da solução corrente, retornando-se, na linha 36, as facilidades

```
Função Primeiro_Aprimorante_BTMA(S, ganho, perda, extra)
1.  Melhoria ← Falso;
2.  Circ Para Todo (r ∈ S) e (Melhoria = Falso) Faça
3.      pms ← F_T(S) − F_T(S\{r});
4.      Se (pms < 0) Então
5.          T(S\{r}) ← T(S);
6.          pms ← 0;
7.      Fim-Se
8.      Circ Para Todo (i ∉ S) e (Melhoria = Falso) Faça
9.          lucroP(i, r) ← ganho(i) − perda(r) + extra(i, r);
10.         Se (lucroP(i, r) + pms > 0) Então
11.             Se (i ∉ T(S\{r})) Então
12.                 MA ← ∞;
13.                 Para Todo (f ∈ Adj(i)) Faça
14.                     Se (f ∈ T(S\{r})) e (c_fi < MA) Então
15.                         MA ← c_fi;
16.                     Fim-Se
17.                 Fim-Para-Todo
18.                 Se (MA − pms − lucroP(i, r) < 0) Então
19.                     lucroS(i, r) ← F_T(S) − F_T(S′);
20.                     lucro(i, r) ← lucroP(i, r) + lucroS(i, r);
21.                     Se (lucro(i, r) > 0) Então
22.                         Melhoria ← Verdadeiro;
23.                     Fim-Se
24.                 Fim-Se
25.             Fim-Se
26.             Senão
27.                 lucroS(i, r) ← pms;
28.                 lucro(i, r) ← lucroP(i, r) + lucroS(i, r);
29.                 Se (lucro(i, r) > 0) Então
30.                     Melhoria ← Verdadeiro;
31.                 Fim-Se
32.             Fim-Senão
33.         Fim-Se
34.     Fim-Circ-Para-Todo
35. Fim-Circ-Para-Todo
36. Retorne i, r, lucro(i, r);
Fim
```

Figura 3.7: Função que determina o primeiro vizinho aprimorante na busca local pelas bordas com teste da menor aresta.

candidatas a inserção e remoção i e r, respectivamente, e o lucro total obtido. Caso contrário, se $lucro(i, r) \leq 0$, retorna-se na linha 36, as facilidades i e r da última avaliação realizada e seu respectivo lucro.

3.3.4 Resultados Computacionais

Essa seção apresenta os resultados computacionais, comparando-se a busca local básica com as três variantes em termos de qualidade das soluções e tempos de processamento. As buscas locais que utilizam as funções apresentadas nas Figuras 3.4, 3.5, 3.6 e 3.7 serão denominadas, respectivamente, de BL_Básica, BL_Teste, BL_B e BL_BTMA. Para cada par instância-algoritmo, foram realizadas 15 execuções com soluções iniciais diferentes geradas inicializando-se o gerador de números aleatórios com sementes diferentes.

Ambiente de Teste, Instâncias e Medidas Utilizadas

A heurística construtiva e os algoritmos de busca local foram implementados em C com o parâmetro de otimização -O3 e executados em uma máquina Pentium IV 3.2 GHz com 1 Gbyte de memória RAM sob o sistema operacional Linux RedHat 9.0. O gerador de números aleatórios utilizado foi o de Matsumoto e Nishimura [53].

Para a comparação dos algoritmos de busca local, três classes de instâncias utilizadas no problema das p-medianas foram adaptadas para o problema híbrido. Devido as características explicadas no próximo parágrafo, serão denominadas de instâncias proporcionais.

A primeira classe é uma modificação das instâncias da ORLIB [5], denominada aqui de ORM_P, contendo 40 grafos com 100 a 900 vértices e p variando entre 5 e 200. A segunda classe é uma modificação das instâncias GR [18], denominada aqui de GRM_P. Contém dois grafos completos, com 100 e 150 vértices e valores de p entre 5 e 50 (resp. entre 5 e 60) para o primeiro (resp. segundo) grafo. A terceira classe é uma modificação das instâncias SL [55], denominada aqui de SLM_P. Essa classe contém três instâncias baseadas em grafos da ORM_P: SLM_P700 utiliza o mesmo grafo que ORM_P34, mas com $p = 233$; SLM_P800 utiliza o mesmo grafo que ORM_P37, mas com $p = 267$; e SLM_P900 utiliza o mesmo grafo que ORM_P40, mas com $p = 300$. Seja então um grafo $G = (V, E)$ com custos associados às arestas. O custo de servir cada usuário i a partir de cada facilidade j é dado pelo valor d_{ij} do caminho mais curto do vértice i ao vértice j no grafo G, calculado a partir dos custos associados às arestas. O custo de instalação c_{ij} de cada aresta é obtido multiplicando-se o custo da mesma pelo fator w, isto é, $c_{ij} = a_{ij} \times w$, onde a_{ij} é o custo associado à aresta (i, j) e w assume os valores 2, 5 e 10.

As Tabelas 3.1 a 3.3 mostram, respectivamente, as 40, 20 e três instâncias com seus respectivos número de vértices (n), número de arestas ($|E|$) e quantidade de facilidades a serem abertas (p). Multiplicando-se pelos três valores assumidos por w, tem-se um total de 189 problemas.

Instância	n	$\|E\|$	p	Instância	n	$\|E\|$	p
ORM_P1	100	198	5	ORM_P21	500	4.909	5
ORM_P2		193	10	ORM_P22		4.896	10
ORM_P3		198	10	ORM_P23		4.903	50
ORM_P4		196	20	ORM_P24		4.914	100
ORM_P5		196	33	ORM_P25		4.894	167
ORM_P6	200	786	5	ORM_P26	600	7.068	5
ORM_P7		779	10	ORM_P27		7.072	10
ORM_P8		792	20	ORM_P28		7.054	60
ORM_P9		785	40	ORM_P29		7.042	120
ORM_P10		786	67	ORM_P30		7.042	200
ORM_P11	300	1.772	5	ORM_P31	700	9.601	5
ORM_P12		1.758	10	ORM_P32		9.584	10
ORM_P13		1.760	30	ORM_P33		9.616	70
ORM_P14		1.771	60	ORM_P34		9.585	140
ORM_P15		1.754	100	ORM_P35	800	12.548	5
ORM_P16	400	3.153	5	ORM_P36		12.560	10
ORM_P17		3.142	10	ORM_P37		12.564	80
ORM_P18		3.134	40	ORM_P38	900	15.898	5
ORM_P19		3.134	80	ORM_P39		15.896	10
ORM_P20		3.144	133	ORM_P40		15.879	90

Tabela 3.1: Classe de instâncias ORM_P para o problemas das p-medianas conectadas.

Instância	n	$\|E\|$	p	Instância	n	$\|E\|$	p
GRM_P1	100	4.950	5	GRM_P11	150	11.175	10
GRM_P2			10	GRM_P12			15
GRM_P3			15	GRM_P13			20
GRM_P4			20	GRM_P14			25
GRM_P5			25	GRM_P15			30
GRM_P6			30	GRM_P16			35
GRM_P7			35	GRM_P17			40
GRM_P8			40	GRM_P18			45
GRM_P9			50	GRM_P19			50
GRM_P10	150	11.175	5	GRM_P20			60

Tabela 3.2: Classe de instâncias GRM_P para o problemas das p-medianas conectadas.

Na avaliação da qualidade das soluções encontradas, foram utilizadas três medidas relativas [48, 52, 61]:

– *Desvio relativo percentual médio (drpm)* - Dada uma instância, define-se o *desvio relativo percentual* de um método como a diferença percentual

Instância	n	$\|E\|$	p
SLM_P700	700	9.585	233
SLM_P800	800	12.564	267
SLM_P900	900	15.879	300

Tabela 3.3: Classe de instâncias SLM_P para o problemas das p-medianas conectadas.

entre a média do valor da solução encontrada pelo método e a média do valor da solução encontrada por todos os métodos. O *desvio relativo percentual médio* é a média dos desvios percentuais obtida pelo método para todas as instâncias do grupo.

– *Classificação média (cm)* - Para cada instância, os métodos são ordenados de acordo com o seu *desvio relativo percentual*. O melhor método possui classificação 1, o segundo melhor possui classificação 2 e assim sucessivamente. Em caso de empate, atribui-se a todos os métodos envolvidos a mesma classificação, calculada como a média das classificações individuais. Como exemplo, se os valores forem -0,03, -0,02, -0,02, 0,01, 0,03 e 0,03, as classificações de cada método serão: 1, 2,5, 2,5, 4, 5,5 e 5,5. A *classificação média* de um método é a média das classificações obtidas para todas as instâncias do grupo.

– *Melhor* - É o número de instâncias para as quais o método obteve o melhor (menor) valor entre todos os métodos, incluindo os casos de empate no primeiro lugar.

Quanto menor o valor da medida *drpm*, melhor o método. O valor da medida *cm* varia entre 1 e o número total de métodos utilizados na comparação. O valor 1 indica que o método é melhor em todas as instâncias do grupo. Assim, valores mais próximos de 1 são melhores. Já para a medida *melhor*, quanto maior seu valor, mais competitiva é a heurística.

Na avaliação dos tempos de processamento obtidos, a seguinte medida absoluta será utilizada:

– *Tempo absoluto médio (tam)* - Para cada instância, considera-se a média do tempo de processamento de um algoritmo em um determinado número de execuções.

Nas tabelas apresentadas no presente capítulo, os valores em negrito destacam o melhor resultado obtido pelas buscas locais nas medidas para cada instância ou grupo de instâncias testadas.

Qualidade das Soluções

As Tabelas 3.4 a 3.6 mostram os valores obtidos para as medidas relativas *drpm*, *cm* e *melhor* utilizadas na avaliação da qualidade das soluções encontradas pelas buscas locais para todos os grupos de instâncias testadas.

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P (40 instâncias)	BL_Básica	**-0,26**	**1,63**	26
	BL_Teste	-0,24	2,11	**27**
	BL_B	-0,02	2,29	21
	BL_BTMA	0,52	3,98	5
GRM_P (20 instâncias)	BL_Básica	-0,06	2,50	18
	BL_Teste	**-0,11**	**1,85**	**19**
	BL_B	**-0,11**	**1,85**	**19**
	BL_BTMA	0,28	3,80	11
SLM_P (3 instâncias)	BL_Básica	**-0,11**	1,67	1
	BL_Teste	-0,10	**1,33**	**2**
	BL_B	0,07	3,33	1
	BL_BTMA	0,14	4,00	0

Tabela 3.4: Qualidade relativa das buscas locais ($w = 2$).

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P (40 instâncias)	BL_Básica	-0,24	1,93	**32**
	BL_Teste	**-0,25**	**1,70**	31
	BL_B	-0,04	2,45	22
	BL_BTMA	0,53	3,93	6
GRM_P (20 instâncias)	BL_Básica	-0,04	2,73	**17**
	BL_Teste	**-0,08**	**1,78**	15
	BL_B	**-0,08**	**1,78**	15
	BL_BTMA	0,19	3,73	11
SLM_P (3 instâncias)	BL_Básica	**-0,06**	**2,00**	1
	BL_Teste	-0,01	**2,00**	**2**
	BL_B	-0,04	**2,00**	1
	BL_BTMA	0,11	4,00	0

Tabela 3.5: Qualidade relativa das buscas locais ($w = 5$).

Para a coluna *drpm*, primeiramente pode-se notar que BL_BTMA obtém os piores resultados para todas as instâncias testadas. Comparando-se os outros métodos entre si, há um grande equilíbrio nos testes. BL_Básica é melhor em cinco grupos de instâncias: $w = 2$ (ORM_P e SLM_P), $w = 5$ (SLM_P) e $w = 10$ (ORM_P e SLM_P). BL_Teste é melhor em quatro grupos de instâncias: $w = 2$ (GRM_P), $w = 5$ (ORM_P e GRM_P) e $w = 10$ (GRM_P), enquanto BL_B é melhor em três grupos de instâncias: $w = 2$, $w = 5$ e $w = 10$ (instâncias GRM_P).

A medida relativa *cm* novamente indica que os piores resultados são obtidos pela BL_BTMA com os valores para todos os grupos de instâncias

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P (40 instâncias)	BL_Básica	**-0,30**	**1,60**	**34**
	BL_Teste	-0,21	1,93	30
	BL_B	0,40	2,90	14
	BL_BTMA	0,51	3,58	10
GRM_P (20 instâncias)	BL_Básica	-0,05	2,70	**19**
	BL_Teste	**-0,54**	**1,85**	17
	BL_B	**-0,54**	**1,85**	17
	BL_BTMA	0,26	3,60	16
SLM_P (3 instâncias)	BL_Básica	**-0,28**	**1,67**	**2**
	BL_Teste	-0,11	2,33	0
	BL_B	-0,15	2,00	1
	BL_BTMA	0,54	4,00	0

Tabela 3.6: Qualidade relativa das buscas locais ($w = 10$).

muito próximos ou iguais a 4 (o menor valor apresentado é 3,58 no grupo ORM_P, $w = 10$). Novamente, existe um equilíbrio comparando-se os outros métodos. BL_Básica é melhor em quatro grupos de instâncias: $w = 2$ (ORM_P), $w = 5$ (SLM_P) e $w = 10$ (ORM_P e SLM_P). BL_Teste é melhor em seis grupos de instâncias: $w = 2$ (GRM_P e SLM_P), $w = 5$ (ORM_P, GRM_P e SLM_P) e $w = 10$ (GRM_P), enquanto BL_B é melhor em quatro grupos de instâncias: $w = 2$ (GRM_P), $w = 5$ (GRM_P e SLM_P) e $w = 10$ (GRM_P).

A coluna *melhor* mostra um equilíbrio entre as buscas BL_Básica e BL_Teste. A primeira é superior às demais em cinco grupos de instâncias: $w = 5$ (ORM_P e GRM_P) e $w = 10$ (ORM_P, GRM_P e SLM_P) enquanto a segunda é superior às demais em quatro grupos de instâncias: $w = 2$ (ORM_P, GRM_P e SLM_P) e $w = 5$ (SLM_P). BL_B é superior às demais somente no grupo $w = 2$ (GRM_P). Novamente, BL_BTMA apresenta os piores resultados.

Assim, analisando-se a qualidade das soluções encontradas pelas buscas locais nas três medidas relativas, destacam-se BL_Básica e BL_Teste, com uma pequena diferença favorecendo uma ou outra busca nas instâncias testadas. Isso vem mostrar que o teste realizado é muito eficiente, descartando vizinhos que não melhorariam a solução corrente e evitando, assim, o trabalho desnecessário ao se executar Prim quando troca-se a facilidade candidata a remoção r pela facilidade candidata a inserção i. Para ressaltar esses resultados, as Tabelas 3.7 a 3.15 mostram a quantidade total média de vizinhos e o percentual médio de vizinhos descartados (coluna Desc.(%)) e avaliados (coluna Aval.(%)) pela busca BL_Teste em 15 iterações realizadas para cada instância.

Os resultados mostram que, em média, para quase todas as instâncias testadas, ocorre o descarte de vizinhos, com exceção de duas delas (GRM_P9 com $w = 2$ e $w = 5$) onde todos os vizinhos são avaliados. Em geral, observou-se que, quando p cresce, mantendo-se fixo o número de vértices do grafo ou quando aumenta-se o fator w, diminui o número de vizinhos descartados. Ocorre esse

mesmo padrão de comportamento nas buscas locais BL_B e BL_BTMA. BL_B descarta, em geral, uma quantidade média de vizinhos um pouco maior do que BL_Teste, pois a facilidade vizinha candidata a inserção deve satisfazer o teste que otimiza a busca local e ser adjacente à árvore de Steiner $T(S\backslash\{r\})$. Já BL_BTMA descarta uma quantidade média de vizinhos bem maior do que BL_Teste e BL_B, pois além dos mesmos requisitos apresentados por BL_B, para a facilidade candidata a inserção, o teste $MA - pms - lucroP(i, r) < 0$ deve ser satisfeito. Em média, BL_BTMA descarta 99% dos vizinhos em todas as instâncias testadas.

Por último, BL_B apresenta rendimento próximo ao das duas melhores buscas nas medidas *drpm* e *cm* e um rendimento bem pior na medida *melhor*. Já BL_BTMA apresenta os piores resultados em todas as instâncias testadas nas três medidas relativas em comparação aos outros algoritmos.

Tempos de Processamento

A complexidade de pior caso da BL_Básica e de suas versões é: $O(p^2\,|E|\,n\log n)$. Para a comparação dos tempos de processamento obtidos pelas buscas locais, serão utilizados os tempos médios de execução em segundos (medida *tam*). As Tabelas 3.16 a 3.24 apresentam esses resultados.

Claramente observa-se que BL_BTMA é muito mais rápida do que as demais em todas as instâncias testadas devido à grande quantidade de vizinhos descartados por essa busca. BL_Básica é muito mais lenta do que as demais em todos os problemas, principalmente nas maiores instâncias da classe ORM_P e nas instâncias SLM_P, pois não apresenta o teste que otimiza a busca local, tendo que executar Prim a cada troca de uma facilidade candidata a remoção por uma facilidade candidata a inserção. Comparando-se BL_B com BL_Teste, tem-se que, em geral, a primeira apresenta tempos de processamento menores do que a segunda, porém sem ganhos significativos. Para as buscas locais com descarte de vizinhos, quando aumenta-se o fator w, em geral, aumenta-se o número de vizinhos avaliados, e conseqüentemente, aumenta-se o tempo de processamento de cada instância.

Assim, em termos de tempo de processamento, destaca-se o algoritmo de BL_BTMA, seguido, com uma pequena diferença entre eles, por BL_B e BL_Teste.

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
ORM_P1	1.129,93	1.058,40	93,67	71,53	6,33
ORM_P2	2.850,33	2.654,53	93,13	195,80	6,87
ORM_P3	3.575,73	3.343,53	93,51	232,20	6,49
ORM_P4	6.147,73	4.363,20	70,97	1.784,53	29,03
ORM_P5	9.938,00	4.562,00	45,90	5.376,00	54,10
ORM_P6	2.890,00	2.830,13	97,93	59,86	2,07
ORM_P7	7.458,46	7.271,93	97,50	186,53	2,50
ORM_P8	13.727,40	12.727,00	92,71	1.000,40	7,29
ORM_P9	30.663,80	23.709,00	77,32	6.954,80	22,68
ORM_P10	42.895,20	20.631,26	48,10	22.263,93	51,90
ORM_P11	3.937,60	3.910,93	99,32	26,66	0,68
ORM_P12	7.923,33	7.861,60	99,22	61,73	0,78
ORM_P13	39.199,73	36.690,53	93,60	2.509,20	6,40
ORM_P14	74.429,40	58.667,00	78,82	15.762,40	21,18
ORM_P15	96.581,53	45.285,06	46,89	51.296,46	53,11
ORM_P16	5.310,86	5.285,33	99,52	25,53	0,48
ORM_P17	16.239,46	16.175,33	99,61	64,13	0,39
ORM_P18	62.846,26	59.940,20	95,38	2.906,06	4,62
ORM_P19	135.898,13	116.377,93	85,64	19.520,20	14,36
ORM_P20	187.727,20	98.911,46	52,69	88.815,73	47,31
ORM_P21	8.035,93	8.024,33	99,86	11,60	0,14
ORM_P22	16.089,20	16.031,86	99,64	57,33	0,36
ORM_P23	105.831,66	101.966,80	96,35	3.864,86	3,65
ORM_P24	240.910,86	199.656,00	82,88	41.254,86	17,12
ORM_P25	257.896,33	131.028,93	50,81	126.867,40	49,19
ORM_P26	8.592,20	8.576,80	99,82	15,40	0,18
ORM_P27	20.669,73	20.596,66	99,65	73,06	0,35
ORM_P28	174.513,93	168.481,53	96,54	6.032,40	3,46
ORM_P29	333.694,13	280.870,46	84,17	52.823,66	15,83
ORM_P30	392.624,20	207.826,93	52,93	184.797,26	47,07
ORM_P31	8.720,80	8.703,26	99,80	17,53	0,20
ORM_P32	24.588,46	24.548,80	99,84	39,66	0,16
ORM_P33	241.098,53	235.402,33	97,64	5.696,20	2,36
ORM_P34	488.564,53	413.862,66	84,71	74.701,86	15,29
ORM_P35	12.122,26	12.108,53	99,89	13,73	0,11
ORM_P36	27.591,20	27.541,13	99,82	50,06	0,18
ORM_P37	324.865,93	318.233,46	97,96	6.632,46	2,04
ORM_P38	10.406,53	10.392,46	99,86	14,06	0,14
ORM_P39	30.708,46	30.666,20	99,86	42,26	0,14
ORM_P40	505.554,80	488.991,26	96,72	16.563,53	3,28

Tabela 3.7: Quantidade média de vizinhos em 15 iterações - ORM_P ($w = 2$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
ORM_P1	1.381,13	1.162,20	84,15	218,93	15,85
ORM_P2	3.504,00	2.005,73	57,24	1.498,26	42,76
ORM_P3	3.623,60	2.252,86	62,17	1.370,73	37,83
ORM_P4	7.725,00	2.342,46	30,32	5.382,53	69,68
ORM_P5	9.817,13	1.587,13	16,17	8.230,00	83,83
ORM_P6	2.513,60	2.346,80	93,36	166,80	6,64
ORM_P7	7.890,13	6.853,53	86,86	1.036,60	13,14
ORM_P8	16.350,80	11.264,20	68,89	5.086,60	31,11
ORM_P9	29.603,46	10.933,60	36,93	18.669,86	63,07
ORM_P10	44.743,20	6.378,00	14,25	38.365,20	85,75
ORM_P11	4.264,40	4.194,53	98,36	69,86	1,64
ORM_P12	9.279,60	8.958,46	96,54	321,13	3,46
ORM_P13	39.587,26	22.537,46	56,93	17.049,80	43,07
ORM_P14	80.882,60	27.768,33	34,33	53.114,26	65,67
ORM_P15	135.787,00	14.308,33	10,54	121.478,66	89,46
ORM_P16	5.041,00	4.987,13	98,93	53,86	1,07
ORM_P17	13.458,13	13.298,13	98,81	160,00	1,19
ORM_P18	84.990,40	56.888,86	66,94	28.101,53	33,06
ORM_P19	139.017,53	41.875,20	30,12	97.142,33	69,88
ORM_P20	221.205,26	32.189,26	14,55	189.016,00	85,45
ORM_P21	8.099,66	8.056,73	99,47	42,93	0,53
ORM_P22	17.073,46	16.850,86	98,70	222,60	1,30
ORM_P23	133.262,80	93.423,46	70,10	39.839,33	29,90
ORM_P24	237.441,06	74.697,46	31,46	162.743,60	68,54
ORM_P25	333.129,86	44.780,20	13,44	288.349,66	86,58
ORM_P26	8.843,73	8.806,86	99,58	36,86	0,42
ORM_P27	20.046,20	19.801,26	98,78	244,93	1,22
ORM_P28	214.298,26	150.419,86	70,19	63.878,40	29,81
ORM_P29	313.796,20	96.069,93	30,62	217.726,26	69,38
ORM_P30	491.802,20	77.931,00	15,85	413.871,20	84,15
ORM_P31	8.880,73	8.846,46	99,61	34,26	0,39
ORM_P32	20.586,86	20.494,86	99,55	92,00	0,45
ORM_P33	263.389,66	197.459,60	74,97	65.930,06	25,03
ORM_P34	474.135,06	162.535,06	34,28	311.600,00	65,72
ORM_P35	11.667,13	11.648,00	99,84	19,13	0,16
ORM_P36	29.294,26	29.124,86	99,42	169,40	0,58
ORM_P37	329.428,66	256.243,73	77,78	73.184,93	22,22
ORM_P38	11.443,20	11.405,80	99,67	37,40	0,33
ORM_P39	27.452,93	27.352,60	99,63	100,33	0,37
ORM_P40	455.507,13	335.938,73	73,75	119.568,40	26,25

Tabela 3.8: Quantidade média de vizinhos em 15 iterações - ORM_P ($w = 5$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
ORM_P1	1.355,20	747,26	55,14	607,93	44,86
ORM_P2	3.650,40	637,13	17,45	3.013,26	82,55
ORM_P3	3.188,33	1.170,33	36,71	2.018,00	63,29
ORM_P4	7.282,86	1.069,13	14,68	6.213,73	85,32
ORM_P5	11.339,13	1.048,00	9,24	10.291,13	90,76
ORM_P6	2.610,33	2.133,93	81,75	476,40	18,25
ORM_P7	7.337,46	4.312,00	58,77	3.025,46	41,23
ORM_P8	18.643,53	4.315,93	23,15	14.327,60	76,85
ORM_P9	36.492,73	5.112,00	14,00	31.380,73	85,99
ORM_P10	51.711,40	5.327,33	10,30	46.384,06	89,70
ORM_P11	4.872,60	4.584,46	94,09	288,13	5,91
ORM_P12	12.365,00	10.994,46	88,92	1.370,53	11,08
ORM_P13	49.140,80	10.074,13	20,50	39.066,66	79,50
ORM_P14	93.939,06	15.114,40	16,09	78.824,66	83,91
ORM_P15	150.449,33	11.467,66	7,62	138.981,66	92,38
ORM_P16	4.974,93	4.874,20	97,98	100,73	2,02
ORM_P17	14.515,66	13.794,73	95,03	720,93	4,97
ORM_P18	75.813,73	21.261,66	28,04	54.552,06	71,96
ORM_P19	177.183,20	24.111,06	13,61	153.072,13	86,39
ORM_P20	245.088,66	20.541,33	8,38	224.547,33	91,62
ORM_P21	8.184,13	8.064,86	98,54	119,26	1,46
ORM_P22	21.385,33	20.495,73	95,84	889,60	4,16
ORM_P23	129.565,40	42.742,73	32,99	86.822,66	67,01
ORM_P24	267.686,20	39.290,93	14,68	228.395,26	85,32
ORM_P25	377.673,13	32.902,93	8,71	344.770,20	91,29
ORM_P26	8.466,20	8.379,86	98,98	86,33	1,02
ORM_P27	20.227,73	19.375,60	95,79	852,13	4,21
ORM_P28	209.621,26	66.124,66	31,54	143.496,60	68,46
ORM_P29	382.507,86	53.714,86	14,04	328.793,00	85,96
ORM_P30	631.629,46	64.224,00	10,17	567.405,46	89,83
ORM_P31	9.747,80	9.670,00	99,20	77,80	0,80
ORM_P32	21.347,13	21.064,80	98,68	282,33	1,32
ORM_P33	265.892,80	81.509,33	30,65	184.383,46	69,35
ORM_P34	543.119,73	89.044,13	16,39	454.075,60	83,61
ORM_P35	8.439,46	8.399,33	99,52	40,13	0,48
ORM_P36	28.282,93	27.610,66	97,62	672,26	2,38
ORM_P37	349.121,40	129.600,86	37,12	219.520,53	62,88
ORM_P38	11.655,33	11.581,06	99,36	74,26	0,64
ORM_P39	27.916,26	27.562,13	98,73	354,13	1,27
ORM_P40	476.902,40	165.215,13	34,64	311.687,26	65,36

Tabela 3.9: Quantidade média de vizinhos em 15 iterações - ORM_P ($w = 10$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
GRM_P1	1.425,20	1.304,40	91,52	120,80	8,48
GRM_P2	2.947,06	2.705,46	91,80	241,60	8,20
GRM_P3	6.894,26	5.722,06	83,00	1.172,20	17,00
GRM_P4	7.647,80	3.475,33	45,44	4.172,46	54,56
GRM_P5	9.762,80	2.049,13	20,99	7.713,66	79,01
GRM_P6	7.866,40	739,46	9,40	7.126,93	90,60
GRM_P7	8.301,66	382,46	4,61	7.919,20	95,39
GRM_P8	8.615,86	223,60	2,60	8.392,26	97,40
GRM_P9	6.239,00	0,00	0,00	6.239,00	100,00
GRM_P10	1.572,86	1.525,13	96,97	47,73	3,03
GRM_P11	5.348,86	4.900,33	91,61	448,53	8,39
GRM_P12	8.709,33	7.893,33	90,63	816,00	9,37
GRM_P13	13.314,66	11.867,86	89,13	1.446,80	10,87
GRM_P14	16.273,20	13.040,53	80,14	3.232,66	19,86
GRM_P15	17.428,66	11.568,46	66,38	5.860,20	33,62
GRM_P16	22.439,06	11.085,46	49,40	11.353,60	50,60
GRM_P17	23.661,26	8.815,46	37,26	14.845,80	62,74
GRM_P18	24.789,53	6.597,20	26,61	18.192,33	73,39
GRM_P19	22.885,66	4.103,80	17,93	18.781,86	82,07
GRM_P20	24.877,66	2.114,33	8,50	22.763,33	91,50

Tabela 3.10: Quantidade média de vizinhos em 15 iterações - GRM_P ($w = 2$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
GRM_P1	1.700,33	1.053,73	61,97	646,60	38,03
GRM_P2	3.315,00	1.940,80	58,55	1.374,20	41,45
GRM_P3	5.765,06	2.744,86	47,61	3.020,20	52,39
GRM_P4	8.221,66	1.893,53	23,03	6.328,13	76,97
GRM_P5	6.736,00	930,00	13,81	5.806,00	86,19
GRM_P6	6.860,26	368,33	5,37	6.491,93	94,63
GRM_P7	8.919,93	156,73	1,76	8.763,20	98,24
GRM_P8	7.645,66	42,26	0,55	7.603,40	99,45
GRM_P9	6.403,00	0,00	0,00	6.403,00	100,00
GRM_P10	2.170,46	1.814,26	83,59	356,20	16,41
GRM_P11	5.974,26	4.222,33	70,68	1.751,93	29,32
GRM_P12	10.577,93	6.003,13	56,75	4.574,80	43,25
GRM_P13	12.275,13	5.953,46	48,50	6.321,66	51,50
GRM_P14	14.534,86	6.542,46	45,01	7.992,40	54,99
GRM_P15	20.085,66	6.924,93	34,48	13.160,73	65,52
GRM_P16	19.496,66	5.342,46	27,40	14.154,20	72,60
GRM_P17	21.362,26	4.879,60	22,84	16.482,66	77,16
GRM_P18	23.552,40	4.109,93	17,45	19.442,46	82,55
GRM_P19	25.591,86	2.838,53	11,09	22.753,33	88,91
GRM_P20	26.403,00	1.043,13	3,95	25.359,86	96,05

Tabela 3.11: Quantidade média de vizinhos em 15 iterações - GRM_P ($w = 5$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
GRM_P1	1.451,33	634,46	43,72	816,86	56,28
GRM_P2	3.421,46	1.534,20	44,84	1.887,26	55,16
GRM_P3	6.045,86	2.560,06	42,34	3.485,80	57,66
GRM_P4	8.591,20	1.804,00	21,00	6.787,20	79,00
GRM_P5	6.690,20	812,26	12,14	5.877,93	87,86
GRM_P6	8.791,13	585,00	6,65	8.206,13	93,35
GRM_P7	7.348,26	166,86	2,27	7.181,40	97,73
GRM_P8	8.320,46	62,66	0,75	8.257,80	99,25
GRM_P9	15.945,00	21,00	0,13	15.924,00	99,87
GRM_P10	2.569,60	1.633,86	63,58	935,73	36,42
GRM_P11	6.458,66	3.301,20	51,11	3.157,46	48,89
GRM_P12	10.576,86	4.663,13	44,09	5.913,73	55,91
GRM_P13	13.350,80	5.506,93	41,25	7.843,86	58,75
GRM_P14	18.605,86	6.880,13	36,98	11.725,73	63,02
GRM_P15	18.673,73	5.442,26	29,14	13.231,46	70,86
GRM_P16	20.380,33	4.851,00	23,80	15.529,33	76,20
GRM_P17	24.361,20	5.094,73	20,91	19.266,46	79,09
GRM_P18	21.486,00	3.273,66	15,24	18.212,33	84,76
GRM_P19	26.423,00	2.383,13	9,02	24.039,86	90,98
GRM_P20	28.064,40	1.093,66	3,90	26.970,73	96,10

Tabela 3.12: Quantidade média de vizinhos em 15 iterações - GRM_P ($w = 10$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
SLM_P1	528.941,40	285.950,86	54,06	242.990,53	45,94
SLM_P2	951.414,40	565.581,13	59,45	385.833,26	40,55
SLM_P3	1.447.763,80	862.978,86	59,61	584.784,93	40,39

Tabela 3.13: Quantidade média de vizinhos em 15 iterações - SLM_P ($w = 2$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
SLM_P1	633.183,26	111.967,66	17,68	521.215,60	82,32
SLM_P2	915.941,66	210.897,26	23,03	705.044,40	76,97
SLM_P3	1.131.112,33	228.465,46	20,20	902.646,86	79,80

Tabela 3.14: Quantidade média de vizinhos em 15 iterações - SLM_P ($w = 5$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
SLM_P1	508.708,00	275.262,93	54,11	233.445,06	45,89
SLM_P2	959.139,93	596.434,33	62,18	362.705,60	37,82
SLM_P3	1.318.937,20	817.906,40	62,01	501.030,80	37,99

Tabela 3.15: Quantidade média de vizinhos em 15 iterações - SLM_P ($w = 10$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
ORM_P1	0,04	**0,00**	**0,00**	**0,00**
ORM_P2	0,10	0,01	0,01	**0,00**
ORM_P3	0,07	0,02	0,01	**0,00**
ORM_P4	0,23	0,08	0,05	**0,00**
ORM_P5	0,58	0,23	0,19	**0,01**
ORM_P6	0,22	0,02	**0,01**	**0,01**
ORM_P7	0,52	0,04	0,03	**0,01**
ORM_P8	1,22	0,14	0,10	**0,03**
ORM_P9	2,95	0,85	0,83	**0,05**
ORM_P10	8,61	2,60	2,36	**0,09**
ORM_P11	0,39	0,02	**0,01**	**0,01**
ORM_P12	1,30	0,03	0,03	**0,02**
ORM_P13	6,36	0,57	0,52	**0,06**
ORM_P14	19,41	3,45	3,22	**0,14**
ORM_P15	39,00	11,90	12,26	**0,23**
ORM_P16	0,66	0,04	**0,03**	**0,03**
ORM_P17	2,39	0,06	0,05	**0,04**
ORM_P18	21,52	1,22	1,06	**0,13**
ORM_P19	57,22	8,40	8,13	**0,28**
ORM_P20	93,43	38,43	31,80	**0,47**
ORM_P21	1,22	0,06	**0,05**	**0,05**
ORM_P22	4,22	0,08	0,08	**0,06**
ORM_P23	49,18	1,95	2,01	**0,23**
ORM_P24	114,18	23,63	21,27	**0,58**
ORM_P25	179,64	74,81	78,56	**0,94**
ORM_P26	2,23	0,08	**0,07**	**0,07**
ORM_P27	7,42	0,13	0,12	**0,08**
ORM_P28	109,14	3,96	4,71	**0,41**
ORM_P29	250,12	41,66	43,78	**1,02**
ORM_P30	328,84	145,29	147,34	**1,68**
ORM_P31	3,55	0,11	0,10	**0,09**
ORM_P32	11,13	0,15	0,15	**0,12**
ORM_P33	158,45	5,33	5,20	**0,63**
ORM_P34	461,26	79,16	80,46	**1,60**
ORM_P35	4,15	0,14	0,13	**0,12**
ORM_P36	17,28	0,22	0,20	**0,15**
ORM_P37	345,57	8,57	9,63	**0,86**
ORM_P38	7,59	0,17	0,17	**0,14**
ORM_P39	17,64	0,25	0,25	**0,19**
ORM_P40	503,35	26,70	21,40	**1,40**

Tabela 3.16: Tempos de execução em segundos *tam* - ORM_P ($w = 2$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
ORM_P1	0,04	0,01	**0,00**	**0,00**
ORM_P2	0,08	0,05	0,02	**0,00**
ORM_P3	0,08	0,05	0,02	**0,00**
ORM_P4	0,36	0,17	0,09	**0,01**
ORM_P5	0,92	0,30	0,27	**0,02**
ORM_P6	0,15	0,02	0,02	**0,01**
ORM_P7	0,45	0,11	0,06	**0,02**
ORM_P8	1,24	0,50	0,35	**0,03**
ORM_P9	4,81	1,82	1,57	**0,06**
ORM_P10	15,48	3,85	3,75	**0,10**
ORM_P11	0,32	0,03	0,02	**0,01**
ORM_P12	1,45	0,07	0,05	**0,02**
ORM_P13	6,44	2,81	2,20	**0,06**
ORM_P14	22,95	9,26	7,88	**0,16**
ORM_P15	74,74	26,68	22,38	**0,29**
ORM_P16	0,67	0,04	**0,03**	**0,03**
ORM_P17	2,39	0,08	0,07	**0,04**
ORM_P18	21,21	7,89	7,21	**0,17**
ORM_P19	70,87	31,28	35,07	**0,33**
ORM_P20	166,28	69,43	65,38	**0,59**
ORM_P21	1,18	0,06	0,06	**0,05**
ORM_P22	3,91	0,13	0,11	**0,06**
ORM_P23	45,59	16,06	12,89	**0,29**
ORM_P24	140,41	69,84	76,02	**0,69**
ORM_P25	334,31	145,56	146,73	**1,24**
ORM_P26	1,89	0,09	**0,07**	**0,07**
ORM_P27	5,93	0,19	0,14	**0,08**
ORM_P28	91,86	36,93	30,18	**0,48**
ORM_P29	283,19	136,87	151,61	**1,16**
ORM_P30	670,77	286,64	263,79	**1,98**
ORM_P31	2,96	0,12	0,10	**0,09**
ORM_P32	8,04	0,18	0,15	**0,11**
ORM_P33	166,04	53,38	55,88	**0,84**
ORM_P34	428,22	247,81	243,00	**1,91**
ORM_P35	4,64	0,14	0,13	**0,12**
ORM_P36	13,97	0,29	0,24	**0,16**
ORM_P37	289,71	80,17	93,04	**1,23**
ORM_P38	7,85	0,20	0,17	**0,15**
ORM_P39	14,34	0,29	0,26	**0,20**
ORM_P40	485,50	156,13	160,21	**1,97**

Tabela 3.17: Tempos de execução em segundos *tam* - ORM_P ($w = 5$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
ORM_P1	0,02	0,02	**0,00**	**0,00**
ORM_P2	0,11	0,07	0,02	**0,00**
ORM_P3	0,09	0,06	0,02	**0,00**
ORM_P4	0,38	0,17	0,11	**0,01**
ORM_P5	0,86	0,36	0,25	**0,02**
ORM_P6	0,15	0,04	0,02	**0,01**
ORM_P7	0,44	0,22	0,08	**0,02**
ORM_P8	1,76	1,01	0,57	**0,03**
ORM_P9	5,94	2,55	1,71	**0,06**
ORM_P10	17,33	4,67	4,38	**0,10**
ORM_P11	0,32	0,05	0,03	**0,02**
ORM_P12	0,97	0,21	0,10	**0,03**
ORM_P13	9,04	4,70	3,19	**0,07**
ORM_P14	32,78	12,73	12,11	**0,17**
ORM_P15	90,19	30,01	28,02	**0,29**
ORM_P16	0,68	0,04	**0,03**	**0,03**
ORM_P17	2,03	0,19	0,13	**0,04**
ORM_P18	24,71	12,34	12,85	**0,16**
ORM_P19	102,98	45,96	40,56	**0,39**
ORM_P20	251,87	81,35	76,02	**0,67**
ORM_P21	1,25	0,08	0,07	**0,05**
ORM_P22	3,88	0,30	0,18	**0,07**
ORM_P23	67,17	28,97	27,93	**0,34**
ORM_P24	249,32	93,15	90,53	**0,78**
ORM_P25	541,13	168,75	193,96	**1,36**
ORM_P26	2,24	0,10	0,09	**0,07**
ORM_P27	5,78	0,41	0,31	**0,09**
ORM_P28	118,80	66,60	54,17	**0,56**
ORM_P29	435,11	185,42	181,79	**1,32**
ORM_P30	1.001,61	379,25	374,18	**2,29**
ORM_P31	3,34	0,14	0,11	**0,09**
ORM_P32	8,70	0,27	0,22	**0,11**
ORM_P33	222,22	113,79	104,48	**0,86**
ORM_P34	784,10	323,75	286,87	**2,06**
ORM_P35	4,30	0,15	0,13	**0,12**
ORM_P36	14,73	0,59	0,45	**0,16**
ORM_P37	413,42	194,84	177,49	**1,33**
ORM_P38	7,88	0,22	0,20	**0,14**
ORM_P39	21,47	0,48	0,45	**0,19**
ORM_P40	694,46	314,49	317,05	**1,84**

Tabela 3.18: Tempos de execução em segundos *tam* - ORM_P ($w = 10$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
GRM_P1	0,10	0,02	0,03	**0,00**
GRM_P2	0,24	0,05	0,05	**0,01**
GRM_P3	0,40	0,18	0,17	**0,02**
GRM_P4	0,62	0,57	0,58	**0,03**
GRM_P5	1,01	1,09	1,10	**0,04**
GRM_P6	1,33	1,07	1,08	**0,05**
GRM_P7	1,67	1,26	1,28	**0,05**
GRM_P8	1,84	1,37	1,38	**0,06**
GRM_P9	2,07	1,07	1,08	**0,06**
GRM_P10	0,38	0,02	0,02	**0,01**
GRM_P11	0,94	0,15	0,15	**0,02**
GRM_P12	1,69	0,30	0,29	**0,04**
GRM_P13	3,10	0,55	0,55	**0,06**
GRM_P14	3,53	1,21	1,21	**0,08**
GRM_P15	4,50	2,24	2,25	**0,13**
GRM_P16	4,78	4,43	4,44	**0,14**
GRM_P17	7,41	6,25	6,04	**0,17**
GRM_P18	8,76	8,27	8,06	**0,18**
GRM_P19	10,41	8,88	8,74	**0,24**
GRM_P20	17,16	11,18	11,00	**0,23**

Tabela 3.19: Tempos de execução em segundos *tam* - GRM_P ($w = 2$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
GRM_P1	0,07	0,07	0,07	**0,01**
GRM_P2	0,22	0,16	0,16	**0,01**
GRM_P3	0,40	0,37	0,37	**0,02**
GRM_P4	0,57	0,76	0,76	**0,03**
GRM_P5	0,89	0,75	0,72	**0,04**
GRM_P6	1,03	0,84	0,84	**0,04**
GRM_P7	1,48	1,15	1,16	**0,05**
GRM_P8	1,87	1,06	1,07	**0,05**
GRM_P9	1,78	1,01	1,02	**0,06**
GRM_P10	0,31	0,08	0,09	**0,01**
GRM_P11	0,95	0,45	0,46	**0,03**
GRM_P12	1,89	1,27	1,27	**0,05**
GRM_P13	3,21	1,94	1,95	**0,08**
GRM_P14	3,57	2,61	2,64	**0,09**
GRM_P15	4,55	4,22	4,26	**0,11**
GRM_P16	4,78	4,64	4,69	**0,15**
GRM_P17	6,83	5,77	6,06	**0,18**
GRM_P18	7,77	7,04	7,77	**0,19**
GRM_P19	12,49	8,67	10,00	**0,21**
GRM_P20	18,32	11,24	12,01	**0,28**

Tabela 3.20: Tempos de execução em segundos *tam* - GRM_P ($w = 5$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
GRM_P1	0,08	0,07	0,07	**0,01**
GRM_P2	0,24	0,20	0,21	**0,01**
GRM_P3	0,46	0,39	0,39	**0,02**
GRM_P4	0,65	0,76	0,76	**0,03**
GRM_P5	0,93	0,71	0,71	**0,03**
GRM_P6	1,18	1,02	1,02	**0,04**
GRM_P7	1,45	0,94	0,94	**0,05**
GRM_P8	1,90	1,13	1,13	**0,05**
GRM_P9	2,54	2,58	2,66	**0,07**
GRM_P10	0,31	0,19	0,20	**0,01**
GRM_P11	0,87	0,73	0,73	**0,03**
GRM_P12	1,73	1,55	1,63	**0,05**
GRM_P13	2,81	2,21	2,46	**0,08**
GRM_P14	3,52	3,58	4,00	**0,11**
GRM_P15	4,81	4,05	4,09	**0,13**
GRM_P16	5,33	4,99	5,09	**0,14**
GRM_P17	7,92	6,34	6,69	**0,16**
GRM_P18	9,54	6,78	7,03	**0,17**
GRM_P19	11,90	9,50	9,76	**0,20**
GRM_P20	21,33	11,54	11,82	**0,25**

Tabela 3.21: Tempos de execução em segundos *tam* - GRM_P ($w = 10$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
SLM_P1	658,54	248,46	249,98	**2,45**
SLM_P2	1.168,80	512,17	503,11	**3,66**
SLM_P3	1.462,00	1.030,78	837,21	**6,19**

Tabela 3.22: Tempos de execução em segundos *tam* - SLM_P ($w = 2$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
SLM_P1	1.152,48	487,34	482,47	**2,99**
SLM_P2	1.872,48	911,50	834,59	**5,32**
SLM_P3	2.595,91	1.356,90	1.346,67	**8,27**

Tabela 3.23: Tempos de execução em segundos *tam* - SLM_P ($w = 5$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
SLM_P1	1.732,61	648,40	619,43	**3,36**
SLM_P2	2.943,57	1.102,94	1.018,01	**5,55**
SLM_P3	3.873,29	1.702,27	1.745,37	**8,91**

Tabela 3.24: Tempos de execução em segundos *tam* - SLM_P ($w = 10$).

3.4 Busca Local Concatenada

BL_BTMA obteve os menores tempos de processamento em relação às outras buscas para todas as instâncias testadas. Entretanto, houve perdas na qualidade das soluções encontradas comparando-se com as demais. Por outro lado, BL_Teste obtém soluções de boa qualidade comparadas as soluções encontradas por BL_Básica, com ganhos significativos em tempos de processamento em relação ao algoritmo básico. Porém, BL_Teste apresenta tempos de processamento superiores às buscas BL_B e, principalmente, BL_BTMA. A partir dessas considerações, pode-se concatenar as duas buscas locais: após a construção da solução inicial, executa-se primeiramente a busca local mais rápida (BL_BTMA) até que se encontre um ótimo local, para, em seguida, executar-se a busca local mais lenta (BL_Teste). Uma estratégia híbrida, concatenando duas buscas locais, foi aplicada ao problema de Steiner em [35].

O objetivo é tentar acelerar BL_Teste mantendo-se a mesma qualidade das soluções encontradas por essa busca. Assim, na análise da qualidade das soluções e dos tempos de processamento, BL_Conc (busca local concatenada) será comparada somente às buscas BL_Básica e BL_Teste.

3.4.1 Qualidade das Soluções

As Tabelas 3.25 a 3.27 mostram um grande equilíbrio entre BL_Básica, BL_Teste e BL_Conc nas medidas *drpm* e *cm*.

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P (40 instâncias)	BL_Básica	0,0009	**1,79**	**30**
	BL_Teste	0,0138	2,23	28
	BL_Conc	**-0,0147**	1,98	27
GRM_P (20 instâncias)	BL_Básica	0,0378	2,30	18
	BL_Teste	-0,0172	**1,83**	**19**
	BL_Conc	**-0,0206**	1,87	17
SLM_P (3 instâncias)	BL_Básica	0,0001	2,33	1
	BL_Teste	0,0120	2,00	**2**
	BL_Conc	**-0,0121**	**1,67**	0

Tabela 3.25: Qualidade relativa das buscas locais, incluindo BL_Conc - ($w = 2$).

Analisando-se a coluna *drpm*, BL_Básica é melhor em três grupos de instâncias: $w = 5$ (SLM_P) e $w = 10$ (ORM_P e SLM_P). BL_Teste também é melhor em três grupos de instâncias: $w = 5$ (ORM_P e GRM_P) e $w = 10$ (GRM_P). BL_Conc é melhor em três grupos de instâncias: $w = 2$ (ORM_P, GRM_P e SLM_P). Na medida relativa *cm*, BL_Básica é melhor em quatro grupos de instâncias: $w = 2$ (ORM_P), $w = 5$ (SLM_P) e $w = 10$ (ORM_P e SLM_P). BL_Teste é melhor em cinco grupos de instâncias: $w = 2$ (GRM_P),

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P (40 instâncias)	BL_Básica	-0,0029	2,10	**33**
	BL_Teste	**-0,0081**	**1,83**	30
	BL_Conc	0,011	2,07	29
GRM_P (20 instâncias)	BL_Básica	0,0209	2,42	**17**
	BL_Teste	**-0,0206**	**1,68**	15
	BL_Conc	-0,0003	1,90	13
SLM_P (3 instâncias)	BL_Básica	**-0,0299**	**2,00**	1
	BL_Teste	0,0206	**2,00**	**2**
	BL_Conc	0,0093	**2,00**	**2**

Tabela 3.26: Qualidade relativa das buscas locais, incluindo BL_Conc - ($w = 5$).

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P (40 instâncias)	BL_Básica	**-0,0917**	**1,64**	**34**
	BL_Teste	0,0033	1,96	31
	BL_Conc	0,0884	2,40	23
GRM_P (20 instâncias)	BL_Básica	0,0394	2,40	17
	BL_Teste	**-0,020**	**1,55**	16
	BL_Conc	-0,019	2,05	**18**
SLM_P (3 instâncias)	BL_Básica	**-0,2264**	**1,33**	**2**
	BL_Teste	-0,0559	1,67	1
	BL_Conc	0,2823	3,00	0

Tabela 3.27: Qualidade relativa das buscas locais, incluindo BL_Conc - ($w = 10$).

$w = 5$ (ORM_P, GRM_P e SLM_P) e $w = 10$ (GRM_P). BL_Conc é melhor em dois grupos de instâncias: $w = 2$ (SLM_P) e $w = 5$ (SLM_P).

Na análise da medida *melhor*, observa-se novamente um equilíbrio entre BL_Básica e BL_Teste, seguidas por BL_Conc. BL_Básica é superior às demais em cinco grupos de instâncias: $w = 2$ (ORM_P), $w = 5$ (ORM_P e GRM_P) e $w = 10$ (ORM_P e SLM_P). BL_Teste é superior às demais em três grupos de instâncias: $w = 2$ (GRM_P e SLM_P) e $w = 5$ (SLM_P). BL_Conc é superior às demais em dois grupos de instâncias: $w = 5$ (SLM_P) e $w = 10$ (GRM_P).

Levando-se em consideração todas as medidas, pode-se dizer que, a concatenação das buscas locais (BL_BTMA seguida pela BL_Teste) mantém, em geral, a mesma qualidade das soluções encontradas pelas buscas locais individualmente.

3.4.2 Tempos de Processamento

Os ganhos obtidos pela concatenação das buscas locais ocorreram principalmente nos tempos de processamento, como mostram as Tabelas 3.28 a 3.36 que apresentam os tempos absolutos médios em 15 iterações realizadas para cada instância-algoritmo, incluindo agora os resultados da busca local concatenada.

Instância	BL_Básica	BL_Teste	BL_Conc
ORM_P1	0,04	**0,00**	**0,00**
ORM_P2	0,10	0,01	**0,00**
ORM_P3	0,07	0,02	**0,01**
ORM_P4	0,23	0,08	**0,05**
ORM_P5	0,58	0,23	**0,17**
ORM_P6	0,22	**0,02**	**0,02**
ORM_P7	0,52	0,04	**0,03**
ORM_P8	1,22	0,14	**0,11**
ORM_P9	2,95	0,85	**0,46**
ORM_P10	8,61	2,60	**1,58**
ORM_P11	0,39	**0,02**	**0,02**
ORM_P12	1,30	**0,03**	**0,03**
ORM_P13	6,36	0,57	**0,43**
ORM_P14	19,41	3,45	**1,95**
ORM_P15	39,00	11,90	**8,08**
ORM_P16	0,66	**0,04**	**0,04**
ORM_P17	2,39	**0,06**	**0,06**
ORM_P18	21,52	1,22	**0,84**
ORM_P19	57,22	8,40	**6,28**
ORM_P20	93,43	38,43	**19,96**
ORM_P21	1,22	**0,06**	**0,06**
ORM_P22	4,22	**0,08**	0,10
ORM_P23	49,18	1,95	**1,51**
ORM_P24	114,18	23,63	**16,14**
ORM_P25	179,64	74,81	**54,13**
ORM_P26	2,23	**0,08**	**0,08**
ORM_P27	7,42	**0,13**	**0,13**
ORM_P28	109,14	3,96	**3,19**
ORM_P29	250,12	41,66	**33,05**
ORM_P30	328,84	145,29	**90,25**
ORM_P31	3,55	**0,11**	0,12
ORM_P32	11,13	**0,15**	0,17
ORM_P33	158,45	5,33	**3,22**
ORM_P34	461,26	79,16	**45,65**
ORM_P35	4,15	**0,14**	**0,14**
ORM_P36	17,28	0,22	**0,21**
ORM_P37	345,57	8,57	**7,21**
ORM_P38	7,59	**0,17**	0,20
ORM_P39	17,64	**0,25**	0,30
ORM_P40	503,35	26,70	**18,12**

Tabela 3.28: Tempos de execução em segundos *tam*, incluindo BL_Conc - ORM_P ($w = 2$).

Instância	BL_Básica	BL_Teste	BL_Conc
ORM_P1	0,04	**0,01**	**0,01**
ORM_P2	0,08	0,05	**0,03**
ORM_P3	0,08	0,05	**0,04**
ORM_P4	0,36	0,17	**0,06**
ORM_P5	0,92	0,30	**0,09**
ORM_P6	0,15	**0,02**	**0,02**
ORM_P7	0,45	0,11	**0,09**
ORM_P8	1,24	0,50	**0,29**
ORM_P9	4,81	1,82	**1,19**
ORM_P10	15,48	3,85	**1,35**
ORM_P11	0,32	0,03	**0,02**
ORM_P12	1,45	**0,07**	**0,07**
ORM_P13	6,44	2,81	**1,64**
ORM_P14	22,95	9,26	**3,97**
ORM_P15	74,74	26,68	**6,30**
ORM_P16	0,67	**0,04**	**0,04**
ORM_P17	2,39	**0,08**	**0,08**
ORM_P18	21,21	7,89	**4,35**
ORM_P19	70,87	31,28	**13,10**
ORM_P20	166,28	69,43	**22,43**
ORM_P21	1,18	**0,06**	0,07
ORM_P22	3,91	**0,13**	0,16
ORM_P23	45,59	16,06	**13,64**
ORM_P24	140,41	69,84	**35,99**
ORM_P25	334,31	145,56	**50,59**
ORM_P26	1,89	0,09	**0,08**
ORM_P27	5,93	0,19	**0,17**
ORM_P28	91,86	36,93	**13,22**
ORM_P29	283,19	136,87	**68,38**
ORM_P30	670,77	286,64	**134,31**
ORM_P31	2,96	**0,12**	**0,12**
ORM_P32	8,04	**0,18**	**0,18**
ORM_P33	166,04	53,38	**31,19**
ORM_P34	428,22	247,81	**101,95**
ORM_P35	4,64	**0,14**	**0,14**
ORM_P36	13,97	0,29	**0,28**
ORM_P37	289,71	80,17	**71,15**
ORM_P38	7,85	**0,20**	**0,20**
ORM_P39	14,34	**0,29**	0,30
ORM_P40	485,50	156,13	**111,72**

Tabela 3.29: Tempos de execução em segundos *tam*, incluindo BL_Conc - ORM_P ($w = 5$).

Instância	BL_Básica	BL_Teste	BL_Conc
ORM_P1	0,02	0,02	**0,01**
ORM_P2	0,11	0,07	**0,03**
ORM_P3	0,09	0,06	**0,03**
ORM_P4	0,38	0,17	**0,05**
ORM_P5	0,86	0,36	**0,09**
ORM_P6	0,15	**0,04**	0,05
ORM_P7	0,44	0,22	**0,12**
ORM_P8	1,76	1,01	**0,24**
ORM_P9	5,94	2,55	**0,71**
ORM_P10	17,33	4,67	**0,99**
ORM_P11	0,32	0,05	**0,04**
ORM_P12	0,97	0,21	**0,16**
ORM_P13	9,04	4,70	**0,90**
ORM_P14	32,78	12,73	**5,32**
ORM_P15	90,19	30,01	**8,48**
ORM_P16	0,68	**0,04**	**0,04**
ORM_P17	2,03	0,19	**0,14**
ORM_P18	24,71	12,34	**2,82**
ORM_P19	102,98	45,96	**18,11**
ORM_P20	251,87	81,35	**22,85**
ORM_P21	1,25	**0,08**	**0,08**
ORM_P22	3,88	0,30	**0,24**
ORM_P23	67,17	28,97	**9,70**
ORM_P24	249,32	93,15	**39,08**
ORM_P25	541,13	168,75	**53,74**
ORM_P26	2,24	**0,10**	**0,10**
ORM_P27	5,78	**0,41**	0,42
ORM_P28	118,80	66,60	**33,10**
ORM_P29	435,11	185,42	**68,13**
ORM_P30	1.001,61	379,25	**134,87**
ORM_P31	3,34	**0,14**	**0,14**
ORM_P32	8,70	0,27	**0,24**
ORM_P33	222,22	113,79	**49,66**
ORM_P34	784,10	323,75	**83,84**
ORM_P35	4,30	**0,15**	**0,15**
ORM_P36	14,73	0,59	**0,45**
ORM_P37	413,42	194,84	**96,01**
ORM_P38	7,88	**0,22**	**0,22**
ORM_P39	21,47	0,48	**0,46**
ORM_P40	694,46	314,49	**208,72**

Tabela 3.30: Tempos de execução em segundos *tam*, incluindo BL_Conc - ORM_P ($w = 10$).

Instância	BL_Básica	BL_Teste	BL_Conc
GRM_P1	0,10	**0,02**	**0,02**
GRM_P2	0,24	0,05	**0,03**
GRM_P3	0,40	0,18	**0,10**
GRM_P4	0,62	0,57	**0,29**
GRM_P5	1,01	1,09	**0,61**
GRM_P6	1,33	1,07	**0,67**
GRM_P7	1,67	1,26	**0,75**
GRM_P8	1,84	1,37	**0,79**
GRM_P9	2,07	1,07	**0,49**
GRM_P10	0,38	**0,02**	**0,02**
GRM_P11	0,94	0,15	**0,09**
GRM_P12	1,69	0,30	**0,17**
GRM_P13	3,10	0,55	**0,27**
GRM_P14	3,53	1,21	**0,46**
GRM_P15	4,50	2,24	**1,01**
GRM_P16	4,78	4,43	**2,82**
GRM_P17	7,41	6,25	**2,61**
GRM_P18	8,76	8,27	**3,86**
GRM_P19	10,41	8,88	**4,76**
GRM_P20	17,16	11,18	**4,69**

Tabela 3.31: Tempos de execução em segundos *tam*, incluindo BL_Conc - GRM_P ($w = 2$).

Analisando-se os resultados das tabelas, BL_Conc obtém ganhos significativos em tempos de processamento em relação às buscas BL_Teste e BL_Básica, principalmente nas maiores instâncias testadas.

Assim, levando-se em consideração a qualidade e o tempo, BL_BTMA consegue acelerar BL_Teste sem perder soluções.

3.5 Considerações Finais

Este capítulo apresentou um estudo sobre algoritmos de busca local para o problema das p-medianas conectadas.

Um algoritmo de busca local necessita de uma solução inicial gerada aleatoriamente ou através de uma heurística construtiva. O método escolhido para a geração das soluções iniciais nos algoritmos de busca local localiza as p facilidades nos vértices do grafo, servindo os usuários com a facilidade aberta mais próxima através da heurística Sample Greedy e conecta as p facilidades abertas por uma árvore de Steiner com a heurística Prim.

Em seguida, descreveu-se um algoritmo de busca local básico baseado naquele para o problema das p-medianas proposto em [47]. Algumas idéias como a estratégia de busca com melhoria iterativa e circularidade foram incorporadas ao algoritmo, com o objetivo de acelerar os passos da busca local.

Instância	BL_Básica	BL_Teste	BL_Conc
GRM_P1	0,07	0,07	**0,03**
GRM_P2	0,22	0,16	**0,08**
GRM_P3	0,40	0,37	**0,24**
GRM_P4	0,57	0,76	**0,45**
GRM_P5	0,89	0,75	**0,31**
GRM_P6	1,03	0,84	**0,34**
GRM_P7	1,48	1,15	**0,33**
GRM_P8	1,87	1,06	**0,37**
GRM_P9	1,78	1,01	**0,42**
GRM_P10	0,31	0,08	**0,05**
GRM_P11	0,95	0,45	**0,17**
GRM_P12	1,89	1,27	**0,41**
GRM_P13	3,21	1,94	**0,65**
GRM_P14	3,57	2,61	**1,18**
GRM_P15	4,55	4,22	**2,13**
GRM_P16	4,78	4,64	**3,04**
GRM_P17	6,83	5,77	**2,82**
GRM_P18	7,77	7,04	**3,16**
GRM_P19	12,49	8,67	**6,09**
GRM_P20	18,32	11,24	**5,52**

Tabela 3.32: Tempos de execução em segundos *tam*, incluindo BL_Conc - GRM_P ($w = 5$).

Instância	BL_Básica	BL_Teste	BL_Conc
GRM_P1	0,08	0,07	**0,03**
GRM_P2	0,24	0,20	**0,15**
GRM_P3	0,46	0,39	**0,23**
GRM_P4	0,65	0,76	**0,20**
GRM_P5	0,93	0,71	**0,24**
GRM_P6	1,18	1,02	**0,28**
GRM_P7	1,45	0,94	**0,35**
GRM_P8	1,90	1,13	**0,39**
GRM_P9	2,54	2,58	**0,49**
GRM_P10	0,31	0,19	**0,07**
GRM_P11	0,87	0,73	**0,36**
GRM_P12	1,73	1,55	**0,60**
GRM_P13	2,81	2,21	**0,73**
GRM_P14	3,52	3,58	**2,04**
GRM_P15	4,81	4,05	**1,71**
GRM_P16	5,33	4,99	**2,40**
GRM_P17	7,92	6,34	**2,25**
GRM_P18	9,54	6,78	**3,02**
GRM_P19	11,90	9,50	**3,54**
GRM_P20	21,33	11,54	**3,61**

Tabela 3.33: Tempos de execução em segundos *tam*, incluindo BL_Conc - GRM_P ($w = 10$).

Instância	BL_Básica	BL_Teste	BL_Conc
SLM_P1	658,54	248,46	**151,17**
SLM_P2	1.168,80	512,17	**237,70**
SLM_P3	1.462,00	1.030,78	**515,74**

Tabela 3.34: Tempos de execução em segundos *tam*, incluindo BL_Conc - SLM_P ($w = 2$).

Instância	BL_Básica	BL_Teste	BL_Conc
SLM_P1	1.152,48	487,34	**177,14**
SLM_P2	1.872,48	911,50	**353,95**
SLM_P3	2.595,91	1.356,90	**509,63**

Tabela 3.35: Tempos de execução em segundos *tam*, incluindo BL_Conc - SLM_P ($w = 5$).

Instância	BL_Básica	BL_Teste	BL_Conc
SLM_P1	1.732,61	648,40	**176,43**
SLM_P2	2.943,57	1.102,94	**481,27**
SLM_P3	3.873,29	1.702,27	**734,23**

Tabela 3.36: Tempos de execução em segundos *tam*, incluindo BL_Conc - SLM_P ($w = 10$).

Porém, conectar as p facilidades abertas por uma árvore de Steiner a cada nova troca realizada é uma tarefa muito custosa computacionalmente. Assim, um teste simples foi proposto com o objetivo de evitar, quando possível, a execução da heurística Prim. Candidatos a inserção que passam no teste são investigados e aqueles que não passam no teste são descartados. Incorporando-se essa idéia, três versões da busca local básica foram propostas: a primeira versão incorpora somente o teste; a segunda e a terceira, além de incorporarem o teste priorizam candidatos a inserção que são vizinhos da árvore de Steiner sem o vértice r. Os resultados computacionais mostraram que, levando-se em consideração a qualidade das soluções encontradas, duas buscas se destacaram: a busca local básica e a busca local com teste, confirmando-se assim a efetividade do descarte de vizinhos realizado nessa última busca. Levando-se em consideração os tempos de processamento, duas situações opostas se apresentaram: a busca local pelas bordas com teste da menor aresta é muito mais rápida que as demais e a busca local básica é muito mais lenta do que as demais devido ao alto custo computacional de se executar Prim a cada troca realizada.

Pelo acima apresentado, destacam-se, então, a busca local pelas bordas com teste da menor aresta, que é rápida mas apresenta perda na qualidade das soluções encontradas, e a busca local com teste, de qualidade comparável à busca local básica, porém mais lenta do que às demais buscas locais com descarte de vizinhos. Uma estratégia híbrida foi então implementada,

primeiramente executando-se a busca local mais rápida de qualidade inferior e, em seguida, executando-se a busca local mais lenta de qualidade superior. Os testes computacionais mostraram exatamente que, ao concatená-las, pode-se ganhar em velocidade sem perder a qualidade das soluções encontradas. Portanto, o algoritmo de busca local proposto para o problema das p-medianas conectadas é composto por duas buscas locais concatenadas, incorporando idéias tais como melhoria iterativa, circularidade e descarte de vizinhos com o objetivo de acelerar os passos da busca local, sem perder, no entanto, a qualidade das soluções encontradas.