5
Implementação de Diferentes Estratégias de Paralelização

A estratégia de paralelização padrão mestre-trabalhador é bastante usada nas plataformas distribuídas. A fim de analisar o comportamento dessa estratégia foram desenvolvidos quatro algoritmos paralelos como variações da estratégia tradicional mestre-trabalhador. As quatro versões são descritas nas Seções 5.2, 5.3, 5.4 e 5.5. Em seguida, na Seção 5.6, é feita uma análise dos resultados computacionais alcançados. Os experimentos computacionais foram todos conduzidos sobre o mTTP, descrito na Seção 2.3.

5.1
Considerações Iniciais

A escolha correta da melhor estratégia de paralelização depende de vários fatores. Dentre os mais importantes estão a arquitetura da máquina paralela usada e as características do problema. A maioria das metaheurísticas paralelas implementadas para executarem em plataformas distribuídas, tem adotado a mesma estratégia de paralelização tradicional mestre-trabalhador usada na paralelização de algoritmos que executam em máquinas superescalares [94]. Nesse contexto, implementou-se quatro algoritmos paralelos para o mTTP, com diferentes variações de uma estratégia tradicional e diversos níveis de cooperação, objetivando avaliar o comportamento de alguns mecanismos.

As implementações paralelas apresentadas neste capítulo basearam-se no algoritmo seqüencial `GRILS-mTTP`, visto na Seção 2.3.2. Dessa forma, é possível investigar como o grau de cooperação e aumento de diversidade em termos do número de trajetórias e da quantidade de informações compartilhada afetam essa heurística.

As quatros versões adotaram o paradigma de programação mestre-trabalhador e a estratégia de paralelização por caminhos múltiplos. Todas as implementações foram desenvolvidas em LAM/MPI, seguindo o modelo padrão de criar um processo por máquina, que só será finalizado quando sua atividade tiver sido concluída.

Em todas as quatro versões paralelas, inicialmente, o processo mestre gera e distribui sementes diferentes para serem usadas pelo gerador de números

pseudo-aleatórios de cada processo trabalhador. Em todas as implementações foi usado o gerador de números aleatórios Mersenne Twister [106], com base na recomendação feita em [133]. As próximas quatro seções descrevem os detalhes específicos das implementações paralelas PAR-I, PAR-O, PAR-1P e PAR-MP.

5.2 Implementação Paralela Independente (PAR-I)

A versão apresentada nesta seção, chamada PAR-I, equivale a executar o algoritmo seqüencial simultaneamente em múltiplas máquinas independentes. Essa implementação é fiel à estratégia de paralelização por caminhos múltiplos com processos independentes (Seção 3.3).

Na implementação desenvolvida durante esta tese, depois de receber suas sementes, cada processo trabalhador inicia um ciclo no qual gera uma nova solução durante a fase de construção GRASP e, em seguida, inicia uma fase de melhoria da solução inicial, através da heurística de busca local ILS, até o critério de re-inicialização ser satisfeito. Esse ciclo é repetido por cada processo escravo até que o critério de parada seja atingido por algum deles.

Nesta implementação, após o mestre enviar a semente, não há mais nenhuma troca de informação entre os processos trabalhadores durante toda a execução. A única troca de mensagem, antes da finalização, tem origem no processo mestre que, ao receber uma mensagem com uma solução que satisfaça o critério de parada, grava a solução recebida e envia uma mensagem de finalização para que todos os processos trabalhadores parem suas execuções.

Essa implementação é representante de uma estratégia trivial de paralelização bastante explorada na literatura, pois sua adaptação da versão seqüencial para a paralela é natural e fácil de ser desenvolvida, sendo necessário implementar somente um processo mestre que seja capaz de distribuir e criar as tarefas entre os processadores, enviar as sementes e coletar os resultados processados. O processo trabalhador pode ser similar ao próprio algoritmo seqüencial, tendo apenas que ser habilitado para receber dados e enviar os resultados. Dessa forma, as soluções encontradas com essa estratégia podem ser alcançadas também pela execução seqüencial. Os resultados obtidos com N buscas locais paralelas independentes são os mesmos obtidos através de N buscas locais executadas de maneira seqüencial em uma máquina monoprocessada.

Na implementação PAR-I, o processo mestre também realiza o papel de processo trabalhador, executando a heurística híbrida `GRILS-mTTP`. Com isso, foi usada a capacidade de processamento da máquina em que o processo mestre estava executando.

Como essa implementação usa o método padrão de paralelismo, é criado um processo por máquina, o qual deve executar até que o critério de parada seja satisfeito. Assim, o tempo de execução de cada processo depende do grau de dificuldade em se atingir o critério de parada.

5.3 Implementação Paralela com Única Cooperação (PAR-O)

Esta versão, denominada PAR-O (do inglês *Parallel One-off Cooperation*), é similar a versão anterior, com exceção da primeira iteração do laço composto por uma execução da fase de construção GRASP seguida da fase de busca local ILS. Depois de cada processo trabalhador executar a fase de construção GRASP, a melhor solução encontrada por cada um deles é enviada para o processo mestre, que seleciona a melhor dentre elas e a envia para todos os demais processos trabalhadores.

Em seguida, todos os processos trabalhadores executam a fase ILS da primeira iteração usando a mesma solução, que é a melhor encontrada por todos os trabalhadores durante a fase de construção GRASP. A estratégia aqui implementada é chamada de cooperação *one-off*, porque só ocorre durante a primeira iteração do algoritmo. As iterações seguintes são todas executadas independentemente. A cada iteração concluída, cada processo trabalhador reinicia sua execução a partir da fase de construção GRASP. Todos esses passos são repetidos até que um critério de parada seja atingido por algum dos processos trabalhadores.

Observa-se que nessa implementação paralela já existe alguma cooperação entre os processos trabalhadores. Todavia, o grau de cooperação implementado ainda é pequeno, pois a troca de informações ocorre somente uma única vez e na primeira iteração. Nesse caso, o processo mestre deve estar habilitado para receber todas as soluções geradas após a fase de construção GRASP inicial, selecionar a melhor e redistribuí-la para os trabalhadores uma única vez. Os processos trabalhadores só se diferenciam da implementação seqüencial no ponto de sincronização criado entre a fase GRASP e a fase ILS na primeira iteração.

5.4 Implementação Paralela com uma Solução de Elite (PAR-1P)

Uma desvantagem da implementação anterior é a falta de uma cooperação contínua entre trabalhadores durante suas execuções, isso é, cada processo trabalhador não aprende a partir das buscas realizadas em paralelo

(ou a partir de soluções encontradas) em iterações anteriores por outros processos trabalhadores.

Nesta implementação, chamada PAR-1P, o mestre mantém sempre a melhor solução encontrada (solução de elite), a fim de viabilizar a troca de informação entre os demais processos trabalhadores durante toda a execução. O nome PAR-1P é exatamente em conseqüência do mestre guardar apenas uma solução, a melhor. Para isso, todas as vezes que um trabalhador finaliza uma iteração completa da heurística, ele compara a solução encontrada na iteração corrente com a melhor solução mantida pelo processo mestre. Se for melhor, o trabalhador envia sua nova solução para o mestre, caso contrário ela é descartada. Assim, a solução mantida pelo mestre pode ser atualizada.

Após os processos trabalhadores enviarem (ou não) sua solução para o mestre, cada trabalhador poderá recomeçar sua execução de qualquer fase da metaheurística, podendo construir uma nova solução a partir da fase de construção GRASP, ou podendo reiniciar a partir da fase de busca local ILS. Nesse último caso, há necessidade do trabalhador mandar uma mensagem ao mestre, solicitando uma cópia da melhor solução. O processo mestre, ao receber essa mensagem, envia uma resposta com a solução. Os processos trabalhadores não podem ser idênticos aos da implementação seqüencial, pois esses processos podem re-iniciar sua execução em fases diferentes, com soluções oriundas de outros processos.

A decisão de qual caminho percorrer é tomada de forma aleatória por cada processo trabalhador após a iteração concluída. Assim, a probabilidade do processo iniciar a partir da fase de construção GRASP é $Q - 1$ e de iniciar a partir da fase de busca ILS é Q, onde a probabilidade é fornecida como parâmetro do programa pelo usuário. Dessa forma, os processos trabalhadores podem tanto melhorar a melhor solução corrente (através da busca local ILS), quanto construir uma nova solução (através da fase de construção GRASP).

Para evitar que os processos trabalhadores enviem soluções que não serão aproveitadas pelo mestre, todas as vezes que a melhor solução armazenada pelo mestre for atualizada, esse envia o custo da nova melhor solução para todos os trabalhadores. Dessa forma, cada trabalhador quando tiver uma solução a ser enviada, ele saberá exatamente se a sua solução é melhor do que a mantida pelo mestre ou não. O objetivo é compartilhar essa informação não somente para atingir uma boa solução mais rapidamente, mas também para encontrar soluções melhores do que as estratégias independentes. Aumenta-se o grau de cooperação entre os processos, sem a necessidade de manter sincronização entre os mesmos.

5.5 Implementação Paralela com uma Lista de Soluções de Elite (PAR-MP)

A implementação PAR-MP objetiva aumentar o número de soluções de elite que podem ser compartilhadas pelos processos. Assim, o mestre é dedicado ao gerenciamento de uma lista de soluções de elite, tendo como uma de suas tarefas coletar e distribuir soluções de elite de acordo com os pedidos realizados pelos trabalhadores. A lista é uma estrutura de dados usada para armazenar soluções de alta qualidade, que podem ser potencialmente usadas durante a busca pelos demais processos trabalhadores. A atualização dessa lista é feita pelo processo mestre, de acordo com a qualidade das soluções já armazenadas e das novas soluções recebidas.

Para manter a lista de soluções de elite sempre atualizada, todas as vezes que um processo trabalhador executar uma iteração completa (fase GRASP seguida da fase ILS), ele enviará sua melhor solução para ser armazenada na lista mantida pelo mestre. Em seguida, o processo trabalhador escolhe aleatoriamente se reinicia sua execução a partir da fase GRASP (criando uma nova solução), ou se reinicia a partir da fase ILS. No último caso, é necessário que o processo solicite uma solução de elite da lista mantida pelo mestre. Esses passos são repetidos pelos processos trabalhadores até que algum critério de parada seja satisfeito.

O desafio principal é encontrar um equilíbrio entre a qualidade das soluções e o grau de diversidade na lista. É importante que as regras de inserção de soluções na lista atendam ambas características, pois a intensificação concentra a busca dentro de uma região específica do espaço de busca, enquanto a diversificação possibilita pesquisar várias regiões diferentes dentro do espaço de solução.

Nesse contexto, a inserção de uma nova solução depende do estado da lista e da fase da heurística em que ela foi gerada. O processo mestre verifica se a nova solução foi criada a partir da fase de busca local ILS. Em caso afirmativo, a nova solução deve ser melhor que a original para que ela possa substituí-la. Caso não seja, a nova solução é descartada. Por outro lado, quando a nova solução é gerada a partir da fase de construção GRASP, a condição para que ela seja inserida como solução de elite é que não exista outra solução igual na lista e, se a lista já estiver preenchida, que a nova seja melhor que a pior solução mantida lá. Assim, esse mecanismo faz com que a chance de existirem soluções muito parecidas na lista diminua.

Outra operação que o mestre deve ser capaz de realizar é a seleção de uma solução a partir da lista de soluções de elite e seu envio para o processo trabalhador que a tiver solicitado. Essa seleção foi implementada de maneira

aleatória entre as soluções mantidas na lista. A próxima seção apresenta a análise computacional do desempenho das quatros versões paralelas PAR-I, PAR-O, PAR-1P e PAR-MP vistas anteriormente.

5.6 Experimentos Computacionais

O objetivo dos experimentos realizados é avaliar o desempenho destas implementações em uma plataforma paralela distribuída. Para isso, os testes foram divididos em três grupos de experimento. O primeiro grupo avalia a qualidade das soluções alcançadas por cada versão paralela. O segundo grupo investiga a escalabilidade das implementações. E o último grupo de testes, analisa o tempo de processamento dos algoritmos paralelos. Todos os algoritmos testados nesta seção foram implementados usando C++ e a versão 7.0.6 do LAM/MPI [142]. A compilação foi realizada com o compilador *gcc* e a opção de otimização $-O3$.

Para efeito de avaliação, todos os testes foram realizados em um ambiente *cluster* dedicado e isolado, a fim de evitar influências externas. O *cluster* é formado por 32 computadores Pentium IV 1.7 GHz e 256 Mbytes de memória RAM, interconectados por uma rede Fast Ethernet. Há uma cópia local do código executável e dos dados do problema em cada máquina.

Os gráficos tttplots (*time-to-target plots*) foram propostos em [4, 5] e utilizados como metodologia para comparação das versões paralelas. Nesse tipo de gráfico, cada ponto da curva associada a um algoritmo representa o par (t_i, P_i) $(i = 1, ..., N)$, onde N é o número total de execuções feitas e t_i é o tempo da i-ésima execução mais rápida do algoritmo considerado e $P_i = (i - 0,5)/N$. Os eixos x e y representam respectivamente o tempo em escala logarítmica e a probabilidade de se alcançar determinado valor alvo. Esses gráficos fornecem uma aproximação da distribuição de probabilidades da variável tempo para alcançar uma solução de custo menor ou igual a um dado alvo. Quanto mais à esquerda está a curva de um algoritmo, mais rapidamente ele atinge o valor alvo com uma dada probabilidade. Quanto menor for a diferença de tempo entre a execução mais rápida e a mais lenta, diz-se que mais robusto é o algoritmo.

5.6.1 Instâncias de Teste do mTTP

Em [56] foram propostas originalmente duas classes de instâncias para o mTTP, todas disponíveis em [145]. A primeira classe é composta pelas instâncias chamadas de circulares. Denota-se por circ$n$ a instância circular com $4 \leq n \leq 20$ equipes. Os nós são colocados em uma circunferência, de

tal forma que a distância de um nó a seu vizinho é igual a 1. Os nós são numerados crescentemente $0, 1, \ldots, n-1$. Existem arcos somente entre os nós i e $i+1 \bmod n$, para $i = 0, 1, \ldots, n-1$. Então, a distância entre os nós i e j (com $i > j$) é dada pelo comprimento do caminho mais curto entre eles e é igual ao mínimo entre $i-j$ e $j-i+n$.

A segunda classe de instâncias foi criada a partir de dados reais, usando-se as distâncias entre as cidades de um sub-conjunto das equipes da *National League* da MLB (*Major League Baseball*) dos Estados Unidos. Denota-se por nl$n$ a instância da *National League* com $4 \leq n \leq 16$ equipes.

Além dessas duas classes de instâncias, foi considerado nos testes também uma instância adicional, criada por Ribeiro e Urrutia em [134] e nomeada $br24$. Essa instância foi construída com dados reais, a partir das distâncias entre as cidades sede das 24 equipes que jogaram na primeira divisão do Campeonato Brasileiro de Futebol de 2003. Essa instância, assim como as outras, pode ser obtida em [145].

Foram desconsideradas as instâncias menores, com $n \leq 6$ equipes, tendo em vista que para essas instâncias os valores ótimos já são conhecidos na literatura.

Os testes realizados foram divididos em grupos de experimentos. Para alguns experimentos, os algoritmos foram programados para encerrar sua execução tão logo seja encontrado uma solução com custo menor ou igual a um valor alvo estabelecido.

5.6.2 Primeiro Experimento - Qualidade das Soluções

Este experimento objetivou investigar o quanto a cooperação entre os processos e a diversidade nas buscas pode melhorar a qualidade das soluções. Na implementação paralela PAR-MP o tamanho da lista de soluções de elite foi definido igual a P, onde P é o número de processos trabalhadores disparados na execução paralela. Para as versões PAR-1P e PAR-MP, após vários testes experimentais realizados, o parâmetro que representa a probabilidade Q de reiniciar a execução de um processo a partir da fase de busca ILS foi fixado em 10%.

Na Tabela 5.1 encontram-se os resultados obtidos através das quatros metaheurísticas paralelas. Nessa tabela são mostrados, para cada instância, o custo das soluções obtidas pela implementação seqüencial da heurística `GRILS-mTTP`, após cinco dias de processamento [145]; os custos das melhores soluções alcançadas pelas implementações paralelas PAR-I, PAR-O e PAR-1P, após 11 horas de processamento em 24 processadores; os custos das melhores soluções

atingidas pela versão PAR-MP, após seis horas de processamento, também em 24 processadores; e a melhoria percentual obtida por PAR-MP em relação ao custo das melhores soluções alcançadas pelo algoritmo seqüencial. Os valores na última coluna são iguais à diferença entre os custos encontrados por PAR-MP e pela versão seqüencial, dividido pelo custo da solução alcançada pela versão PAR-MP.

Tabela 5.1: Soluções encontradas pelas implementações seqüencial e paralelas.

Instância	Seqüencial	PAR-I/O/1P	PAR-MP	Melhoria (%)
circ8	140	140	140	-
circ10	276	**272**	272	1,45
circ12	456	456	456	-
circ14	714	714	714	-
circ16	1004	**984**	**978**	2,59
circ18	1364	**1308**	**1306**	4,25
circ20	1882	1882	1882	-
nl8	41928	41928	41928	-
nl10	63832	63832	63832	-
nl12	120655	120655	120655	-
nl14	208086	208086	208086	-
nl16	285614	**280174**	**279618**	2,09
br24	506433	**503158**	503158	0,65

Como pode ser observado na Tabela 5.1, PAR-I, PAR-O e PAR-1P encontraram soluções de mesmo custo para todas as instâncias e foram capazes de melhorar as soluções de cinco das 13 instâncias testadas. A implementação PAR-MP ainda foi capaz de melhorar as melhores soluções encontradas pelas outras implementações paralelas, no caso de três instâncias: $circ16$, $circ18$ e $nl16$. A versão PAR-MP demonstrou ser melhor que as demais implementações, sendo capaz de atingir soluções não alcançadas por outro algoritmo paralelo.

Embora as versões PAR-I, PAR-O e PAR-1P tenham alcançado soluções de mesmo custo para todas as instâncias, o tempo de processamento exigido por cada versão para atingir esses valores foi diferente. Dentre as três versões, PAR-1P foi a que apresentou os menores tempo de processamento, devido à cooperação entre os processos trabalhadores. A convergência para boas soluções é beneficiada pela troca de informação entre os processos.

Para os demais experimentos, definiram-se três classes de alvos. Os custos das soluções alcançadas pela versão seqüencial foram chamados de *alvos fáceis*. Os custos melhorados pelas versões paralelas PAR-I, PAR-O e PAR-1P foram

classificados como *alvos médios*. Os custos das soluções que só a versão PAR-MP alcançou são classificados como *alvos difíceis* ($circ16$, $circ18$ e $nl16$).

5.6.3 Segundo Experimento - Escalabilidade

Este experimento objetiva medir a escalabilidade das versões paralelas, a fim de avaliar os benefícios de progressivamente aumentar o número de processadores e de buscas realizadas em paralelo.

Os valores apresentados nas Tabelas 5.2 e 5.3 representam o tempo médio de processamento sobre cinco execuções para cada alvo, utilizando-se sementes diferentes para o gerador de números aleatórios. O critério de parada utilizado consiste em interromper cada algoritmo após encontrar uma solução tão boa quanto o alvo fácil, apresentado na Tabela 5.1.

Na Tabela 5.2 é mostrado o tempo, em segundos, requerido para encontrar uma solução com custo no mínimo tão bom quanto o valor alvo para a versão seqüencial e para as versões paralelas PAR-I e PAR-O, em oito, 16 e 24 processadores. A Tabela 5.3 fornece as mesmas informações, porém para as versões paralelas PAR-1P e PAR-MP.

Tabela 5.2: Tempo médio de processamento em segundos para encontrar um alvo fácil em um processador com a versão seqüencial, e em oito, 16 e 24 processadores com as versões PAR-I e PAR-O.

	Seqüencial	PAR-I			PAR-O		
Instância	$P = 1$	$P = 8$	$P = 16$	$P = 24$	$P = 8$	$P = 16$	$P = 24$
circ8	0,87	0,27	0,15	0,14	0,26	0,15	0,14
circ10	197,64	26,32	18,32	15,98	26,31	18,32	15,97
circ12	4,48	1,33	1,11	0,55	1,32	1,11	0,54
circ14	3,46	0,87	0,82	0,73	0,86	0,73	0,71
circ16	413,73	56,54	33,93	24,78	56,58	33,97	24,77
circ18	175,14	43,19	30,06	13,62	43,24	29,99	13,62
circ20	800,94	177,28	82,99	43,47	176,67	83,05	43,47
nl8	0,79	0,23	0,08	0,07	0,22	0,08	0,06
nl10	453,54	113,22	39,03	19,52	113,22	39,03	19,51
nl12	22,13	4,23	1,83	1,34	4,23	1,83	1,33
nl14	33,23	5,32	4,80	4,34	5,32	4,80	4,35
nl16	1.433,05	474,12	243,14	73,62	474,26	243,11	73,58
br24	156,32	40,87	33,83	29,08	40,54	33,70	28,97

Dentre as quatros versões paralelas, a que mais se destacou foi a versão PAR-MP. Essa paralelização conseguiu os melhores tempos de processamento, pois com o crescimento do número de processadores, cresce também a quanti-

Tabela 5.3: Tempo médio de processamento em segundos para encontrar um alvo fácil em um processador com a versão seqüencial, e em oito, 16 e 24 processadores com as versões PAR-1P e PAR-MP.

	Seqüential	PAR-1P			PAR-MP		
Instância	$P = 1$	$P = 8$	$P = 16$	$P = 24$	$P = 8$	$P = 16$	$P = 24$
circ8	0,87	0,19	0,16	0,12	0,19	0,15	0,11
circ10	197,64	26,31	17,04	15,97	26,31	17,04	15,96
circ12	4,48	1,33	0,39	0,35	1,33	0,38	0,34
circ14	3,46	0,65	0,53	0,52	0,65	0,53	0,51
circ16	413,73	55,30	33,93	24,45	55,31	33,94	24,50
circ18	175,14	24,73	21,96	13,60	24,70	23,96	13,62
circ20	800,94	106,34	70,50	43,47	106,30	70,47	43,46
nl8	0,79	0,17	0,10	0,08	0,16	0,08	0,07
nl10	453,54	113,21	39,01	19,52	125,43	39,05	19,51
nl12	22,13	3,91	1,61	1,33	3,91	1,64	1,33
nl14	33,23	5,85	4,80	4,35	5,84	4,80	4,34
nl16	1.433,05	323,33	236,36	73,59	322,27	223,61	73,54
br24	156,32	33,99	25,29	20,23	30,77	23,71	16,55

dade de processos em paralelo que mandam soluções para a lista de soluções de elite. Com isso, a troca de informação entre os trabalhadores aumenta a probabilidade de que boas soluções sejam encontradas em um menor tempo.

Como PAR-MP apresentou o melhor desempenho, outro experimento foi feito com o objetivo de melhor avaliar a escalabilidade dessa implementação. O experimento consistiu em executar PAR-MP com a instância $br24$ em quatro, oito, 16 e 24 processadores 200 vezes. A Figura 5.1 apresenta um gráfico do tipo tttplot, que mostra a distribuição de probabilidade do tempo necessário para o algoritmo PAR-MP encontrar uma solução cujo custo seja pelo menos tão bom quanto o valor alvo igual a 506433.

Por esse gráfico, observa-se que a versão PAR-MP atingiu o alvo definido mais rapidamente com o maior número de processadores. Nota-se também que existem diferenças de tempos consideráveis entre a execução mais rápida e mais lenta para todas as quantidades de processadores testadas. Todavia, a medida que o número de processadores aumenta, essas diferenças de tempos são reduzidas.

A aceleração S de um programa paralelo pode ser definida como a relação entre o tempo $F(1)$ gasto para executar o melhor algoritmo seqüencial e o tempo $F(N)$ gasto por algum algoritmo paralelo quando executado em N processadores [63]:

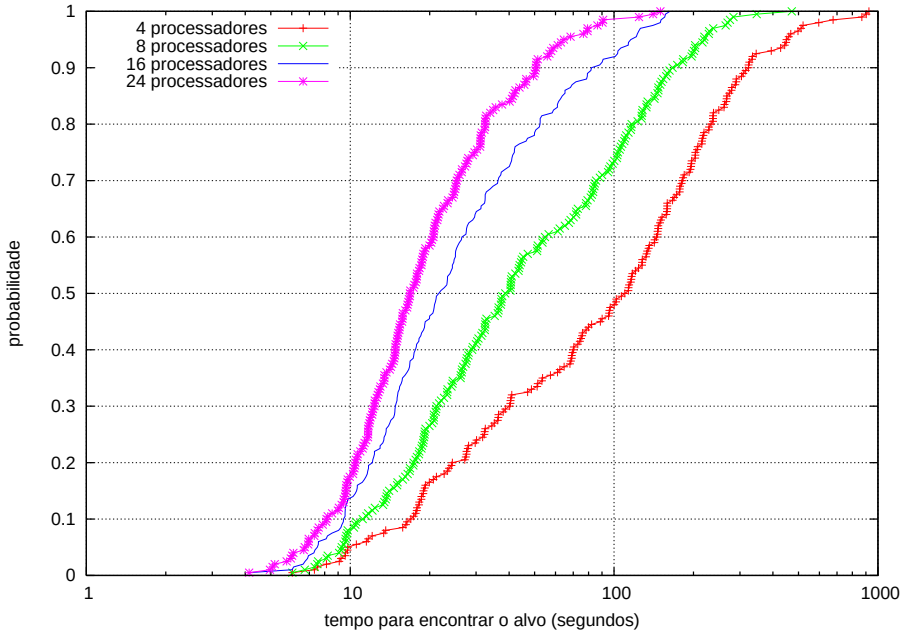

Figura 5.1: Distribuição de probabilidade empírica do tempo para atingir uma solução com custo tão bom quanto o valor alvo, considerando-se o alvo 506433 para a instância *br*24. A versão PAR-*M*P foi executada 200 vezes em quatro, oito, 16 e 24 processadores.

$$S = \frac{F(1)}{F(N)}$$

Outra medida usada para avaliar os programas paralelos é a eficiência E, que trata da relação entre a aceleração S e o número de processadores N [63]:

$$E = \frac{S}{N}$$

As Tabelas 5.4 e 5.5 mostram a aceleração obtida pelas versões paralelas. Essas acelerações foram calculadas considerando-se os tempos da execução seqüencial de `GRILS-mTTP`, e os tempos médios das execuções paralelas mostrados nas Tabela 5.2 e 5.3. As acelerações médias para 24 máquinas foram iguais a 12,51 (PAR-I), 12,62 (PAR-O), 13,19 (PAR-1P) e 13,41 (PAR-MP).

Como pode ser notado, com o aumento no número de processadores usados na execução, houve uma diminuição do tempo de processamento das versões paralelas. A aceleração da versão paralela PAR-*M*P foi maior do que a dos demais algoritmos. Isso ocorre porque com o aumento de processadores há mais processos trabalhadores gerando soluções para serem armazenadas na lista de soluções de elite. Além disso, como a funcionalidade do processo mestre é só gerenciar essa lista, o gargalo no mestre, apesar de ser um risco, torna-se mais distante de ocorrer.

Tabela 5.4: Aceleração em oito, 16 e 24 processadores (versões paralelas PAR-I e PAR-O).

	PAR-I			PAR-O		
Instância	$P = 8$	$P = 16$	$P = 24$	$P = 8$	$P = 16$	$P = 24$
circ8	3,23	5,69	5,87	3,28	5,70	6,21
circ10	7,50	10,79	12,37	7,51	10,79	12,38
circ12	3,37	4,02	8,21	3,39	4,03	8,21
circ14	3,99	4,21	4,74	4,00	4,72	4,74
circ16	7,32	12,20	16,70	7,31	12,18	16,71
circ18	4,06	5,83	12,87	4,05	5,84	12,86
circ20	4,52	9,65	18,42	4,53	9,64	18,43
nl8	3,51	9,36	11,15	3,61	9,38	12,34
nl10	4,00	11,62	23,24	4,01	11,62	23,25
nl12	5,23	12,09	16,56	5,23	12,12	16,63
nl14	6,25	6,93	7,65	6,24	6,93	7,64
nl16	3,02	5,89	19,47	3,02	5,89	19,48
br24	3,83	4,62	5,38	3,86	4,64	5,40
média	4,60	7,91	12,51	4,62	7,96	12,62

Tabela 5.5: Aceleração em oito, 16 e 24 processadores (versões paralelas PAR-1P e PAR-MP).

	PAR-1P			PAR-MP		
Instância	$P = 8$	$P = 16$	$P = 24$	$P = 8$	$P = 16$	$P = 24$
circ8	4,42	5,40	7,23	4,57	5,65	7,51
circ10	7,51	11,60	12,37	7,51	11,60	12,39
circ12	3,36	11,23	12,67	3,35	11,58	13,07
circ14	5,30	6,50	6,69	5,29	6,54	6,78
circ16	7,48	12,19	16,92	7,48	12,19	16,89
circ18	7,08	7,98	12,87	7,09	7,31	12,86
circ20	7,53	11,36	18,42	7,53	11,37	18,43
nl8	4,80	7,24	9,58	4,99	9,90	9,95
nl10	4,01	11,63	23,24	3,62	11,62	23,25
nl12	5,66	13,75	16,59	5,67	13,50	16,64
nl14	5,68	6,92	7,64	5,68	6,92	7,66
nl16	4,43	6,06	19,47	4,45	6,40	19,49
br24	4,60	6,18	7,72	5,08	6,59	9,45
média	5,53	9,08	13,19	5,56	9,32	13,41

5.6.4
Terceiro Experimento - Tempos

O objetivo deste experimento é avaliar o desempenho das quatro versões paralelas, em relação ao tempo necessário para que boas soluções sejam alcançadas. Para isso, os algoritmos foram programados para encerrarem suas execuções tão logo encontrem uma solução com custo menor ou igual ao valor alvo estabelecido. O alvo escolhido foi o médio porque é o valor alcançado por todas as quatro versões paralelas. Foram consideradas exclusivamente as instâncias que tiveram suas soluções melhoradas pelas versões paralelas.

Os resultados são mostrados na Tabela 5.6, na qual há uma coluna com o tempo médio em segundos, de cinco execuções em 24 máquinas, para cada versão paralela. Os resultados mostram que PAR-MP apresenta os menores tempos para três das cinco instâncias.

Tabela 5.6: Tempos médios em segundos sobre cinco execuções das implementações PAR-I, PAR-O, PAR-1P e PAR-MP em 24 processadores, para alcançar os alvos médios.

Instância	Alvo	PAR-I	PAR-O	PAR-1P	PAR-MP
circ10	272	5.768,81	4.650,31	7.209,82	3.725,68
circ16	984	5.366,12	735,73	3.881,09	959,24
circ18	1308	9.323,88	8.565,20	13.972,76	10.620,86
nl16	280174	12.207,18	11.488,44	7.058,52	3.171,40
br24	503158	4.322,45	4.268,41	5.046,40	2.220,69

PAR-MP atingiu soluções para as instâncias $circ16$, $circ18$ e $nl16$ que nenhuma outra versão paralela conseguiu. Assim, a Tabela 5.7 sumariza os melhores resultados obtidos por PAR-MP para essas três instâncias, a melhoria relativa do custo da solução encontrada com relação às melhores soluções encontradas pelas outras versões paralelas, e os tempos médios em segundos sobre cinco execuções, usando 24 processadores. O critério de parada usado nesse teste foi o alvo difícil.

Tabela 5.7: Melhores soluções obtidas pela versão PAR-MP.

Instância	Custo da melhor solução	Melhoria (%)	Tempo (s)
circ16	978	0,61	4.690,83
circ18	1306	0,15	20.883,81
nl16	279618	0,20	14.586,73

Como pode ser observado na Tabela 5.7, o custo da solução encontrada pela versão paralela PAR-MP teve uma melhoria de até 0,61% em relação às melhores soluções encontradas pelas demais versões paralelas.

A Tabela 5.8 apresenta resultados de um experimento cujo objetivo é avaliar o melhor resultado encontrado pelas versões paralelas PAR-I, PAR-O e PAR-1P caso elas executassem o dobro do tempo usado pela versão PAR-MP para atingir as soluções relatadas na Tabela 5.7 (ou seja, os alvos difíceis). Para isso, as implementações PAR-I, PAR-O e PAR-1P foram executadas em 24 processadores até que o tempo limite dado esgotasse. O custo da melhor solução encontrada por cada versão para as três instâncias testadas é mostrado na Tabela 5.8.

Os resultados mostram que outras implementações paralelas não foram capazes de encontrar soluções tão boas quanto aquelas atingidas por PAR-MP, mesmo com o dobro do tempo de processamento. Fica comprovada a superioridade da versão PAR-MP em relação às demais implementações paralelas, tanto em relação à qualidade das soluções quanto em relação ao tempo para obter boas soluções, pois as demais não foram capazes de encontrar todas as soluções alcançadas por PAR-MP.

Tabela 5.8: Soluções encontradas pelas implementações PAR-I, PAR-O e PAR-1P quando executadas pelo dobro do tempo gasto por PAR-MP.

Instância	Tempo (s)	Alvo	PAR-I	PAR-O	PAR-1P
circ16	10.000	978	984	984	984
circ18	40.000	1306	1308	1308	1308
nl16	30.000	279618	280174	280174	280174

Para finalizar, a Figura 5.2 apresenta um gráfico do tipo tttplot, que mostra a distribuição de probabilidade do tempo necessário para os algoritmos paralelos PAR-I, PAR-O, PAR-1P e PAR-MP encontrarem uma solução cujo custo seja pelo menos tão bom quanto o valor alvo igual a 284000 para a instância $nl16$. Esse gráfico corresponde aos resultados obtidos após 200 execuções independentes de cada versão paralela, executadas em oito processadores.

Observa-se na Figura 5.2 que a versão PAR-MP teve um desempenho melhor que as demais versões. Por exemplo, ela encontra o valor alvo em menos de 1.400 segundos com probabilidade de 95%, comparado com 2.445 segundos com a mesma probabilidade para a próxima versão mais rápida entre as demais (PAR-1P). Além disso, nota-se que PAR-MP gasta aproximadamente 1.500

segundos na sua execução mais lenta, enquanto PAR-1P leva mais de 4.300 segundos na sua pior execução.

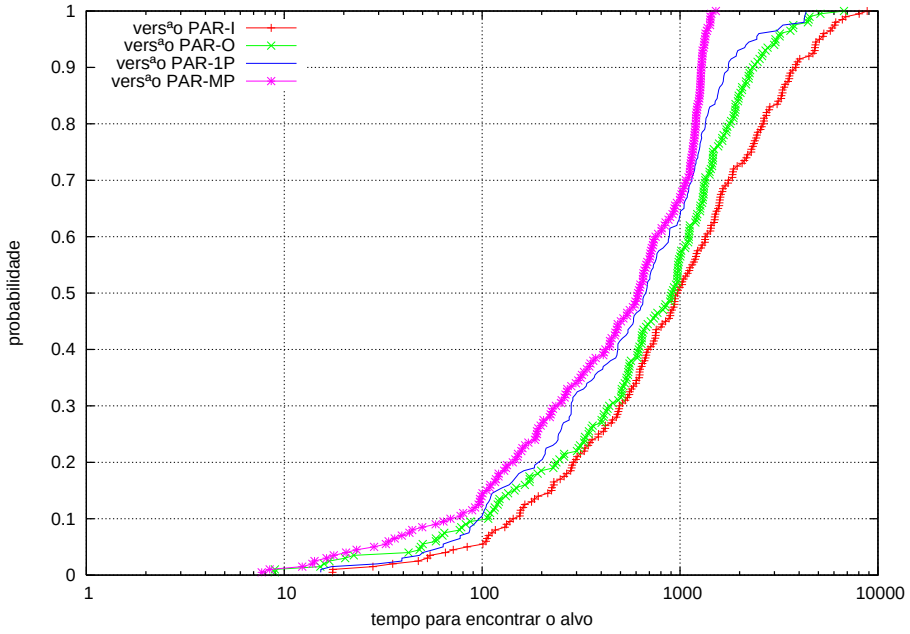

Figura 5.2: Distribuição de probabilidade empírica do tempo para atingir uma solução com custo tão bom quanto o valor alvo, considerando-se o alvo 284000 para a instância *nl*16. As quatro versões paralelas PAR-I, PAR-O, PAR-1P, e PAR-*M*P foram executadas 200 vezes cada uma, em um ambiente formado por oito processadores.

5.7 Considerações Finais

Este capítulo apresentou quatro implementações paralelas da heurística híbrida `GRILS-mTTP` para o mTTP, seguindo estratégias tradicionais de paralelização. Como conseqüência do trabalho desenvolvido neste capítulo foram publicados os artigos [13, 16, 17].

A versão PAR-*M*P apresentou o melhor desempenho, tanto na qualidade das soluções encontradas, quanto nos tempos de processamento. Contudo, apesar do desempenho apresentado por essas implementações, as mesmas não são totalmente adequadas para ambientes distribuídos dinâmicos como os *grids*. Os quatros algoritmos têm como característica comum a presença de um único processo gerenciador (o mestre), que pode tornar-se um gargalo potencial, causando impactos negativos no desempenho da aplicação. Esse impacto pode ser ainda maior em ambientes de *grid*, nos quais a variação de carga e a disponibilidade dos recursos são características importantes.

Além disso, nestas implementações paralelas as tarefas são criadas estaticamente, ou seja, a aplicação paralela ao ser ativada precisa informar quantos

processos trabalhadores devem ser criados. Essa é uma limitação do ambiente de troca de mensagens MPI, que adota o modelo de execução em que as tarefas são criadas e realizam processamento até que o critério de parada da aplicação seja atingido. Em ambientes dinâmicos, a situação ideal consiste em trabalhar com tarefas que possam ser criadas e destruídas dinamicamente, demonstrando que a aplicação pode se adaptar às mudanças sofridas no ambiente ao longo de sua execução.

Apesar das estratégias padrão de paralelização apresentarem um bom desempenho em plataformas distribuídas como *cluster*, as mesmas não se adequam às características do ambiente *grid*, tais como a variação na carga e a disponibilidade dos recursos. Dessa forma, é necessário desenvolver estratégias capazes de se adaptarem às mudanças dinâmicas ocorridas em um *grid* computacional. Essas estratégias devem ser capazes de tratar a localização remota dos recursos, a heterogeneidade e a escalabilidade do ambiente.

Ademais, nenhuma das implementações PAR-I, PAR-O, PAR-1P e PAR-*M*P tratam de questões relacionadas ao gerenciamento do ambiente de execução, tais como descoberta de recursos, tolerância a falhas e monitoramento. Dessa forma, para que a aplicação possa aproveitar ao máximo a alta capacidade de processamento do *grid*, faz-se necessário que cada aplicação seja capaz de gerenciá-lo da maneira mais adequada às suas características específicas.

Para facilitar a implementação eficiente de metaheurísticas em ambiente de *grid*, esta tese propõe no próximo capítulo uma nova estratégia para paralelizar metaheurísticas, integrado com uma camada de gerência do ambiente.