4
GRASP com religamento de caminhos para o problema de atribuição de três índices

4.1
Definição do problema

Pierskalla [94] mostrou originalmente que o problema de atribuição de três índices (AP3) é uma extensão do problema clássico de atribuição bidimensional. O AP3 pode ser visto como um problema de otimização em um grafo completo tripartido $K_{n,n,n} = (I \cup J \cup K, (I \times J) \cup (I \times K) \cup (J \times K))$, onde I, J e K são conjuntos disjuntos de tamanho n. Caso um custo $c_{i,j,k}$ seja associado a cada tripla $(i, j, k) \in I \times J \times K$, então o AP3 consiste em encontrar um subconjunto $A \in I \times J \times K$ de n triplas, tal que todos os elementos de $I \cup J \cup K$ ocorram em exatamente uma tripla de A e que a soma dos custos de todas as triplas seja minimizada.

O AP3 possui a seguinte formulação como um problema de programação inteira em variáveis 0-1:

$$\min \sum_{i \in I, j \in J, k \in K} c_{ijk} x_{ijk}$$

sujeito a

$$\sum_{j \in J, k \in K} x_{ijk} = 1, \ \forall \ i \in I, \tag{4-1}$$

$$\sum_{i \in I, k \in K} x_{ijk} = 1, \ \forall \ j \in J, \tag{4-2}$$

$$\sum_{i \in I, j \in J} x_{ijk} = 1, \ \forall \ k \in K, \tag{4-3}$$

$$x_{ijk} \in \{0, 1\}, \ \forall \ i \in I, \ j \in J, \ k \in K,$$

onde $I = J = K = \{1, 2, \ldots, n\}$.

A formulação acima modela, por exemplo, o problema de atribuir tarefas a trabalhadores e a máquinas a custo mínimo. Dessa forma, c_{ijk} é o

custo de atribuir a tarefa i ao trabalhador j na máquina k. A variável de decisão x_{ijk} é igual a 1 se, e somente se, a tarefa i é atribuída ao trabalhador j na máquina k. As restrições acima implicam em que cada elemento de um dos conjuntos I, J e K seja atribuído a exatamente um elemento de cada um dos outros dois conjuntos.

O AP3 também pode ser formulado usando-se funções de permutação. Existem n^3 elementos de custo c_{ijk}. A obtenção da solução ótima do AP3 consiste na escolha dos n menores elementos de custo, tais que as restrições do problema não sejam violadas. A formulação baseada em permutações para o AP3 é

$$\min_{p,q \in \pi_N} \sum_{i=1}^{n} c_{ip(i)q(i)}$$

onde π_N denota o conjunto de todas as permutações do conjunto de números inteiros $N = \{1, 2, \ldots, n\}$. Deve-se notar que $|\pi_N| = n!$. Observa-se que nenhum dos elementos de custo c_{ijk} escolhidos podem ter os mesmos valores para os índices i, j e k que algum outro elemento. Por exemplo, $x_{1,2,4} = x_{3,2,5} = 1$ nunca irá ocorrer, visto que essas atribuições usam o mesmo valor para $j = 2$. A formulação baseada em permutações incorpora à função objetivo os três conjuntos de restrições mostrados em 4-1, 4-2 e 4-3, facilitando a implementação de heurísticas para o AP3.

Uma formulação equivalente para o AP3 (também chamado de problema de *matching tridimensional*) é mostrada a seguir. Dados três conjuntos disjuntos I, J e K, tais que $|I| = |J| = |K| = n$ e um custo c_{ijk} associado a cada tripla ordenada $(i, j, k) \in I \times J \times K$, encontre uma coleção de custo mínimo de n triplas disjuntas $(i, j, k) \in I \times J \times K$.

O AP3 na sua versão de problema de decisão é NP-Completo [52, 55]. Aplicações do AP3 podem ser encontradas em Pierskalla [95, 94], Frieze e Yadegar [51] e Crama et al. [36] e incluem problemas em siderurgia, escalonamento de capitais de investimento, atribuição de tropas militares, otimização de cobertura por satélites, escalonamento de práticas de ensino e produção de placas de circuitos integrados.

Algoritmos exatos e heurísticas têm sido propostos para o problema de atribuição de três índices, incluindo-se [13, 22, 24, 25, 37, 53, 66, 75, 95, 94, 130].

A seguir será apresentada uma heurística GRASP para o AP3. Também será mostrado como estratégias de religamento de caminhos podem ser incorporadas a essa heurística. O religamento de caminhos é uma estratégia de intensificação que será usada para explorar trajetórias que conectam soluções de qualidade. Várias estratégias de GRASP com religa-

```
procedimento CONSTRUTIVA(semente, n, c, S)
1     Escolha α ∈ [0,1] aleatoriamente;
2     S = ∅;
3     C = {(i,j,k) ∈ I × J × K};
4     para p = 1,...,n−1 faça
5           c̲ = min{c_ijk | (i,j,k) ∈ C};
6           c̄ = max{c_ijk | (i,j,k) ∈ C};
7           C' = {(i,j,k) ∈ C | c_ijk ≤ c̲ + α(c̄ − c̲)};
8           Escolha (i_p, j_p, k_p) ∈ C' aleatoriamente;
9           S = S ∪ {(i_p, j_p, k_p)};
10          Γ^i(p) = {(i,j,k) ∈ C | i = i_p};
11          Γ^j(p) = {(i,j,k) ∈ C | j = j_p};
12          Γ^k(p) = {(i,j,k) ∈ C | k = k_p};
13          C = C \ {Γ^i(p) ∪ Γ^j(p) ∪ Γ^k(p)};
14    fim-para;
15    S = S ∪ C;
fim CONSTRUTIVA
```

Figura 4.1: Fase construtiva para o AP3.

mento de caminhos serão propostas e testadas.

4.2 GRASP para o AP3

Vários algoritmos GRASP aplicados a problemas de atribuição podem ser encontrados na literatura [3, 46, 49, 76, 93, 86, 90, 92, 98, 97, 104, 103, 106, 118]. A seguir serão descritas as fases construtivas e de busca local da heurística GRASP para o AP3.

4.2.1 Fase construtiva

Para apresentar o algoritmo usado na fase construtiva, uma solução será representada por uma coleção de triplas ordenadas. Uma solução viável S será construída selecionando-se n triplas, uma a cada iteração. A Figura 4.1 mostra a fase construtiva em pseudo-código.

Um parâmetro α, usado para construir a lista restrita de candidatos é selecionado aleatoriamente do intervalo $[0, 1]$ (linha 1). O valor de α é mantido durante a fase construtiva. A solução S é inicialmente vazia e o conjunto C de triplas candidatas é inicialmente o conjunto de todas as triplas (linhas 2 e 3).

Os comandos que vão da linha 4 a 14 fazem a seleção das primeiras $n-1$ triplas. Para selecionar a p-ésima tripla que será adicionada à solução,

uma lista restrita de candidatos C' é definida (nas linhas 5 a 7) pelas triplas (i, j, k) do conjunto de candidatos C que possuam custo $c_{ijk} \leq \underline{c} + \alpha(\overline{c} - \underline{c})$, onde

$$\underline{c} = \min\{c_{ijk} \mid (i, j, k) \in C\} \text{ e } \overline{c} = \max\{c_{ijk} \mid (i, j, k) \in C\}.$$

Uma tripla $(i_p, j_p, k_p) \in C'$ é escolhida aleatoriamente na linha 8 e é adicionada à solução, fazendo-se $S = S \cup \{(i_p, j_p, k_p)\}$ (linha 9).

Uma vez que (i_p, j_p, k_p) tenha sido selecionada, o conjunto C de triplas candidatas deve ser ajustado para levar em conta que (i_p, j_p, k_p) faz parte da solução. Todas as triplas (i, j, k) tais que $i = i_p$, $j = j_p$ ou $k = k_p$ são removidas do conjunto corrente de triplas candidatas nas linhas 10 a 13. Esse processo de atualização é o gargalo computacional da fase construtiva. Uma implementação básica consiste em verificar todos os $O(n^3)$ elementos de custo para atualizar a lista de candidatos a cada iteração. Elementos retirados da lista de candidatos são simplesmente marcados como inválidos. Como são feitas $n - 1$ iterações, a complexidade desse algoritmo básico é $O(n^4)$. Na implementação proposta nesse trabalho, quatro listas duplamente encadeadas foram usadas para acelerar a etapa de atualização da lista de candidatos, reduzindo a complexidade da fase construtiva de $O(n^4)$ para $O(n^3)$.

O conjunto de triplas (i, j, k) que fazem parte da lista de candidatos C é usado para criar uma lista duplamente encadeada $L_c = \{c_{i,j,k}; (i, j, k)\}$. Os elementos de L_c aparecem em ordem crescente de custo. A lista L_c é usada durante a construção da lista restrita de candidatos C'. Os valores do menor e do maior custo na lista de candidatos (computados nas linhas 5 e 6 do pseudo-código da Figura 4.1) são mantidos no primeiro e no último elemento de L_c, respectivamente. Esses elementos são endereçados diretamente por ponteiros. Para computar C', L_c é percorrida até que o custo associado ao elemento corrente seja maior que o valor limite dado por $\underline{c} + \alpha(\overline{c} - \underline{c})$. Os elementos percorridos (exceto o último elemento visitado) irão formar a lista C'.

Um ponteiro $P_{i,j,k}$ endereça cada elemento $\{c_{i,j,k}; (i, j, k)\}$ da lista L_c. Os índices i, j e k, que ainda aparecem em elementos de C, são colocados em três outras listas duplamente encadeadas L_i, L_j e L_k, respectivamente.

Para atualizar C após a escolha de uma tripla (i_p, j_p, k_p) (linhas 10 a 13 no pseudo-código da Figura 4.1), primeiramente remove-se i_p de L_i em $O(n)$, pois L_i possui no máximo n elementos. Em seguida, percorre-se L_j, também em complexidade $O(n)$. Para cada elemento $j \in L_j$, a lista L_k é percorrida

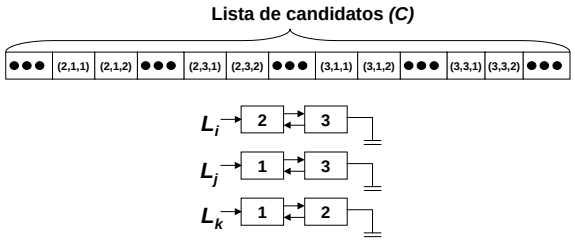

Figura 4.2: Valores da lista de candidatos C e das listas L_i, L_j e L_k após a primeira iteração da fase construtiva para um problema de tamanho $n = 3$. A solução parcial é formada pela tripla $(1, 2, 3)$.

em $O(n)$. Dessa forma, todas as triplas $(i_p, j, k) \in C$ são identificadas em $O(n + n^2)$. Para cada tripla $(i_p, j, k) \in C$, remove-se de L_c o elemento endereçado por $P_{i_p,j,k}$ em $O(1)$. Como no máximo n^2 triplas (i_p, j, k) são removidas a cada iteração, a remoção de todas as triplas $(i_p, j, k) \in C$ é feita em $O(n^2)$. Portanto, a atualização de C, através da identificação e remoção de todas as triplas (i_p, j, k) é feita em $O(n^2)$. Em seguida, j_p é removido de L_j e L_i é percorrida. Para cada elemento $i \in L_i$, a lista L_k é percorrida. Dessa forma, todas as triplas $(i, j_p, k) \in C$ restantes são identificadas. Para cada tripla $(i, j_p, k) \in C$, o elemento endereçado por $P_{i,j_p,k}$ é removido de L_c. Analogamente à remoção das triplas $(i_p, j, k) \in C$, a complexidade para atualizar C removendo-se todas as triplas (i, j_p, k) é $O(n^2)$. Finalmente, k_p é removido de L_k e L_i é percorrida. Para cada elemento $i \in L_i$, a lista L_j é percorrida. Dessa forma, todas as triplas $(i, j, k_p) \in C$ restantes são identificadas. Para cada tripla $(i, j, k_p) \in C$, o elemento endereçado por P_{i,j,k_p} é removido de L_c. Analogamente à remoção das triplas (i_p, j, k) e $(i, j_p, k) \in C$, a complexidade para atualizar C removendo-se todas as triplas $(i, j, k_p) \in C$ é $O(n^2)$. Como a atualização de C é feita em $n - 1$ iterações, a complexidade dessa etapa do algoritmo é $O(n^3)$.

Após a seleção de $n - 1$ triplas, o conjunto C de triplas candidatas contém uma única tripla que é adicionada a S na linha 15, completando assim, a fase construtiva.

Para melhor mostrar o funcionamento desse algoritmo, será considerado um problema de tamanho $n = 3$. Suponha que a tripla $(1, 2, 3)$ tenha sido escolhida para fazer parte da solução durante a primeira iteração do algoritmo. A Figura 4.2 mostra esquematicamente os valores de C e das listas L_i, L_j e L_k ao final da primeira iteração. Elementos (i, j, k) onde $i = 1$, $j = 2$ ou $k = 3$ não fazem parte da lista de candidatos C, tendo sido retirados durante a primeira iteração. Além disso, os índices I, J e K válidos são dados respectivamente por $L_i = \{2, 3\}$, $L_j = \{1, 3\}$ e $L_k = \{1, 2\}$.

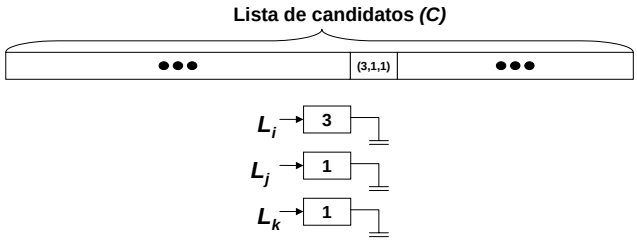

Figura 4.3: Valores da lista de candidatos C e das listas L_i, L_j e L_k após a segunda iteração da fase construtiva para um problema de tamanho $n = 3$. A solução parcial é formada pelas triplas $(1, 2, 3)$ e $(2, 3, 2)$.

Suponha que a tripla $(2, 3, 2)$ tenha sido escolhida para fazer parte da solução durante a segunda iteração do algoritmo. Primeiramente, o elemento de L_i correspondente ao índice $i = 2$ deve ser removido. Para isso, a lista L_i é percorrida até que o elemento seja encontrado, em $O(n)$. Em seguida, para cada elemento em L_j, percorre-se L_k. Dessa forma, as triplas $(2, 1, 1)$, $(2, 1, 2)$, $(2, 3, 1)$ e $(2, 3, 2)$ são identificadas em $O(n+n^2)$ e removidas de C. Como existem ponteiros para cada elemento em C, cada remoção é feita em $O(1)$. Como no máximo n^2 elementos podem ser removidos a cada iteração, a remoção desses elementos é feita em $O(n^2)$.

Em seguida, o elemento de L_j correspondente ao índice $j = 3$ é removido em $O(n)$. Além disso, para cada elemento de L_i, percorre-se L_k. As triplas $(3, 3, 1)$ e $(3, 3, 2)$ são identificadas em $O(n+n^2)$ e cada uma delas é retirada de C em $O(1)$.

Finalmente, retira-se de L_k o elemento correspondente ao índice de valor $k = 2$ em $O(n)$. Para cada elemento ainda em L_i, percorre-se L_j. A tripla $(3, 1, 2)$ é identificada em $O(n + n^2)$ e retirada de C em $O(1)$.

Observa-se que a iteração é calculada em $O(n^2)$. Como são feitas $n-1$ iterações, o algoritmo é computado em $O(n^3)$.

4.2.2 Fase de busca local

No procedimento de busca local, a solução corrente é melhorada quando é encontrada uma solução na sua vizinhança que possua um custo mais baixo do que o seu. A solução corrente é então atualizada e a busca é reiniciada partindo da nova solução. A definição da vizinhança $N(s)$ é crucial para o desempenho da busca local.

As soluções do AP3 podem ser representadas por um par de permutações (p, q). Portanto, o espaço de solução consiste de todas as $(n!)^2$ combinações de permutações possíveis.

```
procedimento LOCAL(n, c, c_S, S)
1     para p = 1, ..., n − 1 faça
2         para q = p + 1, ..., n faça
3             c_j = c_S − c_{i_p,j_p,k_p} − c_{i_q,j_q,k_q} + c_{i_p,j_q,k_p} + c_{i_q,j_p,k_q};
4             se (c_j < c_S) então
5                 S̄ = S \ {(i_p, j_p, k_p)} \ {(i_q, j_q, k_q)};
6                 S = S̄ ∪ {(i_p, j_q, k_p)} ∪ {(i_q, j_p, k_q)};
7                 LOCAL(n, c, c_j, S);
8                 retorne;
9             fim-se;
10            c_k = c_S − c_{i_p,j_p,k_p} − c_{i_q,j_q,k_q} + c_{i_p,j_p,k_q} + c_{i_q,j_q,k_p};
11            se (c_k < c_S) então
12                S̄ = S \ {(i_p, j_p, k_p)} \ {(i_q, j_q, k_q)};
13                S = S̄ ∪ {(i_p, j_p, k_q)} ∪ {(i_q, j_q, k_p)};
14                LOCAL(n, c, c_k, S);
15                retorne;
16            fim-se;
17        fim-para;
18    fim-para;
19    retorne;
fim LOCAL
```

Figura 4.4: Fase de busca local para o AP3.

Define-se a diferença entre duas permutações s e s' como sendo

$$\delta(s, s') = \{i \mid s(i) \neq s'(i)\},$$

onde $s(i)$ é o elemento na posição i da permutação s. A distância entre as permutações s e s' é definida como

$$d(s, s') = |\delta(s, s')|.$$

Na busca local para o AP3 uma vizinhança de trocas-(2,2) é usada. Uma vizinhança N_2 de trocas-(2,2) é definida como

$$N_2(s) = \{s' \mid d(s, s') = 2\}.$$

Em geral, uma vizinhança de trocas-(k, k) poderia ser usada. Uma forma natural de definir a vizinhança de uma solução (p, q) usando-se movimentos de trocas-(2,2) seria permitir que ela fosse gerada por todas as combinações de permutações produzidas por movimentos de troca-(2,2) tanto em p, como em q. Entretanto, o tamanho dessa vizinhança é $\binom{n}{2}^2$, o que é muito grande, mesmo para valores pequenos de n. Um esquema diferente é então proposto, onde a vizinhança de uma solução (p, q) é formada por todas as permutações geradas por trocas-(2,2) em p, mais todas as permutações

geradas por trocas-(2,2) em q. Isso significa que para uma solução $p, q \in \pi_N$, a vizinhaça de trocas-(2,2) é

$$N_2(p,q) = \{p', q' \mid d(p,p') + d(q,q') = 2\}.$$

Portanto, o tamanho da vizinhança é $|N_2(p)| + |N_2(q)| = 2\binom{n}{2}$.

A Figura 4.4 ilustra o procedimento de busca local usando a representação de triplas. A solução S, produzida na fase construtiva, é usada como ponto inicial para a busca local. O custo de S, dado por c_S, é fornecido como dado de entrada para o algoritmo. Nas linhas 1 a 18 as soluções vizinhas são analisadas. O custo de uma solução vizinha produzida a partir da permutação p é calculado na linha 3. Caso o seu custo seja menor do que o custo da solução corrente, a solução corrente é substituída por essa nova solução nas linhas 5 e 6. Caso uma substituição seja feita, o procedimento de busca local é chamado recursivamente, sendo iniciado pela nova solução corrente na linha 7. Da mesma forma, o custo de uma solução vizinha produzida a partir da permutação q é calculado na linha 10. Caso o seu custo seja melhor do que o custo da solução corrente, a solução corrente é substituída por essa nova solução nas linhas 12 e 13. Caso uma substituição seja feita, o procedimento de busca local é chamado recursivamente, sendo iniciado por essa nova solução na linha 14. O procedimento termina quando uma solução localmente ótima em relação à vizinhança definida é encontrada.

4.3 Religamento de caminhos

A técnica de religamento de caminhos consiste em explorar trajetórias que partem de uma *solução inicial*, gerando um caminho na vizinhança dessa solução na direção de outra solução, chamada de *solução guia*. Esse caminho é gerado selecionando-se movimentos que introduzem atributos da solução guia na solução inicial. A cada passo, todos os movimentos que incorporam atributos da solução guia são analisados e o melhor movimento é escolhido. O religamento de caminhos no contexto da metaheurística GRASP foi originalmente introduzido por Laguna e Martí [73]. Outras aplicações são descritas em [7, 27, 113, 117].

Para o problema de atribuição de três índices, o religamento de caminhos é feito entre uma solução inicial $S = \{(1, j_1^S, k_1^S), (2, j_2^S, k_2^S), \ldots, (n, j_n^S, k_n^S)\}$ e uma solução guia $T = \{(1, j_1^T, k_1^T), (2, j_2^T, k_2^T), \ldots, (n, j_n^T, k_n^T)\}$. Esse procedimento de religamento

de caminhos é mostrado em pseudo-código na Figura 4.5. Seja a diferença entre as soluções S e T definida da seguinte maneira:

$$\delta J = \{i = 1, \ldots, n \mid j_i^S \neq j_i^T\}$$

e

$$\delta K = \{i = 1, \ldots, n \mid k_i^S \neq k_i^T\}.$$

Esses conjuntos são computados nas linhas 4 e 5 do pseudo-código.

Uma solução intermediária da trajetória é visitada a cada passo do laço das linhas 6 a 34. Dois tipos de movimentos elementares podem ser feitos. Em um movimento do tipo 1, triplas

$$\{(i_1, j_1, k_1), (i_2, j_2, k_2)\}$$

são substituídas por triplas

$$\{(i_1, j_2, k_1), (i_2, j_1, k_2)\},$$

enquanto que em um movimento do tipo 2, triplas

$$\{(i_1, j_1, k_1), (i_2, j_2, k_2)\}$$

são substituídas por triplas

$$\{(i_1, j_1, k_2), (i_2, j_2, k_1)\}.$$

Os conjuntos δJ e δK são usados para guiar os movimentos. Movimentos do tipo 1 são guiados pelo conjunto δJ, enquanto que movimentos do tipo 2 são guiados pelo conjunto δK.

O laço da linha 8 a 17 considera movimentos do tipo 1. Para todo $i \in \delta J$, seja q tal que $j_q^T = j_i^S$. Em um movimento do tipo 1, triplas

$$\{(i, j_i^S, k_i^S), (q, j_q^S, k_q^S)\}$$

são substituídas por

$$\{(i, j_q^S, k_i^S), (q, j_i^S, k_q^S)\}.$$

```
procedimento RELIGAMENTO(n, c, c_S, S, T)
1     Seja S = {(1, j_1^S, k_1^S), (2, j_2^S, k_2^S), ..., (n, j_n^S, k_n^S)};
2     Seja T = {(1, j_1^T, k_1^T), (2, j_2^T, k_2^T), ..., (n, j_n^T, k_n^T)});
3     c_gmin = c_S; S_gmin = S;
4     δJ = {i = 1, ..., n | j_i^S ≠ j_i^T};
5     δK = {i = 1, ..., n | k_i^S ≠ k_i^T};
6     para |δJ| + |δK| > 2 faça
7           c_min = ∞;
8           para i ∈ δJ faça
9                 Seja q tal que j_q^T == j_i^S;
10                S̄ = S \ {(i, j_i^S, k_i^S)} \ {(q, j_q^S, k_q^S)};
11                S̄ = S̄ ∪ {(i, j_q^S, k_i^S)} ∪ {(q, j_i^S, k_q^S)};
12                c̄ = c_S − c_{i,j_i^S,k_i^S} − c_{q,j_q^S,k_q^S} + c_{i,j_q^S,k_i^S} + c_{q,j_i^S,k_q^S};
13                se c̄ < c_min então
14                      c_min = c̄; S_min = S̄; i_min = i;
15                      flag = 0;
16                fim-se;
17          fim-para;
18          para i ∈ δK faça
19                Seja q tal que k_q^T == k_i^S;
20                S̄ = S \ {(i, j_i^S, k_i^S)} \ {(q, j_q^S, k_q^S)};
21                S̄ = S̄ ∪ {(i, j_i^S, k_q^S)} ∪ {(q, j_q^S, k_i^S)};
22                c̄ = c_S − c_{i,j_i^S,k_i^S} − c_{q,j_q^S,k_q^S} + c_{i,j_i^S,k_q^S} + c_{q,j_q^S,k_i^S};
23                se c̄ < c_min então
24                      c_min = c̄; S_min = S̄; i_min = i;
25                      flag = 1;
26                fim-se;
27          fim-para;
28          S = S_min; c_S = c_min;
29          se flag == 0 então δJ = δJ \ {i_min};
30          se flag == 1 então δK = δK \ {i_min};
31          se c_S < c_gmin então
32                c_gmin = c_S; S_gmin = S;
33          fim-se;
34    fim-para;
35    retorne (S_gmin);
fim RELIGAMENTO
```

Figura 4.5: Religamento de caminhos entre a solução inicial S e a solução guia T.

Analogamente, o laço da linha 18 a 27 considera movimentos do tipo 2. Para cada $i \in \delta K$, seja q tal que $k_q^T = k_i^S$. Em um movimento do tipo 2, triplas

$$\{(i, j_i^S, k_i^S), (q, j_q^S, k_q^S)\}$$

são substituídas por

$$\{(i, j_i^S, k_q^S), (q, j_q^S, k_i^S)\}.$$

A cada passo, o movimento que produz a solução de menor custo é selecionado e o índice correspondente é removido de um dos conjuntos δJ ou δK (linhas 29 e 30). Esse processo continua até que existam apenas dois índices de movimento em um dos conjuntos δJ ou δK. Nesse ponto, o movimento produzirá a *solução guia* e, por esse motivo, não é realizado. A melhor solução encontrada no caminho (S_{gmin}) é retornada pelo procedimento.

A abordagem híbrida proposta nesse capítulo é semelhante à estratégia proposta por Laguna e Martí [73]. Laguna e Martí mantêm um *pool* com três soluções de elite, consistindo das três melhores soluções produzidas até o momento. Na implementação proposta nesse capítulo, um *pool* P de soluções de elite é formado pelas soluções encontradas nas primeiras $|P|$ iterações do algoritmo.

Após essa fase inicial, cada solução s_g produzida pela fase de busca local é religada com uma ou várias soluções de elite. Laguna e Martí selecionam uma solução de elite $s_e \in P$ e geram um caminho de s_g a s_e. Na implementação proposta nesse capítulo, dois caminhos são sempre gerados entre s_g e s_e, um indo de s_g para s_e e outro, de s_e para s_g. Isso é feito porque esses caminhos freqüentemente visitam soluções intermediárias diferentes. Duas estratégias para selecionar s_e são usadas. A primeira foi proposta por Laguna e Martí e consiste em selecionar aleatoriamente s_e de P. A segunda, religa s_g com todas as soluções de elite em P.

Laguna e Martí mantêm seu *pool* atualizado armazenando as três melhores soluções encontradas até o momento. No algoritmo GRASP com religamento de caminhos para o AP3 é usada uma abordagem desenvolvida por Fleurent e Glover [49] para incorporar soluções de elite em um algoritmo GRASP.

Sejam c_{melhor} e c_{pior} os valores das funções objetivo da melhor e da pior solução em P, respectivamente. Dadas duas soluções

$$S = \{(1, j_1^S, k_1^S), (2, j_2^S, k_2^S), \ldots, (n, j_n^S, k_n^S)\}$$

e

$$T = \{(1, j_1^T, k_1^T), (2, j_2^T, k_2^T), \ldots, (n, j_n^T, k_n^T)\}),$$

seja $\Delta(S,T) = |\delta J| + |\delta K|$ uma medida da diferença entre S e T.

A solução S_{gmin} produzida pelo religamento de caminhos é candidata à inserção no *pool* e será aceita caso satisfaça a um dos seguintes critérios:

1. $c_{gmin} < c_{melhor}$, isto é, S_{gmin} é a melhor solução encontrada até o momento;

2. $c_{melhor} \leq c_{gmin} < c_{pior}$ e para todas as soluções de elite $S_p \in P$, $\Delta(S_{gmin}, S_p) > \Delta_{\min}$, onde $\Delta_{\min}$ é um fator de diferenciação para inserção no *pool*, fornecido como entrada.

Uma vez aceita para inserção em P, S_{gmin} irá substituir a pior solução de elite, que será eliminada de P.

O religamento de caminhos também pode ser usado como uma fase de intensificação no conjunto de soluções de elite. Isso é feito aplicando-se o religamento de caminhos (nos dois sentidos) a cada par de soluções de elite em P e atualizando P, caso seja necessário. O procedimento é repetido até que não ocorram mais mudanças em P. Esse procedimento de intensificação pode ser usado em uma fase de pós-otimização (usando o *pool* de soluções do final do algoritmo GRASP), ou durante a otimização (usando o conjunto corrente de soluções de elite).

A etapa de pós-otimização consiste em aplicar o procedimento de intensificação no *pool* de soluções resultante ao final do algoritmo. Em seguida, aplica-se o procedimento de busca local explicado na Subseção 4.2.2 a cada solução de elite, visto que as soluções produzidas pelo religamento de caminhos não são necessariamente mínimos locais. Os mínimos locais produzidos dessa forma são candidatos a inserção no *pool*. Caso ocorra alguma mudança no *pool*, o procedimento de pós-otimização é repetido desde o início.

4.4 GRASP com religamento de caminhos

Nessa seção será mostrado como os procedimentos descritos anteriormente são inseridos no algoritmo GRASP, conforme o pseudo-código apresentado na Figura 4.6.

O algoritmo usa dois critérios de parada. Ele é interrompido após *maxitr* iterações terem sido executadas ou caso uma solução com valor

```
procedimento GRASP_RC_3AP(seed, n, c, alvo, maxitr, maxpool,ifreq)
1     P = ∅;
2     para i = 1, ..., maxitr faça
3           CONSTRUTIVA(seed, n, c, S);
4           LOCAL(n, c, c_S, S);
5           se |P| == maxpool então
6                 selecione P' ⊆ P;
7                 para T ∈ P' faça
8                       S_gmin = RELIGAMENTO(n, c, c_S, S, T);
9                       ATUALIZAR_POOL(S_gmin, c_gmin, P);
10                      S_gmin = RELIGAMENTO(n, c, c_T, T, S);
11                      ATUALIZAR_POOL(S_gmin, c_gmin, P);
12                fim-para;
13          senão P = P ∪ {S} fim-se;
14          se  mod (i,ifreq) == 0 então
15                INTENSIFICAÇÃO(P);
16          fim-se;
17          S_melhor = POOLMIN(P);
18          se c_melhor ≤ alvo então
19                Saia do laço;
20          fim-se;
21    fim-para;
22    POS-OTIMIZAÇÃO(P);
23    S_melhor = POOLMIN(P);
24    retorne (S_melhor);
fim GRASP_RC_3AP;
```

Figura 4.6: Heurística GRASP com religamento de caminhos para o AP3.

objetivo menor ou igual a *alvo* tenha sido encontrada. Cada iteração consiste na construção de uma solução (linha 3), em uma busca local usando a solução construída como solução inicial (linha 4), e uma vez que o *pool* de soluções de elite esteja completo (ou seja, $|P| = maxpool$), em uma fase de religamento de caminhos. Na fase de religamento de caminhos, são examinadas as soluções no caminho entre a solução produzida pela busca local e um subconjunto de soluções de elite (P'), que poderá ser formado por apenas uma solução de P (escolhida aleatoriamente) ou por todas as soluções de P. O religamento de caminhos é feito tanto iniciando-se da solução produzida pela busca local em direção de uma solução de P', como iniciando-se de uma solução de P' em direção à solução produzida pela busca local (linhas 8 e 10). Na abordagem proposta nesse capítulo, P' pode ser formado por uma única solução selecionada aleatoriamente de P, ou pelo *pool* inteiro. Após cada trajetória de religamento de caminhos ter sido percorrida, a melhor solução visitada é testada para inserção em P (linhas 9 e 11). Caso o *pool* não esteja completo, a solução produzida pela busca local é simplesmente adicionada a ele.

A fase de intensificação no conjunto de elite, descrita na Seção 4.3, é

feita a cada intervalo de *ifreq* iterações (linhas 14 a 16). Finalmente, quando um dos critérios de parada é satisfeito, a fase de pós otimização descrita na Seção 4.3 é computada (linha 22).

4.5 Resultados computacionais

Nessa seção serão apresentados os resultados computacionais dos métodos GRASP descritos nesse capítulo. O ambiente computacional usado para conduzir os experimentos será descrito, assim como os problemas testes selecionados para testar as nove variantes propostas. Serão apresentados os resultados da comparação entre estas variantes. Esses resultados atestam que o método estudado pode produzir soluções próximas do ótimo para problemas testes do AP3.

4.5.1 Ambiente computacional

Os experimentos foram feitos em um computador SGI Challenge com 16 processadores MIPS R10000 de 196 MHz, 12 processadores MIPS R10000 de 194 MHz e 7.6 Gb de memória.

Os programas foram escritos em Fortran e foram compilados com o compilador SGI MIPSpro F77 usando as opções `-O3 -static -u`. Os tempos de CPU foram medidos com a função `etime` do sistema operacional. Foi usado o gerador de números pseudo-aleatórios descrito em [124].

4.5.2 Problemas testes

Três classes de problemas testes tiradas da literatura foram usadas nos experimentos.

A primeira classe foi proposta por Balas e Saltzman [13]. Coeficientes inteiros de custo $c_{i,j,k}$ são gerados uniformemente no intervalo $[0, 100]$ para esses problemas. Os experimentos foram limitados a todos os problemas testes de tamanhos $n = 12, 14, 16, 18, 20, 22, 24$ e 26, gerados por Balas e Saltzman. Para cada tamanho, foram considerados todos os cinco problemas testes fornecidos por Balas e Saltzman.

A segunda classe é a classe $T\Delta$, proposta por Crama e Spieksma [37]. O AP3 é visto como um problema de otimização em um grafo

tripartido completo $K_{n,n,n}$, como explicado na Seção 4.1. Um comprimento $d_{u,v} \geq 0$ é atribuído a cada aresta de $K_{n,n,n}$ e o custo $c_{i,j,k}$ é dado por $c_{i,j,k} = d_{i,j} + d_{i,k} + d_{j,k}$. Três tipos de problemas gerados aleatoriamente são considerados. Eles diferem em como os comprimentos $d_{u,v}$ são calculados (veja [37] para maiores detalhes). Cada tipo consiste de três problemas testes de tamanho $n = 33$ e de três problemas testes de tamanho $n = 66$, num total de 18 problemas testes.

A última classe de problemas foi proposta por Burkard, Rudolf e Woeginger [25]. Sejam α_i, β_j e γ_k os elementos de três seqüências de n elementos. Um coeficiente de custo $c_{i,j,k}$ é dado por $c_{i,j,k} = \alpha_i \cdot \beta_j \cdot \gamma_k$. Os problemas testes considerados nos experimentos são os maiores testados por Burkard, Rudolf e Woeginger [25], de tamanhos $n = 12, 14$ e 16. Todos possuem coeficientes de custo inteiros, distribuídos uniformemente no intervalo $[0, 10]$. Para cada tamanho de problema, todos os 100 casos testes fornecidos por Burkard, Rudolf e Woeginger foram considerados.

4.5.3 Variantes do algoritmo GRASP

São consideradas as nove variantes da estratégia GRASP com religamento de caminhos propostas nesse capítulo:

1. `GRASP`: é um algoritmo GRASP puro, sem religamento de caminhos.

2. `GRC(RAND)`: acrescenta a `GRASP` o religamento de caminhos entre a solução produzida pelo algoritmo básico e uma solução selecionada aleatoriamente do conjunto de soluções de elite. O religamento de caminhos é realizado tanto iniciando-se da solução produzida pela busca local em direção à solução de elite, como iniciando-se da solução de elite em direção à solução produzida pela busca local.

3. `GRC(ALL)`: acrescenta a `GRASP` o religamento de caminhos entre a solução produzida pelo algoritmo básico e todas as $|P|$ soluções do conjunto de soluções de elite P. O religamento de caminhos é realizado tanto iniciando-se da solução produzida pela busca local em direção a cada solução de elite, como iniciando-se de cada solução de elite em direção à solução produzida pela busca local.

4. `GRC(RAND,INT)`: acrescenta a `GRC(RAND)` um esquema de intensificação que é repetido periodicamente após um intervalo fixo de iterações.

5. `GRC(ALL,INT)`: acrescenta a `GRC(ALL)` um esquema de intensificação que é repetido periodicamente após um intervalo fixo de iterações.

6. `GRC(RAND,POST)`: acrescenta a `GRC(RAND)` uma fase de pós-otimização onde o religamento de caminhos é feito entre todas as soluções do conjunto de elite e as soluções resultantes são otimizadas localmente.

7. `GRC(ALL,POST)`: acrescenta a `GRC(ALL)` uma fase de pós-otimização onde o religamento de caminhos é feito entre todas as soluções do conjunto de elite e as soluções resultantes são otimizadas localmente.

8. `GRC(RAND,POST,INT)`: acrescenta a `GRC(RAND,POST)` um esquema de intensificação que é repetido periodicamente após um intervalo fixo de iterações.

9. `GRC(ALL,POST,INT)`: acrescenta a `GRC(ALL,POST)` um esquema de intensificação que é repetido periodicamente após um intervalo fixo de iterações.

Em todas as variantes, o parâmetro α da LRC é selecionado aleatoriamente de uma distribuição uniforme no intervalo $[0, 1]$ e permanece fixo a cada iteração. Fixou-se o tamanho do *pool* como $|P|$=30. O valor do fator de diferenciação para inserção no *pool* $\Delta_{\min}$ foi fixado com sendo n, ou seja, metade do número de elementos dos vetores de permutação candidatos a inserção no *pool*. As variantes `GRC(RAND)`, `GRC(RAND,INT)`, `GRC(RAND,POST,INT)` e `GRC(RAND,POST,INT)` usam duas seqüências do gerador de números aleatórios. A primeira é usada na fase construtiva para selecionar o parâmetro α da LRC e para selecionar o próximo elemento a ser inserido na solução. A outra seqüência é usada para fazer a seleção aleatória de uma solução do *pool*. Dessa forma, garante-se que para todas as variantes, as etapas acrescentadas nunca irão deteriorar o resultado que seria obtido por uma estratégia GRASP pura iniciada com a mesma semente do gerador de números aleatórios.

4.5.4 Os experimentos

O objetivo dos experimentos desse capítulo é avaliar a eficiência do religamento de caminhos usado como uma etapa do algoritmo GRASP. Espera-se responder três perguntas:

Tabela 4.1: Estimativas das probabilidades de encontrar uma solução com custo melhor ou igual ao valor alvo, em função do tempo de solução. Os problemas testes considerados são B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman, para os valores alvos 16, 16, 16 e 17, respectivamente. Os algoritmos testados são GRC(RAND) e GRASP.

	B-S 20.1		B-S 22.1		B-S 24.1		B-S 26.1	
tempo	GRC	GRASP	GRC	GRASP	GRC	GRASP	GRC	GRASP
50s	.89	.44	.52	.25	.45	.22	.35	.16
100s	.98	.68	.83	.43	.70	.32	.66	.25
150s	1.00	.80	.93	.56	.84	.42	.85	.37
200s	1.00	.87	.98	.66	.91	.49	.96	.50

1. O religamento de caminhos melhora o desempenho do algoritmo GRASP e como são os seus tempos de processamento quando comparados aos do algoritmo puro?

2. Como são comparados os tempo das diferentes variantes do método GRASP com religamento de caminhos descritas nesse capítulo, fixando-se a qualidade das soluções obtidas?

3. A variável aleatória *tempo para valor alvo* das diferentes variantes do algoritmo GRASP com religamento de caminhos é exponencialmente distribuída?

Para estudar o efeito do religamento de caminhos, o algoritmo GRASP puro (`GRASP`) e o método com religamento de caminhos mais simples (`GRC(RAND)`) foram comparados para os problemas `B-S 20.1`, `B-S 22.1`, `B-S 24.1` e `B-S 26.1` de Balas e Saltzman [13]. Duzentas execuções independentes de cada uma dessas variantes foram feitas para os quatro problemas. Cada execução foi terminada quando uma solução com custo igual ou inferior a um valor `alvo` foi encontrado. Os valores de `alvo` testados para os problemas `B-S 20.1`,`B-S 22.1`, `B-S 24.1` e `B-S 26.1` foram 16, 16, 16 e 17, respectivamente. Esses valores estão longe do valor ótimo e, em geral, podem ser obtidos em poucas iterações. Distribuições empíricas de probabilidade do tempo para valor alvo são plotadas na Figura 4.7. Para plotar essas distribuições empíricas, os tempos medidos são primeiramente ordenados. Em seguida, associa-se ao i-ésimo tempo de execução (t_i) uma probabilidade $p_i = (i - \frac{1}{2})/200$. Finalmente, os pontos $z_i = (t_i, p_i)$ são plotados para $i = 1, \ldots, 200$. Os gráficos mostram claramente que o religamento de caminhos permite reduzir o tempo necessário para encontrar uma solução com valor melhor ou igual ao alvo. Por exemplo, para o problema 20.1 a probabilidade de se encontrar uma solução com valor pelo menos tão bom quanto o valor alvo em no máximo 50 segundos é 89% para

GRC(RAND), enquanto que para GRASP é 44%. Considerando-se um tempo de execução de no máximo 100 segundos, estas probabilidades são 98% e 68% para GRC(RAND) e GRASP, respectivamente. Essas probabilidades são mostradas na Tabela 4.1 para os quatro problemas testes considerados. Esses resultados mostram que, apesar de GRC(RAND) precisar de mais tempo por iteração, esse tempo é compensado pelo número reduzido de iterações necessárias para encontrar a solução.

Da mesma forma que os gráficos da Figura 4.7, usados para comparar GRASP e GRC(RAND), os gráficos das Figuras 4.8 e 4.9 mostram as distribuições empíricas de probabilidade do tempo para valor alvo das variantes GRC(RAND), GRC(RAND,INT), GRC(ALL) e GRC(ALL,INT). São considerados os mesmos problemas testes B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1, com valores alvos mais difíceis de obter: 7, 8, 7 e 8, respectivamente. As variantes GRC(RAND,INT) e GRC(ALL,INT) fazem a intensificação a cada intervalo de *ifreq*= 2000 iterações. Por exemplo, a Figura 4.8 mostra que para o problema 20.1, a probabilidade de GRC(RAND) encontrar uma solução de valor no máximo igual a 7 em menos de 2063s é 0.5, enquanto que, com a mesma probabilidade, GRC(RAND,INT), GRC(ALL) e GRC(ALL,INT) encontram uma solução com valor no máximo 7 em menos de 1744s, 851s e 718s, respectivamente. Na Tabela 4.2 são mostrados os tempos em função da probabilidade para os quatro problemas testes, considerando-se as quatro variantes do algoritmo GRASP com religamento de caminhos. Os gráficos e a tabela mostram que GRC(ALL) e GRC(ALL,INT) encontram uma solução com custo pelo menos tão bom quanto o valor alvo em menos tempo do que GRC(RAND) e GRC(RAND,INT), apesar do fato de que GRC(RAND) e GRC(RAND,INT) precisam de significativamente menos tempo por iteração do que GRC(ALL) e GRC(ALL,INT). Deve-se notar também que os resultados mostram que a intensificação parece beneficiar mais GRC(RAND,INT) quando comparado a GRC(RAND), do que GRC(ALL,INT) quando comparado a GRC(ALL), e que à medida que o tamanho do problema cresce, o benefício da intensificação diminui.

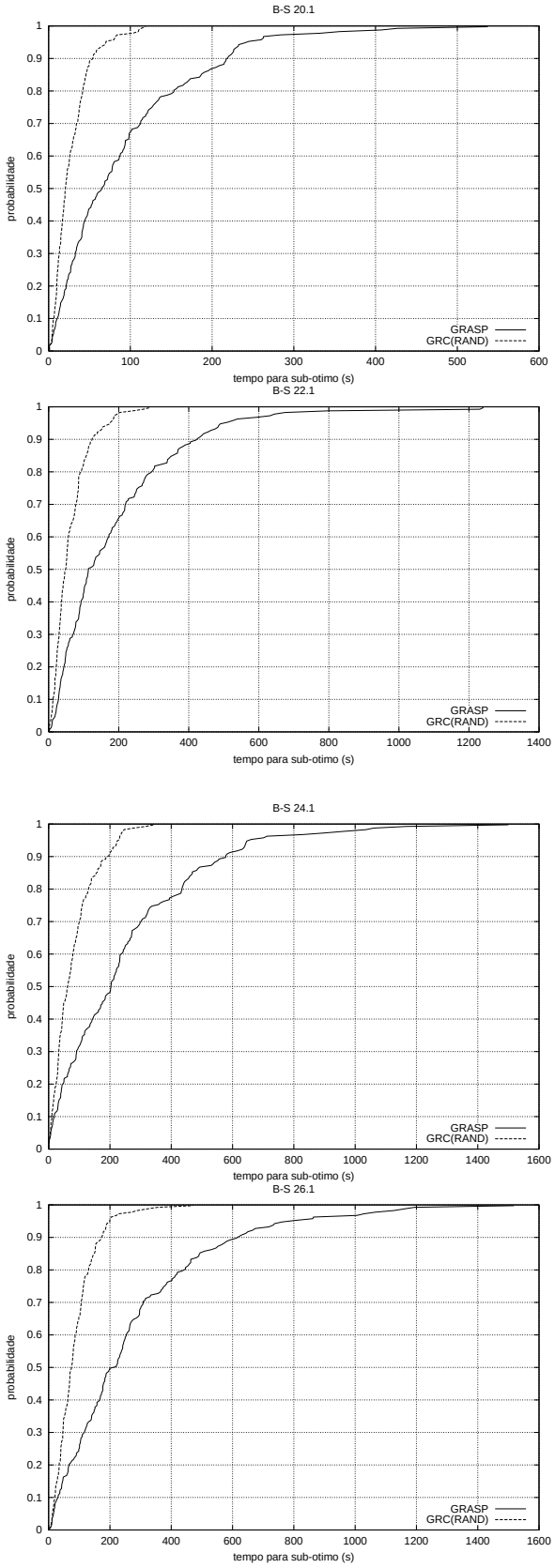

Figura 4.7: Distribuição empírica de probabilidade do tempo para valor alvo das estratégias GRASP e GRC(RAND) (problemas testes B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman). Os problemas considerados foram B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1, para os valores alvos 16, 16, 16 e 17, respectivamente.

Tabela 4.2: Tempo para encontrar uma solução com custo melhor ou igual ao valor alvo, em função da probabilidade. Os problemas testes considerados são B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman, para os valores alvos 7, 8, 7 e 8, respectivamente.

	B-S 20.1			
probabilidade	GRC(ALL,INT)	GRC(ALL)	GRC(RAND,INT)	GRC(RAND)
0.2	238s	266s	338s	428s
0.5	718s	851s	1744s	2063s
0.8	1856s	1918s	4887s	5331s

	B-S 22.1			
probabilidade	GRC(ALL,INT)	GRC(ALL)	GRC(RAND,INT)	GRC(RAND)
0.2	544s	546s	710s	852s
0.5	1502s	1584s	2916s	2763s
0.8	4731s	4780s	8244s	8161s

	B-S 24.1			
probabilidade	GRC(ALL,INT)	GRC(ALL)	GRC(RAND,INT)	GRC(RAND)
0.2	742s	856s	1714s	1697s
0.5	2274s	2603s	5378s	4084s
0.8	6198s	6306s	12467s	11336s

	B-S 26.1			
probabilidade	GRC(ALL,INT)	GRC(ALL)	GRC(RAND,INT)	GRC(RAND)
0.2	594s	592s	852s	963s
0.5	1411s	1411s	2315s	2315s
0.8	3698s	3691s	5941s	6049s

Em um outro tipo de experimento, as diferentes variantes foram executadas por um número fixo de iterações, considerando-se todos os problemas testes descritos na Subseção 4.5.2. Dois tipos de execuções foram feitas. No primeiro tipo, os programas foram executados por 100 iterações (execuções curtas) e no segundo tipo, por 10000 iterações (execuções longas). Foram feitas cinco execuções independentes curtas e cinco execuções independentes longas para cada par problema teste/variante. As variantes `GRC(RAND,POST,INT)` e `GRC(ALL,POST,INT)` fazem a intensificação a cada intervalo de *ifreq*= 30 iterações nas execuções curtas e *ifreq*= 250 iterações nas execuções longas.

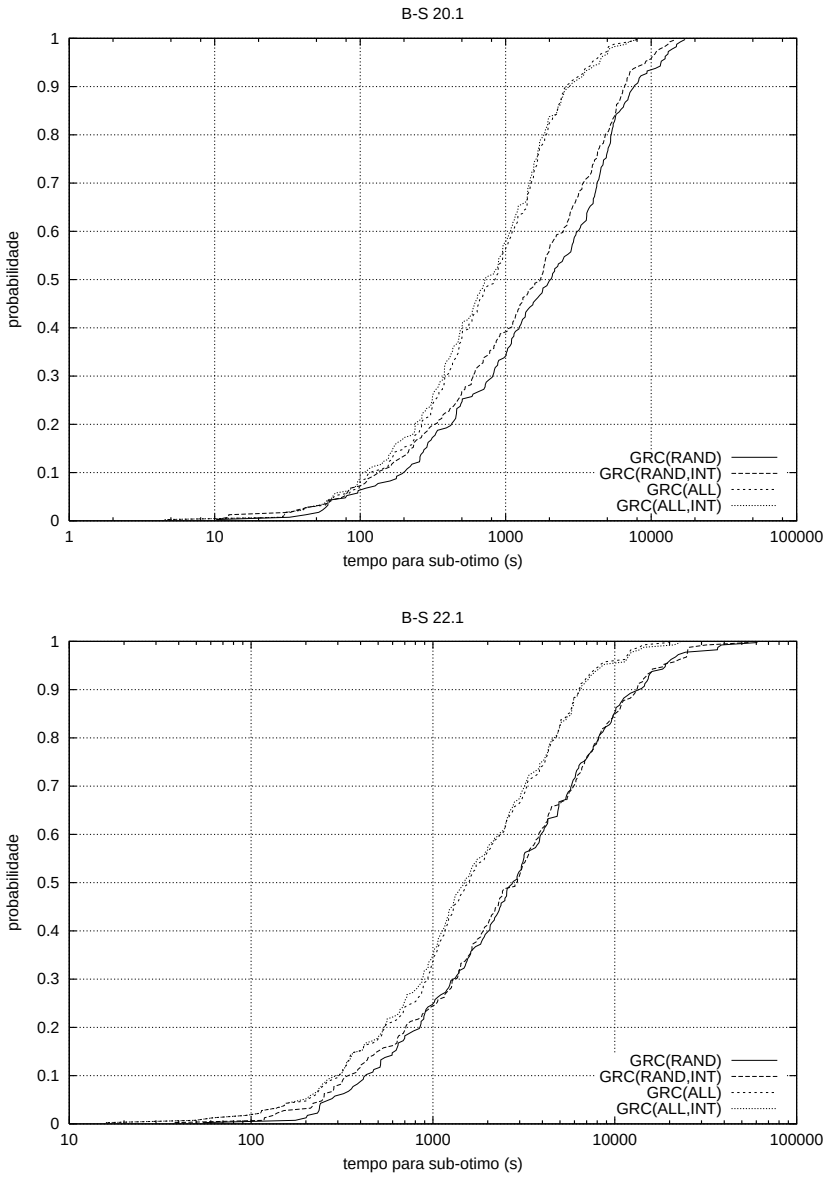

Figura 4.8: Distribuição empírica de probabilidade do tempo para valor alvo das diferentes variantes do algoritmo GRASP com religamento de caminhos (problemas testes B-S 20.1 e B-S 22.1 de Balas e Saltzman). Os problemas considerados foram B-S 20.1 e B-S 22.1, para os valores alvos 7 e 8, respectivamente.

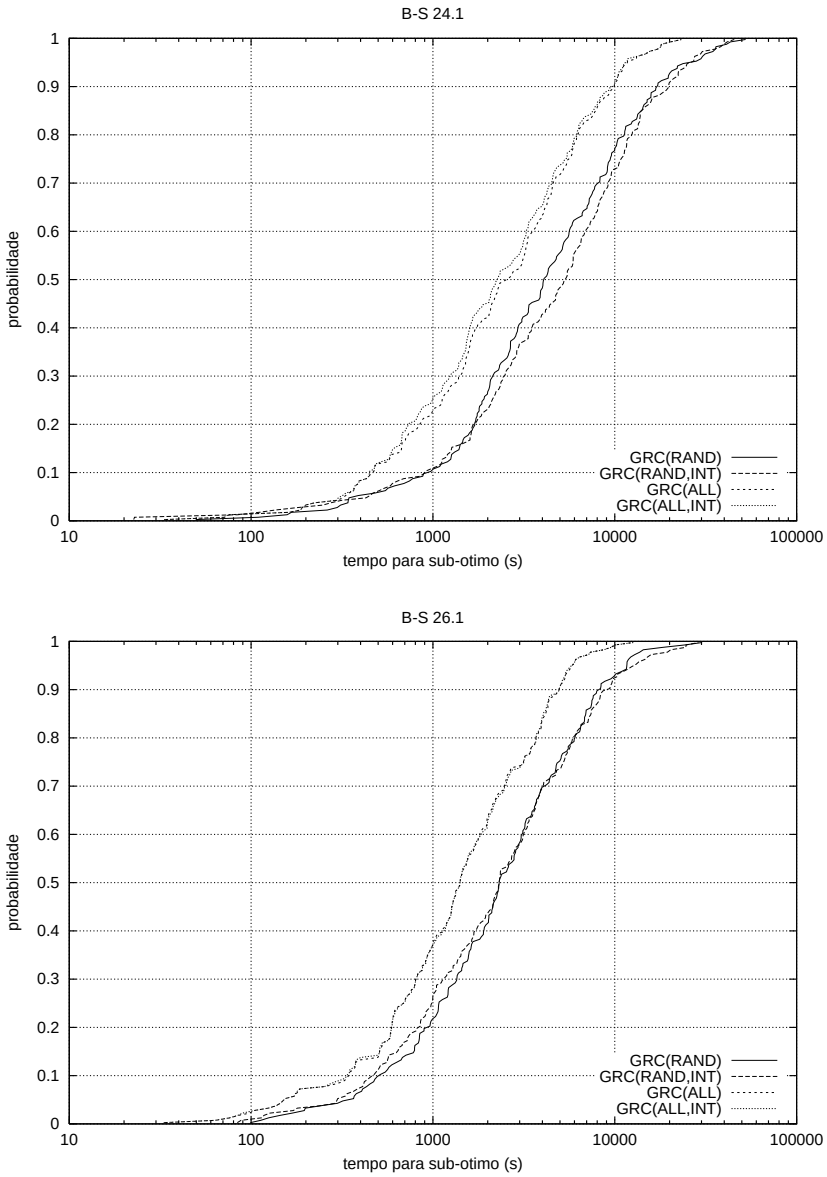

Figura 4.9: Distribuição empírica de probabilidade do tempo para valor alvo das diferentes variantes do algoritmo GRASP com religamento de caminhos (problemas testes B-S 24.1 e B-S 26.1 de Balas e Saltzman). Os problemas considerados foram B-S 24.1 e B-S 26.1, para os valores alvos 7 e 8, respectivamente.

As Tabelas 4.3 e 4.4 mostram os resultados para os problemas testes propostos por Balas e Saltzman [13], para 100 e 10000 iterações, respectivamente. Cada linha mostra as estatísticas tiradas sobre os cinco problemas testes de um mesmo tamanho. Para cada problema teste são feitas cinco execuções independentes. Por exemplo, para $n = 12$, os problemas testes são `B-S 12.1`, `B-S 12.2`, `B-S 12.3`, `B-S 12.4`, e `B-S 12.5` e um total de 25 execuções foram feitas. A coluna descrita por n é a dimensão do problema. A coluna descrita por B-S é a média dos custos obtidos pela heurística de troca em profundidade variável (*variable depth interchange*) proposta por Balas e Saltzman [13] e a coluna OPT lista a média dos ótimos publicados por Balas e Saltzman. Nessas tabelas, assim como nas demais tabelas mostradas a seguir, os resultados do algoritmo básico e das variantes que usam religamento de caminhos `GRC(RAND)`, `GRC(RAND,POST)`, `GRC(RAND,POST,INT)`, `GRC(ALL)`, `GRC(ALL,POST)` e `GRC(ALL,POST,INT)` são mostrados nas colunas GRASP, GRC(R), GRC(R,P), GRC(R,P,I), GRC(A), GRC(A,P), e GRC(A,P,I), respectivamente. Para cada tamanho de problema teste, as células em destaque em cada linha correspondem às variantes com as menores médias de soluções. Para cada dimensão de problema e variante do algoritmo, a célula correspondente nas Tabelas 4.3 e 4.4 lista a média dos custos (calculada sobre as melhores soluções encontradas nas cinco execuções independentes para cada problema teste), o número de vezes que a variante encontrou a melhor solução (mlh) para os cinco problemas testes daquela dimensão (por exemplo, `GRC(ALL,POST)` encontrou a melhor solução para dois dos cinco problemas testes de dimensão 12, enquanto `GRC(ALL,POST,INT)` encontrou a melhor solução para todos os cinco problemas testes de dimensão 12), e a média dos tempos de processamento para as 25 execuções da célula. Na última linha (total de mlh) dessas tabelas, é mostrada a soma do número de vezes em que cada variante encontrou a melhor solução.

São feitas as seguintes observações sobre os resultados nas Tabelas 4.3 e 4.4:

- Nas execuções curtas, o algoritmo GRASP com religamento de caminhos encontra soluções melhores do que a heurística de troca em profundidade variável de Balas e Saltzman para $n \leq 20$, mas obtém soluções piores para $n \geq 22$. Nas execuções longas, porém, todas as variantes (inclusive o algoritmo GRASP puro) encontram soluções melhores do que as encontradas pela heurística de troca em profundidade variável.
- Soluções ótimas são encontradas nas execuções longas para $n \leq 14$.

- Todas as variantes com religamento de caminhos encontraram soluções que são em média, melhores do que as encontradas pelo algoritmo puro.
- Nas curtas e longas, a pós-otimização e a intensificação ajudam as variantes que fazem religamento de caminhos a obter melhores soluções.
- Nas execuções longas, soluções melhores são observadas movendo-se do lado esquerdo em direção ao lado direito da tabela.
- Nas execuções curtas, quando procedimentos de pós-otimização ou de pós-otimização e de intensificação são acrescentados a `GRC(RAND)`, melhores soluções são encontradas quando comparadas àquelas encontradas pela variante `GRC(ALL)`. Porém, a variante `GRC(ALL,POST)` encontra melhores soluções do que `GRC(RAND,POST)` e a variante `GRC(ALL,POST,INT)` encontra soluções melhores do que `GRC(RAND,POST,INT)`.

As Tabelas 4.5, 4.6 e 4.7 mostram os resultados computacionais para os tipos I, II e III, respectivamente, de problemas testes propostos por Crama e Spieksma [37]. Cada tabela mostra o número de iterações, a dimensão do problema n, o valor obtido pela melhor heurística H de Crama e Spieksma [37] e o limite inferior publicado em [37]. A heurística de Crama e Spieksma será chamada de C-S. Cada célula correspondente a um par problema teste/variante mostra o valor da melhor solução encontrada nas cinco execuções independentes, o número de vezes que a melhor solução foi encontrada e a média dos tempos das cinco execuções.

Para os problemas do tipo I, a Tabela 4.5 mostra o seguinte:

- Nas execuções curtas, as variantes com religamento de caminhos encontraram melhores soluções do que C-S para quase todos os problemas testes, com exceção de um. Em execuções longas, melhores soluções foram encontradas pelas variantes do método GRASP com religamento de caminhos quando comparadas àquelas encontradas por C-S para todos os problemas testes.
- Nas execuções curtas, o algoritmo GRASP puro obteve soluções melhores do que as obtidas por C-S para apenas um problema teste de tamanho $n = 33$, enquanto que para $n = 66$, C-S encontrou soluções melhores do que as encontradas pelo algoritmo puro para todos os problemas testes.

Tabela 4.3: Problemas testes de Balas e Saltzman (100 iterações).

			GRASP		GRC(R)		GRC(R,P)		GRC(R,P,I)		GRC(A)		GRC(A,P)		GRC(A,P,I)	
n	B-S	OPT	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh
12	24.0	15.6	31.4 (0.09s)	0	28.8 (0.11s)	0	20.0 (0.35s)	2	20.0 (0.86s)	3	21.2 (0.54s)	0	20.0 (0.71s)	2	18.0 (1.24s)	5
14	22.4	10.0	27.8 (0.16s)	0	25.4 (0.18s)	0	16.8 (0.54s)	1	16.2 (1.27s)	2	17.8 (0.78s)	0	16.4 (1.05s)	0	15.0 (1.81s)	4
16	25.0	10.0	25.0 (0.26s)	0	25.0 (0.29s)	0	18.8 (0.80s)	0	18.4 (1.72s)	3	21.2 (1.07s)	0	20.8 (1.44s)	1	18.4 (2.45s)	4
18	17.6	6.4	26.8 (0.40s)	0	26.8 (0.44s)	0	19.2 (1.04s)	3	18.6 (2.25s)	2	21.2 (1.43s)	0	17.2 (1.88s)	2	17.6 (3.19s)	3
20	27.4	4.8	27.0 (0.58s)	1	27.0 (0.63s)	1	18.6 (1.33s)	3	20.8 (2.80s)	2	21.4 (1.87s)	2	21.0 (2.39s)	2	18.8 (3.97s)	2
22	18.8	4.0	24.6 (0.80s)	1	24.6 (0.86s)	1	22.4 (1.63s)	2	20.8 (3.46s)	4	23.0 (2.37s)	3	23.0 (2.99s)	3	22.6 (4.85s)	4
24	14.0	1.8	31.2 (1.09s)	0	31.2 (1.16s)	0	25.2 (2.10s)	0	20.0 (4.20s)	2	23.4 (2.96s)	1	20.2 (3.64s)	2	16.8 (5.83s)	2
26	15.7	1.3	28.0 (1.44s)	1	28.0 (1.52s)	1	21.2 (2.69s)	2	22.4 (4.79s)	2	23.4 (3.64s)	2	22.4 (4.40s)	2	21.8 (6.71s)	3
total de mlh				3		3		13		20		8		14		27

Tabela 4.4: Problemas testes de Balas e Saltzman (10000 iterações).

			GRASP		GRC(R)		GRC(R,P)		GRC(R,P,I)		GRC(A)		GRC(A,P)		GRC(A,P,I)	
n	B-S	OPT	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh
12	24.0	15.6	16.4	3	15.6	5	15.6	5	15.6	5	15.6	5	15.6	5	15.6	5
			(8.37s)		(10.77s)		(10.85s)		(14.32s)		(71.63s)		(71.70s)		(74.79s)	
14	22.4	10.0	12.0	2	10.6	3	10.2	4	10.2	4	10.0	5	10.0	5	10.0	5
			(14.37s)		(17.71s)		(17.83s)		(22.67s)		(102.22s)		(102.30s)		(106.55s)	
16	25.0	10.0	13.2	0	12.8	1	12.8	1	11.0	2	10.6	2	10.6	2	10.2	4
			(23.51s)		(27.64s)		(27.85s)		(34.33s)		(138.31s)		(138.37s)		(143.89s)	
18	17.6	6.4	11.8	0	9.8	0	8.0	2	9.8	0	7.2	3	7.2	3	7.4	2
			(36.58s)		(42.04s)		(42.20s)		(50.39s)		(184.86s)		(183.70s)		(190.88s)	
20	27.4	4.8	14.2	0	9.8	1	9.4	1	10.2	1	7.2	3	6.6	4	6.4	5
			(53.40s)		(60.20s)		(60.53s)		(70.64s)		(237.02s)		(237.35s)		(246.70s)	
22	18.8	4.0	13.6	0	12.4	0	10.6	1	10.2	1	7.2	4	7.2	4	7.8	3
			(74.82s)		(83.18s)		(83.27s)		(95.63s)		(298.17s)		(298.40s)		(309.64s)	
24	14.0	1.8	13.4	0	9.8	1	9.8	1	8.4	3	7.4	5	7.4	5	7.4	5
			(101.66s)		(111.14s)		(111.98s)		(127.09s)		(368.45s)		(368.82s)		(382.45s)	
26	15.7	1.3	13.4	0	9.8	1	9.8	1	9.8	2	8.4	5	8.4	5	8.4	5
			(134.53s)		(145.60s)		(146.27s)		(164.15s)		(448.54s)		(449.06s)		(465.20s)	
total de mlh			5		12		16		18		32		33		34	

Tabela 4.5: Problemas testes do tipo I de Crama e Spieksma.

				GRASP		GRC(R)		GRC(R,P)		GRC(R,P,I)		GRC(A)		GRC(A,P)		GRC(A,P,I)	
Iterações	n	C-S	LB	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh
100	33	1618	1607	1617 (4.88s)	0	1613 (4.94s)	0	1608 (5.45s)	1	1608 (6.15s)	1	1608 (6.14s)	1	1608 (6.41s)	1	1608 (7.35s)	1
100	33	1411	1395	1419 (5.08s)	0	1405 (5.15s)	0	1401 (5.73s)	1	1401 (6.42s)	1	1401 (6.48s)	1	1401 (6.76s)	1	1401 (7.70s)	1
100	33	1609	1604	1616 (4.70s)	0	1609 (4.79s)	0	1604 (5.32s)	1	1604 (6.25s)	1	1604 (6.22s)	1	1604 (6.49s)	1	1604 (7.54s)	1
100	66	2668	2654	2767 (151.34s)	0	2753 (151.95s)	0	2687 (154.97s)	0	2682 (165.08s)	0	2679 (158.54s)	0	2678 (159.69s)	1	2681 (171.20s)	0
100	66	2469	2433	2515 (147.97s)	0	2490 (149.14s)	0	2461 (152.15s)	0	2452 (163.34s)	1	2454 (155.04s)	0	2454 (156.22s)	0	2452 (166.80s)	1
100	66	2775	2748	2822 (149.07s)	0	2788 (149.53s)	0	2766 (151.93s)	1	2778 (162.79s)	0	2783 (154.77s)	0	2779 (155.87s)	0	2774 (171.40s)	0
total de mlh					0		0		4		4		3		4		4

				GRASP		GRC(R)		GRC(R,P)		GRC(R,P,I)		GRC(A)		GRC(A,P)		GRC(A,P,I)	
Iterações	n	C-S	LB	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh
10000	33	1618	1607	1609 (459.35s)	0	1608 (467.50s)	1	1608 (466.48s)	1	1608 (477.70s)	1	1608 (654.70s)	1	1608 (652.26s)	1	1608 (660.49s)	1
10000	33	1411	1395	1401 (474.90s)	1	1401 (489.25s)	1	1401 (485.71s)	1	1401 (495.97s)	1	1401 (672.43s)	1	1401 (671.76s)	1	1401 (680.50s)	1
10000	33	1609	1604	1606 (443.60s)	0	1604 (460.63s)	1	1604 (459.33s)	1	1604 (465.86s)	1	1604 (648.24s)	1	1604 (673.43s)	1	1604 (676.07s)	1
10000	66	2668	2654	2714 (14432.23s)	0	2670 (14470.68s)	0	2670 (14380.15s)	0	2664 (14440.98s)	1	2664 (15352.03s)	1	2664 (15662.94s)	1	2664 (15470.11s)	1
10000	66	2469	2433	2484 (14157.01s)	0	2454 (14197.33s)	0	2454 (14220.41s)	0	2454 (14238.75s)	0	2449 (14986.12s)	1	2449 (15018.97s)	1	2449 (15010.90s)	1
10000	66	2775	2748	2801 (14196.91s)	0	2758 (14244.77s)	1	2758 (14265.57s)	1	2758 (14283.52s)	1	2759 (15023.19s)	0	2759 (15011.29s)	0	2759 (15084.57s)	0
total de mlh					1		4		4		5		5		5		5

Tabela 4.6: Problemas testes do tipo II de Crama e Spieksma.

				GRASP		GRC(R)		GRC(R,P)		GRC(R,P,I)		GRC(A)		GRC(A,P)		GRC(A,P,I)	
Iterações	n	C-S	LB	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh
100	33	4861	4772	4805	0	4805	0	4799	0	4799	0	4798	1	4798	1	4798	1
				(4.49s)		(4.59s)		(5.50s)		(7.26s)		(6.89s)		(7.50s)		(9.50s)	
100	33	5142	5035	5086	0	5081	0	5070	1	5072	0	5075	0	5071	0	5071	0
				(4.25s)		(4.37s)		(5.49s)		(7.54s)		(6.83s)		(7.62s)		(9.97s)	
100	33	4352	4260	4306	0	4301	0	4288	1	4290	0	4290	0	4290	0	4289	0
				(4.36s)		(4.42s)		(5.54s)		(7.58s)		(6.72s)		(7.25s)		(9.19s)	
100	66	9780	9633	9728	0	9728	0	9709	0	9707	0	9710	0	9709	0	9703	1
				(133.99s)		(134.10s)		(141.95s)		(154.09s)		(145.42s)		(149.12s)		(163.67s)	
100	66	9142	8831	8990	0	8981	0	8962	1	8966	0	8966	0	8966	0	8964	0
				(138.28s)		(139.04s)		(144.25s)		(155.43s)		(149.31s)		(150.97s)		(167.24s)	
100	66	9888	9670	9803	0	9791	0	9768	0	9770	0	9766	0	9764	1	9767	0
				(133.23s)		(133.81s)		(139.25s)		(154.42s)		(145.58s)		(157.98s)		(161.25s)	
total de mlh					0		0		3		0		1		2		2

				GRASP		GRC(R)		GRC(R,P)		GRC(R,P,I)		GRC(A)		GRC(A,P)		GRC(A,P,I)	
Iterações	n	C-S	LB	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh
10000	33	4861	4772	4804	0	4797	1	4797	1	4797	1	4797	1	4797	1	4797	1
				(419.43s)		(433.18s)		(434.92s)		(446.75s)		(752.91s)		(754.34s)		(766.06s)	
10000	33	5142	5035	5076	0	5067	1	5067	1	5069	0	5068	0	5068	0	5068	0
				(398.35s)		(414.93s)		(413.58s)		(429.32s)		(759.88s)		(761.77s)		(772.84s)	
10000	33	4352	4260	4296	0	4287	1	4287	1	4288	0	4287	1	4287	1	4287	1
				(405.03s)		(418.35s)		(420.94s)		(430.96s)		(747.17s)		(748.03s)		(762.19s)	
10000	66	9780	9633	9720	0	9703	0	9703	0	9699	0	9694	1	9694	1	9694	1
				(13453.17s)		(13449.01s)		(13387.55s)		(13545.63s)		(14676.68s)		(14553.60s)		(14629.08s)	
10000	66	9142	8831	8976	0	8957	0	8957	0	8956	0	8951	1	8951	1	8954	0
				(13238.00s)		(13257.29s)		(13335.41s)		(13412.89s)		(14871.51s)		(14706.02s)		(14922.91s)	
10000	66	9888	9670	9784	0	9757	0	9756	0	9759	0	9753	0	9753	0	9751	1
				(12689.67s)		(12739.95s)		(12748.95s)		(12784.51s)		(14446.85s)		(14326.70s)		(14391.67s)	
total de mlh					0		3		3		1		4		4		4

Tabela 4.7: Problemas testes do tipo III de Crama e Spieksma.

				GRASP		GRC(R)		GRC(R,P)		GRC(R,P,I)		GRC(A)		GRC(A,P)		GRC(A,P,I)	
Iterações	n	C-S	LB	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh
100	33	135	133	137 (1.75s)	0	136 (1.87s)	0	133 (2.69s)	1	133 (4.44s)	1	133 (4.13s)	1	133 (4.53s)	1	133 (6.56s)	1
100	33	137	130	136 (1.73s)	0	133 (1.83s)	0	131 (2.58s)	1	132 (4.38s)	0	131 (3.97s)	1	131 (4.50s)	1	131 (6.15s)	1
100	33	135	130	136 (1.73s)	0	133 (1.84s)	0	132 (2.60s)	0	131 (3.89s)	1	131 (3.82s)	1	131 (4.24s)	1	131 (5.84s)	1
100	66	293	283	297 (42.73s)	0	291 (43.28s)	0	286 (47.14s)	1	286 (55.57s)	1	286 (52.29s)	1	286 (53.18s)	1	286 (63.08s)	1
100	66	294	281	295 (42.24s)	0	291 (42.24s)	0	286 (45.63s)	1	287 (55.09s)	0	287 (50.79s)	0	286 (52.54s)	1	287 (62.16s)	0
100	66	293	280	295 (41.77s)	0	288 (42.05s)	0	282 (45.37s)	1	283 (54.55s)	0	284 (50.73s)	0	283 (52.69s)	0	283 (61.55s)	0
total de mlh				0		0		5		3		4		5		4	

				GRASP		GRC(R)		GRC(R,P)		GRC(R,P,I)		GRC(A)		GRC(A,P)		GRC(A,P,I)	
Iterações	n	C-S	LB	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh
10000	33	135	133	135 (149.44s)	0	133 (162.57s)	1	133 (164.11s)	1	133 (179.94s)	1	133 (475.23s)	1	133 (476.97s)	1	133 (490.79s)	1
10000	33	137	130	134 (149.74s)	0	131 (162.42s)	1	131 (163.33s)	1	131 (178.10s)	1	131 (459.16s)	1	131 (459.70s)	1	131 (471.21s)	1
10000	33	135	130	134 (148.27s)	0	132 (162.04s)	0	131 (161.64s)	1	131 (175.95s)	1	131 (439.93s)	1	131 (439.06s)	1	131 (451.72s)	1
10000	66	293	283	293 (4280.75s)	0	286 (4222.88s)	1	286 (4082.55s)	1	286 (4160.62s)	1	286 (5303.95s)	1	286 (5243.44s)	1	286 (5322.97s)	1
10000	66	294	281	292 (3955.23s)	0	286 (3951.28s)	1	286 (3943.24s)	1	286 (3967.02s)	1	286 (5117.01s)	1	286 (4980.35s)	1	286 (5126.86s)	1
10000	66	293	280	292 (4073.55s)	0	283 (3861.92s)	0	282 (3865.62s)	1	282 (3909.43s)	1	282 (5008.44s)	1	282 (5015.85s)	1	282 (5059.06s)	1
total de mlh				0		4		6		6		6		6		6	

- Em execuções longas, o algoritmo GRASP encontrou soluções melhores do que as obtidas por C-S para os problemas testes de tamanho $n = 33$, mas obteve soluções piores para $n = 66$.
- Em todas as execuções curtas, e em quase todas as execuções longas (com exceção de uma), `GRC(RAND)` e `GRC(ALL)` encontraram soluções melhores do que as obtidas pelo algoritmo puro.
- Em todas as execuções curtas, o procedimento de pós-otimização permitiu melhorar as soluções obtidas por `GRC(RAND)`. Em duas das seis execuções curtas, o procedimento de pós-otimização permitiu melhorar as soluções obtidas por `GRC(ALL)`. Porém, nas execuções longas, o procedimento de pós-otimização não fez nenhuma diferença.
- A diferença entre as variantes com religamento de caminhos nas execuções longas é desprezível.
- Em um problema teste de tamanho $n = 33$, mais de uma variante do método GRASP com religamento de caminhos encontrou uma solução ótima.
- O erro em relação ao limite inferior foi de no máximo 0.9% para as execuções curtas e de no máximo 0.65% para as execuções longas.

Para problemas do tipo II, a Tabela 4.6 mostra o seguinte:

- Em todas as execuções curtas e longas, o algoritmo GRASP puro encontrou melhores soluções do que as obtidas por C-S.
- Em quatro das seis execuções curtas e em todas as execuções longas, `GRC(RAND)` obteve melhores soluções do que o algoritmo puro. Em todas as execuções curtas e longas, `GRC(ALL)` obteve melhores soluções do que o método puro.
- Nas execuções curtas, o procedimento de pós-otimização melhorou as soluções obtidas por `GRC(RAND)`. Porém, o procedimento de pós-otimização melhorou a solução obtidas por `GRC(RAND)` em apenas uma execução longa.
- Em metade das execuções curtas o procedimento de pós-otimização permitiu melhorar as soluções obtidas por `GRC(ALL)`, enquanto que nas execuções longas a pós-otimização não teve nenhuma influência na qualidade das soluções obtidas por `GRC(ALL,INT)`.
- A intensificação deteriorou a qualidade das soluções obtidas mais vezes do que melhorou.

– O erro em relação ao limite inferior foi de no máximo 1.5% para as execuções curtas e de no máximo 1.35% para as execuções longas.

Para problemas do tipo III, a Tabela 4.7 mostra o seguinte:

– Em todas as execuções curtas e longas, o algoritmo com religamento de caminhos encontrou soluções melhores do que as obtidas por C-S.
– Em apenas uma execução curta, o algoritmo GRASP puro encontrou uma solução melhor do que a encontrada por C-S. Para os cinco casos testes restantes, C-S encontrou melhores soluções.
– Em quatro das seis execuções longas, o algoritmo GRASP puro obteve melhores soluções do que C-S, enquanto que nas duas execuções restantes, os dois algoritmos encontraram soluções com os mesmos custos.
– As variantes `GRC(RAND)` e `GRC(ALL)` encontraram melhores soluções do que as encontradas pelo método puro em todas as execuções curtas e longas.
– Em todas execuções curtas e em duas execuções longas, o procedimento de pós-otimização melhorou a solução obtida por `GRC(RAND)`.
– O procedimento de pós-otimização melhorou `GRC(ALL)` somente em duas execuções curtas.
– Nas execuções curtas, a intensificação deteriorou a qualidade das soluções obtidas pelas variantes `GRC(RAND)` e `GRC(ALL)` mais do que melhorou. Nas execuções longas, ela não teve nenhuma influência em relação à qualidade da solução obtida.
– O erro em relação ao limite inferior foi no máximo 1.8%.
– Para um problema teste de tamanho $n = 33$, todas as execuções produziram a solução ótima.

A Tabela 4.8 mostra os resultados para os problemas testes de Burkard, Rudolf e Woeginger [25]. São mostrados os resultados das execuções com 100 e 10000 iterações na primeira e na segunda tabela de cima para baixo, respectivamente. Em cada tabela são mostrados a dimensão do problema n, o valor obtido pela heurística Simple_LSH de Burkard, Rudolf e Woeginger usada para problemas testes de tamanho $n = 12$, o valor obtido pela heurística LSH de Burkard, Rudolf e Woeginger usada para problemas testes de tamanhos $n \geq 14$ e os resultados para as variantes do algoritmo GRASP. As heurísticas propostas por Burkard, Rudolf e Woeginger serão chamadas de B-R-W. Cinco execuções independentes foram feitas para cada

variante do método estudado, considerando-se todos os 100 problemas testes de cada tamanho. Cada célula corresponde a um par tamanho de problema teste/variante e mostra a média dos custos calculada para os 100 problemas testes (é considerada a melhor solução obtida nas cinco execuções independentes para cada problema teste), o número de vezes que a melhor solução foi encontrada e o tempo médio de execução para as quinhentas execuções. Esta tabela mostra o seguinte:

- O algoritmo GRASP puro encontrou em média melhores soluções do que B-R-W em todas as execuções curtas e longas.
- Nas execuções curtas, a adição do religamento de caminhos possibilitou a redução do custo médio das soluções para todos os tamanhos de problemas. Nas execuções longas, isso ocorreu apenas para os tamanhos de problemas $n \geq 14$.
- Nas execuções curtas, o procedimento de pós-otimização reduziu o custo das soluções obtidas pelas variantes `GRC(RAND)` e `GRC(ALL)`. Novamente nas execuções curtas, o procedimento de intensificação permitiu reduzir ainda mais o custo das soluções obtidas por `GRC(RAND,POST)` e `GRC(ALL,POST)`. Nas execuções longas, todas as variantes com religamento de caminhos encontraram as mesmas soluções.
- Nas execuções curtas, `GRC(ALL)` foi melhor do que `GRC(RAND)`, `GRC(ALL,POST)` foi melhor do que `GRC(RAND,POST)` e `GRC(ALL,POST,INT)` obteve melhores soluções do que `GRC(RAND,POST,INT)`.
- Nas execuções longas, o algoritmo GRASP puro foi capaz de encontrar as melhores médias de soluções, igualando-se às variantes que fazem religamento de caminhos, para 295 dos 300 problemas testes testados.

4.5.5 Distribuição da probabilidade do tempo para valor alvo

No capítulo anterior foi estudada a distribuição empírica de probabilidade da variável aleatória *tempo para valor alvo* de cinco implementações do método GRASP. Concluiu-se que, dado um valor alvo, o tempo para encontrar uma solução pelo menos tão boa quanto este alvo ajusta-se a uma distribuição exponencial de dois parâmetros. Uma metodologia padrão para análise gráfica [30] foi usada para computar as distribuições empíricas e teóricas e para estimar os parâmetros das distribuições.

Tabela 4.8: Problemas testes de Burkard, Rudolf e Woeginger.

100 iterações		GRASP		GRC(R)		GRC(R,P)		GRC(R,P,I)		GRC(A)		GRC(A,P)		GRC(A,P,I)	
n	B-R-W	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh
12	1188.02	1186.92	92	1186.85	97	1186.82	99	1186.81	100	1186.81	100	1186.81	100	1186.81	100
		(0.12s)		(0.14s)		(0.31s)		(0.73s)		(0.51s)		(0.62s)		(1.03s)	
14	1469.19	1468.18	73	1467.91	87	1467.76	98	1467.74	100	1467.75	98	1467.75	99	1467.74	100
		(0.21s)		(0.23s)		(0.49s)		(1.10s)		(0.75s)		(0.91s)		(1.52)	
16	1476.80	1475.93	56	1475.65	64	1475.17	96	1475.14	99	1475.15	98	1475.13	100	1475.13	100
		(0.35s)		(0.38s)		(0.76s)		(1.61s)		(1.07s)		(1.33s)		(2.21)	
total de mlh			221		248		293		299		297		299		300

10000 iterações		GRASP		GRC(R)		GRC(R,P)		GRC(R,P,I)		GRC(A)		GRC(A,P)		GRC(A,P,I)	
n	B-R-W	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh	custo	mlh
12	1188.02	1186.81	100	1186.81	100	1186.81	100	1186.81	100	1186.81	100	1186.81	100	1186.81	100
		(11.19s)		(13.29s)		(13.35s)		(15.96s)		(65.90s)		(65.96s)		(68.30s)	
14	1469.19	1467.75	99	1467.74	100	1467.74	100	1467.74	100	1467.74	100	1467.74	100	1467.74	100
		(19.09s)		(21.92s)		(22.01s)		(25.67s)		(94.70s)		(94.79s)		(98.02s)	
16	1476.80	1475.18	96	1475.13	100	1475.13	100	1475.13	100	1475.13	100	1475.13	100	1475.13	100
		(31.77s)		(35.58s)		(35.74s)		(41.08s)		(134.62s)		(134.86s)		(139.30s)	
total de mlh			295		300		300		300		300		300		300

Tabela 4.9: Problemas testes usados para estudar a distribuição empírica de probabilidade da variável aleatória *tempo para valor alvo* para `GRASP`. O limite superior (ls) é o melhor valor obtido pelos algoritmos GRASP com religamento de caminhos. São mostrados os valores alvos e os parâmetros estimados.

			parâmetros estimados	
problema	ls	alvo	$\hat{\mu}$	$\hat{\lambda}$
`20.1`	5	20	-0.051	21.538
		18	1.438	41.654
		16	1.288	90.184
`22.1`	9	42	0.064	0.774
		29	0.319	4.336
		16	-2.607	185.218
`24.1`	9	39	-0.059	2.101
		28	-0.003	8.672
		16	-2.890	246.557
`26.1`	9	50	0.134	0.748
		34	-0.076	5.334
		17	26.363	252.903

As Figuras 4.10 e 4.11 mostram, respectivamente, os gráficos Q-Q e os gráficos com as distribuições teóricas e empíricas sobrepostas para `GRASP`. Esses gráficos foram plotados usando a metodologia proposta no capítulo anterior. Foram considerados três valores alvos para cada um dos problemas testes `B-S 20.1`, `B-S 22.1`, `B-S 24.1` e `B-S 26.1` de Balas e Saltzman. Da mesma forma que nos testes feitos no capítulo anterior, foram realizadas 200 execuções independentes do método GRASP para cada par problema teste/valor alvo. Os 200 pontos de dados gerados foram usados para plotar o gráfico Q-Q com informação de variabilidade e o gráfico das distribuições empírica e teórica sobrepostas. A Figura 4.10 é formada por 12 gráficos Q-Q, um para cada par problema teste/valor alvo. Da mesma forma, a Figura 4.11 é formada por 12 gráficos com as distribuições empírica e teórica sobrepostas, um para cada par problema teste/valor alvo. Cada figura é formada por quatro linhas onde são dispostos os gráficos. Cada linha corresponde a um dos quatro problemas testes escolhidos para testar a heurística proposta para o AP3. Em cada linha das figuras, a dificuldade de se encontrar uma solução para um dado valor alvo aumenta da esquerda para a direita. De acordo com as conclusões do capítulo anterior, observa-se que os tempos para valor alvo medidos também adaptam-se bem a uma distribuição exponencial de dois parâmetros. Os valores alvos considerados, assim como os parâmetros estimados pela metodologia, são mostrados na Tabela 4.9.

A metodologia padrão usada para estudar a distribuição empírica de

Tabela 4.10: Problemas testes usados para estudar a distribuição empírica de probabilidade da variável aleatória *tempo para valor alvo* para `GRC(RAND)`. O limite superior (ls) é o melhor valor obtido pelos algoritmos GRASP com religamento de caminhos. São mostrados os valores alvos e os parâmetros estimados.

			parâmetros estimados	
problema	ls	alvo	$\hat{\mu}$	$\hat{\lambda}$
`20.1`	5	18	3.244	12.622
		16	4.265	24.305
		7	-593.345	3852.271
`22.1`	9	29	0.550	3.349
		16	8.776	55.171
		8	-438.551	5032.154
`24.1`	9	28	0.427	5.879
		16	9.536	72.319
		7	-107.661	6924.395
`26.1`	9	34	0.246	4.254
		17	21.009	66.803
		8	101.853	3421.181

probabilidade da variável aleatória *tempo para valor alvo* do método GRASP puro é usada nessa seção para estudar o tempo para valor alvo das quatro variantes com religamento de caminhos `GRC(RAND)`, `GRC(RAND,INT)`,`GRC(ALL)` e `GRC(ALL,INT)`. Quatro problemas testes foram considerados: `B-S 20.1`, `B-S 22.1`, `B-S 24.1` e `B-S 26.1` de Balas e Saltzman. O objetivo é mostrar que as distribuições do tempo para valor alvo das quatro variantes com religamento de caminhos também encaixam-se em uma distribuição exponencial de dois parâmetros.

Nas Figuras 4.12, 4.14, 4.16 e 4.18 são mostrados os gráficos Q-Q para `GRC(RAND)`, `GRC(RAND,INT)`, `GRC(ALL)` e `GRC(ALL,INT)`, respectivamente. Da mesma forma, as Figuras 4.13, 4.15, 4.17 e 4.19 mostram as distribuições empíricas e teóricas da variável aleatória *tempo para valor alvo* para `GRC(RAND)`, `GRC(RAND,INT)`, `GRC(ALL)` e `GRC(ALL,INT)`, respectivamente. Essas figuras são semelhantes às Figuras 4.10 e 4.11 mostradas para `GRASP`. Cada figura é formada por quatro linhas. Cada linha corresponde a um dos quatro problemas considerados. Para cada problema teste foram testados três valores alvos. Em cada linha das figuras, a dificuldade de se encontrar uma solução para um dado valor alvo aumenta da esquerda para a direita. Os gráficos foram plotados com os tempos de 200 execuções da variante do algoritmo GRASP com religamento de caminhos correspondente. Os valores alvos considerados para cada problema e os parâmetros estimados para as distribuições plotadas para as variantes `GRC(RAND)`, `GRC(RAND,INT)`,`GRC(ALL)` e `GRC(ALL,INT)` são mostrados nas Tabelas 4.10, 4.11, 4.12 e 4.13, respec-

Tabela 4.11: Problemas testes usados para estudar a distribuição empírica de probabilidade da variável aleatória *tempo para valor alvo* para `GRC(RAND,INT)`. O limite superior (ls) é o melhor valor obtido pelos algoritmos GRASP com religamento de caminhos. São mostrados os valores alvos e os parâmetros estimados.

			parâmetros estimados	
problema	ls	alvo	$\hat{\mu}$	$\hat{\lambda}$
`20.1`	5	18	1.128	8.663
		16	2.714	15.148
		7	-479.423	3311.963
`22.1`	9	29	0.685	2.884
		16	4.060	35.847
		8	-473.328	5298.755
`24.1`	9	28	0.995	3.913
		16	1.149	60.890
		7	-144.399	7890.621
`26.1`	9	34	0.237	4.258
		17	14.773	58.888
		8	-121.385	3860.867

Tabela 4.12: Problemas testes usados para estudar a distribuição empírica de probabilidade da variável aleatória *tempo para valor alvo* para `GRC(ALL)`. O limite superior (ls) é o melhor valor obtido pelos algoritmos GRASP com religamento de caminhos. São mostrados os valores alvos e os parâmetros estimados.

			parâmetros estimados	
problema	ls	alvo	$\hat{\mu}$	$\hat{\lambda}$
`20.1`	5	18	1.105	12.263
		16	1.683	20.433
		7	-26.468	1223.800
`22.1`	9	29	0.385	3.674
		16	7.585	48.587
		8	-135.125	3085.320
`24.1`	9	28	0.008	6.183
		16	1.053	63.019
		7	-16.763	4004.119
`26.1`	9	34	0.418	3.233
		17	13.692	50.474
		8	32.120	2255.550

Tabela 4.13: Problemas testes usados para estudar a distribuição empírica de probabilidade da variável aleatória *tempo para valor alvo* para `GRC(ALL,INT)`. O limite superior (ls) é o melhor valor obtido pelos algoritmos GRASP com religamento de caminhos. São mostrados os valores alvos e os parâmetros estimados.

			parâmetros estimados	
problema	ls	alvo	$\hat{\mu}$	$\hat{\lambda}$
`20.1`	5	18	0.994	12.425
		16	2.475	17.620
		7	-23.089	1176.373
`22.1`	9	29	0.378	3.677
		16	8.253	43.112
		8	-139.506	2912.195
`24.1`	9	28	0.014	6.167
		16	-0.039	65.575
		7	-130.586	3933.269
`26.1`	9	34	0.419	3.226
		17	10.459	53.769
		8	33.111	2264.880

tivamente. As variantes `GRC(RAND,INT)` e `GRC(ALL,INT)` fazem a intensificação a cada intervalo de *ifreq*= 500, *ifreq*= 500 e *ifreq*= 2000 iterações para os valores alvos fáceis (colunas mais à esquerda nas tabelas), médios (colunas do meio nas tabelas) e difíceis (colunas mais à direita nas tabelas), respectivamente.

As Figuras 4.12, 4.14, 4.16 e 4.18 mostram que existe pouco afastamento dos pontos em relação à reta estimada e, conseqüentemente, as distribuições empíricas dos dados adaptam-se bem a distribuições exponenciais de dois parâmetros para todas as variantes com religamento de caminhos testadas. Além disso, quanto maior for a dificuldade de se obter um dado valor alvo, melhor é a adequação da sobreposição da distribuição teórica à distribuição empírica nos gráficos das Figuras 4.13, 4.15, 4.17 e 4.19.

Segundo a Proposição 3.2 mostrada no capítulo anterior para a distribuição exponencial de dois parâmetros, a probabilidade de se encontrar uma solução com um dado valor em tempo ρt executando-se um processo seqüencial é igual a $1 - e^{-(\rho t - \mu)/\lambda}$, enquanto que a probabilidade de se encontrar uma solução com custo pelo menos tão bom quanto o valor dado em tempo t com ρ processos paralelos independentes é $1 - e^{-\rho(t-\mu)/\lambda}$. Deve-se notar que se $\mu = 0$, então as duas probabilidades são iguais e correspondem à distribuição exponencial não deslocada. Além disso, se $\rho|\mu| \ll \lambda$, então as duas probabilidades são aproximadamente iguais e é possível atingir uma aceleração aproximadamente linear no tempo de solução para valor alvo usando-se múltiplos processos independentes. No Capítulo 6 serão propos-

tas estratégias de paralelização para as estratégias GRASP para o AP3. Uma análise baseada nos parâmetros estimados para as distribuições plotadas será proposta para estudar a aceleração das estratégias paralelas.

4.6 Conclusão

Nesse capítulo, foram apresentadas novas técnicas de construção e de busca local para o problema de atribuição de três índices. Também foi apresentado um algoritmo GRASP com religamento de caminhos para esse problema. Mostrou-se como a técnica de religamento de caminhos pode ser usada para melhorar o desempenho do algoritmo GRASP. Novas formas de incorporar o religamento de caminhos ao algoritmo básico foram descritas. O religamento de caminhos em dois sentidos, o religamento de caminhos aplicado ao conjunto inteiro de soluções de elite e as etapas de pós-otimização e intensificação baseadas em religamento de caminhos permitiram melhorar a estratégia básica.

Experimentos computacionais extensivos foram feitos com os diferentes algoritmos apresentados nesse capítulo. Observou-se que as variantes com religamento de caminhos melhoram o desempenho do algoritmo GRASP puro, permitindo encontrar mais rapidamente uma solução de uma dada qualidade, além de melhorar a qualidade da solução obtida quando o algoritmo é executado por um número fixo de iterações.

Para todos os problemas testes considerados, observou-se que a variante `GRC(RAND)` encontra mais rapidamente uma solução com custo menor ou igual a um valor alvo fornecido como entrada do que a variante `GRASP`. Da mesma forma, observa-se que as variantes `GRC(ALL)` e `GRC(ALL,INT)` encontram uma solução com um dado valor alvo em um tempo computacional inferior ao tempo necessário para `GRC(RAND)` e `GRC(RAND,INT)` obterem uma solução de qualidade semelhante. Portanto, observa-se que as variantes que requerem mais trabalho em cada iteração encontram soluções de uma dada qualidade em tempo menor do que as variantes que realizam menos trabalho por iteração. Além disso, as variantes que requerem mais trabalho em cada iteração, em geral, encontraram soluções melhores do que as demais variantes, quando um mesmo número de iterações é realizado. Mostrou-se que o método GRASP com religamento de caminhos proposto permitiu melhorar os resultados obtidos pelas heurísticas previamente descritas em [13, 25, 37].

A distribuição da probabilidade da variável aleatória *tempo para valor alvo* das variantes com religamento de caminhos foi estudada nesse capítulo. Concluiu-se que esses tempos adaptam-se bem a uma distribuição exponencial de dois parâmetros. Além disso, mostrou-se como as curvas dadas pelas distribuições empíricas dos tempos medidos para as diferentes variantes podem ser usadas como uma ferramenta útil para facilitar a comparação entre algoritmos.

No próximo capítulo será apresentado um algoritmo GRASP com religamento de caminhos para o problema de escalonamento de tarefas JSP. A distribuição do tempo para valor alvo será estudada para esse algoritmo. O algoritmo GRASP para o problema de escalonamento de tarefas JSP será usado juntamente com o algoritmo proposto nesse capítulo para estudar o comportamento das estratégias de paralelização propostas no Capítulo 5.

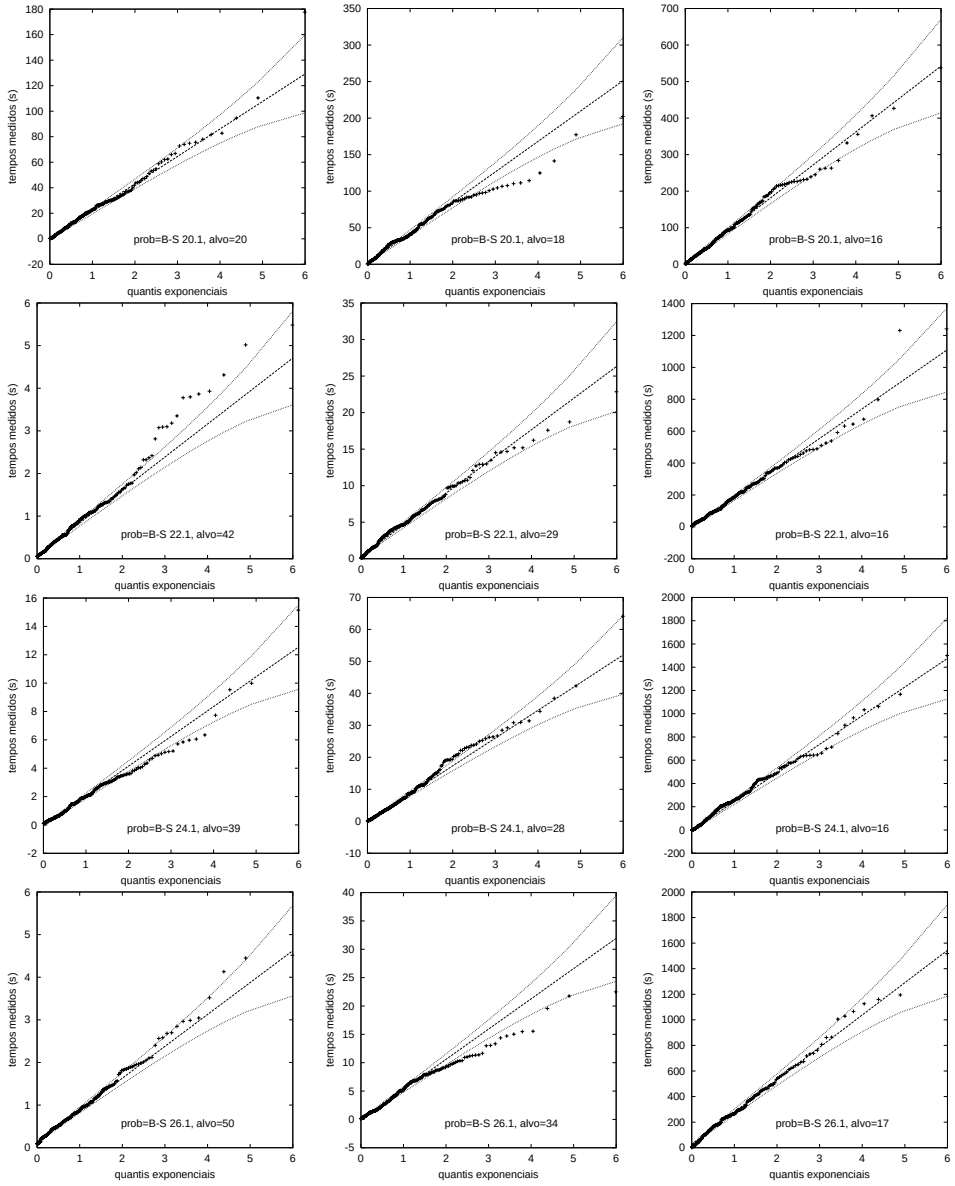

Figura 4.10: Gráficos Q-Q para `GRASP`, problemas B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman.

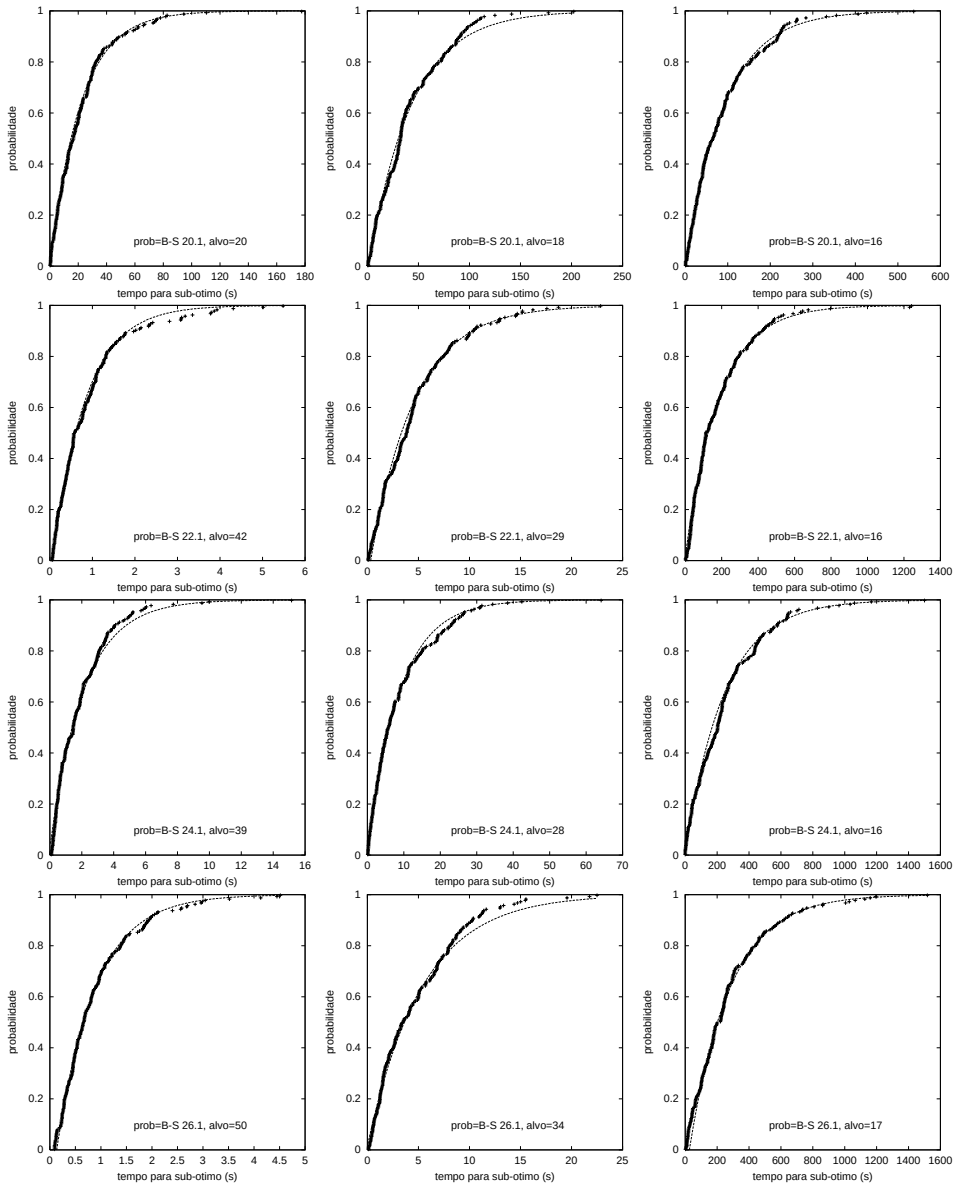

Figura 4.11: Gráficos das distribuições exponenciais para GRASP, problemas B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman.

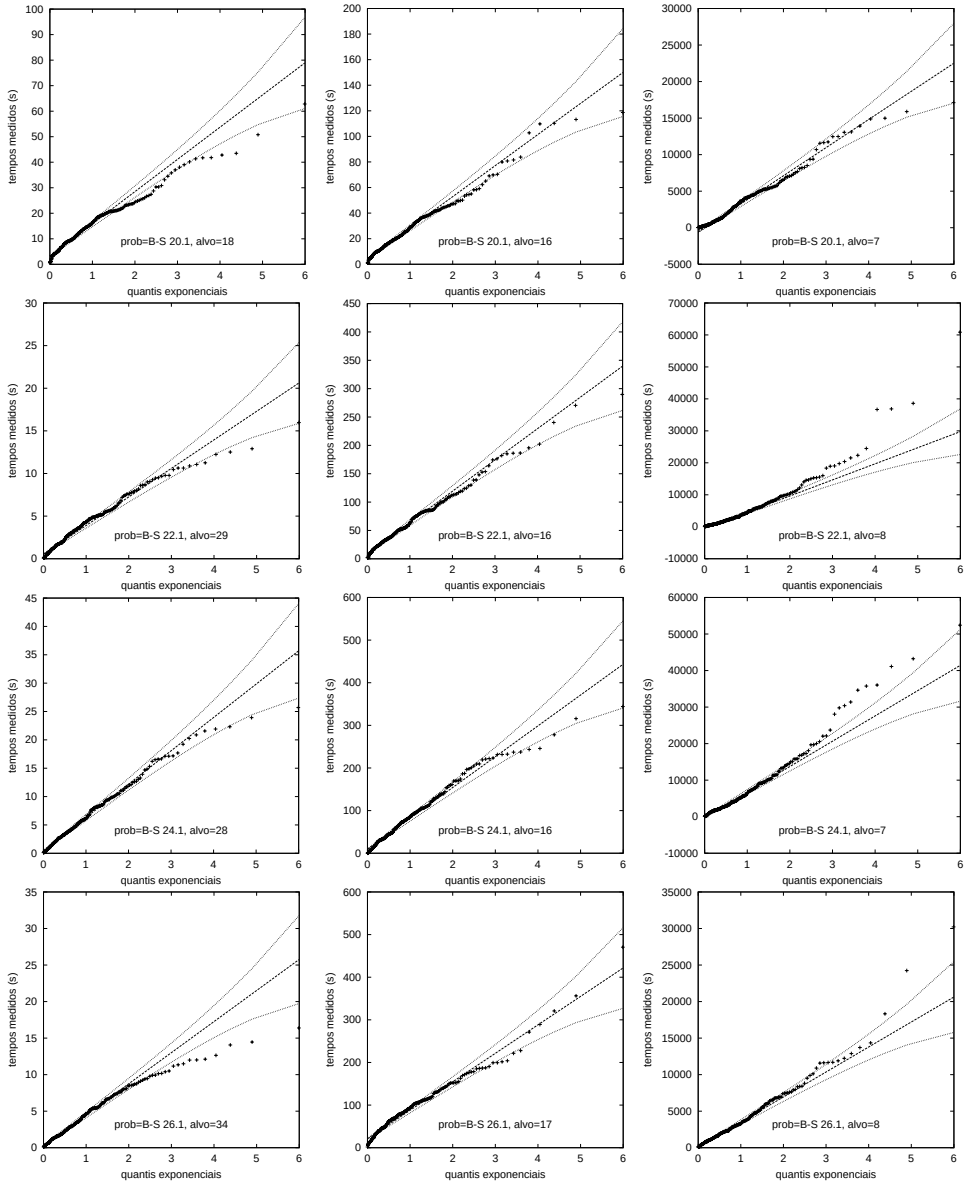

Figura 4.12: Gráficos Q-Q para `GRC(RAND)`, problemas B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman.

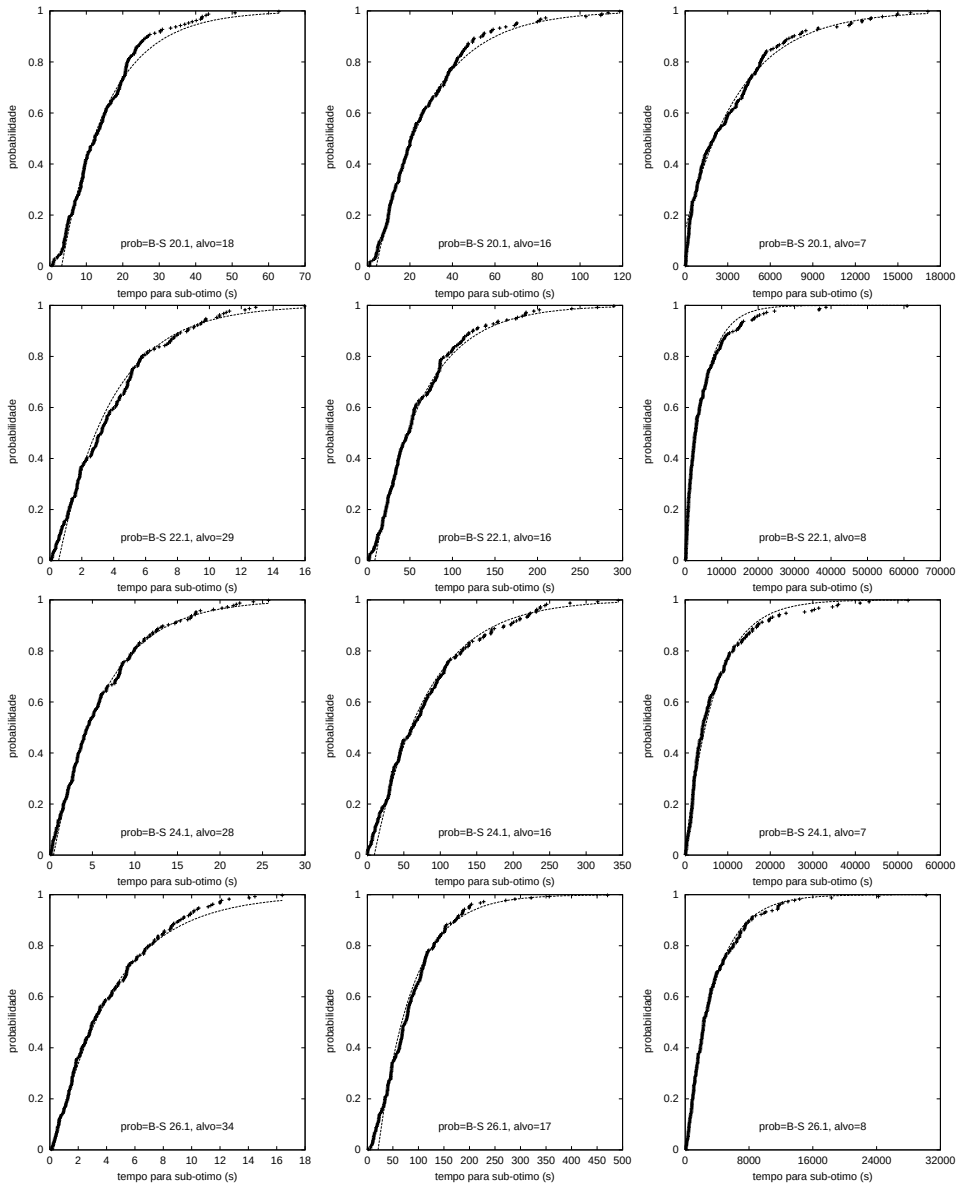

Figura 4.13: Gráficos das distribuições exponenciais para `GRC(RAND)`, problemas B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman.

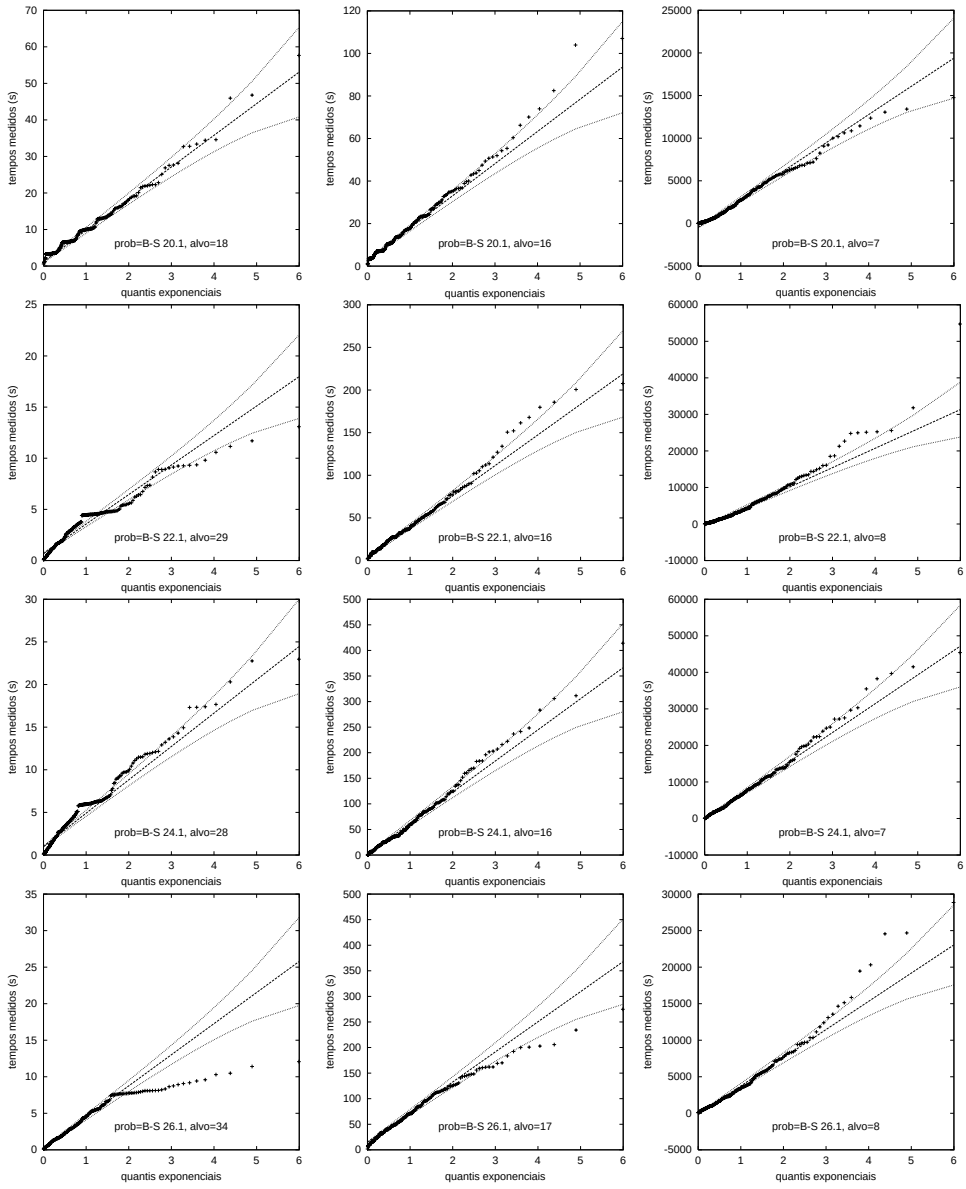

Figura 4.14: Gráficos Q-Q para `GRC(RAND,INT)`, problemas B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman.

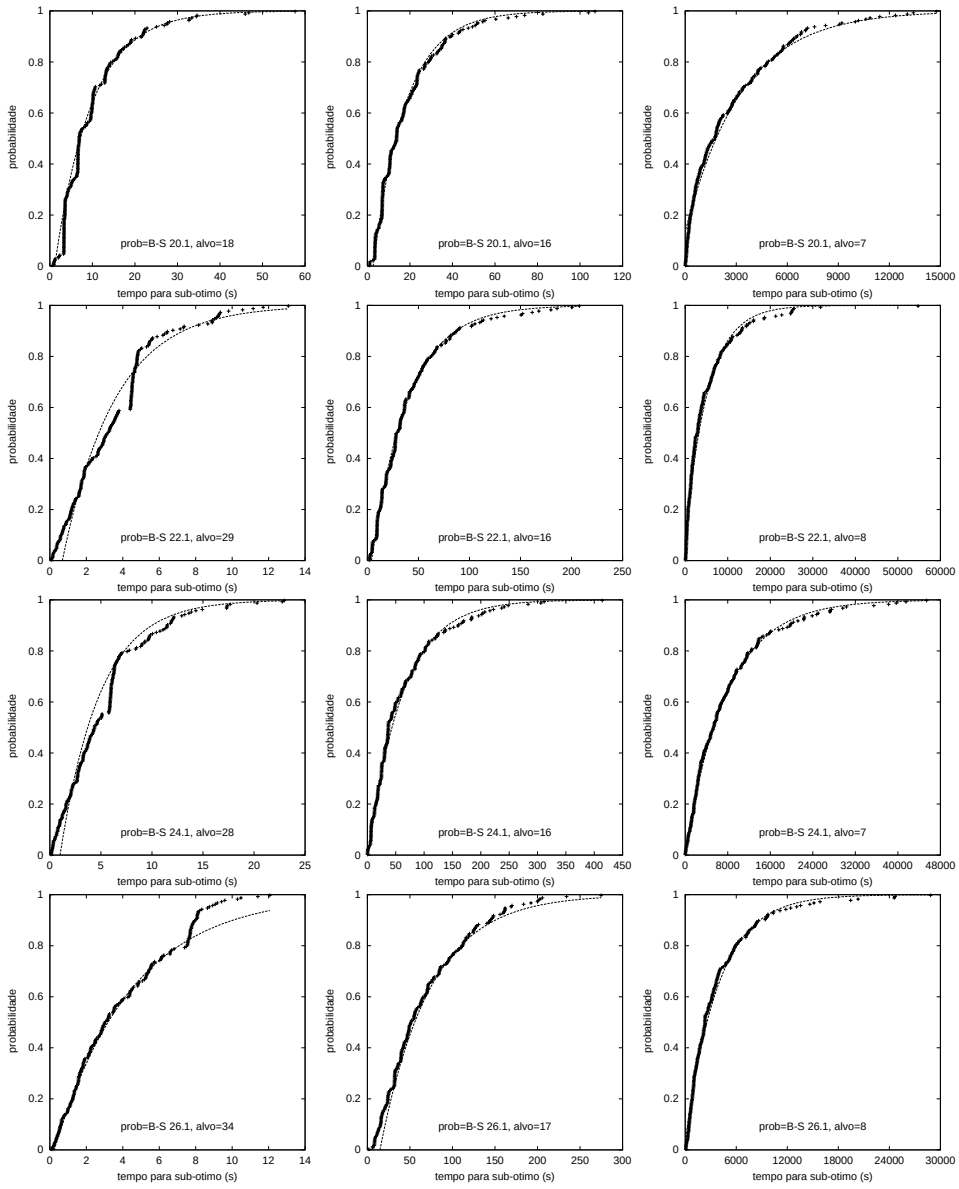

Figura 4.15: Gráficos das distribuições exponenciais para `GRC(RAND,INT)`, problemas B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman.

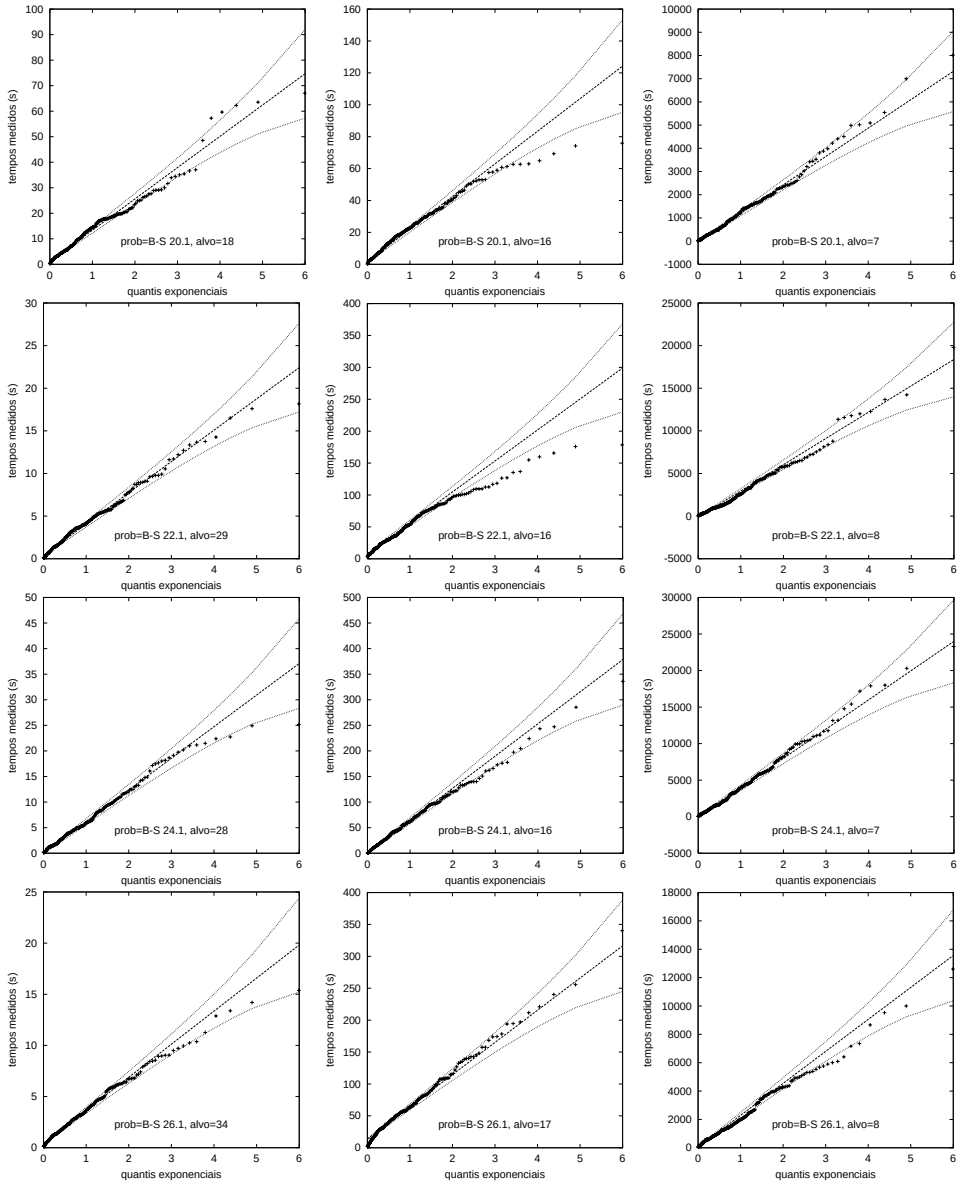

Figura 4.16: Gráficos Q-Q para `GRC(ALL)`, problemas B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman.

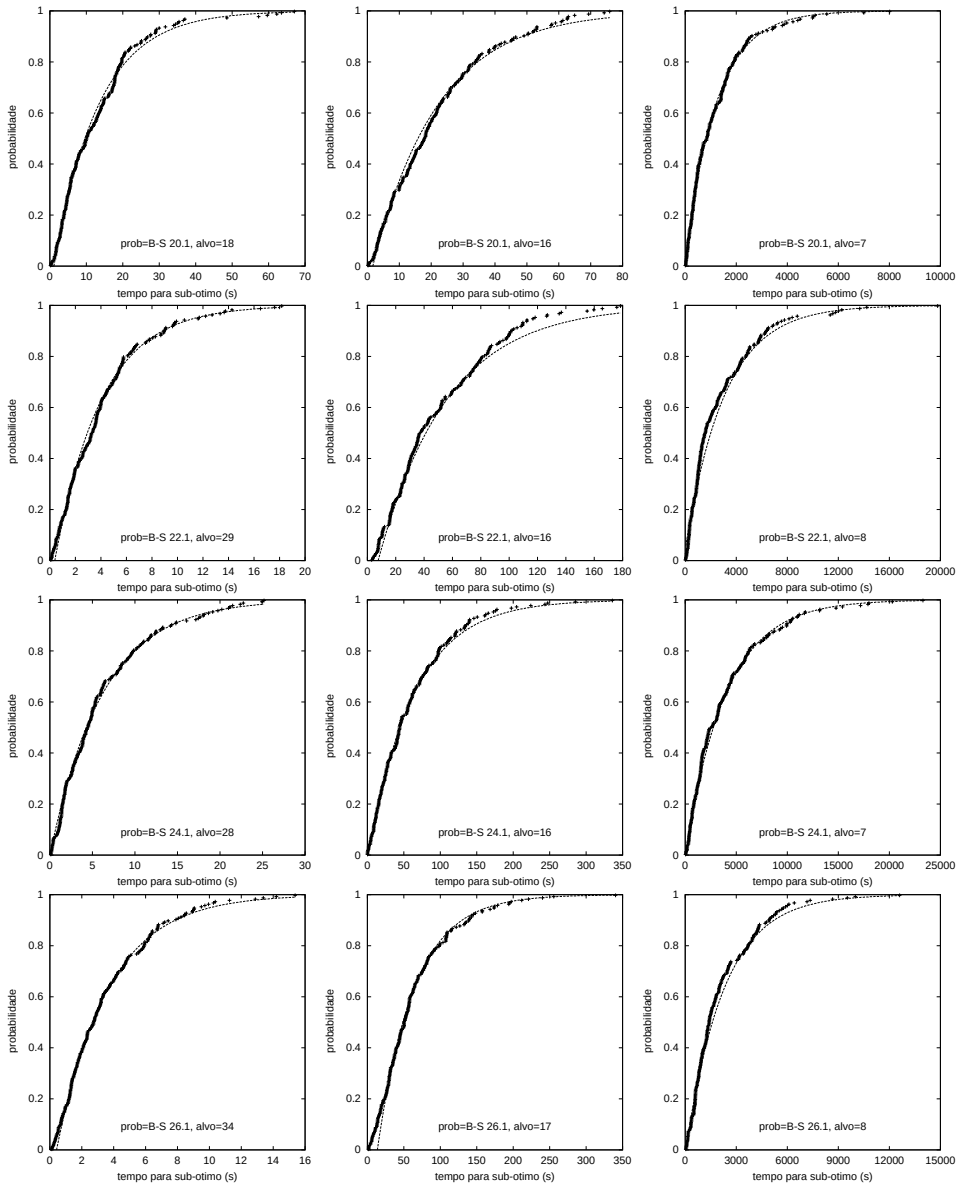

Figura 4.17: Gráficos das distribuições exponenciais para `GRC(ALL)`, problemas B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman.

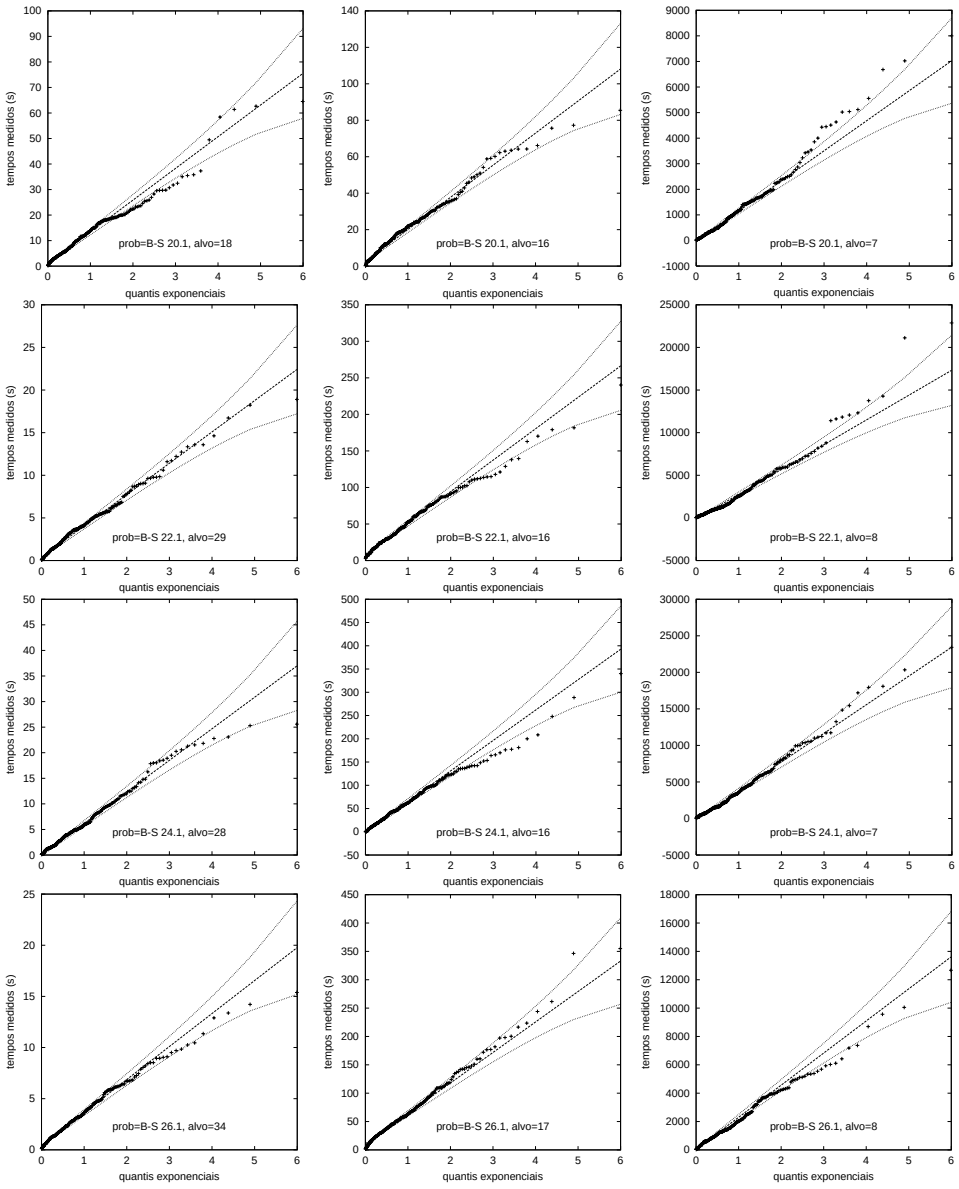

Figura 4.18: Gráficos Q-Q para `GRC(ALL,INT)`, problemas B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman.

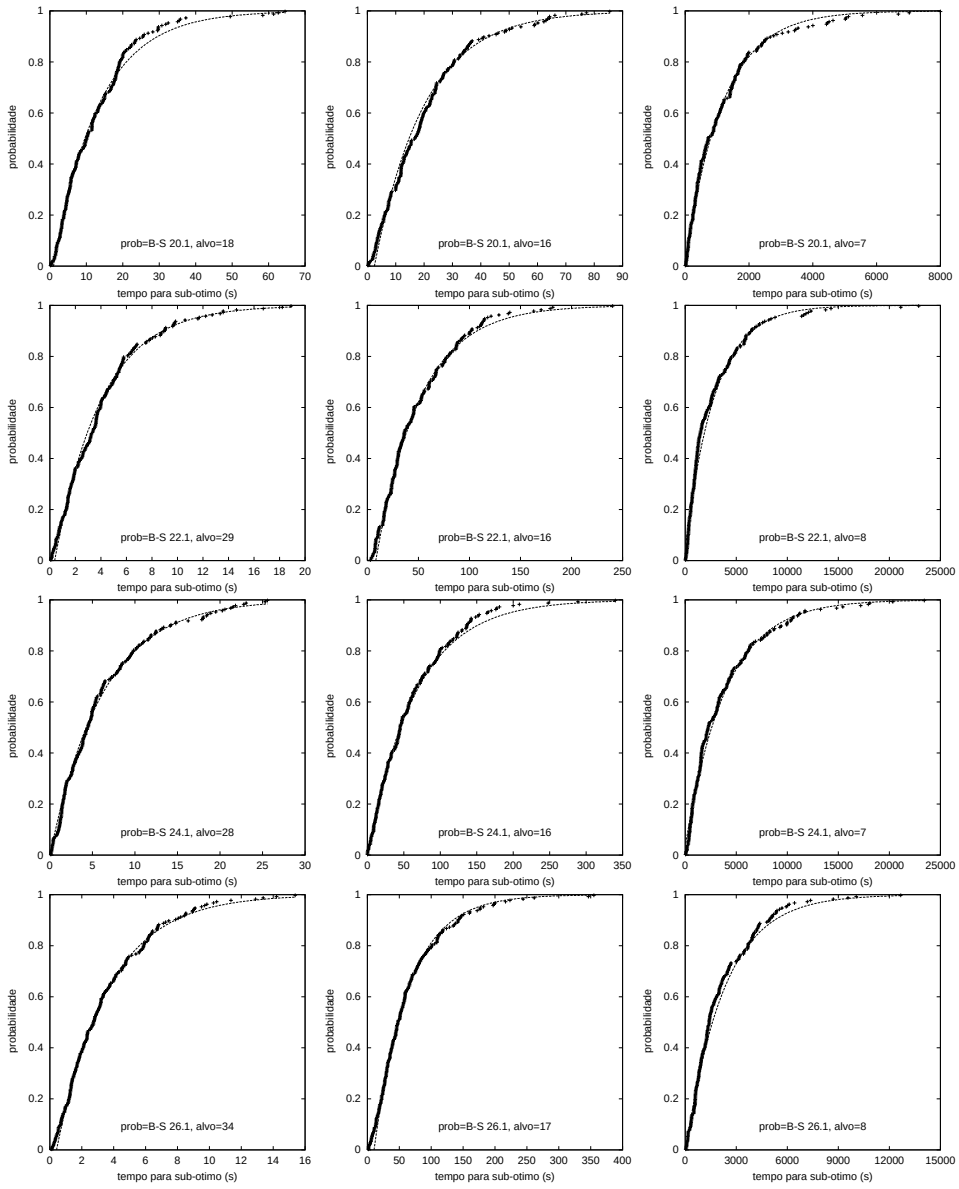

Figura 4.19: Gráficos das distribuições exponenciais para `GRC(ALL,INT)`, problemas B-S 20.1, B-S 22.1, B-S 24.1 e B-S 26.1 de Balas e Saltzman.