3
Heurística Híbrida de Melhoria para BP

A heurística híbrida de melhoria proposta para o problema de *bin packing* é composta de três fases: `Construção`, `Redistribuição` e `Melhoria`. Neste capítulo apresenta-se a descrição detalhada de cada uma das três fases que compõem a heurística.

São dados o conjunto de objetos $N = \{1, \ldots, n\}$, os pesos (ou tamanhos) w_i dos objetos $i = 1, \ldots, n$ e a capacidade da caixa C. Deve-se lembrar (Seção 2.1) que na abordagem dual aqui utilizada o número m de caixas (processadores) também é um dado do problema e a capacidade da caixa (*makespan*) pode ser estendida. Seja S uma solução (não necessariamente viável) com m caixas. Associado à cada solução S existe uma lista de subconjuntos $(B_1, \ldots, B_m)$, representando, cada um deles, uma caixa. B_j será usado para identificar a j-ésima caixa, assim como o conjunto de objetos que nela estão contidos, para todo $j = 1, \ldots, m$. O peso da caixa B_j é dado pela soma dos pesos dos seus objetos $w_S(B_j) = \sum_{i \in B_j} w_i$. Uma caixa B_j $(j = 1, \ldots, m)$ pode estar em uma das seguintes situações:

- *não-utilizada*: $w_S(B_j) = 0$
- *completa*: $w_S(B_j) = C$
- *incompleta*: $0 < w_S(B_j) < C$
- *violada*: $w_S(B_j) > C$

Uma caixa está *saturada* se ela estiver completa ou violada. Uma caixa está *utilizada* se ela estiver completa, incompleta ou violada. A *capacidade residual* de uma caixa incompleta B_j é a diferença entre C e o peso de B_j. Objetos *livres* ou *restantes* são aqueles que ainda não foram empacotados em uma caixa. A função $\texttt{Viável}_{BP}(S)$ retorna `.VERDADEIRO.` caso a solução S seja viável para o problema de BP (isto é, $\max_{j=1}^{m} w_S(B_j) \leq C$) e `.FALSO.` caso contrário.

3.1 Fase de Construção

Cada iteração do procedimento híbrido de melhoria é iniciada pela fase de `Construção`. Nesta fase, cria-se uma solução viável $S = (B_1, \ldots, B_m)$ para o problema dual do *bin packing* (DBP) com exatamente m caixas. Inicialmente, m é igual a um determinado limite inferior calculado. Caso esta solução não seja viável para o problema de *bin-packing*, na próxima fase tenta-se viabilizar S utilizando-se o mesmo número de caixas. Quatro heurísticas construtivas são propostas. Cada uma delas considera uma diferente regra de seleção da caixa onde será colocado o próximo objeto, conforme explicado a seguir.

3.1.1 Dual Best-Fit Decreasing (`DBFD`)

Esta heurística utiliza uma regra de seleção da caixa semelhante à utilizada pela heurística `BFD`, descrita na Seção 2.2.2. Seleciona-se a caixa mais pesada cuja capacidade residual seja suficiente para o objeto corrente. Se tal caixa não existir, coloca-se o objeto na caixa mais leve, violando-se necessariamente a restrição de capacidade desta caixa.

A descrição detalhada do algoritmo `DBFD` é ilustrada pela Figura 3.1. Após a inicialização das m caixas na primeira linha, na linha 2 coloca-se o objeto mais pesado na caixa de índice 1. Na linha 3, j denota o índice da caixa corrente, $\bar{j}$ o maior índice das caixas incompletas e $\underline{j}$ o menor índice das caixas incompletas cuja capacidade residual seja suficiente para o objeto livre mais leve. O laço que vai das linhas 4 a 12 considera um objeto de cada vez. Na linha 5, a variável j é inicializada com o índice da caixa mais pesada cuja capacidade residual seja suficiente para pelo menos um objeto. Na próxima linha, verifica-se se existe alguma caixa incompleta que tenha capacidade residual suficiente para o objeto corrente e, se for o caso, seleciona-se a mais pesada delas. Na linha 7 trata-se o caso em que não existe caixa incompleta com capacidade residual suficiente, porém existe uma caixa não-utilizada. Atualiza-se a variável que guarda o maior índice das caixas incompletas com o índice da próxima caixa não-utilizada. Se não existe caixa com capacidade residual suficiente nem caixa não-utilizada, a caixa mais leve é selecionada na linha 8 como a caixa corrente. Na linha 9 atualiza-se o conteúdo da caixa corrente, que recebe o objeto corrente. A ordenação das caixas é refeita na linha 10. Por fim, na linha 11, a variável que

guarda o índice da primeira caixa incompleta que ainda possui capacidade residual para algum objeto é atualizado.

Deve-se notar que, considerando-se m caixas disponíveis, se a caixa mais pesada da solução S ultrapassar a capacidade máxima ($w_S(B_1) > C$), então uma solução gerada com $m + 1$ caixas disponíveis necessariamente utiliza todas as caixas.

Procedimento DBFD
Entrada: $N = \{1, \ldots, n\}$, C,
w_i $(i = 1, \ldots, n)$ onde $w_h \geq w_{h+1}, h = 1, \ldots, n-1$,
número m de caixas disponíveis.
Saída: $S = (B_1, \ldots, B_m)$ com $w_S(B_1) \geq \ldots \geq w_S(B_m)$.
1 **para-todo** $j = 1, \ldots, m$ **faça** $B_j = \emptyset$;
2 $B_1 \leftarrow B_1 \cup \{1\}$;
3 $j, \bar{j}, \underline{j} \leftarrow 1$;
4 **para-todo** $i = 2, \ldots, n$ **faça**
5 $j \leftarrow \underline{j}$;
6 **enquanto** $w_S(B_j) + w_i > C$ **e** $j \leq \bar{j}$ **faça** $j \leftarrow j + 1$;
7 **se** $j > \bar{j}$ **e** $j \leq m$ **então** $\bar{j} \leftarrow j$;
8 **senão se** $j > m$ **então** $j \leftarrow m$;
9 $B_j \leftarrow B_j \cup \{i\}$;
10 Reordena $(B_1, \ldots, B_m)$ tal que $w_S(B_1) \geq \ldots \geq w_S(B_m)$;
11 **enquanto** $C - w_S(B_{\underline{j}}) < w_n$ **e** $\underline{j} < m$ **faça** $\underline{j} \leftarrow \underline{j} + 1$;
12 **fim-para-todo**
13 **retorne**
fim DBFD

Figura 3.1: Pseudo-código da heurística construtiva DBFD

A complexidade do algoritmo é dominada pelo laço compreendido entre as linhas 4 e 12. No pior caso, as operações das linhas 6, 10 e 11 consideram todas as caixas e têm complexidade igual a $O(m)$. O laço é executado para cada um dos n objetos e portanto a complexidade final do algoritmo é igual a $O(n\,m)$.

3.1.2 Dual Worst Sum-Fit Decreasing (DWSFD)

A idéia desta heurística é tentar completar uma caixa de cada vez, inicialmente respeitando a restrição de capacidade. Para isso, o objeto livre mais pesado é colocado na caixa sendo considerada. A seguir, utiliza-se uma heurística para o problema de soma de conjuntos (*Subset Sum Problem* - SSP) visando encontrar um subconjunto de objetos cuja soma dos pesos mais se aproxime da capacidade residual desta caixa. Este processo é repetido para cada uma das caixas não-utilizadas. Para finalizar, os objetos

restantes, considerados em ordem não-crescente de peso, são colocados na caixa mais pesada com capacidade residual suficiente ou na mais leve das saturadas. A idéia de fixar o objeto mais pesado e tentar completar a capacidade residual da caixa também foi explorada em uma variante da heurística `MBS` [43], descrita em [25], em outra heurística pseudopolinomial [83] e em uma variante do procedimento *Fill Bin* apresentado em [87]. Procedimentos para o problema de soma de conjuntos também foram utilizados em [16, 62, 83].

Procedimento `DWSFD`
Entrada: $N = \{1, \ldots, n\}$, C,
w_i $(i = 1, \ldots, n)$ onde $w_h \geq w_{h+1}, h = 1, \ldots, n-1$,
número m de caixas disponíveis.
Saída: $S = (B_1, \ldots, B_m)$ com $w_S(B_1) \geq \ldots \geq w_S(B_m)$.
1 **para-todo** $j = 1, \ldots, m$ **faça**
2 $N = \{i_1, \ldots, i_l\}$ com $w_{i_1} \geq \ldots \geq w_{i_l}$;
3 $B_j \leftarrow \{i_1\}$;
4 $N \leftarrow N \setminus \{i_1\}$;
5 `folga` $\leftarrow C - w_S(B_j)$;
6 $N' \leftarrow \{i_k \in N : w_{i_k} \leq$ `folga`$\}$;
7 $\overline{N} \leftarrow$ `MTSS`$(N',$ `folga`$)$;
8 $B_j \leftarrow B_j \cup \overline{N}$;
9 $N \leftarrow N \setminus \overline{N}$;
10 **fim-para-todo**
11 Ordene $(B_1, \ldots, B_m)$ tal que $w_S(B_1) \geq \ldots \geq w_S(B_m)$;
12 $N = \{i_1, \ldots, i_l\}$ com $w_{i_1} \geq \ldots \geq w_{i_l}$;
13 **para-todo** $k = 1, \ldots, l$ **faça**
14 $B_m \leftarrow B_m \cup \{i_k\}$;
15 Reordena $(B_1, \ldots, B_m)$ tal que $w_S(B_1) \geq \ldots \geq w_S(B_m)$;
16 **fim-para-todo**
17 **retorne**
fim `DWSFD`

Figura 3.2: Pseudo-código da heurística construtiva `DWSFD`

A Figura 3.2 ilustra o algoritmo `DWSFD`. O primeiro laço compreende as linhas 1-10. Estando o vetor de pesos ordenado, o objeto livre mais pesado de N é identificado pelo índice i_1, o seguinte mais pesado pelo índice i_2 e assim sucessivamente (linhas 2 e 12). O objeto mais pesado é colocado na caixa corrente na linha 3 e o conjunto de objetos livres é atualizado na linha 4. A seguir, calcula-se, na linha 5, o valor da variável `folga` que representa a capacidade residual da caixa corrente. Cria-se, na linha 6, o subconjunto N' formado pelos objetos de N com peso menor ou igual a `folga`, isto é, que podem ser empacotados na caixa corrente. O procedimento `MTSS` [64], com $k = 3$, é executado na linha 7 para resolver o problema SSP. Dados o conjunto de objetos livres N' e a capacidade residual da caixa corrente (`folga`) a heurística retorna um subconjunto $\bar{N} \subseteq N'$ de objetos livres cujo

peso total não ultrapassa o valor da `folga`. Na linha 8 atualiza-se a caixa corrente e na linha 9 o conjunto de objetos livres. Na linha 11 faz-se a ordenação das caixas. O último laço das linhas 13-16 considera cada objeto livre, em ordem não-crescente de peso, empacotando-o na caixa mais leve, na linha 14.

Deve-se notar que, considerando-se uma solução S que possua alguma das m caixas com peso maior do que a capacidade máxima C, então a solução gerada com $m+1$ caixas, no laço compreendido entre as linhas 1-10, coloca, pelo menos, um objeto em cada uma das $m+1$ caixas.

O laço das linhas 1-10 é dominado pela complexidade da heurística `MTSS`, que é $O(n^3)$. Este laço é repetido para cada uma das m caixas. A ordenação das caixas na linha 11 tem complexidade igual a $O(m \log m)$. O laço das linhas 13-16 tem complexidade igual a $O(n\,m)$. A complexidade do algoritmo `DWSFD` é $O(m\,n^3)$.

3.1.3 Dual Best 3-Fit Decreasing (`DB3FD`)

`DB3FD` reserva uma caixa para cada um dos m objetos mais pesados. Em seguida, considerando-se cada uma das m caixas incompletas, tenta criar caixas completas com exatamente três objetos. Para os objetos restantes aplica-se a mesma regra utilizada em `DBFD`: coloca-se o objeto mais pesado livre na caixa mais pesada com capacidade residual suficiente ou na mais leve saturada. Considera-se que uma boa solução inicial possui o maior número possível de caixas completas. Isto pois, quanto menos caixas incompletas, mais espaço em cada uma delas, o que facilita empacotar os objetos livres. O princípio desta heurística é que verificar caixas completas com exatamente três objetos é computacionalmente viável. Uma variante possível seria criar caixas completas com mais objetos. Caixas completas com dois objetos são criadas pelo procedimento de redução `MTRP`, que será utilizado na fase de pré-processamento.

A Figura 3.3 ilustra a implementação proposta para o algoritmo `DB3FD`. O laço das linhas 1-4 coloca, em cada uma das m caixas, o objeto mais pesado ainda não empacotado. No próximo laço (linhas 5-21) cada caixa é inspecionada (da mais leve para a mais pesada) à procura de um par de objetos que complete sua capacidade residual (folga). Estando o vetor de pesos ordenado, o objeto livre mais pesado de N é identificado pelo índice i_1, o seguinte mais pesado pelo índice i_2 e assim sucessivamente (linhas 7 e 23). Na linha 8 inicializam-se os apontadores a para o objeto mais pesado e b para

Procedimento DB3FD
Entrada: $N = \{1, \ldots, n\}$, C,
w_i $(i = 1, \ldots, n)$ onde $w_h \geq w_{h+1}, h = 1, \ldots, n-1$
número m de caixas disponíveis.
Saída: $S = (B_1, \ldots, B_m)$ com $w_S(B_1) \geq \ldots \geq w_S(B_m)$.

1 **para-todo** $j = 1, \ldots, m$ **faça**
2 $B_j \leftarrow \{j\}$;
3 $N \leftarrow N \setminus \{j\}$;
4 **fim-para-todo**
5 **para-todo** $j = m, \ldots, 1$ **faça**
6 folga $\leftarrow C - w_S(B_j)$;
7 $N = \{i_1, \ldots, i_l\}$ com $w_{i_1} \geq \ldots \geq w_{i_l}$;
8 $a \leftarrow 1; b \leftarrow l$;
9 trocou $\leftarrow$ .FALSO.;
10 **enquanto** $a < b$ **e** $w_{i_a} \geq \frac{\text{folga}}{2}$ **e não** trocou **faça**
11 temp $\leftarrow$ folga $- w_{i_a}$;
12 **enquanto** $a < b$ **e** $w_{i_b} <$ temp **faça** $b \leftarrow b - 1$
13 **se** $w_{i_b} =$ temp **e** $a \neq b$ **então faça**
14 $B_j \leftarrow B_j \cup \{i_a, i_b\}$;
15 $N \leftarrow N \setminus \{i_a, i_b\}$;
16 trocou $\leftarrow$ .VERDADEIRO.;
17 **senão**
18 $a \leftarrow a + 1$
19 **fim-se**
20 **fim-enquanto**
21 **fim-para-todo**
22 Ordene $(B_1, \ldots, B_m)$ tal que $w_S(B_1) \geq \ldots \geq w_S(B_m)$;
23 $N = \{i_1, \ldots, i_l\}$ com $w_{i_1} \geq \ldots \geq w_{i_l}$;
24 $\bar{j} \leftarrow 1$;
25 **enquanto** $w_S(B_{\bar{j}}) + w_{i_l} > C$ **e** $\bar{j} < m$ **faça** $\bar{j} \leftarrow \bar{j} + 1$;
26 **para-todo** $k = 1, \ldots, l$ **faça**
27 $j \leftarrow \bar{j}$;
28 **enquanto** $w_S(B_j) + w_{i_k} > C$ **e** $j < m$ **faça** $j \leftarrow j + 1$;
29 $B_j \leftarrow B_j \cup \{i_k\}$;
30 Reordena $(B_1, \ldots, B_m)$ tal que $w_S(B_1) \geq \ldots \geq w_S(B_m)$;
31 **enquanto** $C - w_S(B_{\bar{j}}) < w_{i_l}$ **e** $\bar{j} < m$ **faça** $\bar{j} \leftarrow \bar{j} + 1$;
32 **fim-para-todo**
33 **retorne**
fim DB3FD

Figura 3.3: Pseudo-código da heurística construtiva DB3FD

o objeto mais leve. Na linha 9 inicializa-se a variável `trocou`. Esta é usada para eventualmente interromper a execução do laço compreendido entre as linhas 10-20. Os dois blocos **enquanto** (linhas 10 e 12) verificam, para cada objeto cujo peso é maior ou igual à metade da folga da caixa se existe um outro objeto complementar, isto é, cuja soma dos pesos dos objetos complete a folga da caixa. Se isto ocorrer, atualiza-se a caixa corrente e o conjunto de objetos livres nas linhas 14-16. Uma nova caixa é inspecionada caso a caixa corrente tenha sido totalmente preenchida (variável `trocou` com o valor `.VERDADEIRO.`) ou não exista um par de objetos que a complete. As caixas são ordenadas na linha 22. A variável $\bar{j}$ que guarda o índice da primeira caixa incompleta que ainda possui capacidade residual para receber algum objeto é atualizada nas linhas 24-25. O último laço (linhas 26-32) percorre o conjunto dos objetos restantes. Na linha 29 cada objeto é colocado na caixa mais pesada com capacidade suficiente ou na caixa mais leve das saturadas. Na linha 30 as caixas são reordenadas e na linha 31 a variável $\bar{j}$ é atualizada.

Este procedimento coloca pelo menos um objeto em cada uma das m caixas (linhas 1-4), portanto, ao final do procedimento, todas as caixas estão utilizadas.

O laço das linhas 1-4 é executado m vezes. O laço mais interno, das linhas 10-20, é dominado pelo laço da linha 12. Os objetos excluídos na linha 15 são identificados por seus apontadores na lista com complexidade $O(1)$. O laço das linhas 5-21 também considera m caixas. Os dois blocos **enquanto** (linhas 10 e 12) são complementares, significando que a cada iteração de j os dois (ou apenas um) apontadores (b na linha 12 e a na linha 18) caminham em sentidos opostos no conjunto N, portanto o comando **para-todo** da linha 5, limitado pelo laço 10-20 que é executado no máximo n vezes, possui complexidade $O(m\,n)$. A ordenação das caixas na linha 21 tem complexidade igual a $O(m \log m)$. O último laço das linhas 26-32 também tem complexidade igual a $O(n\,m)$. O comando da linha 27 percorre o conjunto dos objetos restantes e é executado no máximo $(n - m)$ vezes. Nas linhas 28 e 31, no pior caso, m caixas são verificadas. A reordenação das caixas na linha 30 tem complexidade igual a $O(m)$. A complexidade do algoritmo `DB3FD` é $O(n\,m)$.

3.1.4 Dual Worst-Fit Decreasing (`DWFD`)

A heurística `WFD`, descrita na Seção 2.2.2, é a base para `DWFD`. Os objetos são considerados em ordem não-crescente de peso. Enquanto houver

caixas com capacidade residual suficiente, a estratégia de seleção da caixa é exatamente igual à utilizada pela heurística `WFD`: seleciona-se a caixa mais leve (não-utilizada ou incompleta) onde exista capacidade residual suficiente para o objeto corrente. Cada objeto restante é colocado na caixa mais leve das saturadas, violando sua restrição de capacidade.

Procedimento `DWFD`
Entrada: $N = \{1, \ldots, n\}$, C,
w_i $(i = 1, \ldots, n)$ onde $w_h \geq w_{h+1}, h = 1, \ldots, n-1$,
número m de caixas disponíveis.
Saída: $S = (B_1, \ldots, B_m)$ com $w_S(B_1) \geq \ldots \geq w_S(B_m)$.
1 **para-todo** $j = 1, \ldots, m$ **faça** $B_j \leftarrow \{j\}$;
2 **para-todo** $i = m+1, \ldots, n$ **faça**
3 $B_m \leftarrow B_m \cup \{i\}$;
4 Reordena $(B_1, \ldots, B_m)$ tal que $w_S(B_1) \geq \ldots \geq w_S(B_m)$;
5 **fim-para-todo**
6 **retorne**
fim `DWFD`

Figura 3.4: Pseudo-código da heurística construtiva `DWFD`

A Figura 3.4 ilustra o algoritmo `DWFD`. No primeiro passo coloca-se os m objetos mais pesados um em cada uma das m caixas disponíveis. Neste ponto as m caixas estão ordenadas em ordem não crescente de peso. O laço que compreende as linhas 2-5 considera cada um dos $m - n$ objetos restantes, empacotando-o na caixa mais leve na linha 3. A seguir, na linha 4, a ordenação das caixas é refeita.

Assim como em `DB3FD`, todas as m caixas da solução S possuem algum objeto (linha 1).

O laço das linhas 2-5 é executado n vezes. Incluir um objeto em uma caixa possui complexidade $O(1)$ e o procedimento para reordenar as caixas possui complexidade igual a $O(m)$. Logo, a complexidade do algoritmo `DWFD` é $O(m\,n)$.

3.2 Fase de Redistribuição

Quando uma solução viável para DBP, produzida na fase de `Construção` descrita na seção anterior, não for viável para BP, aplica-se estratégias de equilíbrio e desequilíbrio de peso das caixas para melhorar a utilização das mesmas. Cada estratégia é precedida pela aplicação de uma regra que verifica e trata objetos dominantes. Se esta fase não for suficiente

para encontrar uma solução viável para BP o algoritmo segue para a fase de `Melhoria` descrita na próxima seção.

3.2.1 Estratégia de Equilíbrio de Peso

Dado um par de caixas B_{j_1}, B_{j_2} de uma solução inviável S com $w_S(B_{j_1}) > C$ e $w_S(B_{j_2}) < C$, seja $\Delta = |w_S(B_{j_1}) - w_S(B_{j_2})|$ a diferença absoluta dos seus pesos. Freqüentemente, é possível obter uma nova solução $\bar{S}$ onde esta diferença é menor, redistribuindo-se os objetos entre as caixas. Considera-se o exemplo em que $C = 150, w = (80, 49, 46, 45, 39, 21, 20)$, $B_{j_1} = \{3, 2, 5, 6\}$, $w_S(B_{j_1}) = 155$, $B_{j_2} = \{1, 4, 7\}$, $w_S(B_{j_2}) = 145$ e $\Delta = 10$. Cria-se o conjunto de objetos $G = B_{j_1} \cup B_{j_2}$. Tem-se então o problema de fazer uma partição do conjunto G de objetos em dois subconjuntos, de tal forma que a diferença entre a soma dos pesos dos objetos de cada subconjunto seja a menor possível (*Number Partitioning Problem* - NPP). Este é um problema NP-difícil [32] para o qual Karmarkar e Karp [55] propuseram o método aproximado de diferenciação (*Differencing Method* - `DM`), descrito a seguir.

3.2.1.1 O Método de Diferenciação de Karmarkar e Karp

Deseja-se particionar o conjunto de objetos G em dois subconjuntos A, B de forma a minimizar a diferença da soma dos pesos dos objetos em cada subconjunto. Suponha-se q o número de objetos inicialmente em G. Seja *diferenciação* a operação que troca dois pesos w_a e w_b pela sua diferença. O método combina soluções parciais para o problema de particionamento baseadas na operação de diferenciação. A cada passo da heurística, os dois maiores pesos w_a, w_b são diferenciados, o que é equivalente a determinar que os objetos a e b ficarão em subconjuntos diferentes, deixando para decidir-se mais tarde em qual subconjunto ficará cada um deles. Recai-se então no problema de particionar $G' = G - \{a, b\} \cup \{q + 1\}$, onde $q + 1$ é o índice do objeto virtual com peso $w_{q+1} = |w_a - w_b|$. Por exemplo, considera-se o problema de criar uma partição do conjunto $G = \{1, 2, 3, 4, 5, 6, 7\}$ com $q = 7$ e pesos $w = (80, 49, 46, 45, 39, 21, 20)$ em dois subconjuntos disjuntos, A e B. Colocar o objeto 1 de peso $w_1 = 80$ no subconjunto A e o objeto 2 de peso $w_2 = 49$ no subconjunto B é equivalente a colocar um objeto virtual de índice igual a $q + 1 = 8$ cujo

peso é igual à diferença $w_{q+1} = w_1 - w_2 = 31$, no subconjunto A, uma vez que é possível subtrair 49 da soma de pesos dos dois subconjuntos sem alterar a diferença. Como foi decidido que os objetos 1 com $w_1 = 80$ e 2 com $w_2 = 49$ devem ficar em subconjuntos diferentes, então pode-se retirar os dois objetos e trocá-los pelo objeto virtual de peso igual à diferença, 31. A seguir deve-se criar uma nova partição do conjunto $G' = \{3, 4, 5, 6, 7, 8\}$ com pesos $w = (46, 45, 39, 21, 20, 31)$ dos objetos restantes. O algoritmo retira os dois objetos de maiores pesos, calcula a diferença dos seus pesos e a trata como se fosse o peso de um novo objeto virtual, colocando-o no conjunto de números restantes. Este processo continua até que sobre apenas um objeto, cujo peso é o valor da diferença entre a soma dos pesos dos dois subconjuntos da partição final. A complexidade da heurística `DM` é $O(n \log n)$.

Procedimento `DM`

k	G_k	w	(a,b)	(w_a, w_b)	$i = q+k$	$\Delta_k = w_{q+k}$
1	$\{1,2,3,4,5,6,7\}$	(80,49,46,45,39,21,20)	(1,2)	(80, 49)	8	31
2	$\{3,4,5,6,7,8\}$	(46,45,39,21,20,31)	(3,4)	(46, 45)	9	1
3	$\{5,6,7,8,9\}$	(39,21,20,31,1)	(5,8)	(39, 31)	10	8
4	$\{6,7,9,10\}$	(21,20,1,8)	(6,7)	(21, 20)	11	1
5	$\{9,10,11\}$	(1,8,1)	(10,9)	(8, 1)	12	7
6	$\{11,12\}$	(1,7)	(12,11)	(7, 1)	13	6

Construção da partição

k	(a,b)	(w_a, w_b)	$i = q+k$	Δ_k	A	B
					$\{13\}$	$\{\}$
6	(12, 11)	(7, 1)	13	6	$\{12\}$	$\{11\}$
5	(10, 9)	(8, 1)	12	7	$\{10\}$	$\{11, 9\}$
4	(6, 7)	(21, 20)	11	1	$\{10, 7\}$	$\{9, 6\}$
3	(5, 8)	(39, 31)	10	8	$\{7, 5\}$	$\{9, 6, 8\}$
2	(3, 4)	(46, 45)	9	1	$\{7, 5, 4\}$	$\{6, 8, 3\}$
1	(1, 2)	(80, 49)	8	31	$\{7, 5, 4, 2\}$	$\{6, 3, 1\}$

Figura 3.5: Ilustração do procedimento `DM` de Karmarkar e Karp

A Figura 3.5 ilustra o procedimento aplicado ao conjunto $G = \{1, \ldots, 7\}$ com $q = 7$ e $w = (80, 49, 46, 45, 39, 21, 20)$. Para qualquer iteração k de `DM`, seja G_k o conjunto dos objetos restantes, w_i os pesos dos objetos de G_k, (a, b) os dois objetos mais pesados com pesos, respectivamente, w_a e w_b, $i = q + k$ o índice do objeto virtual criado na iteração k e $\Delta_k = w_a - w_b$ seu peso. `DM` troca os objetos a e b com pesos w_a e w_b pelo objeto $q+k$ com peso $w_{q+k} = \Delta_k$ até que sobre apenas um objeto em $G_{k+1} = G_k \setminus \{a, b\} \cup \{q+k\}$. No exemplo abaixo, foram realizadas seis iterações resultando na diferença $\Delta_6 = 6$. A construção da partição é feita analisando-se a sequência de volta das operações de diferenciação, como ilustrado pela Figura 3.5. Inicia-se

colocando o objeto cujo peso é igual a diferença final ($\Delta_6 = 6$) em um subconjunto, por exemplo, A, deixando o outro subconjunto B vazio. Para $k = q - 1$ até $k = 1$, se o objeto $q + k$ com peso $w_{q+k} = \Delta_k$ pertence a A então faz-se $A \leftarrow A \setminus \{q+k\} \cup \{a\}$ e $B \leftarrow B \cup \{b\}$, do contrário $A \leftarrow A \cup \{b\}$ e $B \leftarrow B \setminus \{q + k\} \cup \{a\}$.

A soma dos pesos do subconjunto A é 153 e a do subconjunto B é 147. A diferença $\Delta = 6$ é menor do que a diferença de peso das duas caixas antes da aplicação do método DM, que era 10. A partição gerada pelo procedimento DM para a instância considerada não resultou em uma partição ótima. Gent e Walsh [34] provam que quando existem quatro ou menos números para particionar, o método de diferenciação produz uma partição ótima. Na próxima seção descreve-se uma estratégia de randomização do DM, que, em alguns casos, pode melhorar a qualidade da solução.

3.2.1.2 Versão Randomizada do Método de Diferenciação

Argüello et al. [4] apresentam versões randomizadas para o método de diferenciação de Karmarkar e Karp. Seja G o conjunto de objetos que se deseja particionar. A heurística *Shuffling* (SH) considera uma lista restrita de candidatos à operação de diferenciação (LCR) formada pelos β objetos mais pesados de G e forma um certo número de conjuntos com $\beta/2$ pares fixos de objetos em cada um. Se o número de combinações for muito grande, determina-se o número de conjuntos desejados e escolhe-se os pares aleatoriamente. Para cada um dos conjuntos de pares fixos, diferencia-se cada par, colocando-se o objeto virtual criado com peso igual à diferença no conjunto de números restantes. A seguir aplica-se DM (descrito na seção anterior) ao conjunto de objetos restantes que não fazem parte dos pares fixos. O conjunto de pares fixos que obteve a melhor diferença é selecionado para reduzir a cardinalidade de G. O processo é repetido enquanto a cardinalidade de G for maior do que β.

Ilustra-se a seguir a aplicação de SH ao mesmo conjunto de objetos $G = \{1, 2, 3, 4, 5, 6, 7\}$ usado no exemplo anterior, com $w = (80, 49, 46, 45, 39, 21, 20)$. Utiliza-se $\beta = 4$. Os quatro objetos mais pesados são os de índices $1, 2, 3$ e 4, que podem ser combinados em pares de três maneiras diferentes. Aplicando-se a operação de diferenciação aos pares $\{1, 2\}$ e $\{3, 4\}$, são criados dois objetos virtuais de pesos $w_8 = |w_1 - w_2| = 31$ e $w_9 = |w_3 - w_4| = 1$. Aplicando-se DM ao novo conjunto de objetos restantes $G \setminus \{1, 2, 3, 4\} \cup \{8, 9\} = \{5, 6, 7, 8, 9\}$ com pesos $w = (39, 21, 20, 31, 1)$

obtém-se $\Delta = 6$. Considera-se agora a segunda combinação possível, formada pelos pares $\{1, 3\}$ e $\{2, 4\}$. Neste caso, são criados dois objetos virtuais de pesos $w_8 = |w_1 - w_3| = 34$ e $w_9 = |w_2 - w_4| = 4$. Aplicando-se `DM` ao conjunto $G = \{5, 6, 7, 8, 9\}$ de pesos $w = (39, 21, 20, 34, 4)$ obtém-se $\Delta = 0$. Neste exemplo, neste ponto do procedimento obteve-se uma uma solução ótima, correspondendo a uma partição em dois conjuntos com o mesmo peso. Quando este não é o caso, seleciona-se o conjunto de objetos restantes resultante da combinação que obteve o menor valor de Δ e continua-se o procedimento recursivamente enquanto a cardinalidade de G for maior do que β. A partição final é construída da mesma forma como em `DM`.

No exemplo acima utilizou-se $\beta = 4$. Como só existem três maneiras diferentes de se combinar quatro números em pares, é viável enumerar todas elas, não tendo havido randomização. Considerando-se $\beta = 4$ e o conjunto G, inicialmente com n elementos, o procedimento acima executa três vezes o procedimento `DM`, e portanto esta etapa tem complexidade $O(n \log n)$. A seguir, o número de elementos do conjunto é reduzido em duas unidades e o procedimento é repetido enquanto houver pelo menos β elementos no conjunto. A complexidade final é $O(n^2 \log n)$.

3.2.1.3 Descrição da Estratégia de Equilíbrio de Peso

O procedimento `Equilíbrio` atua sobre uma solução corrente S inviável para BP. Aplica os procedimentos `DM` e `SH` com $\beta = 4$, descritos anteriormente, a todos os pares de caixas possíveis formados por uma caixa violada e outra incompleta. O primeiro par investigado é aquele formado pela caixa violada mais pesada e pela caixa incompleta mais leve. As caixas violadas são investigadas em ordem não-crescente de peso, enquanto que as caixas incompletas na ordem oposta. A função `DM+SH`(j_1, j_2, S) é aplicada ao par de caixas B_{j_1} e B_{j_2} da solução S, produzindo e retornando a melhor solução $\bar{S}$ (isto é, com a menor diferença de pesos entre os dois subconjuntos) encontrada pelos procedimentos `DM` e `SH`. Quando a diferença dos pesos das caixas $\bar{B}_{j_1}$ e $\bar{B}_{j_2}$ da nova solução for menor do que a diferença dos pesos das caixas da solução original ($|w_{\bar{S}}(\bar{B}_{j_1}) - w_{\bar{S}}(\bar{B}_{j_2})| < |w_S(B_{j_1}) - w_S(B_{j_2})|$) a função `DM+SH` retorna `.VERDADEIRO.`, do contrário retorna `.FALSO.`. A busca termina quando não existir mais possibilidade de melhoria. Um mecanismo de controle é utilizado para evitar comparações desnecessárias. Esta estratégia é capaz de viabilizar uma solução.

A seguir detalha-se o procedimento ilustrado pela Figura 3.6. A

variável de retorno `alterou` é um indicador de que a solução S foi alterada e assume os valores `.VERDADEIRO.` ou `.FALSO.`. A variável `aprimorou`, inicializada nas linhas 1 e 6, é usada para indicar que a composição de um par de caixas da solução corrente foi alterada. O contador `iteração` é usado para atualizar o controle que verifica se ainda existe alteração possível. O laço das linhas 7-23 compara todas as caixas violadas com todas as caixas incompletas. Se a execução das rotinas `DM` e `SH` na linha 14 resultarem em melhoria, as atualizações são feitas nas linhas 16-19 e o laço das linhas 7-23 é imediatamente reinicializado. Na primeira iteração o laço completo das linhas 5-24 é executado. A partir de então, enquanto houver pelo menos uma alteração na composição das caixas (indicado pelo valor da variável `aprimorou`) as rotinas `DM` e `SH` serão aplicadas somente aos pares de caixas que tiveram sua composição alterada depois da última vez que eles foram analisados. Para implementar este controle, seja $\texttt{Tempo}(j)$ a iteração mais recente em que a composição da caixa B_j foi alterada. Seja ainda $\texttt{ListaTabu}(j_1, j_2)$ a iteração mais recente em que as rotinas `DM` e `SH` foram aplicadas às caixas B_{j_1} e B_{j_2}, não importando se resultou em alteração da composição das caixas ou não. As rotinas `DM` e `SH` somente serão aplicadas ao par de caixas (B_{j_1}, B_{j_2}) quando $max(\texttt{Tempo}(j_1), \texttt{Tempo}(j_2)) > \texttt{ListaTabu}(j_1, j_2)$. Do contrário, o resultado não poderia alterar a composição das caixas. Para inicializar este controle $\texttt{ListaTabu}(j_1, j_2) = -1$ para todo $j_1, j_2 \in \{1, \ldots, m\}$ e $\texttt{Tempo}(j) = 0$ para todo $j \in \{1, \ldots, m\}$ (linhas 3 e 4). Quando o par de caixas (B_{j_1}, B_{j_2}) for examinado pelas rotinas `DM` e `SH`, $\texttt{ListaTabu}(j_1, j_2) =$ `iteração` é atualizado na linha 13 e, se a composição das caixas for alterada em resultado da aplicação das rotinas `DM` e `SH` também $\texttt{Tempo}(j_1) =$ `iteração` e $\texttt{Tempo}(j_2) =$ `iteração` são atualizados na linha 18. O procedimento pára quando encontra uma solução viável ou quando tiver executado uma iteração do laço 7-23 sem que a variável `aprimorou` fique com o valor igual a `.VERDADEIRO.`. O procedimento retorna a solução S, eventualmente modificada, o que é indicado pelo valor da variável `alterou`.

A solução resultante da aplicação do procedimento `Equilíbrio` à solução inviável S com m caixas continua utilizando todas elas. Uma caixa violada possui necessariamente mais de um objeto. Logo, este procedimento gera uma solução com pelo menos um objeto em cada uma das duas caixas examinadas.

A seguir analisa-se a complexidade deste procedimento. As inicializações das linhas 1 e 2 têm complexidade $O(1)$. A inicialização da linha 3 tem complexidade $O(m)$ e a da linha 4 $O(m^2)$. O laço mais interno (linhas 8-22) é limitado pela execução das rotinas `DM` ($O(n \log n)$) e de sua versão

randomizada SH ($O(n^2 \log n)$). O laço das linhas 7-23 compara todas as caixas violadas com todas as caixas incompletas ou não-utilizadas. Na pior hipótese, estas totalizam m e o número de comparações é $O(m^2)$. O procedimento que refaz a ordenação das caixas na linha 19 tem complexidade $O(m)$. Considerando-se o pior caso em que existem n objetos em uma caixa, cada iteração do laço das linhas 5-24 tem complexidade $O(m^2\, n^2 \log n)$.

```
Procedimento Equilíbrio
Entrada:    N = {1,...,n}, C,
            w_i (i = 1,...,n) onde w_h ≥ w_{h+1}, h = 1,...,n−1,
            S = (B_1,...,B_m) com w_S(B_1) ≥ w_S(B_2) ≥ ... ≥ w_S(B_m).
Saída:      S = (B_1,...,B_m), alterou.
1  alterou ← .FALSO.; aprimorou ← .VERDADEIRO.;
2  iteração ← 0;
3  Tempo(j) ← 0, ∀ j ∈ {1,...,m};
4  ListaTabu(j_1, j_2) ← −1, ∀j_1, j_2 ∈ {1,...,m};
5  enquanto w_S(B_1) > C e aprimorou faça
6      aprimorou ← .FALSO.;
7      para-todo j_1 = 1,...,m − 1 : w_S(B_{j_1}) > C enquanto não aprimorou faça
8          para-todo j_2 = m,...,1 : w_S(B_{j_2}) < C enquanto não aprimorou faça
9              se Tempo(j_1) > Tempo(j_2) então max ← Tempo(j_1);
10             senão max ← Tempo(j_2);
11             se max > ListaTabu(j_1, j_2) então faça
12                 iteração ← iteração + 1;
13                 ListaTabu(j_1, j_2), ListaTabu(j_2, j_1) ← iteração;
14                 S̄, aprimorou ← DM+SH(j_1, j_2, S);
15                 se aprimorou então faça
16                     B_{j_1} ← B̄_{j_1}; B_{j_2} ← B̄_{j_2};
17                     alterou ← .VERDADEIRO.;
18                     Tempo(j_1), Tempo(j_2) ← iteração;
19                     Reordena B_1,...,B_m tal que w_S(B_1) ≥ ... ≥ w_S(B_m);
20                 fim-se
21             fim-se
22         fim-para-todo
23     fim-para-todo
24 fim-enquanto
25 retorne
fim Equilíbrio
```

Figura 3.6: Pseudo-código da estratégia de equilíbrio de peso

3.2.2 Estratégia de Desequilíbrio de Peso

Dado um par de caixas B_{j_1} e B_{j_2} da solução inviável S, seja $\bar{N} = B_{j_1} \cup B_{j_2}$. O problema de soma de conjuntos (*Subset Sum Problem* - SSP) consiste em encontrar um subconjunto $\hat{N}$ de objetos de $\bar{N}$ cuja soma dos seus pesos seja o mais próxima possível de C. Para cada par de caixas

incompletas da solução S, a estratégia de desequilíbrio faz uma tentativa de redistribuir os objetos destas duas caixas de tal forma que o peso da caixa mais pesada da nova solução seja maior do que o peso da caixa mais pesada na solução original. Desta forma, aumenta-se a capacidade residual da caixa mais leve do par. Por investigar somente caixas incompletas, esta estratégia não tem como tornar viável uma solução inviável. Entretanto, aumentando a capacidade residual de uma das caixas incompletas, aumenta a chance da estratégia `Equilíbrio` viabilizá-la posteriormente. Para resolver SSP considerou-se o algoritmo aproximado `MTSS` (com $k = 3$) de Martello e Toth [65], de complexidade $O(n^3)$.

Procedimento `Desequilíbrio`
Entrada: $N = \{1, \ldots, n\}$, C,
w_i $(i = 1, \ldots, n)$ onde $w_h \geq w_{h+1}, h = 1, \ldots, n-1$,
$S = (B_1, \ldots, B_m)$ com $w_S(B_1) \geq w_S(B_2) \geq \ldots \geq w_S(B_m)$.
Saída: $S = (B_1, \ldots, B_m)$, `alterou`.

```
1  alterou ← .FALSO.; aprimorou ← .VERDADEIRO.;
2  iteração ← 0;
3  Tempo(j) ← 0, ∀ j ∈ {1, ..., m};
4  ListaTabu(j1, j2) ← −1, ∀j1, j2 ∈ {1, ..., m};
5  enquanto aprimorou faça
6      aprimorou ← .FALSO.;
7      para-todo j1 = 1, ..., m − 1 : 0 < wS(Bj1) < C faça
8          para-todo j2 = j1 + 1, ..., m : 0 < wS(Bj2) < C faça
9              se Tempo(j1) > Tempo(j2) então max ← Tempo(j1);
10             senão max ← Tempo(j2);
11             se max > ListaTabu(j1, j2) então faça
12                 iteração ← iteração + 1;
13                 ListaTabu(j1, j2), ListaTabu(j2, j1) ← iteração;
14                 N̄ ← Bj1 ∪ Bj2 ;
15                 N̂ ← MTSS(N̄, C, w);
16                 se Σ_{i∈N̂} wi > wS(Bj1) e Σ_{i∈N̂} wi > wS(Bj2) então faça
17                     Bj1 ← N̂; Bj2 ← N̄ \ N̂;
18                     alterou, aprimorou ← .VERDADEIRO.;
19                     Tempo(j1), Tempo(j2) ← iteração;
20                 fim-se
21             fim-se
22         fim-para-todo
23     fim-para-todo
24 fim-enquanto
25 retorne
fim Desequilíbrio
```

Figura 3.7: Pseudo-código da estratégia de desequilíbrio de peso

A Figura 3.7 descreve o procedimento `Desequilíbrio`. As inicializações são realizadas nas linhas 1-4. A variável de retorno `alterou` é um indicador de que a solução S foi alterada podendo assumir os valores `.VERDADEIRO.` ou `.FALSO.`. A variável `aprimorou`, incializada nas linhas 1

e 6, é usada para indicar que a composição de um par de caixas da solução corrente foi alterada. O contador `iteração` é usado para atualizar o controle que verifica se ainda existe alteração possível. O laço das linhas 7-23 compara cada par de caixas incompletas. Na linha 14 cria-se o conjunto $\bar{N}$ formado pelos objetos das duas caixas sendo investigadas. Se a execução da rotina `MTSS` na linha 15 resulta em melhoria (a soma dos pesos dos objetos do subconjunto resultante $\hat{N}$ é maior do que o peso da caixa mais pesada da solução original), a solução S é atualizada nas linhas 17-19. Uma das caixas, por exemplo B_{j_1}, recebe o conjunto de objetos resultante da aplicação de `MTSS` ao conjunto $\bar{N}$ e a outra caixa, B_{j_2}, recebe os objetos restantes. Para evitar comparações desnecessárias utilizou-se o mesmo controle descrito no procedimento `Equilíbrio`. Este procedimento pode retirar todos os objetos de uma caixa que se torna não-utilizada, por isso o teste $w_S(B_i) > 0$ nas linhas 7 e 8.

A seguir analisa-se a complexidade do procedimento `Desequilíbrio`. As inicializações das linhas 1 e 2 possuem complexidade $O(1)$. A inicialização da linha 3 tem complexidade $O(m)$ e a inicialização na linha 4 $O(m^2)$. O laço mais interno (linhas 8-21) é limitado pela execução da rotina `MTSS` de complexidade $O(n^3)$, onde n é o número de objetos da caixa considerada. O laço das linhas 7-23 compara todas as caixas incompletas duas a duas e é feito portanto $O(m^2)$ vezes. Considerando-se o pior caso em que existem n objetos em cada caixa, uma iteração do laço das linhas 5-24 tem complexidade $O(m^2\, n^3)$.

3.2.3 Regra de Dominância de Objetos

A regra de dominância de objetos será aplicada antes das estratégias `Equilíbrio` e `Desequilíbrio`. Diz-se que os objetos i_1, i_2 da caixa B_{j_1} são dominados por outro objeto i_3 de outra caixa B_{j_2} se $w_{i_3} = w_{i_1} + w_{i_2}$. Esta regra é aplicada da seguinte forma. Para cada par formado por uma caixa completa B_{j_1} e por outra violada ou incompleta B_{j_2} da solução corrente, sempre que existirem dois objetos $i_1, i_2 \in B_{j_1}$ e um objeto $i_3 \in B_{j_2}$ tal que $w_{i_3} = w_{i_1} + w_{i_2}$ os objetos i_1, i_2 são trocados com i_3 na solução corrente. Embora os pesos das duas caixas envolvidas na troca permaneçam iguais, aumentar o número de objetos leves nas caixas incompletas e violadas aumenta as possibilidades de trocas realizadas pelas duas estratégias descritas anteriormente e consequentemente as possibilidades de melhoria nesta fase. Este procedimento produz soluções com todas as caixas utilizadas.

```
Procedimento Dominância
Entrada:    N = {1,...,n}, C,
            w_i (i = 1,...,n) onde w_h ≥ w_{h+1}, h = 1,...,n−1,
            S = (B_1,...,B_m) com w_S(B_1) ≥ w_S(B_2) ≥ ... ≥ w_S(B_m).
Saída:      S = (B_1,...,B_m), alterou.
1   alterou ← .FALSO.;
2   para-todo j_1 = 1,...,m : w_S(B_{j_1}) = C faça
3       aprimorou ← .FALSO.;
4       para-todo i_1, i_2 ∈ B_{j_1} enquanto não aprimorou faça
5           aprimorou ← ∃ i_3 ∈ B_{j_2} : j_1 ≠ j_2, w_S(B_{j_2}) ≠ C, w_{i_3} = w_{i_1} + w_{i_2};
6           se aprimorou então faça
7               B_{j_1} ← B_{j_1} \ {i_1, i_2} ∪ {i_3}; B_{j_2} ← B_{j_2} \ {i_3} ∪ {i_1, i_2};
8               alterou, aprimorou ← .VERDADEIRO.;
9           fim-se
10      fim-para-todo
11  fim-para-todo
12  retorne
fim Dominância
```

Figura 3.8: Pseudo-código da regra de dominância de objeto

A Figura 3.8 ilustra o procedimento `Dominância`. Este procedimento compara, na linha 4, todos os pares de objetos das caixas completas (linha 2) com cada objeto das caixas incompletas ou violadas (linha 5). Para cada caixa B_{j_1}, se existe um objeto $i_3 \in B_{j_2}$ que domina dois objetos $i_1, i_2 \in B_{j_1}$, a troca é realizada na linha 7, a busca com a caixa B_{j_1} é interrompida e uma próxima caixa completa é inspecionada na linha 2.

No pior caso, tem-se n objetos das caixas completas combinados dois a dois na linha 4, comparados com n objetos das caixas violadas ou incompletas na linha 5. Logo, uma aplicação da regra de dominância possui complexidade $O(n^3)$.

3.2.4 Descrição do Algoritmo de Redistribuição

A Figura 3.9 ilustra o procedimento `Redistribuição` que combina as três rotinas descritas anteriormente. A estratégia usada aplica alternadamente as rotinas `Equilíbrio` e `Desequilíbrio` (precedidas da rotina `Dominância`) enquanto a solução for inviável e existir possibilidade de redistribuição. Na primeira linha as variáveis usadas no controle da aplicação dos procedimentos de redistribuição são atualizadas. O contador do número de iterações é inicializado na linha 2 e atualizado na linha 4. O teste da linha 5 é usado para verificar se a solução gerada pelo procedimento `Desequilíbrio` na linha 9 resultou em alteração da solução, já que em caso negativo a ro-

tina `Dominância` não faria efeito algum sobre a solução corrente. Da mesma forma, o teste da linha 8 é necessário para verificar se a solução eventualmente gerada pelo procedimento `Dominância` na linha 5 sofreu alteração com a aplicação da rotina `Equilíbrio`, na linha 6. Caso contrário, a rotina `Dominância`, aplicada na linha 8, não teria efeito algum sobre a solução corrente S. A única rotina que pode viabilizar a solução é o procedimento `Equilíbrio` executado na linha 6. Só faz sentido executar novamente o laço se uma das soluções geradas posteriormente à aplicação dessa rotina tiverem alterado a solução corrente, o que é indicado pelas variáveis `alterou_DO_2` e `alterou_D`. O procedimento pára quando não houver mais possibilidades de redistribuição ou quando o número de iterações for igual ao número máximo de iterações `MaxRedistribuição`.

Nota-se que o único procedimento que gera solução com alguma caixa não-utilizada é o procedimento `Desequilíbrio`. Este, porém, não tem como viabilizar a solução corrente já que realiza movimentos somente entre caixas incompletas. Desta forma, se a solução S com m caixas, gerada pelo procedimento `Desequilíbrio`, possui uma única caixa não-utilizada, esta caixa receberá forçosamente pelo menos um objeto quando o procedimento `Equilíbrio` for aplicado a S. Se, neste passo, a solução corrente tornou-se viável, o procedimento pára com m caixas, e todas utilizadas. A mesma afirmação não pode ser feita se `Desequilíbrio` gerou mais de uma caixa não-utilizada. Na linha 11 a variável m' recebe o número exato de caixas utilizadas da solução S.

Esta fase pode tornar viável uma solução inviável para BP. Quando isso não é alcançado, inicia-se a fase de `Melhoria` descrita a seguir.

3.3 Fase de Melhoria

Na fase anterior utilizaram-se técnicas para redistribuir o peso de pares de caixas. Quando uma solução S chega à fase de `Melhoria` tem-se, necessariamente, que pelo menos uma caixa viola a restrição de capacidade máxima da caixa.

Uma estratégia de busca tabu é aplicada para reduzir a violação de capacidade das caixas na solução corrente. Para qualquer solução S, seja E_S a soma dos excessos de todas as caixas violadas. Logo, $E_S = 0$ quando S é viável para BP. Partindo-se de uma solução inviável S, investiga-se vizinhanças definidas por movimentos de troca que permutam pares de objetos, sendo, pelo menos um deles, de uma caixa violada. Para todos os

Procedimento Redistribuição
Entrada: $N = \{1, \ldots, n\}$, C,
w_i $(i = 1, \ldots, n)$ onde $w_h \geq w_{h+1}, h = 1, \ldots, n-1$,
$S = (B_1, \ldots, B_m)$ com $w_S(B_1) \geq w_S(B_2) \geq \ldots \geq w_S(B_m)$,
MaxRedistribuição.
Saída: $S = (B_1, \ldots, B_{m'})$.
1 alterou_DO_2, alterou_D ← .VERDADEIRO.;
2 iteração ← 0;
3 **enquanto** (alterou_DO_2 **ou** alterou_D) **e**
(iteração < MaxRedistribuição) **faça**
4 iteração ← iteração + 1;
5 **se** alterou_D **então** S, alterou_DO_1 ← Dominância(N, C, w, S);
6 S, alterou_E ← Equilíbrio(N, C, w, S);
7 **se** Viável$_{BP}(S)$ **então** vá para passo 11;
8 **se** alterou_E **então** S, alterou_DO_2 ← Dominância(N, C, w, S);
9 S, alterou_D ← Desequilíbrio(N, C, w, S);
10 **fim-enquanto**
11 m' ← número de caixas utilizadas pela solução S;
12 **retorne**
fim Redistribuição

Figura 3.9: Pseudo-código da fase de redistribuição

objetos $i = 1, \ldots, n$, seja $S(i_1)$ a caixa onde o objeto i_1 está na solução S. Cada movimento $i_1 \leftrightarrow i_2$ é definido por um par ordenado (i_1, i_2) de objetos pertencentes a diferentes caixas. O primeiro elemento do par é sempre um objeto da caixa violada alvo, cujo excesso deseja-se reduzir. A solução $\bar{S}$, resultante da aplicação deste movimento à solução S, é caracterizada por $\bar{S}(i_1) = S(i_2), \bar{S}(i_2) = S(i_1), \bar{S}(l) = S(l)\, \forall\, l \neq i_1, i_2$.

Uma vez que o principal objetivo da busca é tornar viável uma solução inviável, é preciso tornar todas as caixas violadas viáveis. Desta forma, restringe-se o conjunto de movimentos candidatos apenas àqueles que possam diminuir o excesso da caixa violada alvo. Logo, cada iteração deverá considerar somente movimentos de troca em que $w_{i_1} > w_{i_2}$. O valor $\Delta(i_1, i_2) = \max\{w_{\bar{S}}(\bar{S}(i_1)) - C, 0\} + \max\{w_{\bar{S}}(\bar{S}(i_2)) - C, 0\}$ denota o excesso associado exclusivamente ao par de caixas envolvido após o movimento de troca.

Considerando-se o objetivo de reduzir a violação (isto é, diminuir o valor de E_S), a quantidade de movimentos de troca de objetos possíveis em uma iteração é muito restrita. Além disso, muitos movimentos possíveis são equivalentes em respeito ao valor da função objetivo, de forma que não fica claro qual movimento deve ser selecionado. Desta forma, criou-se uma estratégia que usa a combinação da avaliação de três regras para se determinar um critério de prioridade para os possíveis movimentos e, ao mesmo tempo, permite-se variar a prioridade de determinados movimentos.

A estratégia que combina múltiplas regras e prioridades variadas é descrita a seguir.

A Tabela 3.1 mostra todas as possíveis situações para um movimento $i_1 \leftrightarrow i_2$ envolvendo as caixas $S(i_1)$ e $S(i_2)$: o tipo de movimento, a situação de cada caixa depois do movimento, o valor do excesso $\Delta(i_1, i_2)$ após o movimento, o número de caixas do par violadas após o movimento, o número de caixas do par completas depois do movimento, e um valor "nível".

A princípio, movimentos que levam a pares com menos caixas violadas são preferíveis. Embora menos importante, movimentos que levam a pares com mais caixas completas também são desejáveis. A estratégia proposta categoriza cada tipo de movimento por um "nível" ou "prioridade". Para uma dada caixa violada alvo, cuja vizinhança está sendo investigada, escolhe-se o movimento com o menor nível (maior prioridade) e, em caso de empate, o de menor excesso $\Delta(i_1, i_2)$. Visto que não é possível estabelecer uma ordem total entre todos os tipos de movimentos (já que um tipo de movimento pode ser melhor do que outro em relação a um critério, mas não em relação a outro), dois diferentes valores de nível são atribuídos para alguns tipos de movimentos e, em cada iteração, um deles é selecionado usando probabilidade 1/2. Esta regra permite escolhas diversas em diferentes iterações, suspendendo preferências que poderiam impedir certos caminhos da busca. A estratégia utilizada é particularmente vantajosa no contexto de um método de solução que aceita movimentos que levam a soluções inviáveis, que podem ser eventualmente tornadas viáveis em um passo mais adiante.

A estratégia proposta pode ser vista como um refinamento ou variante do princípio de "voto persistente" [37][pág. 134], que também explora a abordagem de "oscilação estratégica". A integração do critério de múltiplas regras de decisão, sugerida pelo voto, fornece uma base útil na criação de melhores escolhas. No voto persistente as avaliações são realizadas em uma única iteração, porém executadas por múltiplas regras de decisão. Como sugerido pelos autores esta técnica fornece os fundamentos para refinamentos futuros. Em comum, existe o fato de que as avaliações são feitas em cada iteração e não sobre uma série de iterações. Além disso, os movimentos são avaliados segundo cada uma das regras, e, pode-se dizer, também recebem um "voto" (ou um valor). Diferentemente porém, as regras podem atribuir o mesmo valor para mais de um movimento. Nas duas versões, analisando-se a relação dos movimentos com os valores, em alguns casos não fica claro qual movimento é preferível. Na estratégia proposta resolve-se esta questão determinando-se um ou mais níveis de prioridade

tipo	situação das caixas depois do movimento $i_1 \leftrightarrow i_2$	$\Delta(i_1, i_2)$	violadas	completas	nível
1	*completa, completa*	0	0	2	1
2	*completa, incompleta*	0	0	1	2
3	*incompleta, incompleta*	0	0	0	3
4	*completa, violada*	> 0	1	1	2 e 4
5	*incompleta, violada*	> 0	1	0	4 e 5
6	*violada, violada*	> 0	2	0	5 e 6

Tabela 3.1: Tipos de movimentos

para cada movimento e escolhendo-se um dos níveis de forma aleatória. Segundo Glover e Laguna [37][pág. 351] muito já se escreveu sobre as vantagens da "oscilação estratégica". Entretanto uma das formas de se usar esta abordagem, aquela que involve a oscilação entre diferentes regras de escolha e vizinhanças, frequentemente é negligenciada. Um dos importantes aspectos da oscilação estratégica é que esta permite que os diferentes tipos de movimentos e regras de seleção, apropriados para visitar diferentes regiões e diferentes direções da busca, apareçam naturalmente.

Sempre que um movimento $i_1 \leftrightarrow i_2$ é executado, proíbe-se por um período de `TabuTenure` iterações todos os movimentos que implicam na volta do objeto i_2 para a caixa $S(i_2)$ e do objeto i_1 para a caixa $S(i_1)$. Na implementação proposta, este valor é escolhido de forma aleatória de uma distribuição uniforme no intervalo discreto $[0.8\sqrt{n}, 1.2\sqrt{n}]$.

Um possível elemento de reestruturação lógica da solução é o uso de análise antecipatória. Neste contexto, a reestruturação lógica busca responder as seguintes questões: "Quais condições asseguram a existência de uma trajetória que leva a uma solução melhor?" e "Qual movimento intermediário permite criar tais condições?". Movimentos intermediários podem ser gerados modificando-se a avaliação utilizada para selecionar a transição entre soluções ou modificando-se a estrutura da vizinhança que determina estas transições [37]. Movimentos do tipo 1 são os mais atraentes, já que resultam em duas caixas completas. Embora movimentos dos tipos 4, 5, e 6 sejam, em princípio, inferiores, em algumas situações eles podem ser eficazes. Este seria o caso, por exemplo, se existisse um movimento do tipo 1 associado a uma das caixas violadas resultantes destes movimentos. Assim, executando-se os dois movimentos subseqüentemente, como movimentos combinados, resultaria em uma nova solução com mais caixas completas. Para implementar esta idéia, cria-se uma solução temporária $\overline{S}$ a partir de cada movimento dos tipos 4, 5, e 6. Seja $B_{\overline{j}}$ a caixa violada resultante (no

caso dos movimentos dos tipos 4 ou 5), senão seja $B_{\bar{j}}$ a caixa alvo corrente (no caso do movimento do tipo 6 que resulta em duas caixas violadas). Se existe um movimento do tipo 1 envolvendo um objeto da caixa $B_{\bar{j}}$ da solução temporária $\overline{S}$, então os dois movimentos combinados são executados.

A Figura 3.10 sumariza o procedimento `Seleção_Movimento`, que calcula e retorna o melhor movimento a ser aplicado a uma dada caixa violada alvo B_j da solução corrente S. Denota-se por $\bar{S} = S(i_1, i_2)$ a solução resultante permutando-se os objetos i_1 e i_2 da solução S. A função `Tabu`(i_1, i_2) retorna `.VERDADEIRO.` se o movimento de troca $i_1 \leftrightarrow i_2$ for proibido, `.FALSO.` caso contrário. `TipoMovimento`$(i_1 \leftrightarrow i_2)$ e `NivelMovimento`$(i_1 \leftrightarrow i_2)$ fornecem o tipo e o valor do nível do movimento $(i_1 \leftrightarrow i_2)$, conforme a Tabela 3.1. Variáveis associadas ao melhor movimento são inicializadas nas linhas 1-2. A variável `movimentos` indica a quantidade de movimentos selecionada e pode assumir os seguintes valores: 0 indica que não existe movimento não tabu capaz de reduzir o peso da caixa alvo; 1 se um único movimento representado pela variável `MelhorMovimento` foi selecionado, e 2 indica que a análise antecipatória realisada resultou na seleção de dois movimentos combinados, determinados pelas variáveis `MelhorMovimento` e `MelhorMovimento2`. Os laços mais internos nas linhas 3-28 e 4-27 enumeram todos os movimentos candidatos, definidos por pares formados por cada objeto da caixa B_j e cada objeto mais leve de outra caixa. Na linha 5 verifica-se se o movimento $i_1 \leftrightarrow i_2$ viabiliza a solução ou se ele é do tipo 1, caso em que é imediatamente selecionado na linha 6 e o procedimento é finalizado. Caso contrário, determina-se na linha 9 se o tipo de movimento é de um dos tipos 4, 5, ou 6, e se a violação total de capacidade $\Delta(i_1, i_2)$ das caixas $S(i_1)$ e $S(i_2)$ na nova solução é menor do que certo valor limite `MaxDelta`. Neste caso, verifica-se nas linhas 10-21 se existe um movimento do tipo 1 para ser combinado com $i_1 \leftrightarrow i_2$. Cria-se, na linha 10, uma solução temporária $\bar{S}$ derivada da aplicação do movimento $i_1 \leftrightarrow i_2$ à solução corrente S. A caixa alvo $B_{\bar{j}}$ do próximo movimento é determinada na linha 11. Os laços internos das linhas 12-21 e 13-20 cumprem o mesmo papel que aqueles das linhas 3-28 e 4-27, gerando todos os movimentos envolvendo um objeto $\bar{i_1}$ da caixa $B_{\bar{j_1}}$ e todos os outros objetos $\bar{i_2}$ das outras caixas. Na linha 14 determina-se se o movimento $\bar{i_1} \leftrightarrow \bar{i_2}$ aplicado à solução temporária $\bar{S}$ é do tipo 1 ou torna a solução viável, caso em que é imediatamente selecionado implicando nas atualizações das variáveis que indicam a existência e a identificação dos dois movimentos nas linhas 15-17 e a finalização do procedimento a seguir. A instrução da linha 23 trata os demais casos de movimentos não tabu candidatos. Se o movimento $i_1 \leftrightarrow i_2$ melhora a melhor solução encontrada

até então, todas as informações a respeito da melhor solução corrente são atualizadas nas linhas 24-25. Para os casos em que nem uma solução viável nem um movimento do tipo 1 tenham sido identificados, o procedimento retorna na linha 29 o melhor movimento ou uma indicação de que não existe movimento não tabu para reduzir a violação da caixa alvo.

Considera-se a complexidade da rotina `Seleção_Movimento`. Este procedimento possui quatro laços (3-28; 4-27; 12-21; 13-20) aninhados. Cada um deles investiga no máximo $O(n)$ objetos. Logo, sua complexidade é $O(n^4)$.

O procedimento completo de melhoria, usado para reduzir a inviabilidade a partir da solução corrente S, é sumarizado na Figura 3.11. As inicializações são executadas nas linhas 1-5. Na primeira linha atualiza-se a `ListaTabu`. Na linha 2 inicializa-se a variável E_S, que armazena o excesso acumulado de todas as caixas da solução corrente. Na próxima linha, inicializa-se a variável `menorE`, usada para armazenar o menor valor de E_S entre todas as iterações tabu. A variável `iterações_sem_sucesso`, usada para controlar o número de iterações sem redução no valor da variável `menorE`, é inicializada na linha 4. A caixa alvo é inicializada na linha 5. O laço compreendido entre as linhas 6 a 31 é executado até que uma solução viável seja encontrada ou algum critério de parada for atingido. A cada iteração, o algoritmo determina na linha 7 o melhor movimento (único ou combinado) associado com a caixa alvo B_j, caso exista. Este movimento é aplicado à solução corrente nas linhas 8-20. Se todos os movimentos associados com a caixa alvo forem proibidos (caso em que a variável `movimentos` recebe o valor igual a zero na linha 7) ou se a caixa alvo foi viabilizada após a execução do movimento selecionado, então o índice da caixa alvo é incrementado de uma unidade na linha 21. Na linha 22 detecta-se se todas as caixas violadas foram investigadas e se a solução corrente continua inviável. Neste caso, as caixas são reordenadas na linha 23 e a caixa alvo volta a ser a caixa mais pesada (aquela de índice igual a 1) na linha 24. O contador do número de iterações sem sucesso e o menor excesso são atualizados nas linhas 26-30 e uma nova iteração é iniciada. A fase de redução de inviabilidade termina quando se encontra uma solução viável para BP ou depois de um total de `MaxTabu` iterações da busca tabu terem sido realizadas sem melhoria, isto é, sem redução do valor da variável `menorE`.

A seguir avalia-se a complexidade da rotina `Melhoria`. A `ListaTabu` foi implementada utilizando-se uma matriz de $m \times m$ posições e sua inicialização na linha 1 implica em um tempo $O(m^2)$. A complexidade de cada iteração do laço 6-31 é dominada pela aplicação do procedimento

```
Procedimento Seleção_Movimento
Entrada:    N = {1,...,n}, C, j
            w_i (i = 1,...,n) onde w_h ≥ w_{h+1}, h = 1,...,n−1,
            S = (B_1,...,B_m) com w_S(B_1) ≥ w_S(B_2) ≥ ... ≥ w_S(B_m),
            MaxDelta, Semente.
Saída:      movimentos, MelhorMovimento, MelhorMovimento2.
1   MelhorNivel, MelhorValor ← ∞;
2   movimentos ← 0; MelhorMovimento, MelhorMovimento2 ← ∅;
3   para-todo i_1 ∈ B_j faça
4       para-todo i_2 ∈ {1,...,n} \ B_j : w_{i_2} < w_{i_1} faça
5           se Viável_BP(S(i_1, i_2)) ou TipoMovimento(i_1 ↔ i_2) = 1 então faça
6               MelhorMovimento ← i_1 ↔ i_2; movimentos ← 1;
7               retorne
8           fim-se
9           se (TipoMovimento(i_1 ↔ i_2) = 4 ou TipoMovimento(i_1 ↔ i_2) = 5 ou
                TipoMovimento(i_1 ↔ i_2) = 6) e Δ(i_1, i_2) < MaxDelta então faça
10              S̄ ← S(i_1, i_2);
11              se w_S̄(S̄(i_1)) > C então j̄ ← S̄(i_1) else j̄ ← S̄(i_2);
12              para-todo ī_1 ∈ B_j̄ faça
13                  para-todo ī_2 ∈ {1,...,n} \ B_j̄ : w_{ī_2} < w_{ī_1} faça
14                      se Viável_BP(S̄(ī_1, ī_2)) ou
                            TipoMovimento(ī_1 ↔ ī_2) = 1 então faça
15                          MelhorMovimento ← i_1 ↔ i_2;
16                          MelhorMovimento2 ← ī_1 ↔ ī_2;
17                          movimentos ← 2;
18                          retorne
19                      fim-se
20                  fim-para-todo
21              fim-para-todo
22          fim-se
23          se .NAO.Tabu(i_1 ↔ i_2) e (NivelMovimento(i_1 ↔ i_2) < MelhorNivel ou
                (NivelMovimento(i_1 ↔ i_2) = MelhorNivel e
                Δ(i_1, i_2) < MelhorValor)) então faça;
24              MelhorMovimento ← i_1 ↔ i_2;
25              MelhorValor ← Δ(i_1, i_2); movimentos ← 1;
26          fim-se
27      fim-para-todo
28  fim-para-todo
29  retorne
fim Seleção_Movimento
```

Figura 3.10: Pseudo-código do procedimento de seleção de movimento

Procedimento `Melhoria`
Entrada: $N = \{1, \ldots, n\}$, C,
w_i $(i = 1, \ldots, n)$ onde $w_h \geq w_{h+1}, h = 1, \ldots, n-1$,
$S = (B_1, \ldots, B_m)$ com $w_S(B_1) \geq w_S(B_2) \geq \ldots \geq w_S(B_m)$,
`MaxTabu`, `MaxDelta`, `Semente`.
Saída: $S = (B_1, \ldots, B_m)$.

```
1  ListaTabu ← ∅;
2  E_S ← Σ_{j=1}^{m} max{w_S(B_j) − C, 0};
3  menorE ← ∞;
4  iterações_sem_sucesso ← 0;
5  j ← 1;
6  enquanto iterações_sem_sucesso < MaxTabu e w_S(B_j) > C faça
7      (movimentos, i_1 ↔ i_2, ī_1 ↔ ī_2) ←
           ← Seleção_Movimento(N, C, B_j, w, S, MaxDelta, Semente);
8      se movimentos ≥ 1 então faça
9          Calcule TabuTenure ∈ U[0.8√n, 1.2√n];
10         Proiba a reinserção do objeto i_1 na caixa S(i_1) e do objeto i_2
               na caixa S(i_2) nas próximas TabuTenure iterações;
11         Atualize a solução S: a ← S(i_1), b ← S(i_2), S(i_1) ← b, e S(i_2) ← a;
12         Atualize E_S;
13     senão
14         se movimentos = 2 então faça
15             Calcule TabuTenure ∈ U[0.8√n, 1.2√n];
16             Proiba a reinserção do objeto ī_1 na caixa S(ī_1) e do objeto ī_2
                   na caixa S(ī_2) nas próximas TabuTenure iterações;
17             Atualize a solução S: a ← S(ī_1), b ← S(ī_2), S(ī_1) ← b, e S(ī_2) ← a;
18             Atualize E_S;
19         fim-se
20     fim-se
21     se w_S(B_j) ≤ C ou movimentos = 0 então j ← j + 1;
22     se j > m ou (j ≤ m e w_S(B_j) ≤ C) então faça
23         Ordene as caixas de tal forma que w_S(B_1) ≥ w_S(B_2) ≥ ... ≥ w_S(B_m);
24         j ← 1;
25     fim-se
26     se E_S < menorE então faça
27         menorE ← E_S;
28         iterações_sem_sucesso ← 0;
29     senão iterações_sem_sucesso ← iterações_sem_sucesso + 1;
30     fim-se
31 fim-enquanto
32 retorne
fim Melhoria
```

Figura 3.11: Pseudo-código do procedimento de melhoria para reduzir inviabilidade

Procedimento C+R+M_BP
Entrada: $N = \{1, \ldots, n\}, C, m$
w_i $(i = 1, \ldots, n)$ onde $w_h \geq w_{h+1}, h = 1, \ldots, n-1$,
H, MaxTentativas, MaxRedistribuição, MaxTabu, MaxDelta, Semente.
Saída: $S = (B_1, \ldots, B'_m)$.
1 **para** $k = 1, \ldots,$ MaxTentativas **faça**
2 Selecione uma heurística construtiva do conjunto H;
3 Construa uma solução gulosa $S = (B_1, \ldots, B_m)$ para DBP;
4 **se** $\text{Viável}_{BP}(S)$ **então retorne**
5 $S \leftarrow$ Redistribuição$(N, C, w, S,$ MaxRedistribuição$)$;
6 **se** $\text{Viável}_{BP}(S)$ **então retorne**
7 $S \leftarrow$ Melhoria$(N, C, w, S,$ MaxTabu, MaxDelta, Semente$)$;
8 **se** $\text{Viável}_{BP}(S)$ **então retorne**
9 **fim-para**
10 **retorne**
fim C+R+M_BP

Figura 3.12: Pseudo-código do procedimento C+R+M_BP

Seleção_Movimento na linha 7, cuja complexidade $O(n^4)$ foi determinada anteriormente. Logo, cada iteração do procedimento Melhoria é $O(n^4)$.

A rotina C+R+M_BP, ilustrada pela Figura 3.12, integra as três fases Construção, Redistribuição e Melhoria, descritas anteriormente. O laço que compreende as linhas 1-9 busca uma solução viável para BP com exatamente m caixas. A busca pára quando uma solução viável for encontrada ou quando o número máximo de tentativas for atingido, retornando a melhor solução encontrada S. Cada tentativa inicia, na linha 2, pela seleção de um algoritmo construtivo guloso do conjunto de heurísticas disponíveis $H = \{$ DB3FD, DBFD, DWSFD, DWFD$\}$ usado para construir uma solução inicial S para DBP na linha 3. Se a solução gerada nesta fase for inviável para BP o procedimento de redistribuição de peso das caixas, descrito na Seção 3.2 e ilustrado na Figura 3.9, é aplicado na linha 5. Se a nova solução continuar inviável para BP aplica-se, na linha 7, o procedimento busca tabu de melhoria para reduzir a inviabilidade da solução, descrito na Seção 3.3 e esquematizado na Figura 3.11.

A heurística híbrida de melhoria para o problema de *bin packing* completa HI_BP é esquematizada na Figura 3.13. O pré-processamento ou fase inicial é realizado entre as linhas 1-8. Na primeira linha calcula-se uma solução heurística usando BFD (descrito na Seção 2.2.2) e inicializa-se o limite superior U. A seguir, nas linhas 2 e 3, calculam-se os limites inferiores L_1 e L_2 (Seção 2.2.4.2). Se a solução encontrada é comprovadamente ótima, termina-se o procedimento. Na linha 4 calcula-se o limite inferior L_3 que utiliza o procedimento de redução MTRP (descritos, respectivamente,

nas Seções 2.2.4.4 e 2.2.4.3). Este último cria uma solução parcial com b caixas. Se a instância original foi totalmente reduzida, tem-se uma solução ótima e o procedimento pára na linha 5. Se a solução encontrada por `BFD` possui L_3 caixas, o procedimento pára na linha 6. Na linha 7 atualizam-se o limite inferior L_3 e o limite superior U diminuindo o número de caixas da solução parcial e na próxima linha atualiza-se o conjunto de objetos. Na próxima linha, o melhor limite inferior L é determinado entre L_3, o limite inferior de Fekete e Schepers e o limite L_ϑ (os dois últimos descritos na Seção 2.2.4). Se identificada uma solução ótima com U caixas o procedimento pára na linha 10. Do contrário, na linha 11, inicializa-se a variável `nCaixas` com L. O laço compreendido entre as linhas 12 e 16 busca encontrar uma solução viável com `nCaixas` caixas usando o procedimento `C+R+M_BP`. Caso esse procedimento não tenha sucesso, o número de caixas `nCaixas` é incrementado de uma unidade, na linha 15, e uma nova tentativa iniciada, enquanto o número de caixas for menor do que o limite superior. Se o procedimento `C+R+M_BP` foi capaz de melhorar a solução criada pela heurística `BFD` esta é retornada na linha 14, do contrário a solução com valor igual a U gerada por `BFD` é retornada na linha 18.

Observa-se que a primeira iteração do laço 6-10 do procedimento `HI_BP` atribui à variável `nCaixas` um determinado limite inferior L no número de caixas e as demais iterações, se existirem, incrementam este valor em uma unidade. Em seguida, o procedimento `C+R+M_BP` recebe o valor desta variável que determina o número m de caixas utilizadas nas fases de `Construção`, `Redistribuição` e `Melhoria`. Deve-se assegurar então que a solução gerada nestas três fases utilizou todas as m caixas. Em relação às heurísticas da fase de `Construção`, sabe-se que qualquer solução gerada na primeira iteração desta fase utiliza todas as m caixas, pois $m = L$. Uma segunda iteração desta fase, com o parâmetro número de caixas igual a $L+1$, só será realizada se a primeira iteração não conseguiu construir uma solução viável com L caixas. Pelas definições das heurísticas, apresentadas na Seção 3.1, pode-se garantir que as $L + 1$ caixas serão utilizadas na segunda iteração, e assim sucessivamente. Como visto na Seção 3.2, a fase de `Redistribuição` pode gerar uma solução com menos de `nCaixas` utilizadas. Se isso ocorrer, a solução corrente também será viável e o procedimento pára e retorna a solução viável com `nCaixas` $- 1$ (Figura 3.9). Para finalizar, a rotina `Melhoria` transforma uma solução com m caixas a partir da troca de objetos entre as caixas. Este movimento mantém todas as caixas utilizadas.

Procedimento HI_BP
Entrada: $N = \{1, \ldots, n\}, C$
w_i $(i = 1, \ldots, n)$ onde $w_h \geq w_{h+1}, h = 1, \ldots, n-1$,
H, MaxTentativas, MaxRedistribuição, MaxTabu, MaxDelta, Semente.
Saída: $S = (B_1, \ldots, B_m)$.
1 Gere solução heurística S_{BFD} utilizando BFD e inicialize U;
2 Calcule limite L_1; **se** $L_1 = U$ **então** $S \leftarrow S_{BFD}$ **e retorne**
3 Calcule limite L_2; **se** $L_2 = U$ **então** $S \leftarrow S_{BFD}$ **e retorne**
4 Calcule limite L_3 e seja $S_{MTRP} = (B_1, \ldots, B_b)$ a solução parcial com b caixas gerada pelo procedimento de redução MTRP;
5 **se** $L_3 = b$ **então** $S \leftarrow S_{MTRP}$ **e retorne**
6 **se** $L_3 = U$ **então** $S \leftarrow S_{BFD}$ **e retorne**
7 $L_3 \leftarrow L_3 - b$; $U \leftarrow U - b$;
8 Elimine de N os objetos de S_{MTRP};
9 $L \leftarrow \max\{L_3, L_*^{(20)}, L_\vartheta\}$;
10 **se** $L = U$ **então** $S \leftarrow S_{BFD}$ **e retorne**
11 nCaixas $\leftarrow L$;
12 **enquanto** nCaixas $< U$ **faça**
13 $S \leftarrow$ C+R+M_BP$(N, C,$ nCaixas$, w, H,$ MaxTentativas, MaxRedistribuição, MaxTabu, MaxDelta, Semente);
14 **se** Viável$_{BP}(S)$ **então** $S \leftarrow S \cup S_{MTRP}$ e **retorne**
15 **senão** nCaixas $\leftarrow$ nCaixas $+ 1$;
16 **fim-enquanto**
17 **se** nCaixas $= U$ **então** $S \leftarrow S_{MTRP} \cup S_{BFD}$;
18 **retorne**
fim HI_BP

Figura 3.13: Pseudo-código da heurística híbrida para o problema de *bin packing* HI_BP

3.4 Experimentos Computacionais

Nesta seção descrevem-se os experimentos computacionais realizados aplicando-se a heurística híbrida de melhoria em um conjunto de problemas testes. A heurística HI_BP e todos os demais procedimentos foram codificados na linguagem C. Para a geração das seqüências de números pseudo-aleatórios implementou-se, também na linguagem C, o gerador descrito em [79]. O procedimento de redução MTRP e o limite inferior L_3, descritos nas Seções 2.2.4.3 e 2.2.4.4, também foram implementados na linguagem C, respeitando a parte correspondente do código original em FORTRAN do procedimento exato MTP de Martello e Toth, disponível em [64]. Todos os programas foram compilados utilizando-se a versão 2.95.2 do compilador gcc com parâmetro de otimização igual a -O3.

Todos os experimentos foram feitos em um computador Pentium IV com relógio de 1.7 GHz e 256 MB de memória RAM. Os tempos computacionais são expressos em segundos. Define-se o *erro absoluto* entre

dois valores x e y como $|x - y|$ e o *erro relativo percentual* como $\frac{|x-y|}{x} \cdot 100$, onde x representa o valor da solução ótima (ou o melhor limite inferior conhecido quando a solução ótima não é conhecida) e y o valor da solução (número de caixas) obtida pela heurística `HI_BP`.

3.4.1 Parâmetros

A implementação desenvolvida nesta tese usou os seguintes valores para os parâmetros da heurística `HI_BP`:

- Número de tentativas realizadas pelo procedimento `C+R+M_BP` para construir uma solução viável para BP com uma determinada quantidade fixa de caixas: `MaxTentativas` $= 4$.
- Controle do número máximo de iterações usado na fase de `Redistribuição`: `MaxRedistribuição` $= 100$.
- Período de proibição de um movimento usado pelo procedimento `Seleção_Movimento`: `TabuTenure` $\in [0.8\sqrt{n}, 1.2\sqrt{n}]$.
- Limite utilizado pelo procedimento `Seleção_Movimento` para buscar detectar a existência de um movimento combinado para uma determinada caixa alvo: `MaxDelta` $= 0.2 \min_{i=1,\ldots,n}\{w_i\}$.
- Número máximo de iterações sem redução no menor valor do excesso acumulado de todas as caixas, usado como critério de parada pelo procedimento `Melhoria`: `MaxTabu` $= 4000$.
- Utilizou-se duas seqüências independentes de números pseudo-aleatórios: uma para gerar o valor `TabuTenure` no procedimento `Melhoria` e a outra para gerar o valor `NivelMovimento`$(i_1 \leftrightarrow i_2)$ no procedimento `Seleção_Movimento`. O primeiro valor de cada seqüência é `Semente` $= 1$.

3.4.2 Problemas Testes

Com o objetivo de avaliar a qualidade da heurística `HI_BP` buscou-se considerar problemas testes para os quais houvesse literatura a respeito e que estivessem publicamente disponíveis. Desta forma, três grupos de problemas testes foram considerados, totalizando 1587 instâncias.

O primeiro grupo (`GRUPO-I`) consiste de dois conjuntos de problemas testes introduzidos por Falkenauer [20] e disponibilizadas em [6].

- `uniforme`: conjunto formado por 80 instâncias com capacidade da caixa $C = 150$ e pesos dos objetos gerados de uma distribuição uniforme entre 20 e 100. Está dividido em quatro classes, uma para cada valor de $n = 120, 250, 500, 1000$, com 20 instâncias em cada. Elas são identificadas, respectivamente, por `u_120`, `u_250`, `u_500` e `u_1000`. O valor da solução ótima para todas estas instâncias é conhecido, c.f. Valério de Carvalho [84] e Gent [33]. Os valores ótimos coincidem com o limite inferior mais simples (L_1) para 79 das 80 instâncias. A única exceção é a instância `u_250_13`, cujo valor ótimo é 103 e o limite L_1 é 102.
- `trinca`: as instâncias deste conjunto são consideradas difíceis. Foram construídas de tal forma que todas as caixas da solução ótima possuem exatamente três objetos que enchem completamente a caixa. O valor da solução ótima coincide com o limite inferior L_1. Os pesos dos objetos pertencem ao intervalo (250,500) e as caixas têm capacidade $C = 1000$. Existem quatro classes, com 20 instâncias em cada, para cada valor de $n = 60, 120, 249, 501$ identificadas, respectivamente, por `t_60`, `t_120`, `t_249` e `t_501`.

O segundo grupo (`GRUPO-II`) é formado por três conjuntos de problemas testes introduzidos por Scholl et al. [77] e disponíveis em [57]. Em cada conjunto, as diferentes classes de problemas foram criadas variando-se o número de objetos n, a capacidade da caixa C e os pesos dos objetos.

- `set_1`: conjunto formado por 720 instâncias. O número de objetos assume os valores $n = 50, 100, 200, 500$, a capacidade da caixa os valores $C = 100, 120, 150$, e os pesos dos objetos são inteiros selecionados aleatoriamente de uma distribuição uniforme nos diferentes intervalos $[1, 100]$, $[20, 100]$ e $[30, 100]$. A combinação dos diferentes parâmetros resulta em 36 classes. Foram geradas 20 instâncias em cada classe.

Estas instâncias foram construídas de forma semelhante a algumas instâncias propostas por Martello e Toth [64].

- `set_2`: conjunto formado por 480 instâncias. O número de objetos assume os valores $n = 50, 100, 200, 500$ e a capacidade da caixa o valor $C = 1000$. Com o objetivo de gerar instâncias cujo número médio de objetos por caixa varie entre três e nove, dois outros parâmetros foram considerados: o parâmetro $\bar{w}$ que representa o peso médio dos objetos desejado, podendo assumir os valores $\bar{w} = C/3, C/5, C/7, C/9$, e o parâmetro $\delta = 20\%, 50\%, 90\%$ que determina o desvio máximo de um dado valor w em relação a $\bar{w}$. Por exemplo, quando $\bar{w} = C/5$ e $\delta = 20\%$ os pesos dos objetos são selecionados aleatoriamente de uma distribuição uniforme no intervalo discreto $[160, 240]$.
- `set_3`: conjunto formado por uma única classe com dez instâncias consideradas como difíceis pelos autores, com $n = 200$, capacidade da caixa $C = 100000$, e peso dos objetos $w \in [20000, 35000]$. Uma característica desta classe, em oposição às dos outros dois conjuntos, é que neste caso existe uma grande quantidade de valores diferentes possíveis para os pesos dos objetos.

O documento disponível em [57] relata 704 valores ótimos para as instâncias do conjunto `set_1`, 477 para as instâncias do conjunto `set_2`, e três do conjunto `set_3`. Do total de 1210 instâncias do `GRUPO-II`, 26 são identificadas na URL como "abertas". A aplicação do novo limite inferior L_ϑ ou do limite inferior L_{CS} para o problema de *cutting stock*, utilizado anteriormente por Schwerin e Wäscher [81], prova a otimalidade destas 26 instâncias.

Os três conjuntos do último grupo (`GRUPO-III`) de problemas testes considerados são:

- `was_1`: conjunto formado por uma única classe de 100 instâncias com capacidade da caixa $C = 1000$, $n = 100$ e peso dos objetos $w \in [150, 200]$.
- `was_2`: conjunto formado por uma única classe de 100 instâncias com capacidade da caixa $C = 1000$, $n = 120$ e peso dos objetos $w \in [150, 200]$.
- `gau_1`: conjunto de 17 instâncias com características variadas, onde a capacidade da caixa $C = 10000$, o número n de objetos varia de 57 a 239 e os pesos variam de 2 a 7332.

As instâncias dos conjuntos `was_1` e `was_2` foram coletadas de [80, 81]. O conjunto `gau_1` é formado por problemas residuais coletados

de [88] e identificados, pelos autores, como difíceis. Os conjuntos `was_1`, `was_2` e `gau_1` correspondem, respectivamente, aos arquivos `sch_wae1.bpp`, `sch_wae2.bpp` e `wae_gau1.bpp` disponíveis na URL `http://www.apdio.pt/sicup/Sicuphomepage/research.htm`. Das 200 instâncias dos conjuntos `was_1` e `was_2`, os valores ótimos de 163 instâncias coincidem com o limite inferior L_1. Os valores ótimos das demais 37 instâncias foram identificados com a utilização do novo limite L_ϑ. Das 17 instâncias de `gau_1` a URL citada identifica o valor ótimo de duas instâncias, enquanto Alvim et al. [3] identificam os valores ótimos de mais oito instâncias e os sete valores ótimos restantes são relatados em [76].

Nas próximas seções, chama-se de `Combinado` o conjunto formado pelas 581 instâncias dos conjuntos `uniforme`, `trinca`, `set_1`, `set_2`, e `set_3`, para as quais o pré-processamento ou a heurística `BFD` não conseguiram obter, por si sós, a solução ótima. As instâncias deste conjunto são destacadas em [1].

3.4.3 Estudo das Fases de `HI_BP`

Nesta seção investiga-se a eficácia das diferentes fases de `HI_BP`. Para tal criou-se três versões do algoritmo. A versão `C` executa somente a fase de `Construção`, a versão `C+R` executa também a fase de `Redistribuição` e a versão `C+R+M` inclui a fase de `Melhoria`, sendo equivalente à versão completa do algoritmo. O conjunto de instâncias `Combinado`, acima definido, será utilizado neste estudo. A Tabela 3.2 apresenta um resumo dos principais resultados. Para cada conjunto identificado na primeira coluna, a próxima coluna lista o número de instâncias que fazem parte do conjunto `Combinado`. A seguir, relata-se os resultados obtidos em cada versão: o número de soluções ótimas encontradas e o tempo total de processamento em segundos (para todas as instâncias). A fração do tempo total correspondente à execução somente da fase `Melhoria` da versão `C+R+M` é mostrada na última coluna.

A versão `C` não encontra a solução ótima de muitas instâncias. Somente cerca de 58% das instâncias são resolvidas. A execução também da fase de `Redistribuição`, versão `C+R`, melhora significativamente o número de soluções ótimas encontradas e até mesmo reduz o tempo de processamento. Isso foi possível pois, nesses casos, executou-se menos iterações para encontrar a melhor solução. Por exemplo, considerando-se o conjunto `trinca`, nenhuma solução ótima foi encontrada pela versão `C` nem pela versão `C+R`.

		C		C+R		C+R+M		
conjunto	instâncias	ótimos	tempo	ótimos	tempo	ótimos	tempo	% M
uniforme	74	41	42.30	68	1.92	74	2.65	87.17
trinca	80	0	144.97	0	123.39	80	78.36	99.06
set_1	173	111	5.28	154	2.94	173	122.00	98.25
set_2	244	185	171.99	240	44.56	244	6.09	71.59
set_3	10	0	4.39	10	1.73	10	46.00	96.48
totais	581	337		472		581		

Tabela 3.2: Resultados do experimento que analisa a importância das fases de HI_BP

Poderia-se esperar que a versão C+R fosse mais demorada, pois executa uma fase a mais. Entretanto, o maior erro absoluto desta versão foi 1, enquanto que na versão C foi 2. Embora nenhuma solução ótima tenha sido encontrada em ambas as versões, melhores soluções foram encontradas pela versão C+R, o que justifica ela ter gasto menos tempo. Exceto pelas instâncias do conjunto trinca, para as quais nenhuma solução ótima foi encontrada, as duas primeiras versões conseguem encontrar 472 soluções ótimas entre 501 instâncias (94%). A fase final de Melhoria é fundamental para resolver as instâncias da classe trinca. Entretanto, perde-se muito tempo nesta fase quando nenhuma melhoria é possível (set_3, por exemplo). Considerando-se todas as fases, a heurística híbrida de melhoria conseguiu encontrar soluções ótimas para todas as instâncias do conjunto Combinado.

3.4.4 Comparação com Outros Métodos

Nesta seção compara-se o resultado obtido pela heurística híbrida de melhoria com aqueles obtidos por outros métodos relatados na literatura. A heurística HI_BP foi aplicada uma única vez para cada instância, com parâmetros conforme definidos na Seção 3.4.1. Os valores ótimos comprovados de todas as instâncias e os resultados detalhados obtidos pela heurística HI_BP são apresentados em [1].

A comparação com outros métodos é favorável para HI_BP. Resolveu-se otimamente todas as instâncias do GRUPO-I, composto pelos conjuntos uniforme e trinca, que também são resolvidas pelo método exato proposto por Valério de Carvalho [84].

Fleszar e Hindi [25] usaram as instâncias dos grupos GRUPO-I e GRUPO-II para avaliar a qualidade da melhor heurística por eles proposta (Perturbation MBS' + VNS). Uma vez que os autores concluem que Perturbation MBS' + VNS é competitiva com BISON [77] para as instâncias do GRUPO-II, comparou-se o desempenho de HI_BP diretamente com o de Perturbation MBS' + VNS. A Tabela 3.3 mostra o número de instâncias, a quantidade de instâncias para as quais a solução ótima foi encontrada, o valor máximo encontrado para o erro absoluto, o valor máximo encontrado para o erro relativo, e a média e valor máximo do tempo de processamento em segundos. As mesmas estatísticas são relatadas para Perturbation MBS' + VNS (obtidas em um Pentium de 400 MHz e extraídas da Tabela VIII em [25]).

HI_BP também encontrou a solução ótima de todas as instâncias do GRUPO-II, e também os mesmos valores para as instâncias do conjunto set_1 para as quais Fleszar e Hindi [25] reportaram melhoria no valor da solução conhecida até então: N4C3W4_O (226), N4C3W4_P (219), N4C3W4_R (214), e N4C3W4_T (216). HI_BP identificou 41 soluções ótimas a mais do que Perturbation MBS' + VNS. Desconsiderando-se as 26 instâncias cujos valores ótimos foram identificados simplesmente com a aplicação do novo limite inferior L_ϑ ou do limite inferior L_{CS}, conclui-se que HI_BP encontrou, pelo menos, 15 soluções ótimas a mais do que Perturbation MBS' + VNS (uma do conjunto u_250, dez do conjunto set_1, três do conjunto set_2 e uma do conjunto set_3). Observou-se a superioridade de HI_BP em relação à qualidade das soluções, embora com tempos de processamento maiores, em alguns casos. Quando os limites utilizados pela heurística HI_BP não conseguem identificar o valor ótimo de uma instância, perde-se tempo tentando encontrar uma solução inviável. Isto acontece em particular com a instância u_250_13 do conjunto u_250, com 43 instâncias do conjunto set_1 e com uma instância do conjunto set_3. Estas instâncias encontram-se em destaque nas tabelas disponíveis em [1].

Comparou-se os resultados obtidos por HI_BP para as instâncias do GRUPO-III (was_1, was_2 e gau_1) com aqueles relatados em [82], onde o melhor resultado é o valor obtido pela heurística MTPCS [81] com limite de *backtracking* igual a 500. A Tabela 3.4 relata os resultados de 11 instâncias para as quais HI_BP melhorou este resultado. Em todos os casos a diferença no valor da solução é a redução de uma unidade no número de caixas. Para todas as outras instâncias desses conjuntos encontrou-se os mesmos resultados relatados em [82]. Do total de 217 instâncias do GRUPO-III, HI_BP não encontrou o valor ótimo de apenas cinco instâncias. Schoenfield [76] relata que obteve os valores ótimos destas cinco instâncias através da combinação de várias técnicas, incluindo programação linear.

3.4.5 Robustez

A Tabela 3.5 mostra algumas estatísticas obtidas a partir de cinco execuções de cada instância envolvida neste experimento (7935 execuções), usando cinco valores distintos (consecutivos a partir de 1) para a semente do gerador de números aleatórios. Para cada conjunto, lista-se nas primeiras colunas o número total de execuções, o número de vezes que uma solução ótima é encontrada, os valores médio e máximo encontrados para o erro

		HI_BP					Perturbation MBS' + VNS				
			erro máx.		tempo			erro máx.		tempo	
conjunto	instâncias	ótimos	abs.	rel.	med.	max.	ótimos	abs.	rel.	med.	max.
u_120	20	20	0	0	0.00	0.01	20	0	0.00	0.02	0.04
u_250	20	20	0	0	0.12	2.20	19	1	0.95	0.03	0.16
u_500	20	20	0	0	0.00	0.01	20	0	0.00	0.04	0.14
u_1000	20	20	0	0	0.01	0.02	20	0	0.00	0.07	0.27
t_60	20	20	0	0	0.37	1.65	20	0	0.00	0.01	0.01
t_120	20	20	0	0	0.85	4.53	20	0	0.00	0.02	0.04
t_249	20	20	0	0	0.22	0.51	20	0	0.00	0.02	0.04
t_501	20	20	0	0	2.49	19.26	20	0	0.00	0.06	0.10
set_1	720	720	0	0	0.19	19.23	694	2	2.44	0.15	1.78
set_2	480	480	0	0	0.01	1.19	474	1	2.94	0.10	4.57
set_3	10	10	0	0	4.60	44.76	2	1	1.85	3.74	5.05

Tabela 3.3: HI_BP versus Perturbation MBS' + VNS

conjunto	instância	HI_BP	tempo
was_1	BPP81	18	0.07
was_2	BPP56	21	0.16
	BPP71	21	0.26
gau_1	TEST0097	12	0.01
	TEST0055	15	0.00
	TEST0049	11	0.00
	TEST0075	13	0.01
	TEST0054	14	0.00
	TEST0044	14	0.01
	TEST0095	16	0.01
	TEST0055	20	0.01

Tabela 3.4: Instâncias do GRUPO-II para as quais HI_BP melhorou as melhores soluções conhecidas.

absoluto, os valores médio e máximo do tempo de processamento em segundos.

A heurística HI_BP pode encontrar uma solução ótima em quatro diferentes estágios da sua execução. Na mesma tabela, relata-se também o número de execuções em que a solução ótima foi encontrada pelo pré-processamento ou pelo algoritmo BFD (I), na fase de Construção (P1), na fase de Redistribuição (P2), ou na fase de Melhoria (P3). Como observado por Scholl et al. [77], muitas instâncias dos conjuntos set_1 e set_2 são facilmente resolvidas na etapa inicial (I). Mais uma vez, percebe-se que a abordagem de busca tabu usada para reduzir a inviabilidade da solução na fase de Melhoria (P3) é absolutamente necessária e que esta fase foi a única que conseguiu obter soluções ótimas para o conjunto trinca.

As quatro últimas colunas indicam o número de execuções em que a solução ótima originou de uma solução inicial construída pelas heurísticas DB3FD (H1), DBFD (H2), DWSFD (H3) ou por DWFD (H4) e mostra a importância do uso de diferentes estratégias na construção de soluções iniciais. Observa-se que, na maior parte das vezes, a solução construída por H1 leva à melhor solução final. Entretanto, isto é devido ao fato de H1 ser a primeira heurística aplicada (observa-se também o efeito oposto para a heurística H4, que é a última a ser aplicada e que não é executada se a solução ótima já tiver sido encontrada). A tabela indica que todas as heurísticas utilizadas foram necessárias. Experimentos computacionais preliminares consideraram cada uma dessas quatro heurísticas de cada vez e então somente uma versão randomizada da mesma. Percebeu-se que mesmo fazendo um número bem maior do que quatro tentativas, nenhuma delas foi capaz de sistematica-

mente construir soluções iniciais que sempre levassem à solução ótima de todas as instâncias. Diversidade na fase de `Construção` foi necessária e a combinação de diferentes heurísticas foi fundamental na obtenção da solução ótima de todas as instâncias. É importante notar que considerando os valores ótimos comprovados para todas as instâncias e para cinco execuções de cada instância (7935 execuções no total), em apenas quatro execuções do conjunto `set_1` e 25 execuções (cinco execuções de cinco instâncias) do conjunto `gau_1` não se encontrou a melhor solução conhecida. Estes resultados ilustram a robustez da heurística híbrida de melhoria.

3.4.6 Considerações

Neste capítulo foi apresentada a heurística híbrida de melhoria `HI_BP` para o problema de *bin packing* (BP). A estrutura básica do algoritmo pode ser dividida em três fases, após uma etapa inicial de pré-processamento. Na etapa de pré-processamento aplica-se um procedimento para reduzir o tamanho do problema, calculam-se limites inferiores (incluindo o novo limite L_ϑ) e um limite superior, e determina-se o intervalo de valores possíveis para o número de caixas da solução. A primeira fase (`Construção`) usa diversas heurísticas gulosas para construir soluções com um determinado número fixo de caixas, permitindo-se violar a restrição de capacidade máxima da caixa. Esta abordagem remete ao problema de escalonamento de tarefas em processadores paralelos idênticos ($P||C_{\max}$), uma vez que as soluções geradas são sempre viáveis para este problema, porém não necessariamente viáveis para BP. Na fase `Redistribuição`, aplicam-se alternadamente estratégias de equilíbrio e desequilíbrio dos pesos das caixas e, freqüentemente, obtém-se uma solução viável para BP. A última fase (`Melhoria`) utiliza uma abordagem de busca tabu para reduzir a violação da capacidade das caixas. Uma nova fase é iniciada somente quando a solução corrente não for viável para BP. Quando a última fase não consegue viabilizar a solução, o número fixo de caixas da solução é incrementado em uma unidade e inicia-se uma nova iteração da fase de `Construção`.

Para lidar com a dificuldade de se definir um movimento promissor para BP, apresentou-se, na fase de `Melhoria`, uma nova estratégia de seleção de movimento para a busca tabu, que combina múltiplas regras e prioridades variadas. Acredita-se que a estratégia apresentada seja uma boa alternativa para problemas em que exista dificuldade para se definir um movimento que leve a uma melhor solução, quando não há clareza de qual movimento deve

			erro abs.		tempo		estágios				heurística			
conjunto	execuções	ótimos	médio	máx.	médio	máx.	I	P1	P2	P3	H1	H2	H3	H4
u_120	100	100	0	0	0.00	0.01	30	40	0	30	70	0	0	0
u_250	100	100	0	0	0.15	3.19	0	55	10	35	100	0	0	0
u_500	100	100	0	0	0.00	0.01	0	20	75	5	100	0	0	0
u_1000	100	100	0	0	0.01	0.03	0	0	90	10	100	0	0	0
t_60	100	100	0	0	0.33	2.53	0	0	0	100	89	9	2	0
t_120	100	100	0	0	1.14	6.88	0	0	0	100	88	9	3	0
t_249	100	100	0	0	0.29	2.91	0	0	0	100	100	0	0	0
t_501	100	100	0	0	1.24	19.26	0	0	0	100	99	1	0	0
set_1	3600	3596	0	1	0.20	23.89	2735	340	230	291	833	22	4	2
set_2	2400	2400	0	0	0.01	1.89	1180	300	760	160	1215	0	5	0
set_3	50	50	0	0	4.71	50.61	0	0	40	10	50	0	0	0
gau_1	85	60	0.29	1	0.60	2.40	10	0	50	0	50	0	0	0
was_1	500	500	0	0	0.02	0.14	0	0	415	85	500	0	0	0
was_2	500	500	0	0	0.02	3.58	0	0	480	20	500	0	0	0

Tabela 3.5: Estatísticas de cinco execuções de HI_BP, cada uma delas com uma diferente semente

ser selecionado.

A heurística foi testada em três grupos de problemas formado por 1587 instâncias da literatura. O valor ótimo de todas estas instâncias está disponível na literatura. `HI_BP` melhorou as melhores soluções conhecidas de onze instâncias e não encontrou o valor ótimo de apenas cinco instâncias. Não parece existir na literatura outro método isolado que sistematicamente tenha sido capaz de resolver tantas destas 1587 instâncias, o que mostra a robustez de `HI_BP`.

O próximo capítulo é dedicado ao $P||C_{\max}$. Apresentam-se o problema e limites inferiores, e faz-se um resumo de diversos métodos de resolução encontrados na literatura.