5
GRASP com religamento de caminhos para o problema de job-shop scheduling

5.1
Descrição do problema

O problema de escalonamento de tarefas estudado nesse capítulo é conhecido na literatura como *job-shop scheduling problem* (JSP). Neste problema, um número finito de tarefas é processado em um número finito de máquinas. Cada tarefa é caracterizada por uma ordem fixa de operações, que devem ser processadas em máquinas específicas, durante um dado intervalo de tempo. Cada máquina pode processar no máximo uma operação a cada vez e, uma vez iniciado o processamento de uma operação em uma determinada máquina, esse processamento deve ser completado sem interrupção (isto é, o sistema é não preemptivo). Um escalonamento é uma atribuição de operações a intervalos de tempos nas máquinas. Chama-se de *makespan* o tempo necessário para que todas as tarefas tenham sido completadas. O objetivo do JSP é encontrar um escalonamento que minimize o *makespan*.

O JSP pode ser formulado da seguinte maneira. Dados um conjunto de máquinas $\mathcal{M}$ e um conjunto de tarefas $\mathcal{J}$, sejam $\sigma_1^j \prec \sigma_2^j \prec \cdots \prec \sigma_{|\mathcal{M}|}^j$ os elementos do conjunto ordenado que contém as $|\mathcal{M}|$ operações da tarefa j, onde $\sigma_k^j \prec \sigma_{k+1}^j$ indica que a operação σ_{k+1}^j só poderá iniciar o seu processamento após o término da operação σ_k^j. Seja $\mathcal{O}$ o conjunto de operações. Cada operação σ_k^j é definida por dois parâmetros: $\mathcal{M}_k^j$ é a máquina onde σ_k^j deve ser processada e $p_k^j = p(\sigma_k^j)$ é o tempo de processamento da operação σ_k^j. Chamando-se o tempo inicial da operação $\sigma_k^j \in \mathcal{O}$ de $t(\sigma_k^j)$, uma formulação para o JSP é dada a seguir:

$$
\begin{aligned}
&\text{Minimizar } C_{\max} \\
&\text{Sujeito a: } C_{\max} \geq t(\sigma_k^j) + p(\sigma_k^j),\ \forall \sigma_k^j \in \mathcal{O}, \\
&\qquad t(\sigma_k^j) \geq t(\sigma_l^j) + p(\sigma_l^j),\ \forall \sigma_l^j \prec \sigma_k^j, && \text{(5-1a)} \\
&\qquad t(\sigma_k^j) \geq t(\sigma_l^i) + p(\sigma_l^i) \text{ ou } t(\sigma_l^i) \geq t(\sigma_k^j) + p(\sigma_k^j), && \text{(5-1b)} \\
&\qquad\quad \forall \sigma_l^i, \sigma_k^j \in \mathcal{O} \text{ tal que } \mathcal{M}_{\sigma_l^i} = \mathcal{M}_{\sigma_k^j}, \\
&\qquad t(\sigma_k^j) \geq 0,\ \forall \sigma_k^j \in \mathcal{O},
\end{aligned}
$$

onde C_{max} é o *makespan* que deverá ser minimizado.

Uma solução viável para o JSP pode ser construída a partir de uma permutação das tarefas em $\mathcal{J}$ em cada uma das máquinas em $\mathcal{M}$, observando-se as restrições de precedência e a restrição de que a máquina só poderá processar uma operação de cada vez e de forma ininterrupta. A viabilidade de um conjunto de $|\mathcal{M}|$ permutações de $\mathcal{J}$ pode ser testada em $O(|\mathcal{J}| \cdot |\mathcal{M}|)$. O procedimento usado para determinar a viabilidade de uma solução também calcula o makespan $\mathcal{C}_{max}$ para soluções viáveis [126]. Como cada conjunto de $|\mathcal{M}|$ permutações viáveis corresponde a um escalonamento, o objetivo do JSP é encontrar um conjunto de $|\mathcal{M}|$ permutações viáveis que minimize o *makespan*.

O JSP é NP-difícil [74]. Este problema permanece NP-difícil mesmo para problemas testes formados por três máquinas e tempos unitários de processamento, assim como para problemas testes com apenas três tarefas. Mesmo quando a preempção é permitida, o problema permanece NP-difícil. Métodos exatos [11, 21, 28, 29, 56] têm sido propostos para resolver problemas testes pequenos. Problemas testes com dimensões acima de 15×15 são considerados fora do alcance dos métodos exatos. Para tais problemas, existe a necessidade de heurísticas eficientes. Estudos de heurísticas para o JSP são descritos em [96, 127]. Entre esses métodos estão as regras de despacho revisadas em [50], a abordagem de *shifting bottleneck* [1, 11], estratégias de busca local [78, 79, 127], *simulated annealing* [78, 128], busca tabu [79, 87, 126] e algoritmos genéticos [39]. Um *survey* de técnicas usadas no problema de escalonamento de tarefas JSP pode ser encontrado em [70].

Em Binato et al. [16] é proposta uma heurística GRASP para o JSP. Essa heurística incorpora à fase construtiva uma estratégia de intensificação, onde atributos de soluções de elite recebem incentivos para serem inseridos na solução. Além disso, o POP (*Proximate Optimality Principle*) é aplicado

a soluções parciais durante a fase construtiva. No restante desse capítulo, será apresentada uma heurística GRASP com religamento de caminhos para o JSP baseada na heurística proposta por Binato et al. No algoritmo proposto nesse capítulo, os mecanismos de intensificação e o procedimento POP são retirados da fase construtiva. Dessa forma, a partir de uma heurística GRASP básica, uma etapa de religamento de caminhos é inserida ao final de cada iteração, de maneira semelhante ao que foi feito para o problema de atribuição de três índices no capítulo anterior. Portanto, a etapa de intensificação é transferida da fase construtiva para o final de cada iteração do algoritmo. Pretende-se com isso reduzir o custo das soluções obtidas por Binato et al, assim como o tempo computacional necessário para obtê-las.

5.2 GRASP para o problema de job-shop scheduling

Nessa seção, será apresentada uma heurística GRASP para o problema de escalonamento de tarefas JSP. Os algoritmos usados para a construção de soluções e para a busca local serão descritos a seguir.

5.2.1 Fase construtiva

Na fase construtiva do algoritmo GRASP, uma solução viável é construída inserindo-se um elemento de cada vez na solução parcial. No problema de escalonamento de tarefas JSP, o elemento incorporado à solução parcial a cada iteração da fase construtiva será uma operação. Dessa forma, um escalonamento viável será construído escalonando-se operações, uma de cada vez, até que todas as operações tenham sido escalonadas.

Como visto na seção anterior, σ_k^j denota a k-ésima operação da tarefa j e é definida pelo par $(\mathcal{M}_k^j, p_k^j)$, onde $\mathcal{M}_k^j$ é a máquina na qual a operação σ_k^j é processada e p_k^j é o tempo de processamento da operação σ_k^j. Durante uma determinada etapa da construção de uma solução viável, apenas um subconjunto das operações ainda não escalonadas será candidato ao escalonamento. Uma operação σ_k^j somente poderá ser escalonada caso todas as operações que a precedem na tarefa j já tenham sido escalonadas. Portanto, em cada iteração da fase construtiva, no máximo $|\mathcal{J}|$ operações são candidatas a serem escalonadas. Seja $\mathcal{O}_c$ o conjunto de operações

candidatas e $\mathcal{O}_s$ o conjunto de operações já escalonadas e denote o valor da função gulosa de uma operação candidata σ_k^j por $h(\sigma_k^j)$.

A escolha gulosa consiste em selecionar a operação $\underline{\sigma_k^j} = \text{argmin}(h(\sigma_k^j) \mid \sigma_k^j \in \mathcal{O}_c)$. Seja $\overline{\sigma_k^j} = \text{argmax}(h(\sigma_k^j) \mid \sigma_k^j \in \mathcal{O}_c)$, $\underline{h} = h(\underline{\sigma_k^j})$ e $\overline{h} = h(\overline{\sigma_k^j})$. Logo, a lista restrita de candidatos (LRC) é definida como sendo

$$\text{LRC} = \{\sigma_k^j \in \mathcal{O}_c \mid \underline{h} \leq h(\sigma_k^j) \leq \underline{h} + \alpha(\overline{h} - \underline{h})\},$$

onde α é um parâmetro tal que $0 \leq \alpha \leq 1$.

Uma iteração típica da fase construtiva pode ser resumida da seguinte maneira: partindo-se de um escalonamento parcial (inicialmente vazio), a próxima operação σ_k^j a ser escalonada é selecionada da LRC e é adicionada ao escalonamento parcial, resultando em um novo escalonamento parcial. A operação selecionada é inserida no primeiro intervalo de tempo disponível e viável na máquina $\mathcal{M}_{\sigma_k^j}$. Sejam a e b os instantes inicial e final de um intervalo de tempo disponível na máquina $\mathcal{M}_{\sigma_k^j}$ e seja $e = t(\sigma_{k-1}^j) + p_{k-1}^j$ o tempo final da operação σ_{k-1}^j. A inserção de σ_k^j nesse intervalo de tempo, iniciando em $\max\{a, e\}$, será viável se e somente se $b - \max\{a, e\} \geq p_k^j$. A fase construtiva é finalizada quando o escalonamento parcial estiver completo, isto é, quando todas as operações tiverem sido escalonadas.

Em Binato et al. [16], a função gulosa usada $h(\sigma_k^j)$ é o *makespan* resultante da inclusão da operação σ_k^j às operações já escalonadas, isto é, $h(\sigma_k^j) = \mathcal{C}_{max}$ para $\mathcal{O} = \{\mathcal{O}_s \cup \sigma_k^j\}$. Essa função será chamada de função gulosa por *makespan*.

Nesse capítulo, uma nova função gulosa será proposta. Como será explicado adiante, a função gulosa proposta será usada juntamente com a função gulosa por *makespan* durante as iterações do algoritmo. Essa função favorece operações de tarefas que possuem tempos de processamento elevados ainda não escalonados. A função gulosa proposta é definida por $h(\sigma_k^j) = -\sum_{\sigma_l^j \notin \mathcal{O}_s} p_l^j$ e mede o tempo de processamento ainda não escalonado da tarefa j. Essa função será chamada de função gulosa por *tempo restante*.

O algoritmo usado na fase construtiva é apresentado na Figura 5.1. Na linha 1 o parâmetro α da LRC é escolhido aleatoriamente. O algoritmo parte de um escalonamento parcial inicialmente vazio (linhas 2 e 3) e de um conjunto de operações candidatas formado pelas operações σ_1^j que iniciam cada tarefa j (linha 4). Nas linhas 5 a 14, $|\mathcal{M}| \cdot |\mathcal{J}| - 1$ operações são escolhidas para fazer parte da solução parcial, uma operação a cada iteração do algoritmo. Nas linhas 6 a 9, a LRC é computada de acordo com a

procedimento CONSTRUTIVA($S, seed, \mathcal{M}, \mathcal{J}, \mathcal{O}, p, Makespan$)
1 Escolha $\alpha \in [0,1]$ aleatoriamente;
2 $\mathcal{O}_s = \emptyset$;
3 $S = NULL$; $Makespan = 0$;
4 $\mathcal{O}_c = \sigma_1^j \in \mathcal{O}$;
5 **para** $iter = 1, \ldots, |\mathcal{M}| \cdot |\mathcal{J}| - 1$ **faça**
6 $\underline{\sigma} = \text{argmin}(h(\sigma) \mid \sigma \in \mathcal{O}_c)$;
7 $\overline{\sigma} = \text{argmax}(h(\sigma) \mid \sigma \in \mathcal{O}_c)$;
8 $\underline{h} = h(\underline{\sigma})$ e $\overline{h} = h(\overline{\sigma})$;
9 LRC $= \{\sigma_k^j \in \mathcal{O}_c \mid \underline{h} \leq h(\sigma_k^j) \leq \underline{h} + \alpha(\overline{h} - \underline{h})\}$;
10 Escolha $\sigma_k^j \in LRC$ aleatoriamente;
11 $\mathcal{O}_s = \mathcal{O}_s \cup \{\sigma_k^j\}$;
12 $\mathcal{O}_c = \mathcal{O}_c \setminus \{\sigma_k^j\} \cup \{\sigma_{k+1}^j\}$;
13 Incorpore σ_k^j ao escalonamento S e atualize $Makespan$;
14 **fim-para**;
15 Incorpore a última operação $\sigma_k^j \in$ LRC ao escalonamento S e atualize $Makespan$;
fim CONSTRUTIVA

Figura 5.1: Algoritmo construtivo para o JSP.

função gulosa adaptativa utilizada. Na linha 10 uma operação da LRC é escolhida para fazer parte do escalonamento. Os conjuntos de operações já escalonadas e de operações candidatas são atualizados nas linha 11 e 12, respectivamente. Na linha 13 a operação σ_k^j escolhida é inserida no escalonamento e o *makespan* é atualizado. Finalmente, na linha 15 após a inserção de $|\mathcal{M}| \cdot |\mathcal{J}| - 1$ operações na solução, a LRC contém apenas uma operação, que é incorporada ao escalonamento e o *makespan* é atualizado.

5.2.2 Fase de busca local

A busca local é feita através de trocas-(2,2) a partir do modelo de grafo disjuntivo proposto por Roy e Sussmann [121]. Segundo esse modelo, o JSP é representado por um grafo $G = (V, A, E)$, tal que

$$V = \{\mathcal{O} \cup \{0, |\mathcal{J}| \cdot |\mathcal{M}| + 1\}\}$$

é o conjunto de nós, onde 0 e $|\mathcal{J}| \cdot |\mathcal{M}| + 1$ são nós artificiais que representam a fonte e o sumidouro, respectivamente.

$$A = \{(v,w) \mid v, w \in \mathcal{O}, v \prec w\} \cup \{(0,w) \mid w \in \mathcal{O}, \nexists v \in \mathcal{O} \ni v \prec w\} \cup \{(v, |\mathcal{J}| \cdot |\mathcal{M}| + 1) \mid v \in \mathcal{O}, \nexists w \in \mathcal{O} \ni v \prec w\}$$

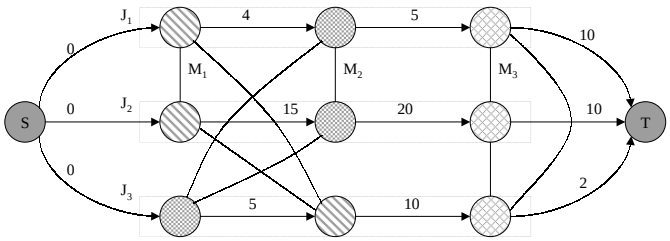

Figura 5.2: Representação de um escalonamento por um grafo disjuntivo.

é o conjunto de arcos que conectam operações consecutivas da mesma tarefa, e

$$E = \{(v, w) \mid \mathcal{M}_v = \mathcal{M}_w\}$$

é o conjunto de arestas que conectam operações que devem ser processadas em uma mesma máquina. O comprimento de um arco ou aresta (v, w), para $v \neq 0$, é igual à duração da operação v. Arcos do tipo $(0, w)$ possuem peso nulo.

Deve-se notar que os arcos em A representam o conjunto de restrições conjuntivas em (5-1a) e as arestas em E representam as restrições disjuntivas em (5-1b). Um exemplo de grafo disjuntivo para um problema teste do JSP com três tarefas e três máquinas é mostrado na Figura 5.2.

Um escalonamento viável consiste em estabelecer uma orientação das arestas em E, de forma que o grafo resultante não tenha ciclos. Dada uma orientação de E, pode-se calcular o tempo de inicio de cada operação, computando-se o caminho mais longo do nó 0 até o nó correspondente à operação. Conseqüentemente, o *makespan* do escalonamento pode ser computado calculando-se o caminho crítico (ou caminho mais longo) do nó 0 até o nó $|\mathcal{J}| \cdot |\mathcal{M}| + 1$. Portanto, o objetivo do JSP é encontrar uma orientação de E de tal forma que o caminho mais longo em G seja minimizado. Na Figura 5.3 as arestas de E são orientadas e o caminho crítico é calculado. Deve-se observar que os valores mostrados dentro de cada círculo correspondem ao valor do caminho mais longo do nó de origem S até o nó correspondente. O caminho mais longo de S até T corresponde ao *makespan* do escalonamento mostrado na figura.

Em Taillard [126] é descrito um algoritmo de complexidade $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ para computar o caminho mais longo em G. Caso todas as arestas disjuntivas redundantes sejam eliminadas, cada operação (exceto as artificiais) terá no máximo dois sucessores e dois predecessores. Para descrever o algoritmo usado para encontrar o caminho mais longo em G, será usada a notação descrita a seguir. Para cada operação σ_i $(1 \leq i \leq |\mathcal{J}| \cdot |\mathcal{M}|)$, define-se:

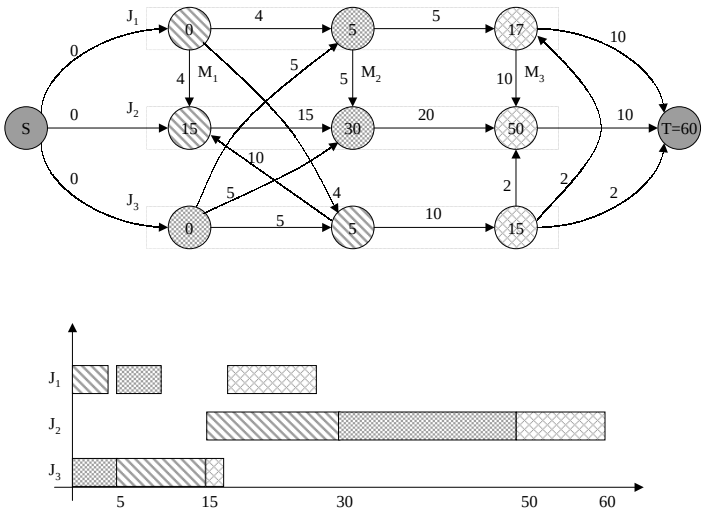

Figura 5.3: Caminho crítico com *makespan* igual a 60.

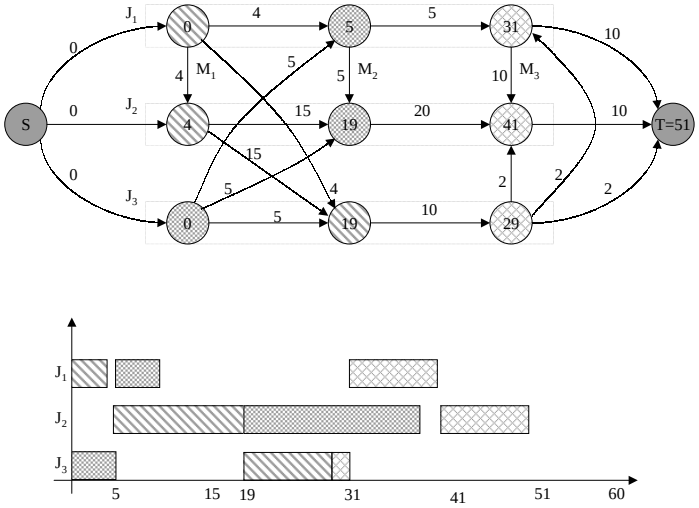

Figura 5.4: Caminho crítico recalculado com *makespan* igual a 51.

- J_i: tarefa da qual σ_i faz parte;
- M_i: máquina onde σ_i deve ser processada;
- p_i: tempo de processamento de σ_i;
- PJ_i: (caso exista) operação pertencente à tarefa J_i, processada antes de σ_i;
- SJ_i: (caso exista) operação pertencente à tarefa J_i, processada após σ_i;
- PM_i: (caso exista) operação processada na máquina M_i antes de σ_i;
- SM_i: (caso exista) operação processada na máquina M_i após σ_i;
- t_i: instante inicial da operação σ_i; e

```
procedimento CALCULA_INICIO_OPERAÇÕES (S, M, J, O, p)
1     Seja Q = {i | PM_i e PJ_i não são definidos};
2     para oper = 1,..., |M| · |J| faça
3           processado(oper) = falso;
4     fim-para;
5     enquanto Q ≠ ∅ faça
6           Q = Q \ {i};
7           processado(i) = verdadeiro;
8           t_i = max(t_PM_i + p_PM_i, t_PJ_i + p_PJ_i);
9           se processado(PM_SJ_i) então
10                Q = Q ∪ {SJ_i};
11          fim-se;
12          se processado(PJ_SM_i) então
13                Q = Q ∪ {SM_i};
14          fim-se;
15    fim-enquanto
fim CALCULA_INICIO_OPERAÇÕES
```

Figura 5.5: Algoritmo para a obtenção do tempo inicial de cada operação em um escalonamento viável.

– q_i: comprimento do caminho mais longo da operação σ_i até o nó $|\mathcal{J}| \cdot |\mathcal{M}| + 1$.

Uma operação σ_i é crítica, caso ela pertença ao caminho mais longo, isto é, caso

$$t_i + q_i = C_{max}.$$

Para cada operação σ_i, tem-se as seguintes identidades (onde $p_i = t_i = q_i = 0$, quando a operação σ_i não é definida):

$$t_i = max(t_{PM_i} + p_{PM_i}, t_{PJ_i} + p_{PJ_i}),$$
$$q_i = max(q_{SM_i}, q_{SJ_i}) + p_i.$$

O algoritmo proposto por Taillard [126] para calcular os instantes iniciais t_i de cada operação σ_i é mostrado na Figura 5.5. Seja Q inicialmente o conjunto de operações para as quais t_i pode ser calculado, ou seja, operações que têm o nó 0 como único antecessor em G (linha 1). Todas as operações são marcadas inicialmente como não processadas (nas linhas 2 a 4). Uma operação σ_i é processada em cada iteração do laço das linhas 5 a 15. O instante inicial de σ_i é calculado na linha 8. Nas linhas 9 a 11, tenta-se incluir a operação SJ_i em Q. Para que SJ_i possa ser incluída em Q é necessário que a operação processada antes de SJ_i na mesma máquina, já tenha sido marcada como processada. Nas linhas 12 a 14, procura-se incluir a operação SM_i em Q. Para que SM_i possa ser incluída em Q é necessário

que a operação processada antes de SM_i na mesma tarefa já tenha sido marcada como processada. O algoritmo termina quando todas as operações tiverem sido marcadas como processadas.

O cálculo dos valores de q_i é feito de forma semelhante ao cálculo dos valores de p_i. Porém, o percurso é iniciado a partir do nó $|\mathcal{J}| \cdot |\mathcal{M}| + 1$ em direção a cada nó do grafo e os arcos são percorridos no sentido contrário.

O procedimento de busca local usa uma vizinhança onde cada solução é gerada pela inversão de duas operações críticas consecutivas que são executadas na mesma máquina. As soluções dessa vizinhança não possuem ciclos [126]. O cálculo de um limite inferior para o *makespan* das soluções de uma vizinhança é feito de forma eficiente em tempo $O(|\mathcal{J}| \cdot |\mathcal{M}|)$. Sejam duas operações críticas σ_a e σ_b ($\sigma_b = SM_a$) que serão invertidas. Os novos valores de t e q são calculados da seguinte forma:

$$
\begin{aligned}
t'_b &= max(t_{PM_a} + p_{PM_a}, t_{PJ_b} + p_{PJ_b}),\\
t'_a &= max(t'_b + p_b, t_{PJ_a} + p_{PJ_a}),\\
q'_a &= max(q_{SM_b}, q_{SJ_a}) + p_a,\\
q'_b &= max(q'_a, q_{SJ_b}) + p_b.
\end{aligned}
$$

$C'_{max} = (t'_b + q'_b, t'_a + q'_a)$ será o valor do novo caminho mais longo caso ele passe por σ_a ou σ_b, ou um limite inferior do novo valor do *makespan*, caso contrário. Um exemplo de movimento é mostrado na Figura 5.4, onde a segunda operação da tarefa 3 e a primeira operação da tarefa 1 foram invertidas, reduzindo o *makespan* de 60 para 51.

Inicialmente, o procedimento de busca local identifica o caminho crítico no grafo correspondente ao escalonamento produzido durante a fase construtiva. Movimentos correspondentes a permutações de pares de operações consecutivas processadas na mesma máquina que estejam no caminho crítico são avaliados. O primeiro movimento encontrado que produza $C'_{max} < C_{max}$, onde C_{max} é o *makespan* da solução corrente, é efetuado. A cada vez que um movimento é feito, o caminho crítico é recalculado e a busca é reiniciada. Caso não seja mais possível reduzir o *makespan* através de permutações de duas operações, o escalonamento corrente é localmente ótimo e a busca local é finalizada.

procedimento RELIGAMENTO $(\mathcal{M}, \mathcal{J}, \mathcal{O}, p, Makespan, S, T)$
1 $S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,|\mathcal{J}|}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,|\mathcal{J}|}), \ldots, (j^S_{|\mathcal{M}|,1}, j^S_{|\mathcal{M}|,2}, \ldots, j^S_{|\mathcal{M}|,|\mathcal{J}|})\}$;
2 $T = \{(j^T_{1,1}, j^T_{1,2}, \ldots, j^T_{1,|\mathcal{J}|}), (j^T_{2,1}, j^T_{2,2}, \ldots, j^T_{2,|\mathcal{J}|}), \ldots, (j^T_{|\mathcal{M}|,1}, j^T_{|\mathcal{M}|,2}, \ldots, j^T_{|\mathcal{M}|,|\mathcal{J}|})\}$;
3 $c_{gmin} = Makespan$; $S_{gmin} = S$;
4 **para** $k = 1, \ldots, |\mathcal{M}|$ **faça**
5 $\quad \delta_k^{S,T} = \{i = 1, \ldots, |\mathcal{J}| \mid j^S_{k,i} \neq j^T_{k,i}\}$;
6 **fim-para**;
7 **enquanto** $(\sum_{k=1}^{|\mathcal{M}|} |\delta_k^{S,T}| > 2)$ **faça**
8 $\quad c_{min} = \infty$;
9 $\quad$ **para** $k = 1, \ldots, |\mathcal{M}|$ **faça**
10 $\quad\quad$ **para** $i \in \delta_k^{S,T}$ **faça**
11 $\quad\quad\quad$ Seja q tal que $j^T_{k,q} == j^S_{k,i}$;
12 $\quad\quad\quad \bar{S} = S \setminus \{(\ldots, j^S_{k,i}, j^S_{k,i+1}, \ldots, j^S_{k,q-1}, j^S_{k,q}, \ldots)\}$;
13 $\quad\quad\quad \bar{S} = \bar{S} \cup \{(\ldots, j^S_{k,q}, j^S_{k,i+1}, \ldots, j^S_{k,q-1}, j^S_{k,i}, \ldots)\}$;
14 $\quad\quad\quad \bar{c} =$ CALCULE_MAKESPAN$(\bar{S})$;
15 $\quad\quad\quad$ **se** $\bar{c} \leq c_{min}$ **então**
16 $\quad\quad\quad\quad c_{min} = \bar{c}$;
17 $\quad\quad\quad\quad S_{min} = \bar{S}$;
18 $\quad\quad\quad\quad i_{min} = i$;
19 $\quad\quad\quad\quad k_{min} = k$;
20 $\quad\quad\quad$ **fim-se**;
21 $\quad\quad$ **fim-para**;
22 $\quad$ **fim-para**;
23 $\quad S = S_{min}$; $Makespan = c_{min}$;
24 $\quad \delta_{k_{min}}^{S,T} = \delta_{k_{min}}^{S,T} \setminus \{i_{min}\}$;
25 $\quad$ **se** $Makespan \leq c_{gmin}$ **então**
26 $\quad\quad c_{gmin} = Makespan$;
27 $\quad\quad S_{gmin} = S$;
28 $\quad$ **fim-se**;
29 **fim-enquanto**;
30 **retorne** (S_{gmin});
fim RELIGAMENTO

Figura 5.6: Religamento de caminhos entre a solução inicial S e a solução guia T.

5.3 Religamento de caminhos para o problema de job-shop scheduling

Para descrever o religamento de caminhos para o JSP, um escalonamento será representado por permutações das operações das tarefas em $\mathcal{J}$ nas máquinas em $\mathcal{M}$. Cada vetor de permutação corresponde a uma máquina. Dessa forma, existem $|\mathcal{M}|$ vetores de permutação, formados por $|\mathcal{J}|$ elementos. Cada permutação implica em uma ordenação das operações na máquina correspondente. Uma solução do JSP será representada por:

$$S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,|\mathcal{J}|}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,|\mathcal{J}|}), \ldots, (j^S_{|\mathcal{M}|,1}, j^S_{|\mathcal{M}|,2}, \ldots, j^S_{|\mathcal{M}|,|\mathcal{J}|})\},$$

onde $j^S_{i,k}$ representa a k-ésima tarefa que será executada na máquina i na solução S.

A técnica de religamento de caminhos consiste em explorar trajetórias que conectam uma *solução inicial* a uma *solução guia*. Para isso, são selecionados 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. Para o JSP, o religamento de caminhos é feito entre uma solução inicial

$$S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,|\mathcal{J}|}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,|\mathcal{J}|}), \ldots, (j^S_{|\mathcal{M}|,1}, j^S_{|\mathcal{M}|,2}, \ldots, j^S_{|\mathcal{M}|,|\mathcal{J}|})\}$$

e uma solução guia

$$T = \{(j^T_{1,1}, j^T_{1,2}, \ldots, j^T_{1,|\mathcal{J}|}), (j^T_{2,1}, j^T_{2,2}, \ldots, j^T_{2,|\mathcal{J}|}), \ldots, (j^T_{|\mathcal{M}|,1}, j^T_{|\mathcal{M}|,2}, \ldots, j^T_{|\mathcal{M}|,|\mathcal{J}|})\}.$$

O pseudo-código do procedimento de religamento de caminhos para o JSP é mostrado na Figura 5.7.

Seja a diferença entre as soluções S e T definida da seguinte maneira:

$$\delta^{S,T}_k = \{i = 1, \ldots, |\mathcal{J}| \mid j^S_{k,i} \neq j^T_{k,i}\}, k = 1, \ldots, |\mathcal{M}|.$$

Esses conjuntos são computados nas linhas 4 a 6 do pseudo-código.

Uma solução intermediária do caminho é visitada a cada passo do laço das linhas 7 a 29. Em um movimento, um vetor de permutação da solução S, dado por

$$(\ldots, j^S_{k,i}, j^S_{k,i+1}, \ldots, j^S_{k,q-1}, j^S_{k,q}, \ldots),$$

é substituído pelo vetor

$$(\ldots, j^S_{k,q}, j^S_{k,i+1}, \ldots, j^S_{k,q-1}, j^S_{k,i}, \ldots),$$

invertendo-se as posições das operações $j^S_{k,i}$ e $j^S_{k,q}$, onde $i \in \delta^{S,T}_k$ e q é tal que $j^T_{k,q} = j^S_{k,i}$. Observa-se que soluções que violam as restrições de precedência podem ser geradas por esse tipo de movimento. A viabilidade de uma solução S é verificada dentro do procedimento `CALCULE_MAKESPAN`(S) (linha 14). Esse procedimento consiste em calcular o caminho crítico até o nó destino do grafo construído, conforme apresentado na seção 5.2.2, usando o algoritmo da Figura 5.5. Um escalonamento inviável é detectado quando encontra-se um ciclo no grafo correspondente. Um custo suficientemente alto é atribuído a soluções inviáveis, para evitar que sejam visitadas pelo religamento de caminhos.

Em cada passo, o movimento que produz a solução de menor custo é selecionado e o seu índice é removido do conjunto $\delta^{S,T}_k$ correspondente (linha 24). Esse processo continua até que existam apenas dois índices de movimento em um dos conjuntos $\delta^{S,T}_k$. Nesse ponto, esse movimento resultará na *solução guia* e, por esse motivo, não é realizado. A melhor solução encontrada no caminho (S_{gmin}) é retornada pelo procedimento.

Na implementação proposta para o JSP, um *pool* P de soluções de elite é formado pelas soluções encontradas nas primeiras $|P|$ iterações do método GRASP. Após essa fase inicial, uma solução s_g é religada com uma ou mais soluções de elite s_e em P. O religamento de caminhos é realizado em dois sentidos: da solução s_g obtida ao final da busca local para a solução s_e do *pool* e da solução s_e para a solução s_g. Isso é feito porque esses caminhos freqüentemente visitam soluções intermediárias diferentes.

Da mesma forma que na heurística GRASP híbrida proposta para o AP3, a estratégia híbrida proposta para o JSP usa a 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

$$S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,|\mathcal{J}|}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,|\mathcal{J}|}), \ldots, (j^S_{|\mathcal{M}|,1}, j^S_{|\mathcal{M}|,2}, \ldots, j^S_{|\mathcal{M}|,|\mathcal{J}|})\}$$

e

$$T = \{(j^T_{1,1}, j^T_{1,2}, \ldots, j^T_{1,|\mathcal{J}|}), (j^T_{2,1}, j^T_{2,2}, \ldots, j^T_{2,|\mathcal{J}|}), \ldots, (j^T_{|\mathcal{M}|,1}, j^T_{|\mathcal{M}|,2}, \ldots, j^T_{|\mathcal{M}|,|\mathcal{J}|})\},$$

seja

$$\Delta(S,T) = \sum_{k=1}^{|\mathcal{M}|} |\delta^{S,T}_k|$$

uma medida da diferença entre S e T.

A solução S_{gmin} produzida pelo religamento de caminhos é candidata à inserção no *pool*. Ela 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 $\Delta(S_{gmin}, S_p) > \Delta_{\min}$ para todas as soluções de elite $S_p \in P$, onde $\Delta_{\min}$ é um fator de diferenciação para inserção no *pool*, fornecido como entrada, isto é, S_{gmin} é melhor do que a pior solução em P e difere significativamente de todas as soluções de elite.

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

Deve-se notar que caso o número de movimentos necessários para percorrer a trajetória de $S \in P$ a S_{gmin} seja no máximo $\Delta_{\min}/2$, então a solução S_{gmin} será tal que $\Delta(S, S_{gmin}) \leq \Delta_{\min}$. Portanto, a diferença entre S_{gmin} e as soluções de P não precisa ser calculada e S_{gmin} somente será incluída no conjunto de elites se o seu custo for menor do que o custo da melhor solução em P.

No algoritmo proposto para o AP3 no capítulo anterior, o custo de uma solução na vizinhança de S era atualizado em $O(1)$ a partir do custo de S. No JSP, o custo de cada solução visitada pelo religamento de caminhos é calculado em $O(|\mathcal{J}| \cdot |\mathcal{M}|)$, através do algoritmo descrito em [126]. Por esse motivo, fazer o religamento de caminhos ao final de cada iteração da heurística para o JSP é computacionalmente muito trabalhoso. Optou-se, então, por realizar o religamento de caminhos apenas em iterações onde a solução gerada após a busca local atenda a um critério de qualidade. Para isso, a média μ_n dos custos das soluções geradas durante as n primeiras iterações é computada:

$$\mu_n = \frac{\sum_{i=1}^{n} c(S_i)}{n},$$

onde $c(S_i)$ é o custo da solução obtida na i-ésima iteração. Computa-se também o desvio padrão σ_n desses custos:

$$\sigma_n = \sqrt{\frac{\sum_{i=1}^{n} (c(S_i) - \mu_n)^2}{n-1}}.$$

Soluções S_i que atendem ao critério de qualidade são tais que:

1. $c(S_i) \leq c_{pior}$, para $i \leq n$;

2. $c(S_i) \leq \max(c_{pior}, \mu_n - 2 \cdot \sigma_n)$, para $i > n$.

O religamento de caminhos também pode ser usado como uma fase de intensificação no conjunto de soluções de elite, conforme foi feito no capítulo anterior para o AP3. Isso é feito aplicando-se o religamento de caminhos a cada par de soluções de elite em P e atualizando P, caso seja necessário. Esse procedimento é repetido até que P estabilize. Esse procedimento de intensificação pode ser usado em uma fase de pós-otimização (usando o *pool* de soluções do final) ou durante a otimização (usando o conjunto corrente de soluções de elite).

Durante a otimização, o procedimento de intensificação é executado a cada intervalo de *ifreq* iterações. Após uma etapa de intensificação, caso P não tenha sido modificado por mais de *ifreq* iterações, atribui-se um valor suficientemente alto aos custos das $|P|/2$ piores soluções de P para garantir que as soluções do *pool* sejam renovadas. As $|P|/2$ piores soluções de P são eventualmente substituídas por soluções geradas nas iterações seguintes pelo algoritmo GRASP com religamento de caminhos. Portanto, soluções com custo alto, porém suficientemente diferentes das soluções em P são aceitas para inserção no *pool*.

A etapa de pós-otimização usada na abordagem híbrida para o JSP é idêntica à fase de pós-otimização proposta para o AP3. Essa etapa consiste em aplicar o procedimento de intensificação no *pool* de soluções resultante ao final do algoritmo GRASP. Em seguida, aplica-se o procedimento de busca local explicado na Subseção 5.2.2 a cada solução de elite. 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 seu início.

5.4 GRASP com religamento de caminhos

Nessa seção será mostrado como os procedimentos descritos anteriormente são inseridos no algoritmo GRASP. Na Figura 5.7 o algoritmo GRASP híbrido para o JSP é descrito em pseudo-código. Seja *maxpool* o tamanho do conjunto de elites P. As primeiras *maxpool* iterações contribuem cada uma com uma solução para o conjunto de elite (linha 19). O religamento de caminhos não é realizado até que o conjunto de soluções de elite esteja completo.

O algoritmo usa dois critérios de parada. Ele é interrompido caso uma solução com valor objetivo menor ou igual a *alvo* tenha sido encontrada, ou após executar *maxitr* iterações. Em cada iteração uma solução é construída

```
procedimento GRASP_RC_JSP (M, J, O, p, alvo, maxitr, maxpool, ifreq)
1     P = ∅;
2     para i = 1, ..., maxitr faça
3           se  mod (i, 2) == 0 então
4                 CONSTRUTIVA_MAKESPAN(S, seed, M, J, O, p, Makespan);
5           senão
6                 CONSTRUTIVA_TEMPO_RESTANTE(S, seed, M, J, O, p, Makespan);
7           fim-se;
8           LOCAL(S, M, J, O, p, Makespan);
9           se |P| == maxpool então
10                aceito = VERIFICAR_QUALIDADE(S, i);
11                se aceito então
12                      para T ∈ P' ⊆ P faça
13                            S_gmin = RELIGAMENTO(M, J, O, p, Makespan, S, T);
14                            ATUALIZAR_POOL(S_gmin, c_gmin, P);
15                            S_gmin = RELIGAMENTO(M, J, O, p, Makespan, T, S);
16                            ATUALIZAR_POOL(S_gmin, c_gmin, P);
17                      fim-para;
18                fim-se;
19          senão P = P ∪ {S} fim-se;
20          se  mod (i,ifreq) == 0 então INTENSIFICAÇÃO(P) fim-se;
21          S_melhor = POOLMIN(P);
22          se MAKESPAN(S_melhor) ≤ alvo então saia do laço; fim-se;
23    fim-para;
24    POS-OTIMIZAçãO(P);
25    S_melhor = POOLMIN(P);
26    retorne (S_melhor);
fim GRASP_RC_JSP;
```

Figura 5.7: Algoritmo GRASP com religamento de caminhos bidirecional para o JSP.

usando-se de forma alternada os dois algoritmos descritos na Seção 5.2.1 (linhas 3 a 7). Iterações ímpares constroem soluções usando a função gulosa por *tempo restante* e iterações pares constroem soluções usando a função gulosa por *makespan*. A busca local é aplicada na linha 8 à solução construída.

Uma vez que o *pool* de soluções de elite esteja completo, verifica-se se a solução do algoritmo GRASP atende ao critério de qualidade (linha 10) e, em caso positivo, a fase de religamento de caminhos é realizada (linhas 12 a 17). 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', considerando-se os dois sentidos (linhas 13 e 15). O subconjunto 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 14 e 16).

A cada intervalo de *ifreq* iterações, a fase de intensificação no conjunto de elite, descrita na Seção 5.3, é computada (linha 20). Finalmente, quando um dos critérios de parada é satisfeito, computa-se a fase de pós-otimização também descrita na Seção 5.3 (linha 24).

5.5 GRASP para o JSP com intensificação e POP durante a fase construtiva

Nessa seção será descrita a abordagem de GRASP proposta por Binato et al. [16]. Essa abordagem incorpora à fase construtiva uma estratégia de intensificação e o POP (*Proximate Optimality Principle*). Esses dois conceitos foram propostos por Fleurent e Glover [49] para o problema de atribuição quadrática. A fase construtiva é feita pelo algoritmo apresentado na Seção 5.2.1, usando a função gulosa por *makespan* e acrescentando-se os mecanismos para fazer a intensificação e o POP. A fase de busca local usa o algoritmo proposto por Taillard [126], apresentado na Seção 5.2.2. Algumas particularidades da heurística GRASP de Binato et al. são explicadas a seguir.

Intensificação

O método proposto em [16] usa informações obtidas a partir de soluções já visitadas para influenciar as decisões tomadas em iterações subseqüentes. Isso é feito através do uso de uma memória de longo prazo, onde são armazenadas informações sobre atributos de soluções de qualidade anteriormente visitadas. Essas informações são usadas para alterar as probabilidades de escolha atribuídas a cada elemento da LRC durante a fase construtiva.

A estratégia de intensificação mantém um conjunto $\mathcal{E}$ de até q soluções de elite distintas, que são usadas para guiar a fase construtiva. Antes de ser inserido no conjunto de soluções de elite, um escalonamento encontrado em uma iteração do algoritmo GRASP é submetido a um critério idêntico ao critério de aceitação no *pool*, apresentado na Seção 5.3. Na Seção 5.3, a medida de diferenciação entre dois escalonamentos é dada comparando-se a ordem das operações processadas em cada máquina nos dois escalonamentos. Na abordagem de Binato et al., a medida de diferenciação entre dois escalonamentos é calculada comparando-se os instantes iniciais de cada operação nos dois escalonamentos. Dessa forma, uma medida da diferenciação

entre dois escalonamentos é dada pelo número de instantes iniciais distintos entre operações correspondentes nos dois escalonamentos.

Assim como em Fleurent e Glover [49], na abordagem de Binato et al. são criados incentivos durante a fase construtiva para favorecer a escolha de atributos das soluções de elite para inserção na solução que está sendo construída. Sejam $t(\mathcal{S}, \sigma)$ o instante inicial da operação σ no escalonamento $\mathcal{S}$ e $\mathcal{S}_p$ o escalonamento parcial que está sendo construído. Para $\sigma \in$ LRC, um índice de *intensidade* é definido como

$$\mathcal{I}(\sigma) = \sum_{S \in \mathcal{H}} \frac{C_{\max}(\mathcal{E}^*)}{C_{\max}(\mathcal{S})},$$

onde $C_{\max}(\mathcal{E}^*)$ é o *makespan* da melhor solução de elite $\mathcal{E}^*$ em $\mathcal{E}$, $C_{\max}(\mathcal{S})$ é o *makespan* do escalonamento $\mathcal{S}$ e

$$\mathcal{H} = \{\mathcal{S} \in \mathcal{E} \mid t(\mathcal{S}, \sigma) = t(\mathcal{S}_p, \sigma), \text{para } \sigma \in \text{LRC}\}$$

é o conjunto de escalonamentos de elite S, onde o instante inicial da operação σ é idêntico ao tempo inicial da operação σ caso ela seja inserida no escalonamento parcial S_p. Observa-se que a *intensidade* da operação σ aumenta com o número de escalonamentos de elite que possuem instantes iniciais idênticos para essa operação.

A função de *intensidade* é usada para alterar a função gulosa usada na fase construtiva do algoritmo GRASP. Para isso, uma nova função heurística

$$h'(\sigma) = \lambda \frac{\underline{h}}{h(\sigma)} + \mathcal{I}(\sigma) \tag{5-2}$$

é definida para toda operação $\sigma \in$ LRC onde $\underline{h} = h(\underline{\sigma})$ e λ é um parâmetro usado para levar em conta o fato de que $0 < \underline{h}/h(\sigma) \leq 1$ e $0 \leq \mathcal{I}(\sigma) \leq q$, onde q é o número de escalonamentos de elite. O parâmetro λ varia no decorrer das iterações do algoritmo para dar maior ênfase a $\underline{h}/h(\sigma)$ ou à função de *intensidade*.

Uso de probabilidades para seleção de um elemento da LRC

Em um algoritmo GRASP básico e no algoritmo proposto nesse capítulo, cada elemento da LRC possui a mesma probabilidade de ser escolhido. Na abordagem proposta por Binato et al. [16] são usadas distribuições de probabilidade para guiar a seleção de um candidato da LRC, criando diferentes probabilidades de escolha para cada elemento. Essa estratégia foi proposta por Bresina [20] e é explicada na Subseção 2.3.2.

Deve-se notar que a estratégia originalmente proposta por Bresina calcula probabilidades de escolha para todos os elementos da lista de candidatos. Em [16], essas probabilidades são calculadas apenas para elementos da LRC e um elemento é selecionado da forma descrita a seguir. Inicialmente, a LRC é computada usando-se a função gulosa por *makespan* como mostrado na Seção 5.2.1. Em seguida, a função heurística gulosa dada por (5-2) é calculada para todas as operações da LRC. As operações são ordenadas de acordo com os novos valores das suas funções heurísticas. A probabilidade de selecionar cada operação da LRC é avaliada e uma operação é escolhida.

POP

Na fase construtiva de um algoritmo GRASP para o JSP, uma solução é construída selecionando-se uma operação a cada iteração. Segundo o POP (*Proximate Optimality Principle*) [57] aplicado ao JSP, soluções parciais de qualidade com v operações estão próximas de soluções parciais de qualidade com $v+1$ operações. Um erro no escalonamento de uma operação no início do processo construtivo pode levar à inserção de outros erros na solução. Em uma heurística GRASP básica, passos incorretos são corrigidos com um procedimento de busca local após a construção da solução. Porém, devido à natureza dos procedimentos de busca local (tais como trocas-(2,2)), pode ser difícil identificar e desfazer decisões incorretas.

O POP no contexto do JSP consiste em, após um dado número de operações terem sido escalonadas, aplicar o algoritmo de busca local à solução parcial com o objetivo de reduzir o seu *makespan*. A busca local é realizada sobre um grafo construído de forma semelhante ao grafo apresentado na Seção 5.2.2. Sejam $\mathcal{S}_p$ um escalonamento parcial e $\mathcal{O}_p$ as suas operações correspondentes. O grafo é construído usando-se apenas operações que já tenham sido escalonadas (operações do conjunto $\mathcal{O}_p$), enquanto as operações restantes são agrupadas no nó destino. Dessa forma, apenas operações que já tenham sido escalonadas serão consideradas durante a busca local. Após a execução do POP no escalonamento parcial, o processo de inserção de operações na solução é retomado.

O POP é computacionalmente trabalhoso e por isso não é aplicado à solução parcial a cada iteração. Na implementação proposta em [16], um parâmetro $freq$ é usado para determinar a freqüência de aplicação do POP.

5.6 Resultados computacionais

Nessa seção serão apresentados os resultados computacionais para a heurística para o JSP proposta nesse capítulo. O ambiente computacional usado para conduzir os experimentos será descrito, assim como os problemas testes usados para estudar a heurística proposta. Serão apresentados os resultados da comparação entre as seguintes variantes:

1. `GP`: é a heurística GRASP pura que constrói soluções usando de forma alternada as duas funções gulosas apresentadas na Seção 5.2 e usa o algoritmo de busca local descrito em [126].

2. `GP+RC`: é a heurística GRASP com religamento de caminhos proposta nesse capítulo;

3. `GP+C-INT`: é a heurística GRASP com intensificação e POP realizados durante a fase construtiva [16].

5.6.1 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].

5.6.2 Problemas testes

Os experimentos foram realizados em 66 problemas testes das cinco classes de problemas padrões para o JSP: `abz`, `car`, `la`, `mt` e `orb`. A dimensão dos problemas variam de 6 a 30 tarefas e de 4 a 20 máquinas. Todos os problemas testes estudados foram obtidos da OR-Library mantida por Beasley [1] [15].

[1] `http://mscmga.ms.ic.ac.uk/jeb/orlib/jobshopinfo.html`

Tabela 5.1: Estimativas das probabilidades de encontrar uma solução melhor ou igual ao valor alvo, em função do tempo máximo de solução. Os problemas testes estudados são `abz6`, `mt10`, `orb5` e `la21`, para os valores alvos 947, 950, 910 e 1110, respectivamente.

tempo	abz6		mt10		orb5		la21	
(s)	GP	GP+RC	GP	GP+RC	GP	GP+RC	GP	GP+RC
100	0.09	0.32	0.03	0.01	0.42	0.61	0.10	0.11
500	0.48	0.96	0.19	0.50	0.92	1.00	0.36	0.89
1000	0.74	1.00	0.37	0.87	0.98	1.00	0.56	1.00
1500	0.84	1.00	0.54	0.97	0.99	1.00	0.69	1.00

5.6.3 Os experimentos

O objetivo desses experimentos é observar o comportamento geral do algoritmo proposto. Pretende-se verificar como as soluções obtidas por `GP+RC` se comparam às melhores soluções conhecidas para um conjunto padrão de problemas testes. O tempo para solução alvo de `GP+RC` será comparado ao tempo de solução de `GP` e de `GP+INT` com a finalidade de verificar a eficiência da heurística GRASP com religamento de caminhos proposta em relação a outras variantes.

Nos experimentos realizados com `GP+RC`, 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 e o fator de diferenciação para inserção no *pool* como $\Delta_{\min} = 0.25 \cdot |\mathcal{J}| \cdot |\mathcal{M}|$. O religamento de caminhos bidirecional é feito entre a solução produzida pela heurística GRASP e todas as soluções armazenadas no *pool*. O desvio padrão usado para verificar se uma solução obtida ao final da busca local será submetida ao procedimento de religamento de caminhos é calculado a partir dos custos das primeiras 10000 soluções. As execuções de `GP+RC` usadas para plotar os gráficos apresentados nesse capítulo não fazem intensificação nem pós-otimização. As execuções de `GP+RC` mostradas nas Tabelas 5.4, 5.5 e 5.6 usadas para testar cada classe de problemas testes fazem a intensificação a cada intervalo de *ifreq*=500000 iterações.

Em todos os testes realizados com `GP+C-INT` nesse capítulo, o programa foi configurado para usar os mesmos valores de parâmetros usados nos testes apresentados em [16]. Portanto, um *pool* de 30 soluções de elite foi mantido para fazer a intensificação e foi usado um fator de diferenciação para inserção no *pool* de $\Delta_{\min} = 0.8 \cdot |\mathcal{J}| \cdot |\mathcal{M}|$. O POP foi executado após a construção de 40% e de 80% da solução parcial. Uma distribuição linear foi usada para guiar a seleção de elementos da LRC.

O efeito do religamento de caminhos na heurística GRASP para o

Tabela 5.2: Tempo para encontrar uma solução com custo melhor ou igual ao valor alvo, em função da probabilidade. O problema teste `mt10` foi testado para os valores alvos 970, 960 e 950. Para cada valor alvo testado, é mostrado o decréscimo percentual do tempo de solução de `GP+RC` em relação ao tempo de solução de `GP`.

	alvo=970			alvo=960		
probabilidade	GP (s)	GP+RC (s)	red.(%)	GP (s)	GP+RC (s)	red.(%)
0.2	44.92	144.61	-221.92	214.94	198.75	7.53
0.5	163.93	211.66	-29.11	667.41	295.71	55.69
0.8	386.94	292.98	24.28	1362.65	416.35	69.44
	alvo=950					
probabilidade	GP (s)	GP+RC (s)	red.(%)			
0.2	546.45	263.02	51.86			
0.5	1422.20	394.51	72.26			
0.8	2951.01	578.53	80.39			

JSP foi estudado nessa seção. Para isso, as variantes `GP` e `GP+RC` foram comparadas para os problemas `abz6`, `mt10`, `orb5` e `la21`. Duzentas execuções independentes das duas variantes foram feitas para esses quatro problemas. Cada execução foi interrompida quando uma solução com custo igual ou inferior ao valor alvo `alvo` foi encontrada. Os valores de `alvo` testados para os problemas `abz6`, `mt10`, `orb5` e `la21` foram 947, 950, 910 e 1110, respectivamente. Esses valores estão longe do valor ótimo e, em geral, podem ser obtidos em poucas iterações. A Figura 5.8 mostra as distribuições empíricas de probabilidade para tempo de solução para `GP` e `GP+RC`. Essas distribuições empíricas são plotadas ordenando-se os tempos medidos e, em seguida, associando-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 $i = 1, \ldots, 200$. Observa-se nesses gráficos que a variante `GP+RC` obtém a solução alvo mais rapidamente, quando comparada a `GP`. Na Tabela 5.1 são mostradas probabilidades para `GP` e `GP+RC` obterem a solução alvo em função do tempo, para os quatro problemas testes considerados. Por exemplo, considerando-se um tempo de execução de no máximo 500 segundos, a probabilidade de encontrar uma solução pelo menos tão boa quanto a solução alvo para o problema `abz6` é 96% para `GP+RC`, enquanto que para `GP` é 48%. Para o problema `la21`, a probabilidade de obter uma solução de custo melhor ou igual ao alvo é de 56% para `GP` e 100% para `GP+RC`, para um tempo de execução de no máximo 1000 segundos. Esses resultados comprovam o fato observado no capítulo anterior de que, apesar da abordagem GRASP com religamento de caminhos precisar de mais tempo por iteração do que um algoritmo puro, esse tempo é compensado pelo menor número de iterações necessárias para obter uma solução de custo melhor ou igual ao alvo.

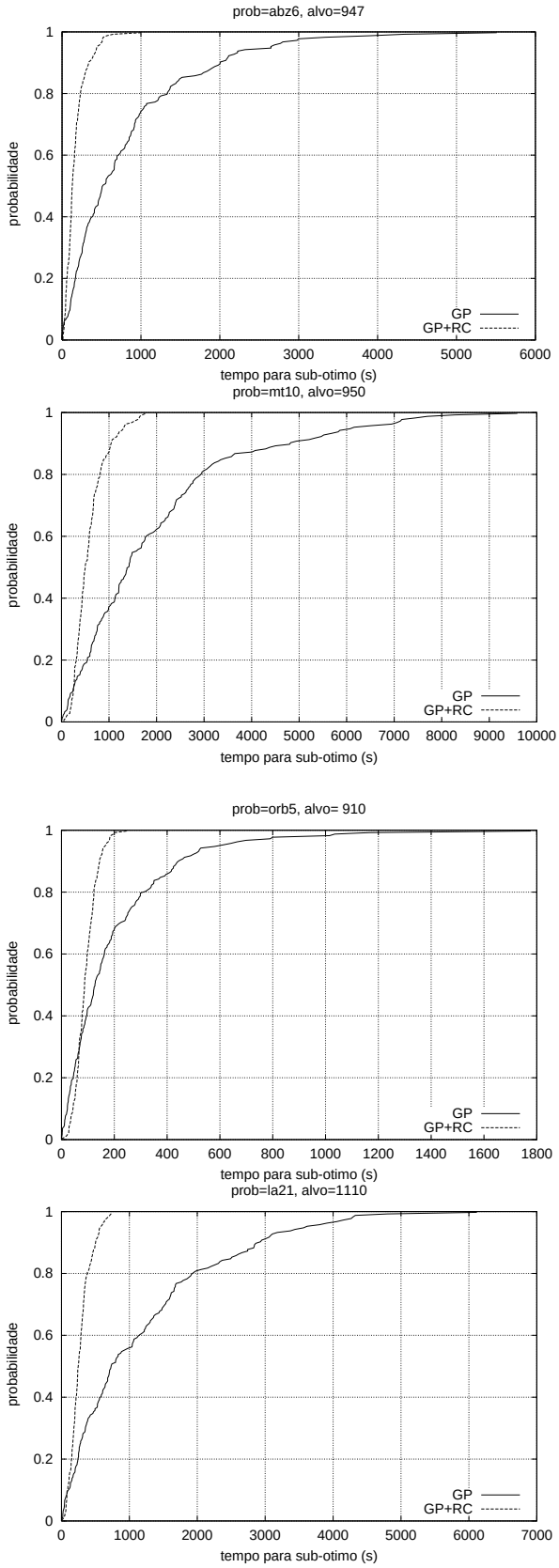

Figura 5.8: Distribuição empírica de probabilidade do tempo para valor alvo para `GP` e `GP+RC`, problemas `abz6`, `mt10`, `orb5` e `la21`.

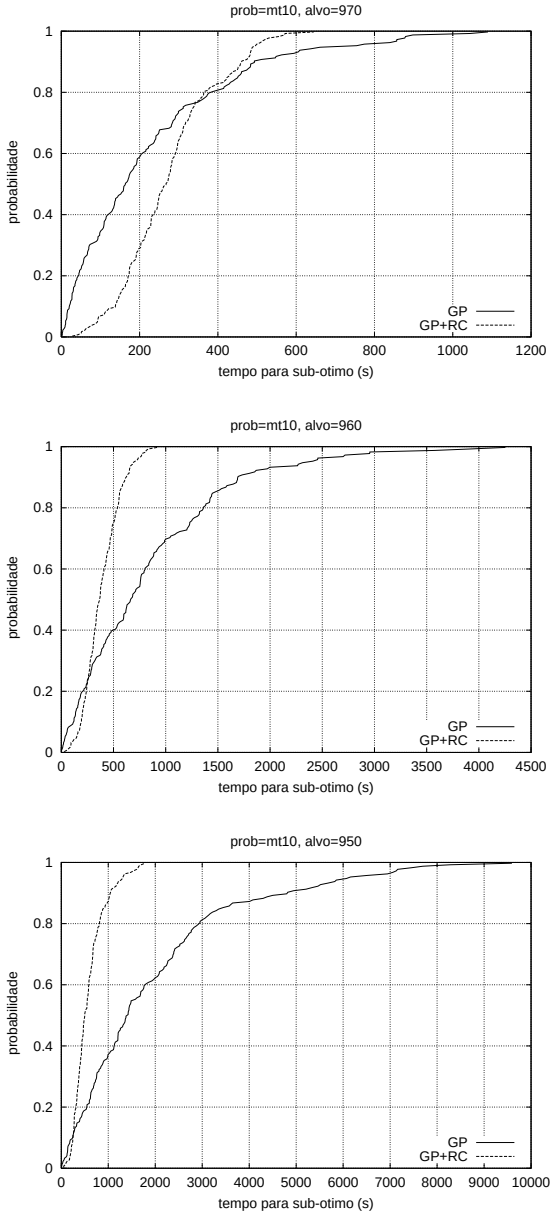

Figura 5.9: Distribuição empírica de probabilidade do tempo para valor alvo para `GP` e `GP+RC`, problema `mt10` e valores alvos 970, 960 e 950.

A Figura 5.9 mostra, para o problema mt10, uma comparação entre GP e GP+RC para três valores alvos: 970, 960 e 950. Esses valores estão a 4.3%, 3.2% e 2.1% do melhor valor conhecido, respectivamente. São mostrados três gráficos, onde a dificuldade de obter o valor alvo cresce de cima para baixo na figura. As distribuições empíricas são plotadas da mesma maneira que nos gráficos da Figura 5.8. Observa-se que, para o primeiro gráfico de cima para baixo, GP encontra a solução alvo mais rapidamente do que GP+RC para probabilidades abaixo de 76%. Isso acontece porque o valor alvo procurado é muito fácil de obter e as iterações de GP precisam de menos tempo computacional do que as iterações de GP+RC. Aumentando-se gradualmente a dificuldade de obter os valores alvos nos dois gráficos seguintes, observa-se que as probabilidades até onde GP encontra a solução alvo mais rapidamente do que GP+RC caem para 24% e 9%, respectivamente. A Tabela 5.2 mostra para probabilidades de 20%, 50% e 80%, os tempos necessários para GP e GP+RC encontrarem cada um dos três valores alvos considerados. Também é mostrado para cada par de variante e valor alvo, o decréscimo percentual do tempo de solução de GP+RC, em relação ao tempo de solução de GP, em função da probabilidade. Observa-se que à medida em que o valor alvo aproxima-se do melhor valor conhecido, o decréscimo percentual do tempo de solução de GP+RC em relação ao tempo de solução de GP cresce. Por exemplo, para um valor alvo de 960 e uma probabilidade de 50%, o decréscimo percentual é de 44.93%. Diminuindo-se o valor alvo para 950, o decréscimo percentual cresce para 64.98%, para a mesma probabilidade de 50%.

As variantes GP+RC e GP+INT são comparadas na Figura 5.10. As distribuições empíricas para esses algoritmos são plotadas da mesma forma que nos gráficos da Figura 5.8, usados para comparar GP e GP+RC. Os mesmos problemas testes abz6, mt10, orb5 e la21 foram usados e os valores alvos considerados foram 965, 960, 930 e 1130, respectivamente. Deve-se observar que os valores alvos usados nesse experimento são mais fáceis de obter do que os usados para comparar GP e GP+RC. Isso foi necessário porque a variante GP+INT precisaria de um tempo computacional muito elevado para obter soluções de qualidade comparável às do primeiro experimento. A Tabela 5.3 mostra algumas estimativas das probabilidades de encontrar uma solução com custo pelo menos tão bom quanto o valor alvo, em função do tempo máximo de solução para GP+C-INT e GP+RC. Observa-se para o problema la21, por exemplo, que a probabilidade de GP+C-INT encontrar uma solução de valor no máximo igual a 1130 em menos de 500 segundos é de 37%, enquanto que para GP+RC a probabilidade é de 100%. Para o problema mt10, observa-se que as probabilidades de GP+C-INT e GP+RC encontrarem

a solução alvo em menos de 1000s são de 27% e 100%, respectivamente. Verifica-se, dessa forma, que o uso de intensificação em um algoritmo GRASP apresenta melhores resultados quando essa etapa é feita após a fase de busca local, ou seja ao final de uma iteração. Isso pode ser explicado porque a intensificação prematura, isto é, feita durante a fase construtiva, pode reduzir drasticamente o número de mínimos locais visitados durante uma execução.

Para verificar como o algoritmo proposto se comporta em termos da qualidade da solução obtida, a implementação foi executada extensivamente para todos os problemas testes. O número de iterações executadas foi freqüentemente da ordem de milhões. Esses resultados são mostrados nas Tabelas 5.4, 5.5 e 5.6. Cada tabela mostra, para cada problema teste, o seu nome, suas dimensões (número de tarefas e número de máquinas), o valor da melhor solução conhecida (MSC), o valor da melhor solução encontrada por `GP+C-INT` e por `GP+RC`, o número de iterações executadas, o tempo de processamento para 1000 iterações, o valor da melhor solução encontrada e o erro percentual relativo da solução de `GP+RC` em relação à MSC.

Dos 66 problemas testes estudados, `GP+RC` encontrou a melhor solução conhecida em 49 dos casos (74.2%). Encontrou uma solução com valor dentro do intervalo de 0.5% da melhor solução conhecida para 50 problemas testes (75.7%). Em 56 problemas testes (84.8%), a solução de `GP+RC` ficou a 1% da melhor solução conhecida e em 61 problemas testes (92.4%) ficou a 2% da melhor solução conhecida. Em 64 problemas testes (97%), a solução de `GP+RC` ficou a 5% da melhor solução conhecida, enquanto que para todos os casos, a solução do GRASP encontrada ficou a menos de 7.5% da melhor solução conhecida.

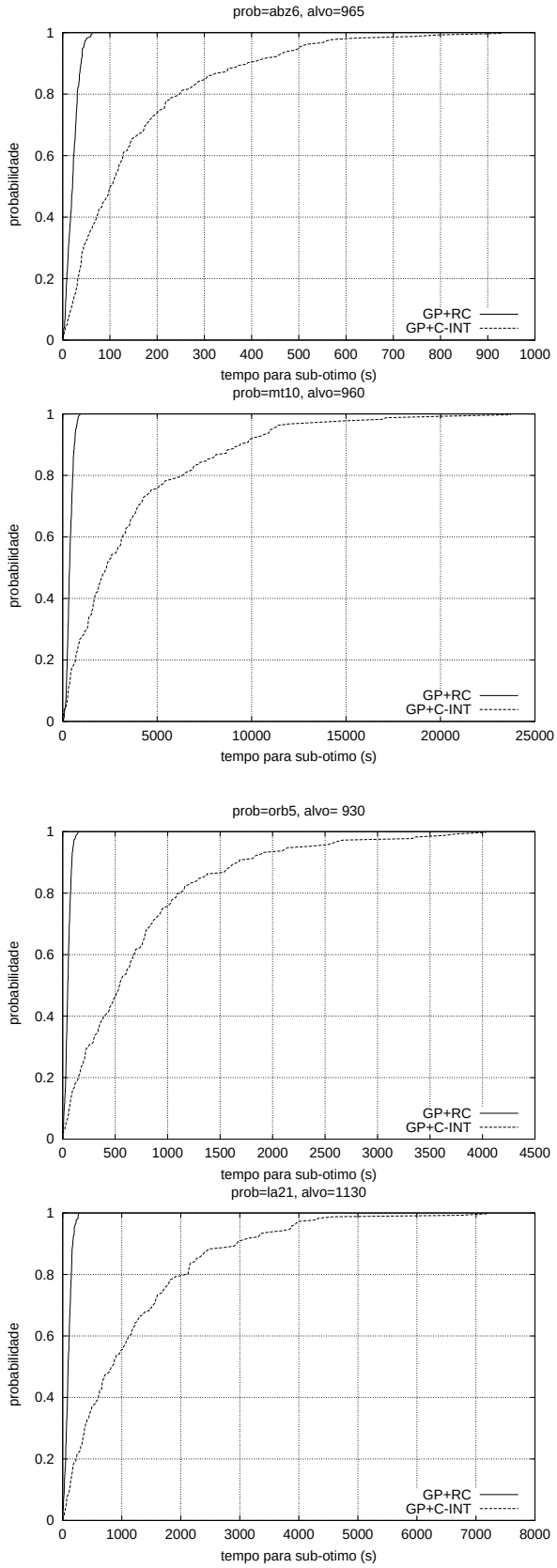

Figura 5.10: Distribuição empírica de probabilidade do tempo para valor alvo para `GP+C-INT` e `GP+RC`, problemas `abz6`, `mt10`, `orb5` e `la21`.

Tabela 5.3: Estimativas das probabilidades de encontrar uma solução com custo melhor ou igual ao valor alvo, em função do tempo máximo de solução para GP+C-INT e GP+RC, respresentados por INT e RC, respectivamente. Os problemas testes estudados são abz6, mt10, orb5 e la21, para os valores alvos 965, 960, 930 e 1130, respectivamente.

tempo	abz6		mt10		orb5		la21	
(s)	INT	RC	INT	RC	INT	RC	INT	RC
100	0.49	1.00	0.04	0.03	0.16	0.95	0.09	0.54
500	0.94	1.00	0.17	0.75	0.46	1.00	0.37	1.00
1000	1.00	1.00	0.27	1.00	0.75	1.00	0.55	1.00
1500	1.00	1.00	0.34	1.00	0.86	1.00	0.69	1.00

Nas Tabelas 5.7 e 5.8 os resultados são resumidos para cada classe de problema. Na Tabela 5.7 é mostrado para cada classe de problema, o seu nome, a soma dos melhores valores conhecidos (MSC), a soma dos melhores valores encontrados por GP+RC e o erro relativo da soma dos melhores valores encontrados por GP+RC em relação à soma dos melhores valores conhecidos. Na Tabela 5.8 é mostrado para cada classe de problemas, o seu nome e o percentual de problemas testes para os quais o método GRASP com religamento de caminhos encontrou uma solução dentro dos intervalos de 0%, .5%, 1%, 2%, 5% e 10% do valor da melhor solução conhecida. Observando-se essas tabelas, conclui-se que as classes de problemas mais fáceis são car e mt, onde o algoritmo com religamento de caminhos encontrou a melhor solução conhecida para todos os problemas testes. Para as classes orb e la, os erros relativos médios de GP+RC ficaram a um intervalo de menos de 0.5% do melhor valor conhecido. A classe de maior dificuldade foi abz, onde o erro relativo médio foi de 2.64% em relação aos melhores valores conhecidos.

5.6.4 Distribuição da probabilidade para tempo de solução

Os gráficos Q-Q e os gráficos com as distribuições teóricas e empíricas sobrepostas para GP são mostrados nas Figuras 5.11 e 5.12, respectivamente. Da mesma forma, as Figuras 5.13 e 5.14 mostram respectivamente, os gráficos Q-Q e os gráficos com as distribuições teóricas e empíricas sobrepostas para GP+RC. Esses gráficos são plotados segundo a metodologia apresentada no Capítulo 3. Foram considerados os problemas testes: abz6, mt10, orb5 e la21. Para cada problema teste, foram analisados três valores alvos. Cada gráfico apresentado nessas figuras é plotado a partir dos pontos gerados por 200 execuções independentes do algoritmo GRASP.

As Figuras 5.11 e 5.13 são formadas por 12 gráficos Q-Q, um para cada par problema teste/valor alvo para GP e GP+RC, respectivamente. Da

Tabela 5.4: Resultados computacionais para as classes de problemas `abz`, `car`, `mt` e `orb`. São mostrados o nome do problema, a dimensão do problema (tarefas e máquinas), o valor da melhor solução conhecida (MSC), o valor da melhor solução encontrada por `GP+C-INT` e para `GP+RC`, o número de iterações executadas, o tempo de processamento para 1000 iterações, o valor da melhor solução encontrada e o erro percentual relativo da solução de `GP+RC` em relação a MSC.

					`GP+RC`			
problema	$\|\mathcal{J}\|$	$\|\mathcal{M}\|$	MSC	`GP+C-INT`	iterações	tempo (s) (10^3 iter.)	sol.	erro (%)
`abz5`	10	10	1234	1238	1000000	2.53	1234	0.0
`abz6`	10	10	943	947	300000	2.26	943	0.0
`abz7`	15	20	665	723	50000000	11.66	692	4.1
`abz8`	15	20	670	729	10000000	49.78	705	5.2
`abz9`	15	20	691	758	1000000	875.92	740	7.1
`car1`	11	5	7038	7038	1000	15.31	7038	0.0
`car2`	13	4	7166	7166	1000	52.14	7166	0.0
`car3`	12	5	7312	7366	50700000	2.08	7312	0.0
`car4`	14	4	8003	8003	10000	2.79	8003	0.0
`car5`	10	6	7702	7702	500000	4.40	7702	0.0
`car6`	8	9	8313	8313	10000	11.85	8313	0.0
`car7`	7	7	6558	6558	1000	16.05	6558	0.0
`car8`	7	7	8264	8264	20000	4.66	8264	0.0
`mt06`	6	6	55	55	10	1.74	55	0.0
`mt10`	10	10	930	938	2500000	4.05	930	0.0
`mt20`	20	5	1165	1169	4500000	46.48	1165	0.0
`orb1`	10	10	1059	1070	1200000	46.75	1059	0.0
`orb2`	10	10	888	889	1100000	11.45	888	0.0
`orb3`	10	10	1005	1021	6500000	33.32	1005	0.0
`orb4`	10	10	1005	1031	100000000	1.94	1011	0.6
`orb5`	10	10	887	891	20000000	14.61	889	0.2
`orb6`	10	10	1010	1013	3500000	43.75	1012	0.2
`orb7`	10	10	397	397	30000	18.72	397	0.0
`orb8`	10	10	899	909	1600000	24.26	899	0.0
`orb9`	10	10	934	945	11100000	4.38	934	0.0
`orb10`	10	10	944	953	300000	33.25	944	0.0

mesma forma, as Figuras 5.12 e 5.14 são formadas por 12 gráficos com as distribuições empíricas e teóricas sobrepostas, um para cada par problema teste/valor alvo, para `GP` e `GP+RC`, respectivamente. 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 GRASP. Em cada linha das figuras, a dificuldade de encontrar uma solução para um dado valor alvo aumenta da esquerda para a direita. As Tabelas 5.9 e 5.10 mostram os valores alvos considerados, assim como os parâmetros estimados pela metodologia para `GP` e `GP+RC`, respectivamente.

As Figuras 5.11 e 5.13 mostram que o afastamento dos pontos em relação à reta estimada é pequeno, tanto para `GP`, como para `GP+RC`. Observa-se também que à medida em que a dificuldade de obter o valor alvo

Tabela 5.5: Resultados computacionais para a classe de problemas `la` (problemas `la01` a `la20`). São mostrados o nome do problema, a dimensão do problema (tarefas e máquinas), o valor da melhor solução conhecida (MSC), o valor da melhor solução encontrada por `GP+C-INT` e para `GP+RC`, o número de iterações executadas, o tempo de processamento para 1000 iterações, o valor da melhor solução encontrada e o erro percentual relativo da solução de `GP+RC` em relação a MSC.

					`GP+RC`			
problema	$\|\mathcal{J}\|$	$\|\mathcal{M}\|$	MSC	`GP+C-INT`	iterações	tempo (s) (10^3 iter.)	sol.	erro (%)
`la01`	10	5	666	666	100	0.82	666	0.0
`la02`	10	5	655	655	4000	2.74	655	0.0
`la03`	10	5	597	604	10000	2.95	597	0.0
`la04`	10	5	590	590	1000	15.71	590	0.0
`la05`	10	5	593	593	100	1.09	593	0.0
`la06`	15	5	926	926	100	8.00	926	0.0
`la07`	15	5	890	890	100	1.58	890	0.0
`la08`	15	5	863	863	300	5.49	863	0.0
`la09`	15	5	951	951	100	3.40	951	0.0
`la10`	15	5	958	958	100	11.50	958	0.0
`la11`	20	5	1222	1222	100	3.14	1222	0.0
`la12`	20	5	1039	1039	100	2.89	1039	0.0
`la13`	20	5	1150	1150	100	3.20	1150	0.0
`la14`	20	5	1292	1292	100	5.70	1292	0.0
`la15`	20	5	1207	1207	200	122.75	1207	0.0
`la16`	10	10	945	946	1300000	2.27	945	0.0
`la17`	10	10	784	784	20000	3.29	784	0.0
`la18`	10	10	848	848	50000	9.07	848	0.0
`la19`	10	10	842	842	20000	15.14	842	0.0
`la20`	10	10	902	907	17000000	1.63	902	0.0

aumenta, os pontos tornam-se mais ajustados à reta estimada. Verifica-se, então, que os tempos para subótimo medidos para `GP` e para `GP+RC` adaptam-se bem a uma distribuição exponencial de dois parâmetros.

Em [16] é mostrado que a distribuição de probabilidade para tempo de solução para `GP+C-INT` adapta-se bem a uma distribuição exponencial de dois parâmetros. Nessa seção mostrou-se que um algoritmo GRASP onde a fase construtiva é computada usando-se de forma alternada duas funções gulosas (`GP`) e um algoritmo GRASP com religamento de caminhos restrito a iterações onde a busca local tenha obtido soluções de qualidade (`GP+RC`) também ajustam-se a uma distribuição exponencial de dois parâmetros. Esses resultados reforçam as conclusões mostradas nos Capítulos 3 e 4.

Tabela 5.6: Resultados computacionais para a classe de problemas `la` (problemas `la21` a `la40`). São mostrados o nome do problema, a dimensão do problema (tarefas e máquinas), o valor da melhor solução conhecida (MSC), o valor da melhor solução encontrada por `GP+C-INT` e para `GP+RC`, o número de iterações executadas, o tempo de processamento para 1000 iterações, o valor da melhor solução encontrada e o erro percentual relativo da solução de `GP+RC` em relação a MSC.

					`GP+RC`			
						tempo (s)		erro
problema	$\|\mathcal{J}\|$	$\|\mathcal{M}\|$	MSC	`GP+C-INT`	iterações	(10^3 iter.)	sol.	(%)
`la21`	15	10	1047	1091	100000000	3.51	1057	1.0
`la22`	15	10	927	960	26000000	3.45	927	0.0
`la23`	15	10	1032	1032	10000	39.39	1032	0.0
`la24`	15	10	935	978	125000000	3.26	954	2.0
`la25`	15	10	977	1028	32000000	3.29	984	0.7
`la26`	20	10	1218	1271	3500000	6.35	1218	0.0
`la27`	20	10	1235	1320	10500000	27.51	1269	2.8
`la28`	20	10	1216	1293	20000000	17.77	1225	0.7
`la29`	20	10	1157	1293	50000000	6.17	1203	4.0
`la30`	20	10	1355	1368	3000000	7.61	1355	0.0
`la31`	30	10	1784	1784	10000	267.60	1784	0.0
`la32`	30	10	1850	1850	100	12.66	1850	0.0
`la33`	30	10	1719	1719	1000	875.11	1719	0.0
`la34`	30	10	1721	1753	50000	80.33	1721	0.0
`la35`	30	10	1888	1888	10000	348.32	1888	0.0
`la36`	15	15	1268	1334	51000000	5.54	1287	1.5
`la37`	15	15	1397	1457	20000000	12.51	1410	0.9
`la38`	15	15	1196	1267	20000000	32.87	1218	1.8
`la39`	15	15	1233	1290	6000000	59.06	1248	1.2
`la40`	15	15	1222	1259	2000000	104.18	1244	1.8

Tabela 5.7: Resultados experimentais: a soma dos melhores valores conhecidos (MSC) é comparada à soma das soluções obtidas por `GP+RC` para cada classe de problemas.

problema	soma das MSC	soma das soluções `GP+RC`	erro (%)
`abz`	4203	4314	2.64
`car`	60356	60356	0.00
`mt`	2150	2150	0.00
`orb`	9028	9038	0.11
`la`	44297	44513	0.49

Tabela 5.8: Resultados experimentais: percentual de soluções de `GP+RC` dentro de um determinado intervalo de tolerância da melhor solução conhecida.

	percentual de soluções do `GP+RC` a					
problema	0%	.5%	1%	2%	5%	10%
`abz`	40.0	40.0	40.0	40.0	60.0	100.0
`car`	100.0	100.0	100.0	100.0	100.0	100.0
`mt`	100.0	100.0	100.0	100.0	100.0	100.0
`orb`	70.0	90.0	100.0	100.0	100.0	100.0
`la`	72.5	72.5	82.5	95.0	100.0	100.0

Tabela 5.9: Problemas testes usados para estudar a distribuição empírica de probabilidade da variável aleatória *tempo para valor alvo* para `GP`. São mostrados para os quatro problemas testados, o custo da melhor solução conhecida (MSC), o valor alvo e os parâmetros estimados.

			parâmetros estimados	
problema	MSC	alvo	$\hat{\mu}$	$\hat{\lambda}$
`abz6`	943	970	0.203	3.804
		960	0.428	15.567
		950	-1.323	68.490
`mt10`	930	970	-9.403	233.092
		960	11.723	885.034
		950	109.528	1827.882
`orb5`	887	930	-0.783	15.757
		920	1.249	38.273
		910	-1.011	191.111
`la21`	1047	1130	-4.115	50.343
		1120	-1.015	206.836
		1110	-87.594	1268.081

Tabela 5.10: problemas testes usados para estudar a distribuição empírica de probabilidade da variável aleatória *tempo para sub-ótimo* para `GP+RC`. São mostrados para os quatro problemas testados, o custo da melhor solução conhecida (MSC), o valor alvo e os parâmetros estimados.

			parâmetros estimados	
problema	MSC	alvo	$\hat{\mu}$	$\hat{\lambda}$
`abz6`	943	950	21.883	47.796
		947	47.884	117.140
		943	47.671	756.562
`mt10`	930	960	145.054	138.369
		950	199.814	215.100
		938	305.278	524.238
`orb5`	887	910	50.615	49.283
		900	94.846	172.376
		895	130.124	395.414
`la21`	1047	1110	122.206	158.215
		1105	154.191	231.769
		1100	175.200	407.736

5.7 Conclusão

Nesse capítulo foi proposta uma heurística GRASP para obtenção de soluções aproximadas para o problema de *job-shop scheduling*. Neste algoritmo, a fase de intensificação é transferida para o final de cada iteração e é realizada na forma de religamento de caminhos. Como o religamento de caminhos para o JSP é computacionalmente muito trabalhoso, somente são submetidas a esse procedimento soluções que satisfazem a um critério de qualidade. Além disso, soluções são construídas usando duas funções gulosas de forma alternada, o que produz uma maior diversidade de soluções iniciais para a busca local.

O algoritmo foi testado para 66 problemas testes. As melhores soluções conhecidas foram encontradas para 74.2% dos problemas testados. Soluções a 5% dos valores das melhores soluções conhecidas foram obtidas para 97% dos problemas testes estudados.

Comparando-se as variantes `GP+RC` e `GP`, verificou-se que o religamento de caminhos quando acrescentado ao método GRASP faz com que o procedimento encontre mais rapidamente soluções de uma dada qualidade. Portanto, o aumento no tempo computacional de cada iteração do método híbrido é compensado por um aumento na robustez do algoritmo. Verificou-se, também, que a intensificação na forma de religamento de caminhos feita após a busca local permite encontrar mais rapidamente soluções de uma dada qualidade do que quando a intensificação é feita durante a fase construtiva, através do incentivo à inserção de atributos de soluções de qualidade. Portanto, a variante `GP+RC` permite a obtenção de melhores resultados quando comparada à variante `GP+C-INT`.

Observou-se que os tempos para valor alvo da heurística GRASP híbrida e da heurística pura que usa alternadamente duas funções gulosas na fase construtiva ajustam-se a distribuições exponenciais de dois parâmetros.

O objetivo desse capítulo foi mostrar que o uso da etapa de religamento de caminhos no GRASP melhora os resultados obtidos pelo algoritmo puro. Além disso, pretendeu-se mostrar que uma etapa de intensificação traz maiores benefícios quando computada após a fase de busca local e não durante a fase construtiva. Por esses motivos, foram comparadas apenas estratégias GRASP. Em [63] o algoritmo proposto nesse capítulo é comparado a outras heurísticas desenvolvidas para o JSP. As heurísticas comparadas são testadas para problemas da OR-Library. Observa-se que os resultados obtidos pelo GRASP com religamento de caminhos apresentado nesse capítulo são

comparáveis aos melhores resultados obtidos pelas heurísticas apresentadas.

No próximo capítulo serão apresentadas e analisadas duas estratégias de paralelização para o método GRASP com religamento de caminhos. As heurísticas híbridas propostas para o problema de atribuição de três índices e para o problema de escalonamento de tarefas JSP serão usadas para estudar o comportamento das estratégias de paralelização propostas.

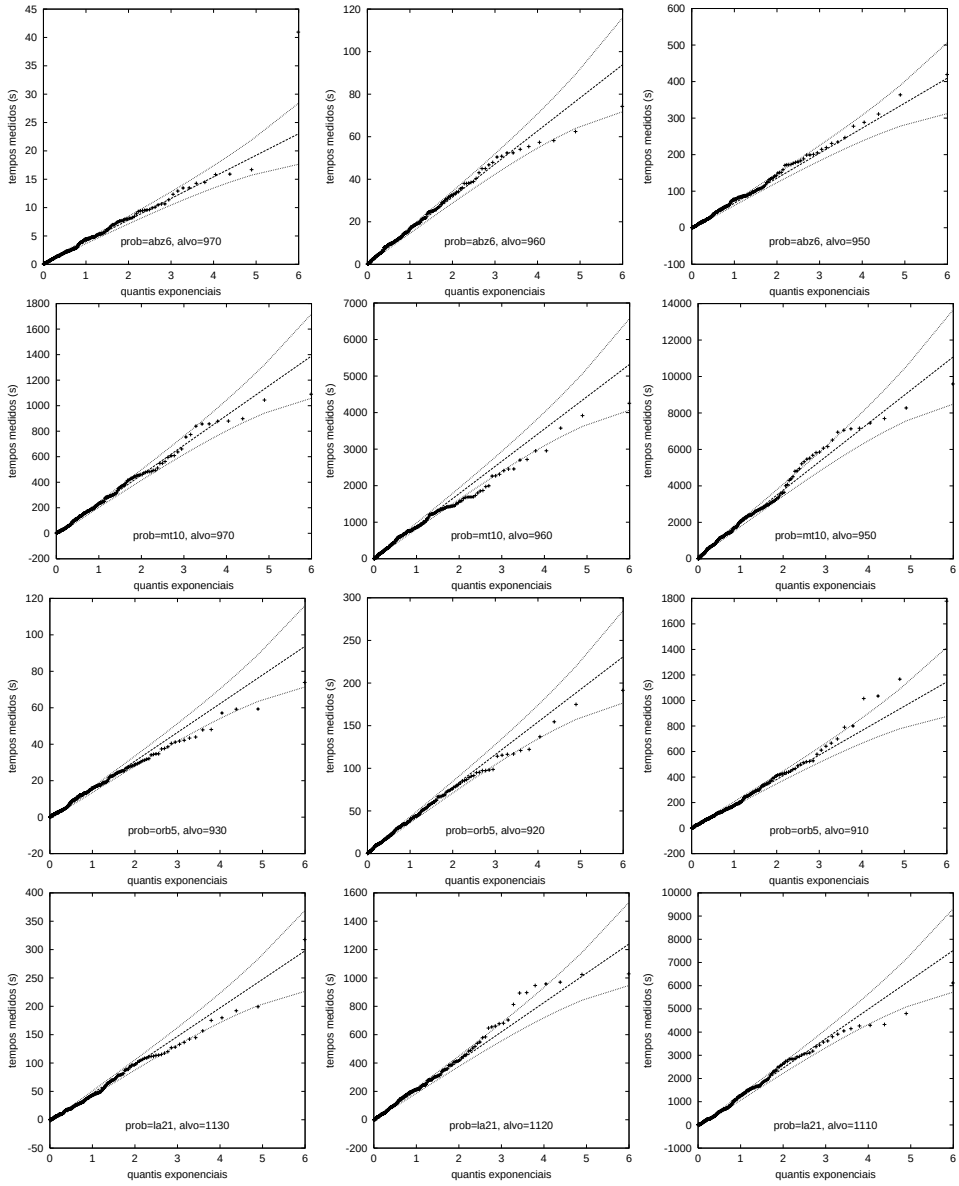

Figura 5.11: Gráficos Q-Q para `GP`: problemas `abz6`, `mt10`, `orb5` e `la21`.

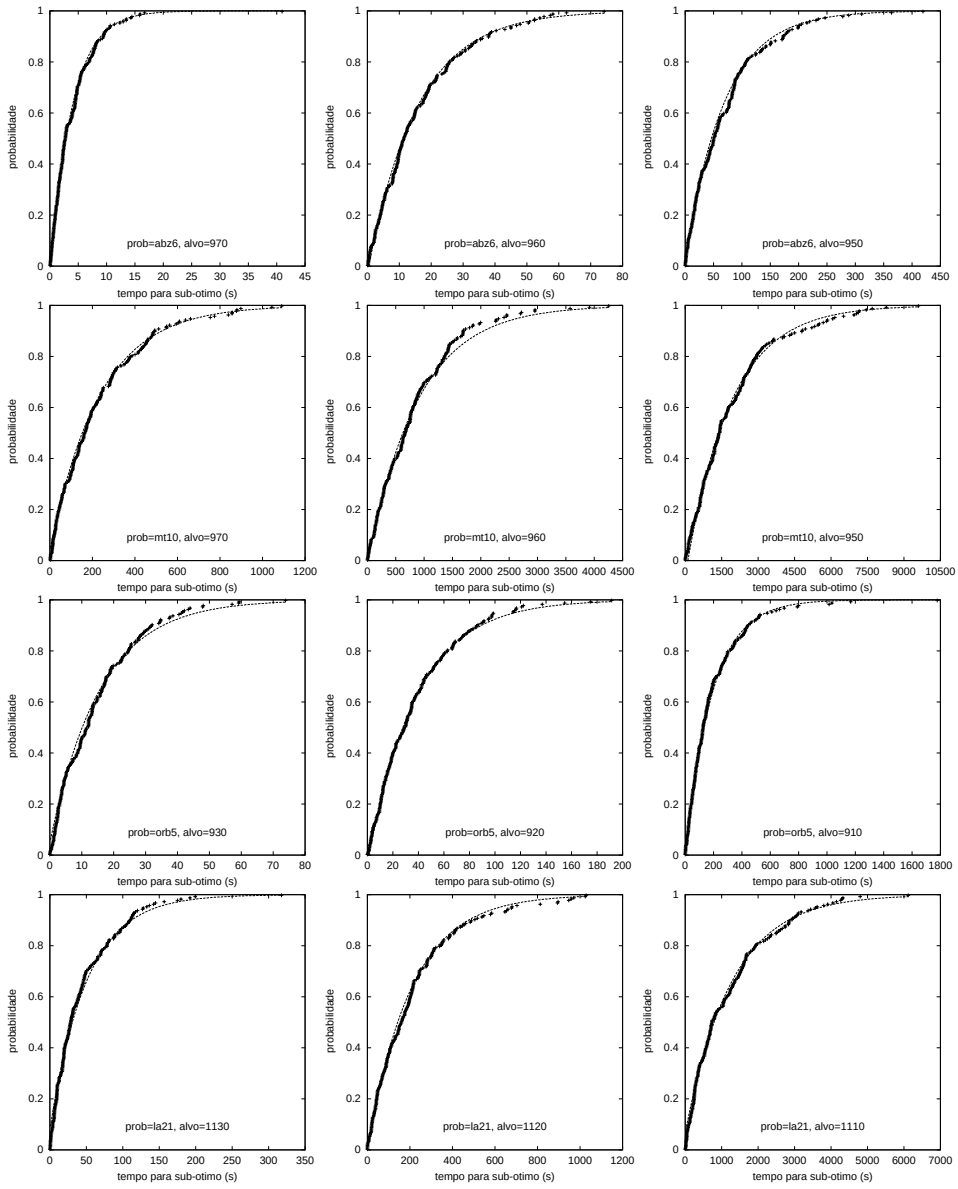

Figura 5.12: Gráficos das distribuições exponenciais para `GP`: problemas `abz6`, `mt10`, `orb5` e `la21`.

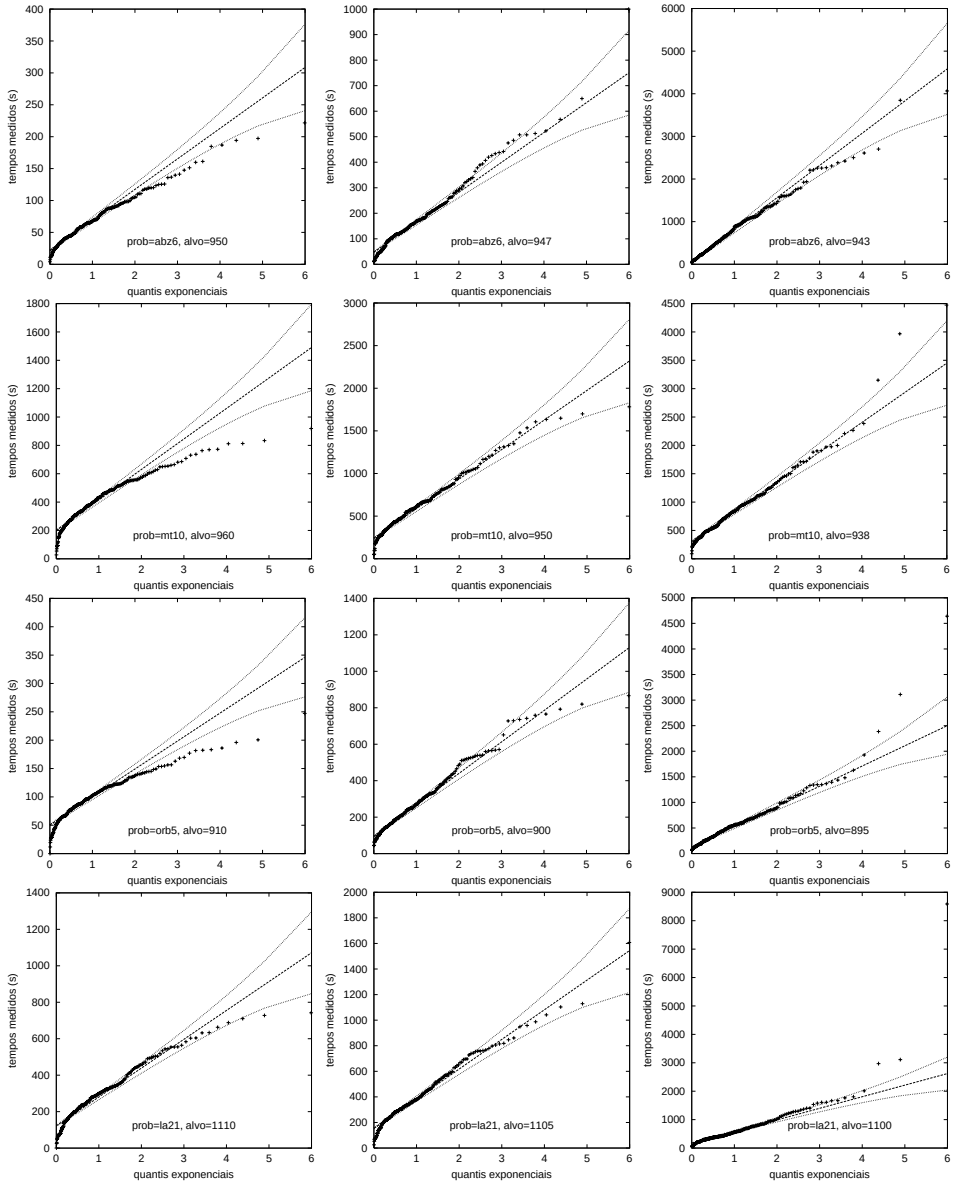

Figura 5.13: Gráficos Q-Q para `GP+RC`: problemas `abz6`, `mt10`, `orb5` e `la21`.

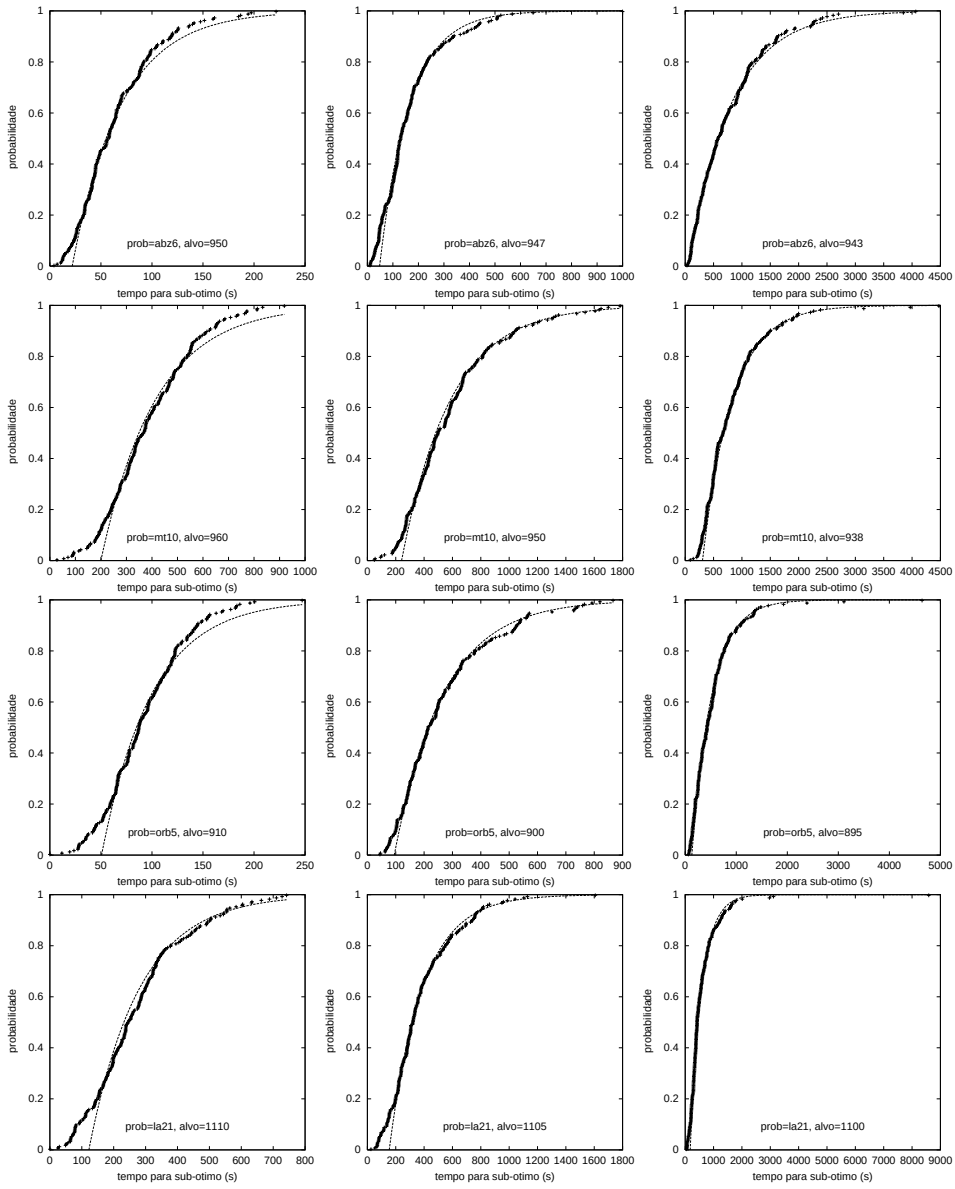

Figura 5.14: Gráficos das distribuições exponenciais para `GP+RC`: problemas `abz6`, `mt10`, `orb5` e `la21`.