Heurísticas Híbridas para Escalonamento Estático de Tarefas em Sistemas com Processadores Heterogêneos

Eyder Franco Sousa Rios

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação Aplicada e Automação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre.

Área de concentração: Otimização e Inteligência Artificial.

Orientador : Luiz Satoru Ochi, DSc
Co-orientadora : Maria Cristina Silva Boeres, PhD
Programa de Pós-Graduação em Computação - IC/UFF

Niterói, Dezembro de 2004.

Heurísticas Híbridas para Escalonamento Estático de Tarefas em Sistemas com Processadores Heterogêneos

Eyder Franco Sousa Rios

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação Aplicada e Automação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Otimização e Inteligência Artificial.

Aprovada por:

Prof. Luiz Satoru Ochi, DSc - IC/UFF (Presidente)

Profa. Maria Cristina Silva Boeres, PhD - IC/UFF

Prof. Eugene Francis Vinod Rebello, PhD - IC/UFF

Prof. Fábio Protti, DSc - DCC/UFRJ

Profa. Maria Cláudia Silva Boeres, DSc - DI/UFES

Niterói, Dezembro de 2004.

Para minha esposa Josiane e meus filhos Juliana e Mateus.

Pois o homem não sabe a sua hora. Como os peixes que se apanham com a rede traiçoeira e como os passarinhos que se prendem com o laço, assim se enredam também os filhos dos homens no tempo da calamidade, quando cai de repente sobre eles.

Eclesiastes, 9:12

Agradecimentos

Agradeço a todos que de alguma forma contribuíram para o desfecho deste trabalho. Primeiramente, agradeço a minha esposa Josiane que sempre me incentivou e assumiu sozinha os compromissos de nosso lar. Aos meus filhos Juliana e Mateus (nascido durante a escrita deste trabalho) que permitiram e suportaram meus diversos períodos de ausência. A vocês dedico meu amor incondicional. Aos meus pais, que desde o início proporcionaram condições para meu desenvolvimento pessoal e intelectual. À minha irmã Leyla, que me apoiou e deu suporte nesta empreitada. Ao meu irmão Herbert (Bebeto), sempre disposto a ajudar apesar da distância.

Agradeço aos colegas do Programa, que proporcionaram um ambiente trabalho agradável marcado pelo companheirismo: Elzenclever, Márcia, Eduardo, Deolinda, Geisa, Helder, Rodrigo, Jacques, Viterbo, Daniela, Viviane, Renatha, Adriana, Alexandre, Aline, Cristiane, Idalmis, Haroldo, Ivairton, Ádria, Luciana, Cristiano e Luciene. Ao pessoal da República Mouseville, que me acolheu num ambiente de convívio alegre e descontraído: André (Totó), Áthila (Taz), Eduardo (Cêcê), Jonivan e Sanderson. Um agradecimento especial ao amigo e conterrâneo Stênio que foi companheiro desde o início do curso e sempre me incentivou nos momentos difíceis. Foi uma grata satisfação descobrir em vocês, além de colegas, verdadeiros amigos.

Ao professores do Departamento com os quais tiver o prazer de conviver neste período: Martinhon, Izabel Cafezeiro, Helena, Lúcia e Plastino. Aos membros

da banca que enriqueceram este trabalho com suas observações e sugestões: Fábio Protti, Cláudia Boeres e Vinod Rebello. Agradecimentos também a Ângela e Izabela que com presteza e dedicação sempre estiveram prontas para ajudar, dando suporte na Secretaria do Programa. Cabe aqui um agradecimento oportuno ao nosso mascote Matheus, filho de Izabela, pelos momentos de alegria proporcionados.

Um agradecimento especial aos meus orientadores Luiz Satoru e Cristina Boeres que, com paciência e dedicação, transmitiram os conhecimentos necessários para a elaboração deste trabalho. Sinto-me orgulhoso em dizer que o exemplo de profissionalismo demonstrado por vocês, despertou em mim o desejo de me tornar um profissional cada vez melhor.

Resumo da Tese apresentada ao Programa de Pós-Graduação em Computação do Instituto de Computação da UFF como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação (MSc).

Heurísticas Híbridas para Escalonamento Estático de Tarefas em Sistemas com Processadores Heterogêneos

Eyder Franco Sousa Rios

Dezembro/2004

Orientador : Luiz Satoru Ochi, DSc
Co-orientadora : Maria Cristina Silva Boeres, PhD
Programa de Pós-Graduação em Computação - IC/UFF

A exploração dos recursos existentes em sistemas computacionais distribuídos através da alocação adequada dos componentes de aplicações paralelas não é um processo elementar. Tal procedimento constitui o Problema de Escalonamento de Tarefas que é, em sua forma geral, um problema NP-Completo e tem sido bastante explorado pela literatura especializada. A alta complexidade deste problema tem incentivado a pesquisa de métodos heurísticos para sua resolução, onde algoritmos da classe *list scheduling* constituem mecanismos comumente empregados. Todavia, a característica inerentemente gulosa destas heurísticas pode, em muitos casos, contribuir negativamente para o desempenho destes algoritmos. Este trabalho investiga a aplicabilidade da integração de heurísticas da classe *list scheduling* com mecanismos de busca baseados em metaheurísticas. O objetivo é combinar o poder de busca das metaheurísticas com a baixa complexidade de heurísticas *list scheduling* para a geração de escalonamentos eficientes em sistemas computacionais distribuídos, cujos recursos são geralmente heterogêneos. Para tanto, são propostas duas heurísticas denominadas HTSGA e HTSG baseadas, respectivamente, em Algoritmo Genético e GRASP. O mecanismo de busca implementado por estas heurísticas procura eliminar o comportamento guloso dos algoritmos *list scheduling* fornecendo novas possibilidades de escalonamento através da determinação de diferentes seqüências de

atribuição de tarefas durante a construção de soluções. Além disso, foi investigada a possibilidade de adaptação das heurísticas propostas a diferentes classes de aplicações paralelas. Como forma de validação deste trabalho, o desempenho das heurísticas propostas foi comparado com algoritmos de construção tradicionais existente na literatura e com uma metaheurística que serviu de base para este trabalho. Os resultados mostraram que os algoritmos propostos são robustos tanto em relação à adaptabilidade às características das instâncias submetidas quanto na rápida convergência para soluções sub-ótimas.

Abstract of Thesis presented to UFF as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

Hybrid Heuristics for Static Task Scheduling in Computing Systems with Heterogeneous Processors

Eyder Franco Sousa Rios

December/2004

Advisor : Luiz Satoru Ochi, DSc
Co-Advisor : Maria Cristina Silva Boeres, PhD
Department of Computer Science - IC/UFF

The utilization of existing resources in distributed computing systems with adequate allocation of parallel application's components is not an elementary issue. Such process constitutes the Task Scheduling Problem which is in general a NP-complete problem, and it has been explored in specialized literature. The complexity of this problem has motivated the development of heuristic methods to solve it, in which algorithms within the list scheduling class constitute commonly used mechanisms. However, the greedy characteristic of such heuristics can, in many cases, contribute negatively on the achievmento of the best solution. This work investigates the applicability of the integration of List Scheduling class heuristics with search mechanisms based on meta-heuristics. The goal is to combine meta-heuristics search power with low-complexity of List Scheduling heuristics for generating efficient schedules on distributed computing systems, in which resources are heterogeneous, in general. For this, two heuristics are proposed: HTSGA and HTSG, which are based on Genetic Algorithm and GRASP, respectively. The search mechanisms implemented in such heuristics aims to eliminate the greedy behaviour of List Scheduling algorithm, providing new scheduling possibilities by determinating different task allocation sequences during the solution construction. Moreover, it was investigated the possibility of adapting proposed heuristics to different parallel application classes. To validate this work, the solutions generated by the proposed heuristics were

compared with traditional construction algorithms presented in the literature and to a meta-heuristic which was the basis of this work. The results showed that the proposed algorithms are robust both related to adaptability to characteristics of the submitted instances and to quick convergency to sub-optimal solutions.

Palavras-chave

1. Escalonamento Estático de Tarefas
2. Metaheurísticas
3. Algoritmo Genético
4. GRASP

Keywords

1. Static Task Scheduling
2. Metaheuristics
3. Genetic Algorithm
4. GRASP

Glossário

AG	:	Algoritmo Genético (*Genetic Algorithm*);
GRASP	:	*Greedy Radomized Adaptative Search*;
GAD	:	Grafo Acíclico Direcionado;
GADE	:	Grafo Acíclico Direcionado Escalonado;

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Desde seu surgimento, a indústria de computadores tem concentrado seus esforços no sentido de desenvolver a tecnologia de construção de sistemas computacionais. A cada ano, os componentes de *hardware* tornam-se mais compactos e mais rápidos, possibilitando a criação de equipamentos cada vez mais poderosos. Apesar deste desenvolvimento, ainda existem aplicações nas diversas áreas do conhecimento que demandam um poder computacional ainda maior. Modelagens meteorológicas, decodificação dos genômas das espécies, aplicações da indústria aeronáutica, simulações da economia, processamento de sinais, são exemplos de aplicações computacionalmente intensivas [1].

Para atender à demanda de tais aplicações, uma alternativa é utilizar equipamentos mais poderosos através da aquisição de processadores mais rápidos e de outros componentes que incrementem a velocidade de processamento destes sistemas. Contudo, características como os limites tecnológicos na fabricação de *hardware* ou mesmo os altos custos destes equipamentos são fatores que restringem estes avanços. Uma alternativa viável de agregar poder computacional a estes equipamentos é a utilização de múltiplos processadores interconectados, de forma que juntos possam aumentar a capacidade de processamento do sistema. Tais sistemas são conhecidos como **computadores paralelos** e possibilitam que uma aplicação seja dividida em diversas tarefas computacionais que são executadas pelos processadores do sistema. Na busca por equipamentos cada vez mais poderosos, **supercomputadores** com

alto poder de processamento chegam a possuir milhares de processadores interconectados. Um exemplo de um sistema deste tipo é o *IBM ASCI White* que, com seus 8.192 processadores, executa até 12,3 trilhões de cálculos por segundo. Todavia, o custo de dezenas de milhões de dólares deste equipamento, o torna proibitivo para a maioria das aplicações [2].

A evolução da microeletrônica tem permitido a melhoria constante dos projetos de construção de computadores, inclusive os de menor porte. De fato, a capacidade computacional dos sistemas vem dobrando a cada 18 meses enquanto o custo permanece constante. Já os avanços tecnológicos na construção de equipamentos para conexão de computadores atinge esta mesma taxa a cada 9 meses [3]. Estes avanços motivaram o desenvolvimento dos **sistemas computacionais distribuídos**, formados por diversos computadores interconectados que compartilham seus recursos de processamento e armazenamento de dados. A proliferação dos equipamentos de computação de baixo custo associado ao desenvolvimento de sistemas de comunicação de alto desempenho, contribuíram para o surgimento de um tipo de arquitetura distribuída conhecida como **clusters de computadores** [4]. Este paradigma, permitiu que a aglomeração de diversos equipamentos de pequeno porte incrementasse o desenvolvimento de aplicações paralelas a um custo acessível. Recentemente, uma outra alternativa que tem se desenvolvido bastante são as chamadas **grades computacionais** [5]. O termo, cunhado no final dos anos 90, é usado para representar um sistema dinâmico de alto desempenho onde computadores interligados e geograficamente distribuídos podem ser vistos como um único grande recurso computacional. Este conceito foi estendido posteriormente para contemplar não só a integração de processadores, mas também de outros recursos como bancos de dados, instrumentos científicos, sensores *etc* [6]. Talvez um dos exemplos mais famosos de processamento distribuído seja o projeto Seti@Home [7], onde diversos voluntários em todo o mundo executam um pequeno programa que utiliza os ciclos de processamento ociosos de seus computadores para analisar sinais de rádio oriundos do espaço. Isso permitiu que mais de 5 milhões de equipamentos conectados via Internet criassem um sistema distribuído com um poder computacional sem precedentes.

No entanto, para que aplicações possam ser executadas de maneira adequada por sistemas paralelos ou distribuídos, é necessário que o paralelismo existente tanto nas tarefas quanto no sistema seja convenientemente explorado [8]. Neste trabalho, as aplicações paralelas são compostas por tarefas atômicas (indivisíveis) que podem trocar informações entre si, gerando relações de dependência. A alocação eficiente de tais aplicações paralelas em sistemas distribuídos corresponde ao problema de escalonamento de tarefas. Em sua forma geral, o escalonamento de tarefas é um problema NP-Completo [9] e tem sido bastante abordado pela literatura especializada [10, 11]. Contudo, muitos dos trabalhos encontrados consideram simplificações na modelagem do problema que podem não corresponder à realidade. Algumas propostas encontradas na literatura assumem, por exemplo, sistemas computacionais com um número infinito de processadores [12, 13] ou ignoram os custos envolvidos nas trocas de informações entre as tarefas de uma aplicação paralela [14, 15, 16]. Tais simplificações tem por objetivo reduzir a complexidade do problema e podem não levar a resultados práticos.

Atualmente, com o crescente desenvolvimento da tecnologia de conexão de computadores, cada vez mais sistemas são interligados formando redes locais ou geograficamente distribuídas. Neste contexto, alguns algoritmos de escalonamento orientados para sistemas heterogêneos tem sido propostos [17, 18, 19], sendo a maioria deles baseados em heurísticas pertencentes à classe *list scheduling* [20, 21, 22, 23]. Os algoritmos desta classe possuem características atraentes como a baixa complexidade e a relativa simplicidade quando comparadas com outras classes de heurísticas [8].

O objetivo deste trabalho é estudar a integração de heurísticas de escalonamento da classe *list scheduling* com mecanismos de busca baseados em metaheurísticas. Para tanto, heurísticas tradicionais como EFT [17], HEFT [21] e ETF [24] foram adaptadas e incorporadas a duas propostas baseadas em Algoritmo Genético e GRASP. A intenção é combinar o poder de busca das metaheurísticas com a baixa complexidade dos algoritmos *list scheduling* para a produção de escalonamentos eficientes para ambientes computacionais heterogêneos distribuídos. No entanto, o uso de metaheurísticas para a solução do problema de escalonamento de tarefas é pe-

nalizado pela alta complexidade envolvida no processo de busca implementado por estes algoritmos. Utilizada como uma alternativa para minimizar este problema, a técnica implementada pelos algoritmos aqui propostos, baseada no trabalho do Ahmad *et al* [17], foi a separação dos processos de busca e construção de soluções. Esta técnica, combinada com recursos adicionais propostos neste trabalho, além de reduzir a complexidade dos algoritmos propostos, levou à produção de heurísticas robustas, eficientes e confiáveis.

Este documento está organizado da seguinte forma. No Capítulo 2 é apresentado o embasamento teórico sobre o problema de escalonamento bem como um levantamento sobre o estado da arte sobre o assunto. Também são definidos os modelos de aplicação e de arquitetura adotados neste estudo. O Capítulo 3 apresenta heurísticas de escalonamento da literatura que estão relacionadas com este trabalho. No Capítulo 4 são propostas duas metaheurísticas baseadas em Algoritmos Genéticos e GRASP para o problema de escalonamento de tarefas em sistemas heterogêneos. No Capítulo 5 são apresentados e analisados os resultados obtidos pelas heurísticas propostas. Finalmente, no Capítulo 6 são apresentadas as conclusões e propostas de trabalhos futuros.

Capítulo 2

O Problema de Escalonamento de Tarefas

O bom desempenho na execução de uma aplicação em um ambiente de computação com multiprocessadores paralelos ou distribuídos, depende criticamente da distribuição das tarefas da aplicação nos processadores do sistema. Diversas características tanto arquiteturais quanto relacionadas com a própria aplicação influenciam o projeto de algoritmos de escalonamento. Um dos fatores principais que deve ser levado em consideração no escalonamento de tarefas em sistemas modernos é o custo de comunicação entre tarefas alocadas em diferentes processadores. Uma outra característica importante está relacionada com a heterogeneidade dos recursos de *hardware* disponíveis no sistema de computação [11]. A heterogeidade de um sistema de computação é determinada pela presença de componentes de *hardware* pertencentes a uma mesma classe que apresentam características distintas. Sistemas que possuem processadores com diferentes capacidades de processamento e/ou canais de comunicação com velocidades de transmissão variadas, podem ser citados como exemplos de ambientes computacionais heterogêneos.

O Problema de Escalonamento de Tarefas consiste, basicamente, em determinar a alocação das tarefas de uma aplicação paralela nos processadores de um

ambiente de computação de forma a minimizar o tempo total de execução. Este problema depende fortemente da estrutura da aplicação paralela, da arquitetura do ambiente de computação alvo, da uniformidade dos custos de computação das tarefas e do critérios de desempenho escolhidos [11]. De fato, o problema de escalonamento de tarefas é, na sua forma geral, um problema NP-Completo [9]. Contudo, para casos restritos, propostas menos complexas podem ser encontradas [25, 26, 27, 28].

Por diversas razões, casos restritos do problema de escalonamento podem não levar a aplicações práticas. Por exemplo, nem sempre é realista assumir que todas as tarefas de uma aplicação paralela possuem o mesmo custo de computação, já que o número de instruções embutidas em cada tarefa certamente é variável. Ou ainda, detalhes arquiteturais do sistema de computação alvo, como o compartilhamento de memória ou a comunicação interprocessadores, influenciam diretamente no desempenho da aplicação [11]. Desta forma, algoritmos de escalonamento devem considerar em seu projeto a manipulação de características intrínsecas da aplicação e da arquitetura de forma a possibilitar um melhor desempenho da aplicação escalonada em um sistema paralelo. Tais características são obtidas através da especificação de um **modelo de escalonamento**, composto pelo **modelo arquitetural** e **modelo de aplicação**.

2.1 Modelo Arquitetural

O modelo de um sistema de computação define as características arquiteturais relevantes ao ambiente paralelo a ser considerado. Podem ser encontrados na literatura diversos modelos para ambientes multiprocessados paralelos ou distribuídos com abordagens abstratas [12, 13] ou realistas [29, 30, 31, 32]. Estes modelos definem características de um sistema paralelo como o tipo de acesso à memória, acoplamento de processadores, heterogeneidade do sistema, entre outras. Durante o desenvolvimento de programas, o ideal é que o programador possa escrever o código sem considerar características específicas do ambiente onde o *hardware* será executado. Para tanto, é fundamental a definição de um modelo de ambiente de

computação que abstraia os detalhes arquiteturais da sistema alvo, mas que, ao mesmo tempo, seja versátil o suficiente para que possa ser aplicado em um grande número de plataformas paralelas reais [33]. Na tentativa de acompanhar o rápido desenvolvimento dos sistemas paralelos e distribuídos, diversos modelos arquiteturais foram propostos [15, 29, 30, 31, 34, 35, 36] com o objetivo de definir as características relacionadas com a heterogeneidade dos processadores e com o fluxo de comunicação nestes sistemas.

Em busca de um modelo de arquitetura paralela mais realista, Valiant desenvolveu o *Bulk Synchronous Parallelism* (BSP) [32], que foi um dos primeiros paradigmas a levar em consideração os custos de sincronização de mensagens na comunicação entre processadores. Uma máquina BSP consiste em um conjunto finito de processadores com memória local que são gerenciados por um roteador e que trocam mensagens entre si. As aplicações executadas numa máquina BSP são organizadas em blocos de sincronização chamados *supersteps*, onde cada processador recebe um conjunto de tarefas independentes; efetua a transmissão e/ou recebimento de mensagens referentes às tarefas independentes; e realiza a sincronização de mensagens entre todos os processadores.

Motivados pelo avanço tecnológico das redes de microcomputadores, Culler *et al* apresentaram o modelo *LogP* [30] para ambientes paralelos, onde os custos de comunicação e de processamento de mensagens são tratados separadamente. Este modelo considera uma arquitetura, com um número determinado de processadores, que enfatiza atributos relacionados com a comunicação, como a velocidade dos canais (latência) e o custo (*overhead*) e a capacidade (*gap*) de processamento de mensagens. Além disso, este modelo [30, 31, 37, 38, 39, 40, 41, 42] assume que a rede de comunicação tem uma capacidade finita para transmissão simultânea de mensagens. Uma extensão desta arquitetura, o modelo *LogGP* [34], não impõe restrições quanto ao número de mensagens transmitidas pela rede de comunicação, considerando como parâmetro adicional a largura de banda da rede. Esta característica, procura incorporar a capacidade que muitas máquinas paralelas possuem para a transmissão de mensagens longas, com o intuito de obter maior eficiência com aplicações que realizam um tráfego denso de mensagens. Em [43], Boeres e Rebello mostraram que o

uso de técnicas de aglomeração de mensagens a serem transmitidas, pode aumentar significativamente o desempenho de aplicações paralelas neste tipo de sistema. Uma variação da arquitetura definida por *LogGP* é o modelo *LogGPS* [31] que, como característica adicional, considera o tempo de sincronização necessário para o envio de mensagens longas. A importância desta característica foi percebida através da observação do desempenho de programas que utilizam bibliotecas de comunicação de alto nível como MPL [44] e MPI [45]. Este parâmetro, determina a partir de que ponto uma mensagem deve ser enviada em modo síncrono ou assíncrono, ou seja, é utilizado para calibrar o sistema em relação à influência do comprimento das mensagens no desempenho da rede de comunicação.

Como pôde ser observado, os modelos previamente descritos enfatizam a identificação de características arquiteturais que influenciam no desempenho da rede de comunicação em sistemas paralelos. Contudo, à medida que novos parâmetros são agregados a um modelo de arquitetura, torna-se mais complexo o projeto de algoritmos de escalonamento. Segundo Papadimitriou [15], o custo de comunicação entre processadores é o parâmetro mais significativo a ser considerado na arquitetura de um sistema paralelo. De fato, a maior parte dos trabalhos relacionados com o problema de escalonamento encontrados na literatura, representam a comunicação através do modelo de latência [10, 11, 15, 17, 18, 21, 24, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56]. Este modelo, considera ambientes computacionais com multiprocessadores interconectados por canais que apresentam retardo na comunicação. Este retardo, provocado pelo tempo de transmissão de dados entre processadores distintos, é chamado de **latência**. Outra característica presente nesta arquitetura determina que os processadores estão habilitados a fazer *multicast*, ou seja, enviar simultaneamente diversas mensagens para processadores distintos. Neste trabalho, é considerado um modelo arquitetural de latência com um número determinado de processadores heterogêneos, conforme descrito a seguir.

2.1.1 Modelo Arquitetural Adotado

Um ambiente de computação não preemptivo pode ser representado por um grafo direcionado completo $M = (P, L)$, onde $P = \{p_1, \ldots, p_m\}$ denota o conjunto de unidades de processamento heterogêneas, ou simplesmente processadores, com memória local distribuída totalmente conectados e $L = \{(p_i, p_j) \mid p_i, p_j \in P\}$ a rede de canais de comunicação do sistema. O termo $\lambda(p_i, p_j) \in \mathbb{R}^*$ representa o retardo do canal de comunicação (latência) que interliga os processadores p_i e p_j. Cada processador $p_i \in P$ possui um fator de heterogeneidade associado, definido por $h(p_i) \in \mathbb{R}^*$, que é inversamente proporcional à velocidade de processamento de p_i.

A Figura 2.1 mostra um grafo que representa um ambiente de computação com três processadores heterogêneos totalmente conectados. No grafo, os vértices indicam os processadores do sistema distribuído cada qual com um fator de heterogeneidade ($h(p_i)$) associado. As arestas bidirecionais entre os processadores indicam os canais de comunicação com seus respectivos fatores de latência ($\lambda(p_i, p_j)$). No exemplo, p_1 e p_2 são os processadores mais rápidos do sistema, enquanto p_3 é o mais lento. O canal de comunicação entre os processadores p_1 e p_3 é mais lento que os outros dois presentes na rede de comunicação do sistema distribuído. Observe que esta modelagem possibilita a existência de canais com diferentes fatores de latência dependendo do sentido de transmissão, como, por exemplo, em $\lambda(p_1, p_2) \neq \lambda(p_2, p_1)$.

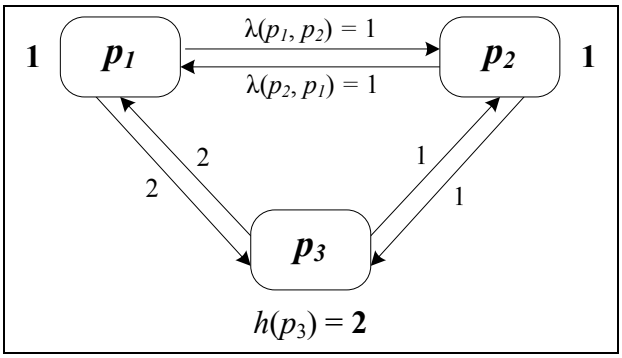


Figura 2.1: Um grafo direcionado completo representando um ambiente computação heterogêneo com três processadores.

Observe que esta modelagem prevê a possibilidade de que um mesmo canal de comunicação apresente diferentes velocidades de transmissão dependendo do sentido em que as messagens trafegam entre os processadores ($\lambda(p_i, p_j) \neq \lambda(p_j, p_i)$).

2.2 Aplicação Paralela

Uma aplicação paralela ou multiprocessada é um conjunto de tarefas de computação interrelacionadas que cooperam através da troca de informações para a obtenção de um resultado comum [57]. Em geral, estas tarefas necessitam se comunicar para trocar informações entre si. Como cada tarefa progride de forma independente das demais, surge a necessidade de sincronizar a troca de informações para garantir a coerência do resultado final da aplicação. Por exemplo, se uma determinada tarefa necessita de informações fornecidas por outro componente da aplicação, esta tarefa não poderá ser concluída até que receba todos os dados que necessita para sua execução.

2.2.1 Modelo de Aplicação Adotado

Uma aplicação paralela pode ser representada por um grafo acíclico direcionado (GAD) denotado por $G = (T, E)$, onde $T = \{t_1, \ldots, t_n\}$ é o conjunto de vértices que representam as tarefas que compõem a aplicação e $E = \{(t_i, t_j) \mid t_i, t_j \in T\}$, o conjunto de arestas do grafo que definem as relações de precedência entre as tarefas. Associado a cada tarefa $t_i \in T$, existe um peso de computação $\varepsilon(t_i) \in \mathbb{R}^*$ indicando a quantidade de computação necessária para executar esta tarefa. A cada aresta $(t_i, t_j) \in E$ existe um peso de comunicação $\omega(t_i, t_j) \in \mathbb{R}$ representando a quantidade de dados que deve ser enviada da tarefa para t_i para a tarefa t_j. Cada aresta (t_i, t_j) define uma relação de precedência que determina que a tarefa t_j não pode ser iniciada até que receba todos os dados enviados por t_i que, por sua vez, só inicia a transmissão imediatamente após o término de sua execução. Se $p(t_i)$ denota o processador alocado à tarefa t_i, o custo de transmissão de dados entre t_i e t_j através

do canal de comunicação que liga os processadores $p(t_i)$ e $p(t_j)$ é dado por:

$$c(t_i, t_j) = \begin{cases} \omega(t_i, t_j) \cdot \lambda(p(t_i), p(t_j)) & \text{, se } p(t_i) \neq p(t_j) \\ 0 & \text{, caso contrário} \end{cases} \tag{2.1}$$

As arestas incidentes sobre uma tarefa t_i, determinam seus conjuntos de predecessores e sucessores imediatos, ou seja, as tarefas que, respectivamente, enviam e recebem mensagens de t_i, definidos por:

$$PRED(t_i) = \{t_j \in T \mid (t_j, t_i) \in E, t_j \in T\} \tag{2.2}$$

$$SUCC(t_i) = \{t_j \in T \mid (t_i, t_j) \in E, t_j \in T\} \tag{2.3}$$

Caso o número de predecessores de uma tarefa t_i é igual a zero ($PRED(t_i) = \emptyset$), esta tarefa é considerada de *entrada*. Analogamente, se uma tarefa não possui sucessores ($SUCC(t_i) = \emptyset$) esta tarefa é considerada de *saída*.

Um GAD de uma aplicação paralela é ilustrado na Figura 2.2. Cada vértice do grafo denota uma tarefa da aplicação paralela. Os números presentes ao lado de cada vértice indicam o peso de execução associado a cada tarefa. Cada aresta representa uma restrição de precedência e indica que uma mensagem deve ser enviada entre as duas tarefas envolvidas. Associado a cada aresta existe um valor, ou peso, que indica a quantidade de dados da mensagem a ser enviada. No diagrama, t_1 e t_2 podem ser identificadas como tarefas de *entrada* e t_9 como uma tarefa de *saída*.

2.3 Escalonamento de Tarefas

Segundo a taxonomia proposta por Casavant e Kuhl [58], o escalonamento de tarefas pode ser classificado como *local* ou *global*, dependendo se as tarefas são atribuídas a um único processador ou a processadores distintos, respectivamente. O escalonamento global pode ser classificado como *dinâmico* ou *estático* de acordo com o momento em que as decisões relativas à atribuição de tarefas são tomadas. No escalonamento dinâmico, as tarefas são atribuídas aos processadores do sistema durante a execução da aplicação, onde parâmetros relacionados com a computação

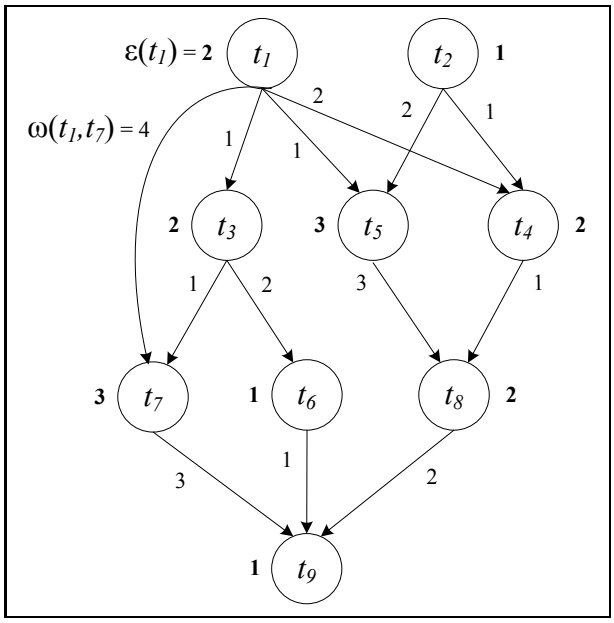


Figura 2.2: Um grafo acíclico direcionado representando uma aplicação paralela.

e troca de informações entre as tarefas, não necessariamente são previamente conhecidos. No escalonamento estático, características da estrutura da aplicação como topologia do grafo, custos de execução e comunicação das tarefas são conhecidos *a priori*, permitindo que, antes do início da execução da aplicação, seja possível determinar a seqüência e a distribuição das tarefas nos processadores do sistema de computação alvo.

2.3.1 Representação de Escalonamento

Um arranjo das tarefas de uma aplicação $G = (T, E)$ nos processadores de um sistema de computação paralelo $M = (P, L)$, ou simplesmente escalonamento, pode ser definido como um conjunto finito $S = \{(t_i, p_j, s(t_i, p_j)) \mid \forall t_i \in T, p_j \in P\}$ onde cada tupla $(t_i, p_j, s(t_i, p_i))$ determina que a tarefa $t_i \in T$ deve ser atribuída ao processador $p_j \in P$ com o tempo de ínicio de execução em p_j igual a $s(t_i, p_j)$. O tempo total de execução de uma aplicação paralela, ou *makespan*, de acordo com um escalonamento S é dado por:

$$makespan(S) = \max_{\forall (t_i, p_j, s(t_i, p_j)) \in S} \{s(t_i, p_j) + \varepsilon(t_i) \cdot h(p_j)\} \tag{2.4}$$

Uma vez que uma tarefa está alocada a um processador, o *tempo de início* de sua execução é determinado pelo instante em que todos os dados de seus predecessores imediatos são recebidos no respectivo processador. O *tempo de fim*, associado ao custo de computação e à velocidade de processamento, é o instante em que uma tarefa encerra sua execução e está pronta para enviar suas mensagens. O *tempo de início*, o *tempo de fim* e o *tempo de execução* de uma tarefa t_i em um processador p_j são definidos, respectivamente, por:

$$s(t_i, p_j) = \max\{disp(p_j), \max_{\forall t_k \in PRED(t_i)} \{f(t_k, p(t_k)) + c(t_k, t_i)\}\} \quad (2.5)$$

$$f(t_i, p_j) = s(t_i, p_j) + e(t_i, p_j) \quad (2.6)$$

$$e(t_i, p_j) = \varepsilon(t_i) \cdot h(p_j) \quad (2.7)$$

onde $disp(p_j)$ é o instante mais cedo em que o processador p_j está livre para executar uma tarefa.

A Figura 2.3 mostra uma representação gráfica de um escalonamento (c) obtido pela alocação das tarefas do GAD (a) nos processadores de um sistema (b). No escalonamento (c), os três grandes retângulos indicam os processadores do sistema distribuído de acordo com os rótulos p_1, p_2 e p_3 correspondentes. Em cada processador, os retângulos menores indicam as tarefas do GAD que são rotuladas por um número inteiro que indica o índice da tarefa correspondente. O comprimento vertical de cada retângulo indica o tempo de execução de cada tarefa. As setas representam as mensagens enviadas entre tarefas do grafo, onde a projeção vertical de cada seta indica a latência no envio da respectiva mensagem. As medidas do tempo de execução das tarefas e da latência no envio de mensagens, são feitas em unidades de tempo de acordo com a escala presente à direita do diagrama. No diagrama, as tarefas t_1 e t_3 apresentam, respectivamente, tempos de execução iguais a 2 e 4 unidades de tempo; a mensagem enviada entre estas duas tarefas tem latência igual a 2. Observe que entre alguns pares de tarefas que trocam mensagens e que estão alocadas em um mesmo processador (t_5 e t_8, por exemplo) a latência no envio é considera nula.

Em ambientes não homogêneos, devidos à heterogeneidade dos processadores e dos canais de comunicação, cada tarefa ou aresta do GAD pode ter seu valor

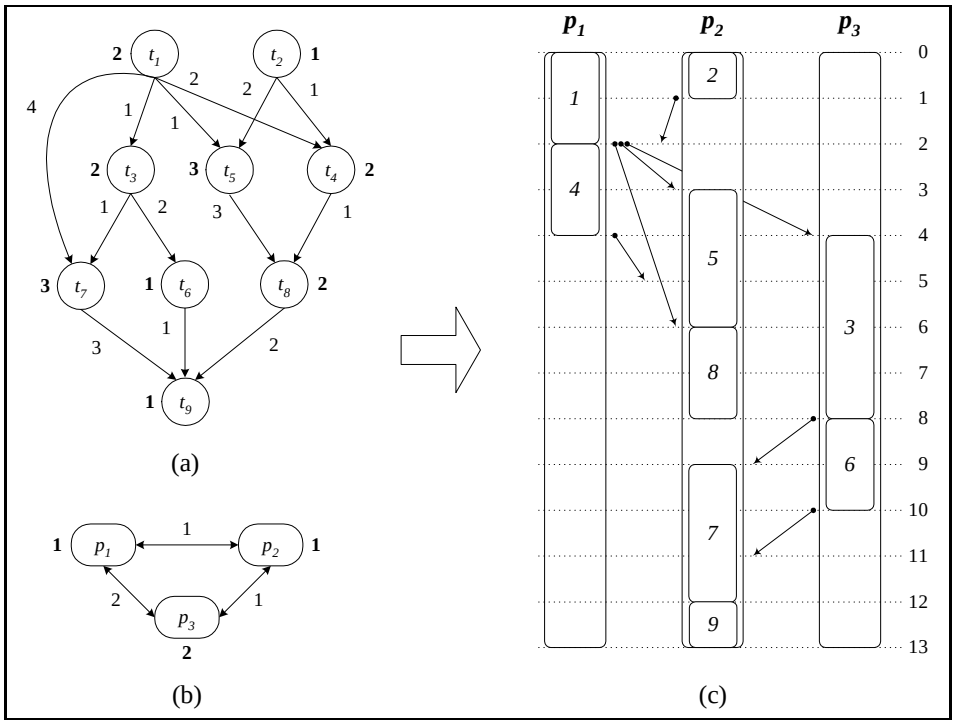


Figura 2.3: Um escalonamento de uma aplicação paralela.

de custo diferentemente avaliado dependendo da alocação das tarefas. Desta forma, antes do escalonamento, fica difícil avaliar previamente a influência do peso de uma tarefa ou comunicação baseados somente nas informações do grafo. Para uma análise mais acurada, são utilizados os custos médios de execução e comunicação associados às tarefas de um GAD em relação aos parâmetros de um sistema distribuído, que são estimados como:

$$\overline{\varepsilon}(t_i) = \frac{1}{m} \cdot \sum_{j=1}^{m} e(t_i, p_j) \tag{2.8}$$

$$\overline{\omega}(t_i, t_j) = \omega(t_i, t_j) \cdot \overline{\lambda} \tag{2.9}$$

$$\overline{\lambda} = \frac{1}{m \cdot (m-1)} \cdot \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda(p_i, p_j) \tag{2.10}$$

onde $\overline{\lambda}$ representa a latência média dos canais de comunicação do sistema, não considerando o retardo de transmissão de mensagens entre tarefas alocadas um mesmo processador que é igual a zero.

Os algoritmos de escalonamento procuram atribuir as tarefas de uma aplicação aos processadores de um sistema de computação, de forma que as relações de precedência sejam obedecidas e o tempo total de execução da aplicação seja o menor

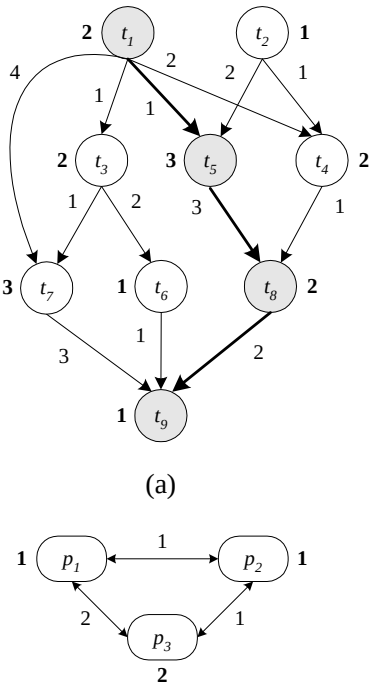
possível. Para atingir este objetivo, alguns algoritmos procuram assimilar atributos associados às tarefas da aplicação paralela e ao ambiente de computação. Alguns dos atributos comumente utilizados são o *nível*, *co-nível* e *caminho crítico* de uma tarefa [17, 21, 59]. O caminho crítico de uma tarefa t_i é o caminho de maior custo que passa por t_i, isto é, o conjunto ordenado de tarefas do grafo, iniciado por uma tarefa de entrada, onde o somatório dos custos médios de execução e de comunicação é máximo. O *nível* de uma tarefa t_i é dado pelo comprimento do caminho mais longo, considerando os custos médios de comunicação e execução, a partir de t_i até uma tarefa de saída do grafo. O *co-nível* é o comprimento do caminho mais longo a partir de uma tarefa de entrada do grafo até t_i, excluindo-se o custo médio de execução de t_i. As expressões para o cálculo do *nível*, *co-nível* de uma tarefa t_i são definidas como se segue:

$$nível(t_i) = \max_{\forall t_j \in SUCC(t_i)} \{nível(t_j) + \overline{\omega}(t_i, t_j)\} + \overline{\varepsilon}(t_i) \quad (2.11)$$

$$co\text{-}nível(t_i) = \max_{\forall t_j \in PRED(t_i)} \{co\text{-}nível(t_j) + \overline{\omega}(t_j, t_i) + \overline{\varepsilon}(t_j)\} \quad (2.12)$$

Uma característica que merece ser observada é que se as tarefas forem classificadas em ordem não crescente de *nível*, esta seqüência determina uma ordem topológica para as tarefas do grafo, isto é, uma seqüência linear de tarefas que preserva as restrições de precedência entre as tarefas do grafo. Outra característica interessante é que o comprimento do *caminho crítico* de uma tarefa t_i é dado pela soma dos seus valores de *nível* e *co-nível*. Estas propriedades podem ser observadas na Figura 2.4 que apresenta uma tabela (c) onde podem ser vistos os valores de *nível*, *co-nível* e comprimento do *caminho crítico* para cada tarefa de um GAD (a) em relação a um sistema distribuído (b). Em (a) pode ser visto em destaque um caminho crítico do GAD que é determinado pelas tarefas que apresentam os maiores comprimentos de caminho crítico no grafo. Observe que, neste exemplo, o GAD apresenta um outro caminho crítico iniciado a partir da tarefa t_2, já que esta tarefa apresenta valor $nível(t_2) + co\text{-}nível(t_2)$ máximo.

Um outro atributo importante relacionado com o grafo que representa uma aplicação paralela é o conceito de granularidade. Em termos simples, a granularidade de uma aplicação paralela pode ser definida como a razão entre a quantidade de co-



tarefa	$\overline{\varepsilon}$ (t_i)	*nível* (t_i)	*co-nível* (t_i)	*nível* (t_i) + *co-nível* (t_i)
t_1	2,67	18,67	0,00	18,67
t_2	1,33	18,67	0,00	18,67
t_3	2,67	13,33	4,00	17,33
t_4	2,67	10,67	5,33	16,00
t_5	4,00	14,67	4,00	18,67
t_6	1,33	4,00	9,33	13,33
t_7	4,00	9,33	8,00	17,33
t_8	2,67	6,67	12,00	18,67
t_9	1,33	1,33	17,34	18,67

(c)

Figura 2.4: Cálculo dos valores de *nível* e *co-nível* e comprimento do *caminho crítico*.

municação e a quantidade de computação existentes em um GAD [52]. Este conceito pode dar um boa noção sobre as características de uma aplicação relacionadas com os custos de comunicação e de computação, isto é, pode indicar se uma aplicação realiza mais computação que comunicação ou vice-versa. A análise da granularidade é importante porque pode determinar se a paralelização de uma aplicação paralela oferece alguma vantagem [43]. Contudo, a granularidade calculada somente a partir de dados da aplicação pode não ser adequada para ambiente heterogêneos, uma vez que os custos de comunicação e execução podem mudar radicalmente se comparados antes e após o escalonamento. De fato, a maioria das formulações para o cálculo da granularidade encontradas na literatura foram efetuadas tendo em vista ambientes de computação homogêneos [10, 52, 60, 61]. Desta forma, o cálculo da granularidade deve considerar aspectos tanto do GAD quanto do ambiente de computação alvo. Neste trabalho, a granularidade de um GAD em relação a um sistema distribuído é

obtido a partir da seguinte expressão:

$$gran(G, M) = \frac{\sum_{(t_i,t_j)\in E} \overline{\omega}(t_i, t_j)}{\sum_{t_k \in T} \overline{\varepsilon}(t_k)} \quad (2.13)$$

onde G é o GAD que representa uma aplicação paralela e M o sistema de computação alvo.

Assim, quando a quantidade de comunicação em um GAD domina a quantidade de computação, diz-se que o grafo possui granularidade *fina* e apresenta valor de $gran(\cdot) \geq 1$. Quando um GAD apresenta quantidade de computação maior que a quantidade de comunicação, o valor de $gran(\cdot) < 1$ e grafo é classificado como de granularidade *grossa*. A Figura 2.5 ilustra dois GADs G_1 e G_2 de granularidades fina (a) e grossa (b) calculadas em relação ao sistema M apresentado na Figura 2.1.

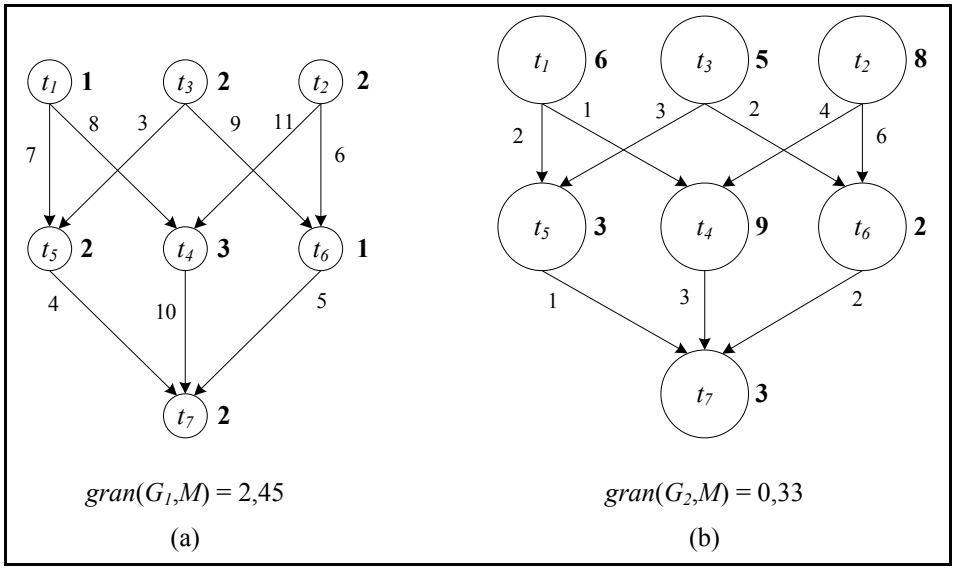


Figura 2.5: GADs com granularidade fina (a) e grossa (b).

2.4 Algoritmos de Escalonamento

A intratabilidade do problema de escalonamento de tarefas tem motivado a pesquisa em busca de soluções em duas grandes frentes [62]. Na primeira estão

as *heurísticas de construção* que, a partir de um modelo de aplicação e arquitetura, constroem um único escalonamento como solução para uma determinada instância de entrada. Noutra frente estão as *heurísticas ou algoritmos de busca*, onde vários escalonamentos são gerados durante a execução do algoritmo objetivando a produção de uma melhor solução final. A seguir são apresentadas as principais classes de algoritmos de escalonamento encontradas na literatura.

2.4.1 Heurísticas de Construção

Heurísticas de construção são algoritmos que constroem um único escalonamento adicionando, a cada passo, uma tarefa do GAD no ambiente de computação alvo. O uso de heurísticas de construção é indicado principalmente quando o tempo de construção de um escalonamento é um fator crítico, já que este tipo de algoritmo normalmente apresenta baixa complexidade. Nesta linha, podemos identificar algumas classes de heurísticas como do tipo *list scheduling* [21, 24, 63, 64, 65, 66], de análise de caminho crítico do grafo [52, 67, 68, 69] e de aglomeração de tarefas [47, 49, 60, 61, 69, 70, 71, 72].

List Scheduling

A maioria dos algoritmos de escalonamento encontrados na literatura são pertencentes à classe das heurísticas do tipo *list scheduling* [11]. Estas heurísticas, efetuam o escalonamento de tarefas em duas fases distintas: *priorização de tarefas* e *seleção de processador*. Na primeira fase, o algoritmo atribui prioridades às tarefas e as organiza em uma lista segundo algum critério específico. Na segunda fase, a tarefa pronta[1] de maior prioridade é atribuída ao processador que minimiza uma determinada função de custo. Esta última fase é repetida até que todas as tarefas do grafo sejam escalonadas. A principal diferença entre os algoritmos baseados nesta técnica é a estratégia utilizada para a priorização de tarefas [43], como por exemplo: BIL (*Best Imaginary Level*) [22], HLF (*Highest Level First*) [21, 26]; LP (*Longest Path*) [26]; LPT (*Longest Processing Time*) [73, 74] e CP (*Critical Path*) [65]. O

[1]Uma tarefa está pronta quando todos os seus predecessores imediatos já foram escalonados.

maior problema com as heurísticas desta classe reside justamente no procedimento de atribuição estática de prioridades, isto porque, no momento da atribuição, quando nenhuma tarefa está escalonada, uma tarefa pode não ter sua devida importância no escalonamento corretamente determinada pelo algoritmo [52].

Análise de Caminho Crítico

O caminho crítico de um GAD é o caminho com maior custo existente no grafo. A partir desta premissa, heurísticas baseadas nesta estratégia buscam minimizar o comprimento do caminho crítico procurando remover os custos de comunicação através do agrupamento de tarefas adjacentes em um mesmo processador. Segundo McCreary *et al* [75], o fato dos algoritmos baseados nesta técnica tomarem decisões gulosas a partir de informações locais sobre o caminho crítico, pode não conduzir a uma boa solução. Exemplos de algoritmos baseados em análise do caminho crítico podem ser encontrados em [68, 72, 76, 77].

Aglomeração de Tarefas

Os algoritmos baseados em aglomeração de tarefas trabalham em dois estágios distintos: aglomeração e escalonamento [78]. No primeiro estágio, são criados agrupamentos unitários contendo as tarefas do GAD, onde cada tarefa é considerada proprietária de seu respectivo grupo. Em seguida, inicia-se um processo de junção dos agrupamentos baseados no valor de uma função de custo, com o objetivo de minimizar o tempo de início de execução da tarefa proprietária. No segundo estágio, um escalonamento é construído através da atribuição aos processadores do sistema dos agrupamentos definidos na etapa anterior. Durante este processo, é analisada a possibilidade de se utilizar um número de processadores menor que o número de agrupamentos de forma a obter minimizar *makespan* do escalonamento gerado. Trabalhos relacionados com algoritmos de aglomeração de tarefas podem ser encontrados em [15, 61, 76, 79, 80, 81, 82].

Heurísticas Híbridas

Algumas heurísticas utilizam mais de uma estratégia durante o processo de escalonamento, não se enquadrando, portanto, nas classes aqui apresentadas. A aplicação de métodos de escalonamento diferentes em um mesmo algoritmo pode ser motivada por várias razões. Uma delas pode ser o comportamento negativo que algumas heurísticas apresentam quando submetidas a instâncias com características específicas. Isto que dizer que determinadas heurísticas normalmente não apresentam um desempenho constante quando submetidas a um conjunto variado de instâncias [10]. Um exemplo de heurística híbrida pode ser encontrado em [83], onde é proposto um algoritmo que une cinco diferentes abordagens. Outros exemplos são os de heurísticas que combinam técnicas de análise do caminho crítico com aglomeração [69, 72]; aglomeração com *list scheduling* [24, 68]; aglomeração com replicação [47] e estratégias de escalonamento estático e dinâmico [84].

2.4.2 Heurísticas de Construção e Busca

Heurísticas de construção e busca podem gerar diversas soluções ao longo de sua execução com o objetivo de, a cada iteração, melhorar a solução encontrada no passo anterior. Uma característica deste tipo de algoritmo é que, apesar de consumirem um maior tempo para a produção de uma solução final, os escalomentos gerados podem apresentar melhor qualidade que os produzidos por heurísticas de construção. Os algoritmos de escalonamento baseados em metaheurísticas agregam estas características. Metaheurísticas são métodos de busca de propósito geral destinados a encontrar uma boa solução, eventualmente a melhor, através da aplicação de heurísticas modeladas para um problema específico. Por serem de caráter geral, as metaheurísticas possuem mecanismos que permitem varrer o espaço de busca do problema escapando de soluções ótimas locais. São exemplos de metaheurísticas: Algoritmos Genéticos (AG) [85], GRASP [86], Busca Tabu (*Tabu Search*) [87, 88], *Simulated Anneling* [89], VNS [90] entre outras.

Diversas aplicações de metaheurísticas para o problema de escalonamento

de tarefas podem ser encontradas na literatura com destaque para Busca Tabu [91] e Algoritmos Genéticos [14, 16, 17, 18, 19, 55, 92, 93, 94]. Dentre estes, poucos trabalhos focados no uso de metaheurísticas para ambientes heterogêneos multiprocessados podem ser encontrados [17, 18, 19, 95]. Destacamos o trabalho de Kwok e Dhodhi [17] que apresentaram um algoritmo genético, combinando o uso de listas de prioridades com uma heurística do tipo *list scheduling*, que serviu de base para uma das metaheurísticas propostas neste trabalho.

2.4.3 Replicação de Tarefas

Apesar de não ser considerada uma classe de heurísticas para o problema de escalonamento, as técnicas de replicação de tarefas tem sido utilizadas com sucesso em diversos trabalhos da literatura afim [10, 47, 49, 51, 96, 97, 98, 99, 100, 101, 102]. Mesmo nos algoritmos de escalonamento mais eficientes, pode ocorrer a existência de espaços ociosos (onde o processador fica inativo) entre tarefas escalonadas em um mesmo processador. Isto ocorre, porque algumas tarefas precisam esperar a chegada de mensagens enviadas por tarefas predecessoras alocadas em outros processadores. O princípio da replicação consiste em alocar de forma redundante uma mesma tarefa em processadores diferentes visando reduzir os custos de comunicação e, assim, viabilizar a alocação prévia de tarefas sucessoras. A principal distinção entre os algoritmos que utilizam este recurso está nas estratégias de seleção das tarefas a serem replicadas [10]. Trabalhos relacionados com replicação de tarefas podem ser encontrados em [10, 47, 49, 51, 97, 98, 99, 100, 101]. Estudos sobre o efeito da replicação no escalonamento de tarefas podem ser vistos em [96, 102].

2.5 Resumo

Neste capítulo foram apresentados os modelos arquitetural e de aplicação, bem como o problema de escalonamento de tarefas. Propriedades relacionadas com a aplicação e a arquitetura dos sistemas de computação heterogêneos foram descritas e definidas. Também foram enumeradas as principais classes e características das

heurísticas de escalonamento de tarefas. Com base nestas informações, no próximo capítulo serão apresentadas algumas heurísticas relacionadas com os algoritmos propostos neste trabalho.

Capítulo 3

Trabalhos Relacionados

Neste trabalho, foram desenvolvidas duas metaheurísticas híbridas para o problema do escalonamento de tarefas em ambientes com processadores e canais de comunicação heterogêneos. A natureza híbrida dos algoritmos propostos, reside na incorporação e adaptação de algumas heurísticas encontradas na literatura de forma que as soluções geradas apresentem boa qualidade independentemente de características específicas das instâncias. Neste capítulo, são apresentadas heurísticas de escalonamento relacionadas com este trabalho, além de fundamentos sobre metaherísticas GRASP e Algoritmo Genético.

3.1 Metaheurísticas GRASP e Algoritmos Genéticos

Esta seção apresenta os fundamentos sobre metaheurísticas GRASP e Algoritmos Genéticos. Este conceitos são importantes para uma melhor compreensão, tanto dos algoritmos propostos, quanto das abordagens extraídas da literatura apresentadas neste capítulo.

3.1.1 GRASP

Proposto originalmente por Resende e Feo [86], um algoritmo GRASP (*Greedy Randomized Adaptive Search Procedures*) é um procedimento multipartida para obtenção de soluções aproximadas, eventualmente ótimas, para um problema de otimização. Cada iteração do algoritmo consiste de duas etapas básicas: uma *fase de construção*, onde uma solução é produzida; e uma *fase de busca local*, onde a vizinhança da solução construída na etapa anterior é investigada na procura de uma melhor solução. A cada iteração GRASP, o algoritmo seleciona a solução que minimiza uma função de avaliação $f(\cdot)$. Após todas as iterações, a melhor solução encontrada pelo algoritmo é retornada. O Algoritmo 3.1 ilustra o procedimento executado por um GRASP.

```
1  algoritmo GRASP(f(·), g(·), α, max_iters)
2      s* = ∞;
3      para i = 1 até max_iters faça
4          s = construção(g(·), α);
5          s = busca_local(f(·), s);
6          se f(s) < f(s*) então
7              s* = s;
8          fim
9      fim
10     retorna s*;
11 fim
```

Algoritmo 3.1: Algoritmo GRASP genérico.

Na fase de construção, cujo pseudocódigo é mostrado no Algoritmo 3.2, uma solução válida para o problema é construída iterativamente elemento a elemento (linhas de 4 a 11). Durante o processo de seleção de um elemento, uma lista de candidatos C é inicializada com os elementos que podem ser adicionados à solução que está sendo construída (linha 3). Uma lista restrita de candidatos (LRC) é então gerada a partir dos elementos contidos em C, de acordo com o valor de uma função gulosa $g(\cdot)$ que avalia o benefício de seleção de cada elemento (linha 7). A cada iteração, o benefício associado à função $g(\cdot)$ é atualizado (linha 10), determinando

o caráter adaptativo do GRASP. A criação da LRC também é influenciada por um outro parâmetro $\alpha \in [0, 1]$ que controla o grau de aleatoriedade da construção (linha 7). Por exemplo, se $\alpha = 0$, o algoritmo seleciona a cada passo o elemento mais benéfico naquele instante, tornando gulosa a construção. Por outro lado, se $\alpha = 1$, a seleção do elemento candidato é aleatória. Após a seleção (linha 8), o elemento é adicionado à solução (linha 9) e a lista de candidatos C é atualizada (linha 10). O processo é então repetido até que não existam mais candidatos para compor a solução.

```
1  procedimento construção(g(·), α)
2      s* = {};
3      Inicializa a lista de candidatos C;
4      enquanto C ≠ ∅ faça
5          g_min = min {g(c) | c ∈ C};
6          g_max = max {g(c) | c ∈ C};
7          LRC = {c ∈ C | g(c) ≤ g_min + α(g_max − g_min)};
8          Selecione c ∈ LRC aleatoriamente;
9          s = s ∪ {c};
10         Atualize a lista de candidatos C;
11     fim
12     retorna s;
13 fim
```

Algoritmo 3.2: GRASP: Procedimento de construção de uma solução.

O caráter aleatório da construção GRASP permite que soluções diferentes possam ser geradas a cada execução do algoritmo de construção. Contudo, este mecanismo não garante que a solução construída represente um ótimo local. Desta forma, é conveniente se aplicar uma busca local em torno da solução inicial com o objetivo de tentar melhorar o resultado obtido a cada construção. A busca local realiza uma varredura numa vizinhança da solução obtida na fase de construção na tentativa de encontrar outra solução de melhor qualidade. Para cada problema específico, uma estrutura $V(s)$ pode ser definida para determinar a vizinhança de uma solução s. Segundo Resende [86], a chave para a eficiência de um procedimento de busca reside na definição de uma estrutura de vizinhança adequada, técnicas eficientes de busca nesta vizinhança e da qualidade da solução inicial.

O Algoritmo 3.3 apresenta o pseudocódigo para a busca local efetuada por um GRASP. Inicialmente, uma lista de soluções vizinhas a uma solução s é determinada (linha 3). Em seguida, todas as soluções pertencentes à vizinhança de s são analizadas (linhas de 4 a 10) e a solução que minimiza uma função de avaliação $f(\cdot)$ é selecionada (linhas de 6 a 8). Finalmente, o procedimento de busca retorna a melhor solução encontrada (linha 11).

```
1  procedimento busca_local(s, f(·))
2      s* = s;
3      V = {v ∈ V(s) | f(v) < f(s)};
4      enquanto V ≠ ∅ faça
5          Selecione v ∈ V;
6          se f(v) < f(s*) então
7              s* = v;
8          fim
9          V = V − {v};
10     fim
11     retorna s*
12 fim
```

Algoritmo 3.3: GRASP: Procedimento de busca local.

Uma análise na estrutura de um algoritmo GRASP nas suas versões iniciais, ela não é adaptativa, ou seja, a solução construída em uma iteração não leva em consideração a informação da iteração anterior. Na verdade, a única informação histórica armazenada é a melhor solução encontrada até o momento. Contudo, apesar desta característica na sua estrutura, o GRASP é geralmente um algoritmo que pode convergir rapidamente para boas soluções [103].

3.1.2 Algoritmos Genéticos

Introduzidos originalmente por John Holland [85], os Algoritmos Genéticos (AGs) são procedimentos iterativos de busca heurística onde a cada iteração uma população de indivíduos (soluções) é produzida. O espaço de busca, que compreende as possíveis soluções viáveis para uma instância do problema, é analisado pelo

algoritmo através de operações que combinam informações dos indivíduos. O ponto de partida para a aplicação de Algoritmos Genéticos é a definição de uma representação para as soluções do problema estudado. A maioria das representações são baseadas no uso de um conjunto finito de elementos utilizados para compor uma solução. Inicialmente, uma população de indivíduos é criada através de um procedimento aleatório ou heurístico. A cada iteração do algoritmo, também chamada de geração, os indivíduos da população corrente são decodificados e avaliados segundo uma função que determina o grau de aptidão de cada indivíduo. Neste esquema, indivíduos que representam soluções de boa qualidade apresentam maior aptidão. A transmissão de características dos indivíduos da população corrente para a geração seguinte é realizada através de um processo de reprodução que implica na aplicação de operadores de seleção, cruzamento e mutação. Para compor uma nova população, os indivíduos são selecionados segundo critérios relacionados com sua aptidão, de forma que indivíduos mais adaptados tenham maior possibilidade de contribuir com o desenvolvimento da geração seguinte. A operação de cruzamento, aplicada com determinada probabilidade, ocorre com a seleção de dois ou mais indivíduos (pais) da população corrente, que trocam partes de seu material genético, gerando novos indivíduos. Como a seleção tende a privilegiar os indivíduos com maior aptidão, existe uma maior probabilidade destes transmitirem suas características para a próxima geração, criando o mecanismo responsável pelo incremento sucessivo na qualidade das soluções. No entanto, este mesmo processo pode levar a uma convergência prematura para regiões do espaço de busca que apresentam boas soluções locais (ótimos locais), mas ainda distantes de um ótimo global. Neste contexto, o operador de mutação é aplicado com o objetivo de gerar indivíduos que representem soluções em regiões ainda pouco exploradas, criando assim, um mecanismo que garante a manutenção da diversidade na população. Este processo reprodutivo é repetido até que algum critério de parada seja atingido, como por exemplo, a execução de um dado número de gerações, ou a obtenção de um indivíduo com um determinado nível de aptidão ou, ainda, a chegada a um ponto específico no espaço de busca do problema.

O Algoritmo 3.4 ilustra o pseudocódigo de um Algoritmo Genético tradicio-

nal. Como dados de entrada o algoritmo recebe o número de gerações (max_gen), o tamanho da população (max_pop) e as probabilidades de cruzamento ($prob\_cruz$) e mutação ($prob\_mut$). Inicialmente, uma população inicial pop_0 é gerada (linha 2) e a aptidão de cada indivíduo avaliada (linha 3). A cada geração i, uma nova população pop_i é gerada através da reprodução dos indivíduos da população anterior (linha 5). Após a geração da nova população, cada indíviduo é avaliado (linha 6) antes da próxima iteração. Ao final, o algoritmo retorna a melhor solução encontrada ao longo das iterações (linha 8).

```
1 algoritmo AG(max_gen, max_pop, prob_cruz, prob_mut)
2 |  pop_0 = pop_inicial(max_pop);
3 |  Avalie pop_0;
4 |  para i = 1 até max_gen faça
5 |  |  pop_i = reprodução(pop_{i-1}, prob_cruz, prob_mut);
6 |  |  Avalie pop_i;
7 |  fim
8 |  retorna Melhor solução encontrada;
9 fim
```

Algoritmo 3.4: Pseudocódigo de um Algoritmo Genético tradicional.

Apesar de estocásticos, os AGs não podem ser classificados como procedimentos de busca totalmente aleatórios. Isto porque, o processo de reprodução nos Algoritmos Genéticos, possibilita que características positivas dos indivíduos pais sejam mantidas nas novas gerações, aumentando com isso as possibilidades de convergência para soluções ótimas ou sub-ótimas.

3.2 Heurísticas de Escalonamento

Nesta seção são apresentados algoritmos para o problema de escalonamento de tarefas selecionados da literatura especializada e que estão relacionados com este trabalho. Tratam-se de três heurísticas de construção, um procedimento de busca local e um Algoritmo Genético, cujos detalhes de funcionamento são apresentados a seguir.

3.2.1 EFT - *Earliest Finish Time*

O *Earliest Finish Time* (EFT) [24] é um algoritmo que utiliza uma metodologia simples para o escalonamento estático de tarefas em sistemas heterogêneos. Pertencente à classe das heurísticas do tipo *list scheduling*, o EFT procura atribuir uma tarefa candidata a escalonamento ao processador onde sua execução termina mais cedo. Inicialmente, o algoritmo atribui às tarefas do GAD prioridades determinadas segundo algum critério específico. Uma lista L de tarefas prontas é então inicializada a partir das tarefas do grafo que não possuem predecessores imediatos. Em seguida, é selecionada a tarefa livre $\hat{t} \in L$ que maximiza uma função $g(\hat{t})$, utilizada para determinar a prioridade associada a $\hat{t}$. A tarefa $\hat{t}$ é então escalonada no processador $\hat{p}$ onde sua execução termina mais cedo, considerando os custos de comunicação com tarefas predecessoras e a disponibilidade do processador. No próximo passo, a lista de tarefas prontas L é atualizada através da remoção de $\hat{t}$ e da inclusão de todos seus sucessores cujos predecessores já foram escalonados, ou seja, as tarefas que ficaram livres após o escalonamento de $\hat{t}$. O processo é então repetido enquanto houver tarefas livres em L.

O Algoritmo 3.5 apresenta o pseudocódigo de EFT. São fornecidos como entrada um sistema distribuído M, um grafo G e uma função de custo $g(\cdot)$ para avaliação de prioridades. Inicialmente, a lista de tarefa prontas L é inicializada (linha 4). No laço principal (linhas 5 a 13), o algoritmo seleciona de L a tarefa que apresenta maior prioridade (linha 6) e determina em qual processador seu *tempo de fim* é minimizado (linha 7) para, em seguida, efetuar seu escalonamento (linha 9). Antes do início da próxima iteração, o conjunto de tarefas prontas é atualizado (linhas 9 a 12), onde a função $nesc(t')$ (linha 11) determina o número de predecessores imediatos da tarefa t' que ainda não foram escalonados. Após o processamento de todas as tarefas do GAD, o algoritmo retorna o escalonamento obtido (linha 14).

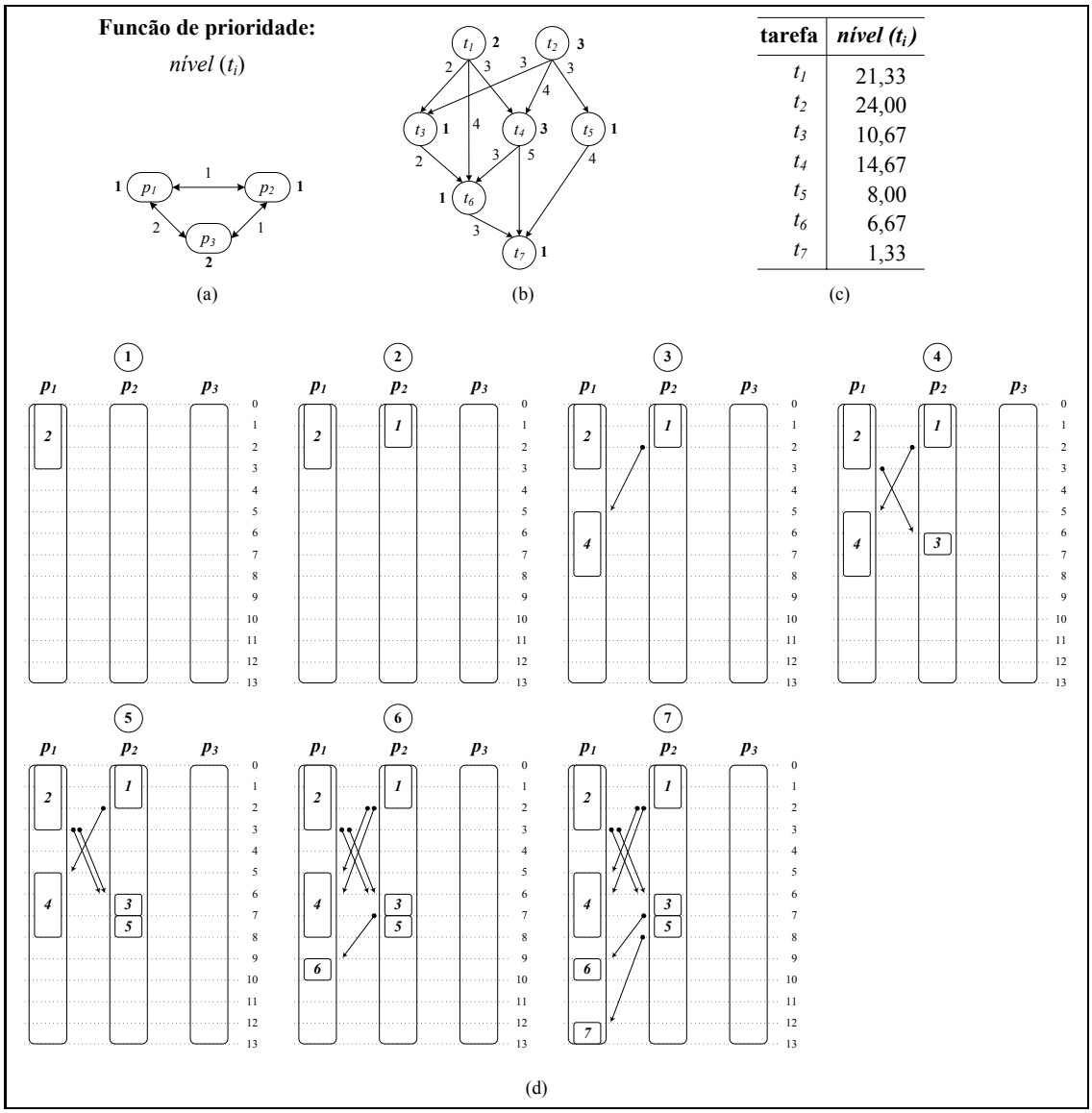
O desempenho de EFT depende fortemente da função de classificação de prioridades $g(\cdot)$ escolhida, uma vez que a seqüência de atribuição das tarefas influencia diretamente na qualidade do escalonamento. A Figura 3.1 apresenta um exemplo do processo de escalonamento efetuado por EFT utilizando o *nível* como função

```
1  algoritmo EFT(M, G, g(·))
2      Atribua prioridades às tarefas de T ∈ G;
3      S = {};
4      L = {t' ∈ T | PRED(t') = ∅};
5      enquanto |L| > 0 faça
6          t̂ = {t' ∈ L | g(t') = max{g(t'') | t'' ∈ L}};
7          p̂ = {p' ∈ P | f(t̂, p') = min{f(t̂, p'') | p'' ∈ P}};
8          S = S ∪ {(t̂, p̂, s(t̂, p̂))};
9          L = L − {t̂};
10         para cada t' ∈ SUCC(t̂) faça
11             se nesc(t') = 0 então L = L ∪ {t'};
12         fim
13     fim
14     retorna S;
15 end
```

Algoritmo 3.5: EFT - Earliest Finishing Time

de prioridade. Os diagramas (a) e (b) representam, respectivamente, o sistema de computação alvo e o GAD de uma aplicação paralela. A tabela (c) indica os valores de prioridade utilizados para determinar a ordem de escalonamento das tarefas do GAD. Cada diagrama em (d) mostra uma etapa do processo de escalonamento realizada pelo algoritmo.

A complexidade de EFT é determinada pelo laço principal iniciado na linha 5 do Algoritmo 3.5. Para um GAD com n tarefas, a busca pela tarefa que maximiza a função de custo $g(\cdot)$ (linha 6), pode ser efetuada em tempo $O(n)$. Na linha 7, a busca pelo valor mínimo do *tempo de fim* de uma tarefa t_i, relacionado com número de processadores (m) e de predecessores de t_i ($|PRED(t_i)|$), consome tempo igual $O(|PRED(t_i)| \times m)$ e determina a complexidade do código interno ao laço principal. Considerando todas as iterações do laço e que o número de arestas do GAD (e) domina o número tarefas, a complexidade de EFT pode ser determinada como igual a $O(e \times m)$. Para grafos densos, onde o número de arestas é próximo de n^2, a complexidade pode ser considerada na ordem de $O(n^2 \times m)$.



tarefa	*nível* (t_i)
t_1	21,33
t_2	24,00
t_3	10,67
t_4	14,67
t_5	8,00
t_6	6,67
t_7	1,33

Figura 3.1: O processo de escalonamento efetuado pelo algoritmo EFT.

3.2.2 HEFT - *Heterogeneous Earliest Finish Time*

O *Heterogeneous Earliest Finish Time* (HEFT) [21] também faz parte da classe *list scheduling* para escalonamento de tarefas em sistemas heterogêneos. O diferencial básico de HEFT em relação a EFT reside na política de inserção adotada pelo algoritmo, que busca alocar tarefas candidatas em espaços ociosos existentes entre tarefas já escalonadas em um mesmo processador. Como toda heurística *list scheduling*, o algoritmo associa a cada tarefa do GAD a ser escalonado um valor de prioridade. A prioridade utilizada por HEFT é o *nível* associado a cada tarefa do

GAD. O algoritmo inicia calculando e atribuindo o valor de *nível* para cada tarefa do grafo. Em seguida, uma lista de tarefas livres L é construída a partir das tarefas do GAD que não possuem predecessores imediatos. A tarefa livre $\hat{t} \in L$ com maior valor de *nível* é então selecionada para escalonamento. No caso de mais de uma tarefa apresentar o mesmo valor de *nível*, o desempate é feito aleatoriamente. A tarefa selecionada é então atribuída ao processador $\hat{p}$ que termina a execução de $\hat{t}$ mais cedo, levando em consideração os espaços ociosos existentes entre tarefas já escalonadas. O algoritmo procura inserir a tarefa candidata dentro destes espaços, considerando as restrições de início de execução impostas pelas mensagens enviadas por tarefas predecessoras, além de verificar se o comprimento do espaço (em unidades de tempo) é suficiente para comportar o tempo de execução da tarefa. No passo seguinte, a lista L é atualizada pela remoção de $\hat{t}$ e adição das tarefas que ficaram livres após o escalonamento de $\hat{t}$. O processo é então repetido até que todas as tarefas do GAD sejam escalonadas.

O Algoritmo 3.6 apresenta o pseudocódigo de HEFT. São fornecidos como entrada para o algoritmo um sistema distribuído M e um grafo G, sendo produzido como resultado um escalonamento S. Inicialmente, a lista de tarefas prontas L é inicializada (linha 4). No laço principal (linha 5 a 13), a tarefa com maior *nível* presente em L é selecionada (linha 6) e seu *tempo de fim* é determinado em cada processador do sistema (linha 7), considerando os espaços ociosos existentes entre tarefas alocadas em um mesmo processador. A tarefa selecionada é então escalonada no processador determinado pela operação anterior (linha 8). Em seguida, a lista de tarefas prontas é atualizada (linhas 9 a 12) preparando o algoritmo para a próxima iteração. Após escalonar todas as tarefas do GAD, o algoritmo retorna o escalonamento produzido (linha 14).

A função $f\_ins(S, \hat{t}, p_j)$, presente nas linhas 7 e 8 do Algoritmo 3.6, computa o *tempo de fim* de uma tarefa t_i considerando a alocação de t_i nos espaços ociosos existentes entre tarefas alocadas ao processador p_j de acordo com um escalonamento S. Para isto, a função obtém inicialmente o conjunto das tarefas alocadas no processador p_j. Seja $S_j =< \overline{t}_1, \overline{t}_2, \ldots, \overline{t}_e >$ o conjunto ordenado, com cardinalidade e, das tarefas escalonadas no processador p_j, tal que $s(\overline{t}_i, p_j) < s(\overline{t}_{i+1}, p_j)$,

```
1  algoritmo HEFT(M, G)
2      Calcule o valor de nível para cada tarefa de G;
3      S = {};
4      L = {t_i ∈ T | PRED(t_i) = ∅};
5      enquanto |L| > 0 faça
6          t̂ = {t_i ∈ L | nível(t_i) = max{nível(t_k) | t_k ∈ L}};
7          p̂ = {p_i ∈ P | f_ins(S, t̂, p_i) = min{f_ins(S, t̂, p_k) | p_k ∈ P}};
8          S = S ∪ {(t̂, p̂, f_ins(S, t̂, p̂) − e(t̂, p̂))};
9          L = L − {t̂};
10         para cada t' ∈ SUCC(t̂) faça
11             se nesc(t') = 0 então  L = L ∪ {t'};
12         fim
13     fim
14     retorna S;
15 fim
```

Algoritmo 3.6: HEFT - Heterogeneous Earliest Finish Time

para $1 \leq i < e$. Em busca de espaços ociosos, o algoritmo visita as tarefas de S_j procurando aquelas que iniciam após o instante de chegada em p_j da última mensagem enviada para a tarefa candidata a escalonamento $\hat{t}$, instante este denotado por $\overline{s}(\hat{t}, p_j)$. Para cada tarefa de $\overline{t}_i \in S_j$ que satisfaz esta condição, é calculado o comprimento do espaço ocioso entre esta tarefa $\overline{t}_i$ e sua predecessora $\overline{t}_{i-1}$, considerando o instante de chegada das mensagens destinadas a $\hat{t}$. Se o custo de execução da tarefa $\hat{t}$ no processador p_j for compatível com o comprimento do espaço ocioso calculado, a tarefa é então alocada nesta posição. Caso contrário, a função verifica o próximo espaço ocioso existente repetindo o processo até que uma posição de inserção seja encontrada. Caso nenhum espaço ocioso seja encontrado ou não comporte a execução de $\hat{t}$, o *tempo de fim* da tarefa é calculado considerando sua alocação após a última tarefa atribuída ao processador p_j.

A Figura 3.2 ilustra o processo de análise de um espaço ocioso em um processador p_j. No exemplo, pode ser observado em (a) um espaço ocioso entre duas tarefas adjacentes ($\overline{t}_{i-1}$ e $\overline{t}_i$). As setas indicam as mensagens destinadas à uma tarefa $\hat{t}$ que chegam ao processador p_j. Note que antes da inserção, o cálculo de comprimento do espaço ocioso (*comp*) leva em consideração o instante de chegada

da última mensagem destinada a $\hat{t}$ $(\overline{s}(\hat{t}, p_j))$ e não todo o espaço ocioso existente entre as duas tarefas adjacentes. Em (b), após a inserção, pode ser verificada a alocação da tarefa $\hat{t}$ no processador p_j, respeitando as restrições impostas pelo tempo de chegada das mensagens enviadas pela tarefas predecessoras de $\hat{t}$.

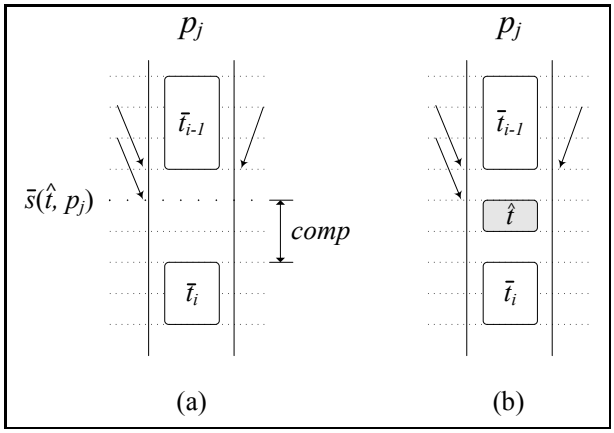


Figura 3.2: Análise de um espaço ocioso efetuado por HEFT.

O pseudocódigo de $f\_ins(\cdot)$ é apresentado pelo Algoritmo 3.7. A função recebe o escalonamento parcial S, a tarefa candidata a escalonamento $\hat{t}$ e o processador alvo p_j e retorna o *tempo de fim* de $\hat{t}$ em p_j, considerando a possível inserção da tarefa em algum espaço ocioso entre as tarefas alocadas neste processador. Inicialmente, o algoritmo percorre as tarefas atribuídas ao processador p_j em ordem crescente de seus tempos de início (linhas 2 a 13). Para cada tarefa $\overline{t}_i$ que inicia após o tempo de chegada da última mensagem destinada a $\hat{t}$ (linha 3) o algoritmo computa o comprimento do espaço ocioso entre duas tarefas adjacentes. É considerado o caso em que existe um espaço ocioso antes da primeira tarefa alocada ao processador (linha 5). Para as demais tarefas, o espaço é calculado levando em conta o *tempo de fim* da tarefa escalonada imediatamente antes de $\overline{t}_i$ (linha 7). Se o custo de execução de $\hat{t}$ em p_j é comportado pelo espaço ocioso (linha 9), o *tempo de fim* é computado de forma a inserir a tarefa nesta posição. Caso $\hat{t}$ não possa ser inserida em nenhum espaço ocioso, a função retorna o *tempo de fim* da tarefa considerando a disponibilidade do processador (linha 14).

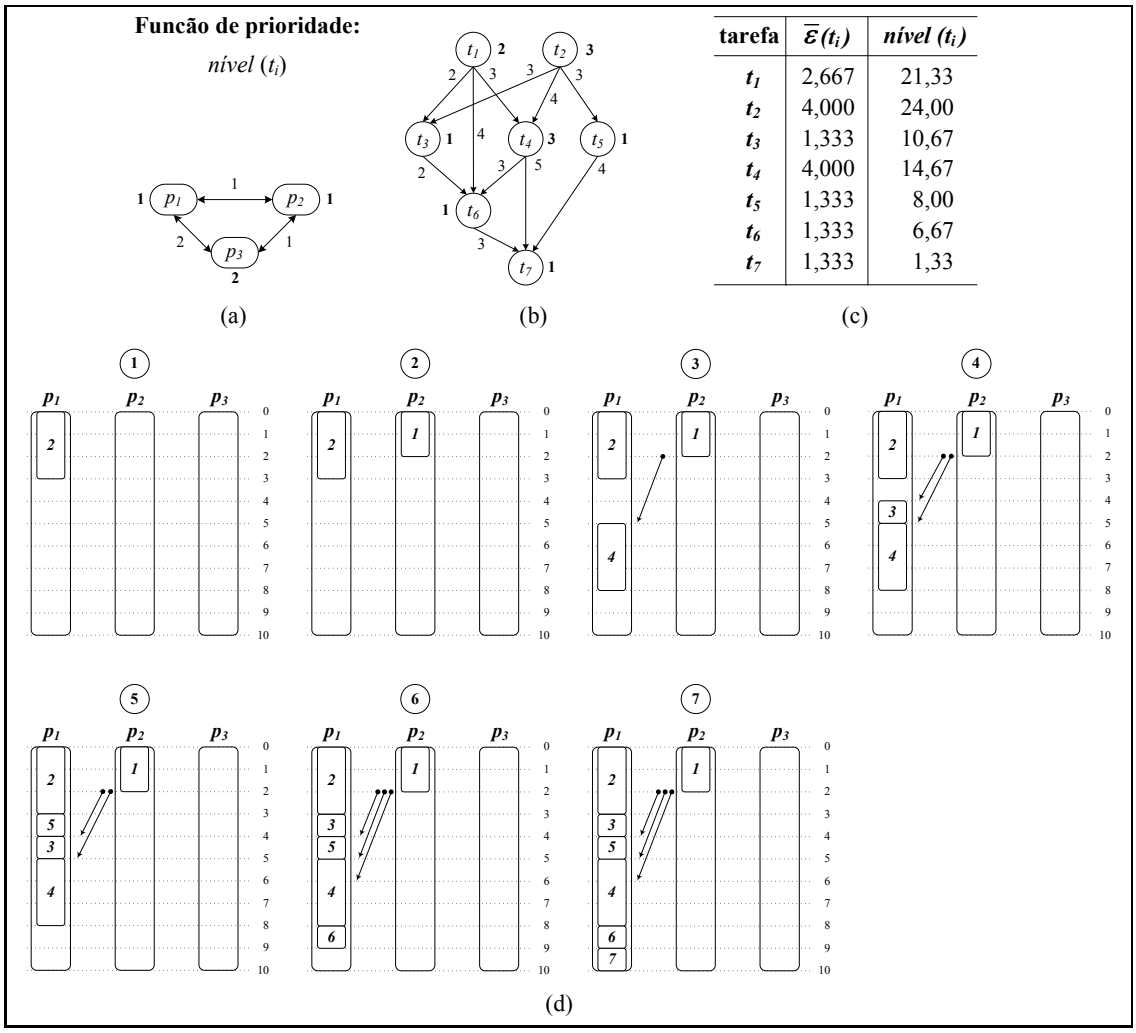
A Figura 3.3 ilustra o processo de escalonamento efetuado por HEFT. No exemplo, a instância de entrada é composta por um sistema distribuído heterogêneo

```
1  função f_ins(S, t̂, p_j)
2      para i = 1 até |S_j| faça
3          se s(t̄_i, p_j) > s̄(t̂, p_j) então
4              se i = 1 então
5                  comp = s(t̄_i, p_j) − s̄(t̂, p_j);
6              senão
7                  comp = s(t̄_i, p_j) − max{s̄(t̂, p_j), f(t̄_{i−1}, p_j)};
8              fim
9              se e(t̂, p_j) ≤ comp então
10                 retorna s(t̄_i, p_j) − comp + e(t̂, p_j);
11             fim
12         fim
13     fim
14     retorna s(t̂, p_j);
15 fim
```

Algoritmo 3.7: A função de inserção de tarefas utilizada por HEFT.

com três processadores (a) e uma aplicação paralela (b). Os custos médios de execução e os valores de prioridade podem ser vistos na tabela (c). Em (d) são mostrados os estados do escalonamento após a atribuição de cada tarefa do GAD. Nos estados 4 e 5, pode-se ver o momento da inserção das tarefas t_3 e t_5, respectivamente.

O cálculo da complexidade de HEFT é similiar ao da heurística anterior (EFT), onde a diferença básica está relacionada à função de inserção $f\_ins(\cdot)$ utilizada (linhas 7 e 8 do Algoritmo 3.7). O cálculo do *tempo de fim* de uma tarefa t_i, assim como em EFT, consome tempo $O(|PRED(t_i)| \times m)$, mas é acrescido pelo tempo de busca por uma posição de inserção que é igual a $O(n-1)$ no pior caso. Assim, o cálculo do *tempo de fim* consome tempo de computação da ordem de $O(n + |PRED(t_i)| \times m)$. Como o número de arestas e domina o número de tarefas n de um grafo, a complexidade de HEFT fica na ordem de $O(n + e \times m)$. Para grafos densos, onde o número de arestas é da ordem de n^2, o algoritmo apresenta complexidade proporcional a $O(n^2 \times m)$.



tarefa	$\overline{\varepsilon}(t_i)$	nível (t_i)
t_1	2,667	21,33
t_2	4,000	24,00
t_3	1,333	10,67
t_4	4,000	14,67
t_5	1,333	8,00
t_6	1,333	6,67
t_7	1,333	1,33

Figura 3.3: O processo de construção de um escalonamento pela heurística HEFT.

3.2.3 ETF - *Earliest Task First*

Introduzido por Hwang *et al* [24], o *Earliest Task First* (ETF) é considerado um dos algoritmos mais eficientes para escalonamento de GADs arbitrários em ambientes com processadores homogêneos. Neste algoritmo, as decisões relativas ao escalonamento de uma tarefa são baseadas no atraso e no volume de dados associados ao envio de mensagens através de uma estratégia simples: a primeira tarefa livre é escalonada primeiro. Diferentemente de outras heurísticas do tipo *list scheduling*, o ETF é orientado por evento, isto é, cada passo do algoritmo é guiado por um relógio que orienta as análises e decisões de escalonamento. Para compreender melhor esta característica, é preciso definir a variável CM (*Current Moment*) que representa o *instante corrente* do relógio do sistema em que uma decisão de escalonamento é

tomada. Além disso, são definidos os conjuntos L e I indicando, respectivamente, as tarefas e processadores ociosos no instante CM. Inicialmente, o conjunto I de processadores ociosos é inicializado para conter todos os processadores disponíveis no sistema e o conjunto de tarefas livres L recebe todas as tarefas sem predecessores do grafo de entrada. O algoritmo inicia o processo de escalonamento analisando o tempo de início de cada tarefa livre pertencente a L em todos os processadores em I, selecionado o par tarefa-processador, $\hat{t}$ e $\hat{p}$, respectivamente, que apresenta o tempo de início de execução mais cedo. No caso de mais de um par tarefa-processador candidato, o desempate é feito através da análise de algum atributo das tarefas envolvidas, como o *nível*, por exemplo. Após a seleção de $\hat{t}$ e $\hat{p}$, o algoritmo verifica se o tempo de início mais cedo entre todas as tarefas e processadores livres, indicado por $\hat{s}$, é maior que o instante de conclusão de alguma tarefa correntemente em execução. Este instante é chamado de *próximo instante de decisão*, denotado por NM (*Next Moment*), que indica o instante mais cedo após CM em que uma tarefa em execução tem seu processamento finalizado. Se houver alguma tarefa nesta condição, o algoritmo adia o escalonamento de $\hat{t}$ em $\hat{p}$ para poder considerar novos processadores que ficam ociosos no instante NM e evitar a formação de espaços ociosos entre tarefas escalonadas neste processador. Caso contrário, se $\hat{s} \leq NM$, a tarefa $\hat{t}$ é escalonada em $\hat{p}$, e os conjuntos de tarefas e processadores livres são atualizados através da remoção de $\hat{t}$ e $\hat{p}$, respectivamente. Após o escalonamento de todas as tarefas livres nos processadores disponíveis em um determinado instante de decisão, a variável CM é atualizada para conter o momento em que a primeira tarefa correntemente em execução terminar, ou seja, o próximo instante de decisão (NM). Tomando o novo valor de CM como base, os processadores que ficarem ociosos neste instante são adicionados ao conjunto I. As tarefas cujos predecessores foram escalonados no último *instante corrente de decisão* são incluídas em L e o processo é repetido até que todas as tarefas do grafo sejam escalonadas.

Conforme mencionado, o ETF foi originalmente projetado para sistemas com processadores homogêneos, motivo pelo qual o *tempo de início* de execução de uma tarefa é utilizado como função de prioridade. Como neste trabalho adotamos o uso de ambientes de computação com processadores e canais de comunicação

heterogêneos, o algoritmo ETF foi adaptado para trabalhar com este tipo de sistema. Basicamente, a adaptação efetuada consistiu em utilizar o *tempo de fim* de execução de uma tarefa como função de prioridade. A partir deste ponto, as referências à heurística ETF estarão relacionadas à versão adaptada para sistemas heterogêneos.

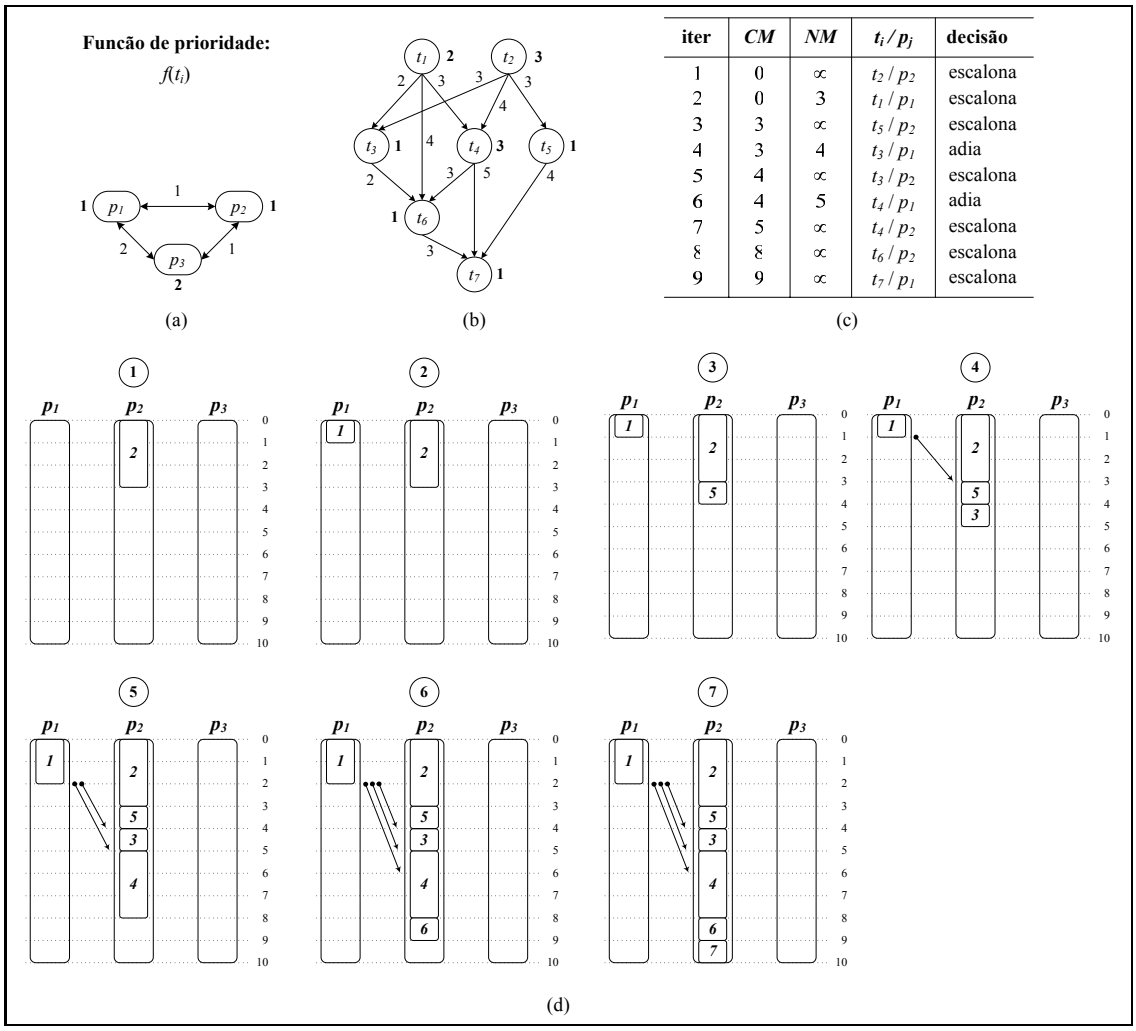
O pseudocódigo de ETF adaptado é apresentado no Algoritmo 3.8. Inicialmente, o algoritmo constrói a lista de tarefas livres a partir das tarefas do GAD que não possuem predecessores (linha 3). O laço externo (linhas 4 a 22) é repetido enquanto houver tarefas ainda não escalonadas. A cada iteração do laço interno (5 a 15), o algoritmo seleciona o par tarefa-processador $\hat{t}$ e $\hat{p}$ que apresenta menor tempo de fim (linha 6), considerando a disponibilidade do processador $\hat{p}$ (linha 7). Em seguida, é verificado se $\hat{t}$ pode iniciar antes do término mais cedo de uma tarefa correntemente em execução (linha 8). Em caso afirmativo, a tarefa $\hat{t}$ é escalonada no processador $\hat{p}$ e as listas de tarefas livres, de tarefas processadas e de processadores ociosos são atualizadas (linhas 9 e 10). Também é verificado se o instante de término de $\hat{t}$ é anterior ao tempo registrado em NM, que, em caso afirmativo, é atualizado (linha 11). Caso $\hat{t}$ inicie sua execução em $\hat{p}$ após o instante registrado em NM (linha 8), o escalonamento é adiado através do abandono do laço interno (linha 13). Após a saída do laço interno, o instante corrente de decisão é atualizado para o instante mais cedo de término de uma tarefa (NM) (linha 16) e NM tem seu valor atualizado (linha 18). Em seguida, é efetuada a atualização das listas de tarefas livres e de processadores ociosos afetados pela(s) última(s) decisões de escalonamento efetuadas (linhas de 18 a 21). Após o escalonamento de todas as tarefas do GAD, o algoritmo retorna o escalonamento construído (linha 23).

Na Figura 3.4 é ilustrada a criação de um escalonamento por ETF. É considerada um sistema distribuído com três processadores heterogêneos (a) e de um GAD de uma aplicação paralela (b). A Tabela (c) mostra os valores de CM e NM a cada iteração do processo de escalonamento, o par tarefa-processador selecionado e a decisão tomada a cada iteração do algoritmo. Os diagramas em (d) mostram os estados do escalonamento após a atribuição de cada tarefa do GAD.

A análise do Algoritmo 3.8 mostra que sua complexidade é dominada pelo

```
1  algoritmo ETF(M, G)
2      S = {}; I = P; Q = ∅; CM = 0; NM = ∞;
3      L = {t' ∈ T | PRED(t') = ∅};
4      enquanto |Q| < n faça
5          enquanto |I| > 0 e |L| > 0 faça
6              Encontre t̂ ∈ L e p̂ ∈ I tal que: w = min_{∀t'∈L,∀p'∈I} {f(t', p')};
7              ŝ = max{CM, w};
8              se ŝ ≤ NM então
9                  S = S ∪ {(t̂, p̂, s(t̂, p̂))};
10                 L = L − {t̂}; I = I − {p̂}; Q = Q ∪ {t̂};
11                 se f(t̂, p̂) ≤ NM então NM = f(t̂, p̂);
12             senão
13                 Abandone o laço;
14             fim
15         fim
16         CM = NM;
17         NM = min_{(t',p',m)∈S} {f(t', p') | f(t', p') > CM};
18         para cada t' ∈ Q e p' ∈ P | f(t', p') = CM faça
19             I = I ∪ {p'};
20             se nesc(t') = 0 então L = L ∪ {t'};
21         fim
22     fim
23     retorna S;
24 fim
```

Algoritmo 3.8: ETF - Earliest Task First



iter	CM	NM	t_i / p_j	decisão
1	0	∝	t_2 / p_2	escalona
2	0	3	t_1 / p_1	escalona
3	3	∝	t_5 / p_2	escalona
4	3	4	t_3 / p_1	adia
5	4	∝	t_3 / p_2	escalona
6	4	5	t_4 / p_1	adia
7	5	∝	t_4 / p_2	escalona
8	8	∝	t_6 / p_2	escalona
9	9	∝	t_7 / p_1	escalona

Figura 3.4: Escalonamento construído pela heurística ETF.

código contido nos dois laços aninhados principais (linhas 4 e 5). O laço externo executa pelo menos n iterações, já que o escalonamento de algumas tarefas pode ser adiado pela decisão tomada na linha 8. Para o laço interno existem duas possibilidades mutuamente exclusivas: (a) quando $\hat{s} \leq NM$, a tarefa candidata $\hat{t}$ é escalonada normalmente; (b) quando $\hat{s} > NM$ ou $|I| = 0$ ou $|L| = 0$, o laço é abandonado. Neste último caso, a variável CM tem seu valor atualizado para NM e pelo menos uma tarefa termina sua execução neste instante (aquela com *tempo de fim* igual a NM). Desta forma, no pior caso, o número de vezes que o *loop* externo é executado é da ordem de $O(2n)$. A complexidade do laço interno é determinada pelo código da linha 6 sendo igual a $O(n \times m)$. O código responsável por atualizações contido nas linhas 16 a 21 também possui complexidade igual a $O(n \times m)$. Desta forma, o

tempo do *loop* externo, que determina a complexidade de ETF, é igual a $O(2n^2 \times m)$ que, por sua vez, é proporcional a $O(n^2 \times m)$.

3.2.4 TASK - *Topological Assignment and Scheduling Kernel*

Até o momento, foram apresentadas heurísticas de construção que produzem escalonamentos através da atribuição, a cada passo, de uma tarefa de um GAD a um processador de um sistema. A cada iteração, a decisão referente ao escalonamento de uma tarefa é baseada em informações que maximizam (ou minimizam) características associadas às tarefas e aos processadores da instância envolvida. Esta estratégia gulosa pode fazer com que o algoritmo tome decisões que podem levar à produção de escalonamentos ineficientes. Neste contexto, uma técnica que pode levar a resultados promissores é a utilização de um procedimento de busca local após o processo de construção de um escalonamento. Um procedimento de busca local é um método que objetiva melhorar uma solução inicial efetuando uma série de movimentos em direção a melhores soluções existentes em sua vizinhança. Entenda-se por vizinhança como o conjunto de soluções que podem ser obtidas a partir da solução inicial através de um movimento (alteração local). Os métodos de busca local tem sido muito utilizados como um meio para encontrar ótimos locais para problemas de alta complexidade como o de escalonamento de tarefas [104].

O algoritmo *Topological Assignment and Scheduling Kernel* (TASK), introduzido por Min-You Wu *et al* [59], é um procedimento de busca local que procura minimizar o *makespan* de um escalonamento através da movimentação das tarefas escalonadas entre os processadores do sistema. Seja um GAD escalonado, ou simplesmente GADE, aquele que incorpora informações sobre a alocação e a seqüência de execução das tarefas nos processadores de acordo com um determinado escalonamento. Desta forma, os *custos* de execução e de comunicação das tarefas podem ser determinados com exatidão, uma vez que a alocação das tarefas já está definida pelo escalonamento. Assim, para incorporar características referentes à alocação das tarefas, um GADE utiliza os *custos* de execução e comunicação em vez dos *pesos* de execução e comunicação determinados por um GAD. Já para mapear a seqüência de

execução das tarefas, um GADE adiciona pseudo-arestas com custo de comunicação igual a zero entre tarefas adjacentes alocadas em um mesmo processador.

A Figura 3.5(c) apresenta um exemplo de um GADE criado a partir de uma instância (a) e de um escalonamento (b). Cada vértice representa uma tarefa da aplicação que está alocada em um determinado processador de acordo com o escalonamento (b). Os números ao lado de cada vértice indicam o *custo de execução* da respectiva tarefa no processador onde está alocada. As arestas representam o fluxo de comunicação entre as tarefas (mensagens) e o número em sua proximidade indica o *custo de comunicação* entre as tarefas envolvidas. Note que as arestas entre tarefas alocadas em um mesmo processador possuem custos de comunicação iguais a zero, considerando o modelo arquitetural adotado. As setas tracejadas representam as pseudo-arestas de custo zero adicionadas para mapear a seqüência de execução de tarefas adjacentes alocadas em um mesmo processador.

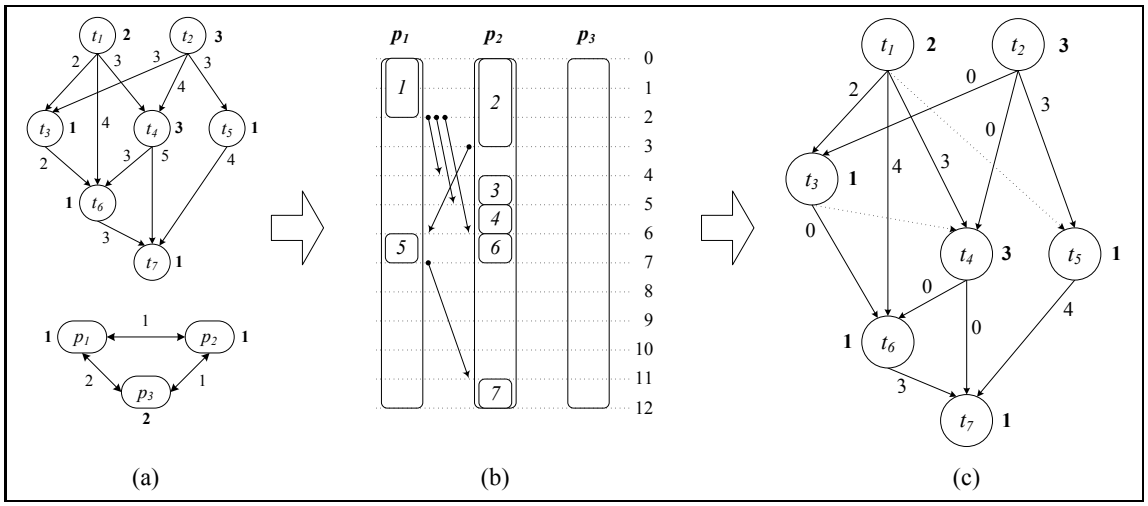


Figura 3.5: Um GAD escalonado.

O mecanismo de busca local implementado por TASK baseia-a na análise do *caminho crítico escalonado* de um GADE. O caminho crítico escalonado de um GADE é dado pelo caminho mais longo existente no grafo. O cálculo do comprimento do caminho crítico escalonado está relacionado com o *nível* e *co-nível* escalonados das tarefas do GADE. O *nível* escalonado de uma tarefa t_i, ou simplesmente $nível_e(t_i)$, é dado pelo somatório dos *custos* de execução e comunicação do maior caminho desde t_i até uma tarefa de saída do grafo. Já o *co-nível* escalonado de t_i, denotado por

co-nível$_e(t_i)$, é obtido pelo somatório dos *custos* de execução e comunicação do maior caminho desde uma tarefa de entrada até t_i, excluindo-se o custo de execução de t_i. Assim, o comprimento do caminho crítico escalonado de uma tarefa t_i, denotado por $L_e(t_i)$, pode ser obtido pela soma dos valores de *nível*$_e(t_i)$ e *co-nível*$_e(t_i)$. Desta forma, para um GAD escalonado $G = (T, E)$, ou simplesmente G_e, o comprimento do caminho escalonado é determinado pela seguinte expressão:

$$L_{cc}(G_e) = \max_{t_i \in T} \{L_e(t_i)\} \quad (3.1)$$

$$L_e(t_i) = \textit{nível}_e(t_i) + \textit{co-nível}_e(t_i) \quad (3.2)$$

$$\textit{nível}_e(t_i) = \max_{t_j \in SUCC(t_i)} \{\textit{nível}_e(t_j) + c(t_i, t_j)\} + e(t_i, p(t_i)) \quad (3.3)$$

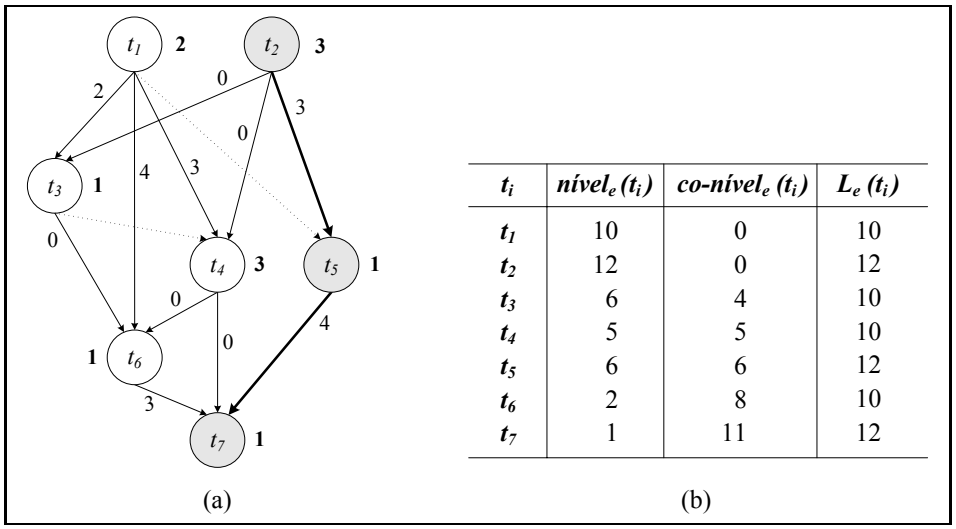
$$\textit{co-nível}_e(t_i) = \max_{t_j \in PRED(t_i)} \{\textit{co-nível}_e(t_j) + c(t_j, t_i) + e(t_j, p(t_j))\} \quad (3.4)$$

onde $L_{cc}(G_e)$ indica o comprimento do caminho crítico escalonado de G_e; *nível*$_e(t_i)$ e *co-nível*$_e(t_i)$ representam, respectivamente, os valores de *nível* e *co-nível* escalonados para a tarefa t_i; $e(t_i, p(t_j))$ (definido na Expressão 2.7) denota o custo de execução da tarefa t_i no processador $p(t_j)$ e $c(t_i, t_j)$ (Expressão 2.1) computa a latência no envio da mensagem entre as tarefas t_i e t_j.

Pode-se perceber que os valores de *nível*$_e(\cdot)$ e *co-nível*$_e(\cdot)$ são obtidos de maneira similar aos de *nível*$(\cdot)$ e *co-nível*$(\cdot)$ (Expressões 2.11 e 2.12, respectivamente), mas, no lugar de se utilizar os *pesos* médios de comunicação e computação determinados pela instância, consideram-se os *custos* de comunicação e execução de acordo com o escalonamento das tarefas nos processadores do sistema.

Um exemplo de cálculo e representação gráfica de um caminho crítico escalonado pode ser visto na Figura 3.6. Em (a), as tarefas e arestas em destaque indicam o caminho crítico escalonado do GADE apresentado na Figura 3.5(b). A Tabela (b) mostra os valores de *nível*, *co-nível* e comprimento de caminho crítico escalonados para cada tarefa do GADE. Analisando esta tabela, pode-se concluir que o comprimento do caminho crítico escalonado do grafo (a) é igual a 12, uma vez que este é o maior valor de $L_e(\cdot)$ para as tarefas do grafo.

A definição de caminho crítico escalonado é fundamental para a compreensão do mecanismo de busca implementado por TASK. Segundo Wu *et al* [21],



t_i	$nível_e(t_i)$	$co\text{-}nível_e(t_i)$	$L_e(t_i)$
t_1	10	0	10
t_2	12	0	12
t_3	6	4	10
t_4	5	5	10
t_5	6	6	12
t_6	2	8	10
t_7	1	11	12

Figura 3.6: Valores de *nível*, *co-nível* e comprimento de caminho crítico escalonados de um GAD escalonado.

o somatório dos custos de execução das tarefas pertencentes ao caminho crítico, representa um limite inferior para o *makespan* de um determinado escalonamento. Baseado neste princípio, TASK procura minimizar o comprimento do caminho crítico do grafo movimentando as tarefas do GADE entre os processadores do sistema.

O processo de busca realizado por TASK é efetuado a partir de um escalonamento S, de um grafo $G = (T, E)$ em um sistema distribuído $M = (P, L)$. Inicialmente, o algoritmo constrói um GAD escalonado G_e a partir de um GAD G e de um escalonamento S. A construção de G_e é efetuada através da identificação das tarefas vizinhas (cronologicamente adjacentes) em cada processador do sistema M, onde, para cada par de tarefas vizinhas, uma pseudo-aresta de custo zero é adicionada a G_e. Caso já exista uma aresta entre um par de tarefas adjacentes, seu custo de comunicação é definido como nulo. Após a construção de G_e, um conjunto Q das tarefas ainda não processadas recebe todas as tarefas do grafo. Em seguida, é efetuado o cálculo dos valores de $nível_e(\cdot)$ e $co\text{-}nível_e(\cdot)$ para cada tarefa de G_e, considerando as pseudo-arestas adicionadas. No passo seguinte, é selecionada a tarefa livre $\hat{t} \in Q$, alocada ao processador $p(\hat{t})$, com maior valor de $L_e(\hat{t})$. Caso mais de uma tarefa apresente o mesmo valor de $L_e(\cdot)$, o desempate é feito pelo menor valor de $co\text{-}nível_e(\cdot)$. Se o empate persistir, a escolha é feita aleatoriamente. Pode-se perceber que como somente as tarefas livres pertencentes a Q são analisadas, este

método de seleção garante que as tarefas são obtidas em ordem topológica, respeitando, desta forma, as restrições de precedência impostas pelas arestas (inclusive as pseudo-arestas) de G_e.

Após a seleção da tarefa $\hat{t}$, é analisada a possibilidade de redução de $L_e(\hat{t})$, avaliando o impacto de sua alocação em todos os outros processadores do sistema. Denota-se por $\overline{L}_e(\hat{t}, p_j)$ o comprimento do caminho crítico escalonado da tarefa $\hat{t}$ assumindo que sua alocação foi realizada no processador p_j, considerando, obviamente, os novos valores de $\mathit{nível}_e(\hat{t})$ e $\mathit{co\text{-}nível}_e(\hat{t})$ e as pseudo-arestas decorrentes desta movimentação. É importante ressaltar que, ao atribuir a tarefa candidata $\hat{t}$ a um novo processador, as restrições de precedência intraprocessador devem ser respeitadas, simulando a alocação de $\hat{t}$ após seu último predecessor presente neste processador. Se o algoritmo verificar que existe um processador $\hat{p}$ no qual a tarefa candidata apresentou redução no comprimento do caminho crítico escalonado, $\hat{t}$ é então atribuída definitivamente para este processador e as pseudo-arestas referentes a esta movimentação são atualizadas. Além disso, são recalculados os valores de $\mathit{co\text{-}nível}(\cdot)$ para todas as tarefas descendentes de $\hat{t}$, isto é, as tarefas pertencentes a todos os caminhos originados em $\hat{t}$. Note que, como as tarefas são visitadas em ordem topológica, não existe necessidade de recalcular os valores de $\mathit{nível}_e(\cdot)$ das tarefas ancestrais de $\hat{t}$ (tarefas pertencentes aos caminhos que terminam em $\hat{t}$), uma vez que estas tarefas já foram analisadas pelo algoritmo e estão definitivamente alocadas. Prosseguindo com a análise, se $\hat{t}$ não apresentar redução no valor do caminho crítico escalonado, a tarefa permanece no processador $p(\hat{t})$ ao qual estava originalmente alocada. Finalmente, o conjunto Q de tarefas não analisadas é atualizado com a remoção de $\hat{t}$ e o processo é reiniciado até que todas as tarefas de G_e sejam analisadas.

O Algoritmo 3.9 detalha o procedimento de busca local executado por TASK onde são fornecidos como entrada um sistema distribuído M com m processadores, um grafo G e um escalonamento S. Inicialmente, um grafo escalonado G_e é construído (linha 2) através da adição de pseudo-arestas de custo zero entre tarefas vizinhas alocadas em um mesmo processador (Algoritmo 3.10). Em seguida, o valor de $L_e(t_i)$ é calculado para cada tarefa $t_i \in G_e$ (linhas 3 a 5). A cada iteração do laço

principal (linhas 7 a 20), a tarefa $\hat{t}$ que maximiza o valor de $L_e(\cdot)$ é selecionada (linha 8) e o valor de $\overline{L}_e(\hat{t}, p_j)$ é calculado simulando a alocação de $\hat{t}$ em cada processador p_j do sistema (linhas 9 a 11). Em seguida, é identificado o processador $\hat{p}$ que apresenta o menor valor de $\overline{L}_e(\cdot)$ (linha 12). Se este processador é diferente do processador onde $\hat{t}$ está atualmente alocado (linha 13), $\hat{t}$ é então escalonada definitivamente em $\hat{p}$ (linha 14 e 15) e as pseudo-arestas de G_e (linha 16) e os valores de *co-nível*$_e(\cdot)$ dos descendentes de $\hat{t}$ são atualizados (linha 17). A tarefa $\hat{t}$ é então removida do conjunto Q (linha 19) e o processo é repetido até que todas as tarefas do grafo sejam analisadas.

```
1  algoritmo TASK(M, G, S)
2      G = cria_GADE(G, S);
3      para cada t_i ∈ G faça
4          Calcule valor de L_e(t_i);
5      fim
6      Q = T;
7      enquanto |Q| > 0 faça
8          t̂ = max{L_e(t_i) | t_i ∈ Q};
9          para cada p_j ∈ P faça
10             L̄_e(t̂, p_j) = L_e(t̂), considerando p(t̂) = p_j;
11         fim
12         p̂ = {p_j ∈ P | L_e(t̂, p_j) = min{L_e(t̂, p_k) | p_k ∈ P}};
13         se p̂ ≠ p(t̂) então
14             S = S − {(t̂, p(t̂), s(t̂, p(t̂))};
15             S = S ∪ {(t̂, p̂, s(t̂, p̂))};
16             Atualize pseudo-arestas no GADE;
17             Calcule o co-nível_e para as tarefas descendentes de t̂;
18         fim
19         Q = Q − {t̂};
20     fim
21     retorna S;
22 fim
```

Algoritmo 3.9: TASK - Topological Assignment and Scheduling Kernel

A Figura 3.7 apresenta um exemplo de como a busca local TASK pode melhorar um escalonamento. Pode-se observar a redução do *makespan* do escalonamento devido à movimentação da tarefa t_3 do processador p_2 para o processador p_1.

```
1  algoritmo cria_GADE(G(T, E), S)
2      G_e(T, E_e) = G(T, E);
3      para j = 1 até m faça
4          para i = 1 até |S_j| − 1 faça
5              E_e = E_e ∪ {(t̄_i, t̄_{i+1})};
6              ω(t̄_i, t̄_{i+1}) = 0;
7          fim
8      fim
9      retorna G_e;
10 fim
```

Algoritmo 3.10: Construção de um GAD escalonado.

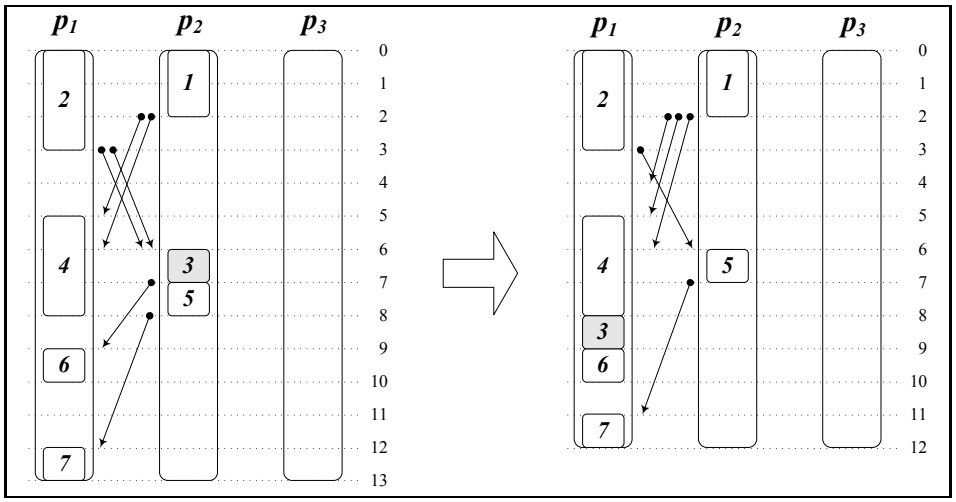


Figura 3.7: Aplicação da busca local TASK sobre um escalonamento.

Analisando a complexidade do algoritmo TASK, pode-se concluir que a construção do GADE G_e efetuada pela função $cria\_GADE(\cdot)$ (linha 2) consome tempo $O(n)$, já que ao procurar por pares de tarefas vizinhas escalonadas nos processadores do sistema, todas as tarefas do grafo são visitadas (veja Algoritmo 3.10). Como o cálculo do comprimento do caminho crítico (linhas 3 a 5) está relacionado com o número de arestas (e) de G_e, o tempo consumido é da ordem de $O(e)$. A busca pela tarefa de maior caminho crítico escalonado leva tempo igual a $O(n)$, considerando todas as iterações do *loop* principal (linhas de 7 a 20). O principal custo de computação do algoritmo está associado ao cálculo de $L_e(\cdot)$ em cada pro-

cessador. O custo deste cálculo para todas as tarefas, que está relacionado como o número de arestas do grafo, fica na ordem de $O(e)$. A verificação deste valor em cada processador, considerando as tarefas já alocadas, leva tempo proporcional a $O(n \times m)$. Desta forma, o tempo total de execução do algoritmo TASK é da ordem de $O(e + n \times m)$.

3.2.5 PSGA - *Problem-Space Genetic Algorithm*

Dentre as técnicas de computação evolutiva existentes, destacam-se os Algoritmos Genéticos (AG), que têm mostrado bastante eficiência na solução de problemas altamente combinatórios [105]. Em relação ao problema de escalonamento de tarefas, modelos que incorporam custos de comunicação e processadores heterogêneos são bastante complexos e têm recebido pouca atenção na literatura afim. Neste contexto, uma das poucas contribuições existentes é o trabalho de Dhodhi *et al* [17] que desenvolveram o Algoritmo Genético PSGA (*Problem-Space Genetic Algorithm*) para o problema. Neste algoritmo, os autores incorporam numa estrutura de AG a heurística EFT (*Earliest Finish Time*) para o escalonamento de tarefas de um GAD genérico em ambientes de computação heterogêneos. Resumidamente, PSGA funciona como descrito a seguir: inicialmente, são definidos os valores dos parâmetros para calibragem do AG, como o número de indivíduos da população (max_pop), número de gerações (max_gen), probabilidade de cruzamento ($prob\_cruz$) e probabilidade de mutação ($prob\_mut$). Além disso, os valores de *nível* e *co-nível* são calculados para cada tarefa do GAD em relação ao ambiente de computação. Em seguida, uma população inicial é criada associando a cada indivíduo valores (prioridades) relacionados com o *nível* e *co-nível* de cada tarefa do grafo. Neste algoritmo os indivíduos não representam uma solução para o problema, mas apenas uma etapa no processo de construção de uma solução, ou seja, o AG nesta caso, determina somente uma ordem de prioridade para cada tarefa. Estas prioridades são então passadas para heurística EFT que, a partir das prioridades associadas a cada indivíduo, produz soluções válidas (escalonamentos) para o problema. Durante cada iteração do AG, indivíduos são selecionados para reprodução através de

operações de cruzamento e mutação, produzindo uma nova geração de indivíduos com características herdadas da população anterior. As iterações ocorrem até que o critério de parada, definido como um número fixo de gerações, seja atingido.

O Algoritmo 3.11 ilustra o procedimento executado por PSGA. Como entrada são fornecidos um sistema distribuído M de m processadores, um grafo G e os parâmetros de calibragem do AG, tais como: número de gerações, tamanho da população e probabilidades de cruzamento e mutação. O algoritmo retorna o escalonamento associado ao melhor indivíduo gerado. O algoritmo começa, produzindo através da função $pop\_inicial(max\_pop)$ (linha 2) a população inicial (pop_0) com max_pop indivíduos. As prioridades atribuídas a cada gene i de um indivíduo, são obtidas através da combinação de valores relacionados com o *nível* e *co-nível* (Expressões 2.11 e 2.12, respectivamente) de cada tarefa t_i correspondente. Cada indivíduo s_k é gerado através da atribuição de uma lista de prioridades obtidas de acordo com a seguinte expressão:

$$s_k(i) = \begin{cases} nível(t_i) & \text{, se } k = 1 \\ nível(t_i) + rand(-\frac{co\text{-}nível(t_i)}{2}, +\frac{co\text{-}nível(t_i)}{2}) & \text{, se } 1 < k \leq max\_pop \end{cases} \tag{3.5}$$

onde $s_k(i)$ denota o valor de prioridade associado à tarefa t_i definido pelo indivíduo s_k e $rand(x, y)$, com $x, y \in \mathbb{R}$, um número gerado aleatoriamente entre x e y, inclusive. Como um indivíduo não representa uma solução final para o problema de escalonamento, é preciso submetê-lo ainda a uma heurística de decodificação que, a partir da lista de prioridades, gera um escalonamento válido das tarefas do GAD no sistema alvo. PSGA utiliza o algoritmo EFT (*Earliest Finish Time*) como heurística de decodificação, conforme definido a seguir:

$$decod(s_k) = EFT(M, G, g_k(\cdot)) \tag{3.6}$$

$$g_k(t_i) = s_k(i) \tag{3.7}$$

onde $g_k(\cdot)$ é a função de avaliação das prioridades relacionadas com o indivíduo s_k.

No Algoritmo 3.11, a cada geração g (laço das linhas 3 a 22), o algoritmo seleciona o melhor indivíduo produzido na população anterior (pop_{g-1}) com a função $pop\_campeão(\cdot)$ (linha 4) e o transfere para a população corrente (linha 5). Durante

o processo de reprodução, PSGA utiliza o *método da roleta* para a seleção de indivíduos (linhas 7 e 8). Neste método, cada indivíduo possui probabilidade de seleção proporcional ao valor de sua função de aptidão em relação aos demais membros da população. O cruzamento (linha 10), realizado com probabilidade $prob\_cruz$, ocorre através da seleção de dois indivíduos (pais) pertencentes à população atual. Em seguida, dois pontos de cruzamento c_1 e c_2, onde $1 \leq c_1 < c_2 \leq n$, são escolhidos aleatoriamente e o segmento dos pais situado entre estes pontos é permutado, resultando desta operação dois novos indivíduos. A mutação (linha 19), realizada com probabilidade $prob\_mut$ sobre a população corrente, é efetuada através da escolha aleatória de um gene i do indivíduo mutante e pertubando seu valor na faixa de $\frac{-co\text{-}nível(t_i)}{2}$ a $\frac{+co\text{-}nível(t_i)}{2}$. Usando um critério elitista, PSGA transporta para a geração seguinte a melhor solução da população corrente (linha 5), visando preservar a melhor solução encontrada até o momento. O algoritmo termina após atingir um número max_gen de gerações produzidas. O melhor indivíduo da população corrente é decodificado e o escalonamento obtido é retornado como resultado final (linha 23).

A Figura 3.8 ilustra o funcionamento de PSGA. A instância de entrada, composta por um sistema distribuído e um GAD, pode ser vista em (a). Em (b) é apresentada uma população de seis indivíduos gerados pelo algoritmo. Observe que cada indivíduo representa um conjunto diferente de prioridades que, ao serem decodificados, podem produzir diferentes escalonamentos. Uma operação de cruzamento é exemplificada em (c). Nesta etapa, os pontos de corte c_1 e c_2 são selecionados aleatoriamente e definem as seqüências de prioridades que devem ser permutadas entre os indivíduos s_1 e s_3 selecionados produzindo, desta forma, dois novos indivíduos s'_2 e s'_3. Uma operação de mutação pode ser observada em (d), onde os valores de prioridade do indivíduo s_6 são pertubados produzindo o novo indivíduo s'_6. Em (e) pode ser visto um processo de decodificação de um indivíduo em um escalonamento. A decodificação associa a cada tarefa do grafo um valor de prioridade que determina a seqüência de escalonamento das tarefas e ser efetuada pela heurística EFT.

```
1  algoritmo PSGA(M, G, max_pop, max_gen, prob_cruz, prob_mut)
2      pop_0 = pop_inicial(max_pop);
3      para g = 1 até max_gen faça
4          ŝ = pop_campeão(pop_{g-1});
5          pop_g = {ŝ};
6          para i = 1 até max_pop/2 faça
7              pai_1 = sel_roleta(pop_{g-1});
8              pai_2 = sel_roleta(pop_{g-1});
9              se prob(prob_cruz) então
10                 cruzamento(pai_1, pai_2, filho_1, filho_2);
11             senão
12                 filho_1 = pai_1;
13                 filho_2 = pai_2;
14             fim
15             pop_g = pop_g ∪ {filho_1, filho_2};
16         fim
17         para cada s_i ∈ pop_g faça
18             se prob(prob_cruz) então
19                 s_i = mutação(s_i);
20             fim
21         fim
22     fim
23     retorna decod(ŝ);
24 fim
```

Algoritmo 3.11: O Algoritmo Genético PSGA.

3.3 Resumo

Neste capítulo, foram apresentados os fundamentos sobre metaheurísticas GRASP e Algoritmos Genético, bem como algoritmos relacionados à proposta deste trabalho. Foram detalhadas três heurísticas do tipo *list scheduling*: *Earliest Finish Time* (EFT), *Heterogeneous Earliest Finish Time* (HEFT) e *Earliest Task First* (ETF); um procedimento de busca local denominado *Topological Assignment and Scheduling Kernel* (TASK); e um algoritmo genético chamado *Problem-Space Genetic Algorithm* (PSGA). Para as heurísticas *list scheduling* e o procedimento de busca local, foram apresentadas as respectivas análises de complexidade. No próximo capítulo, serão propostas duas novas heurísticas para o problema de escalonamento de

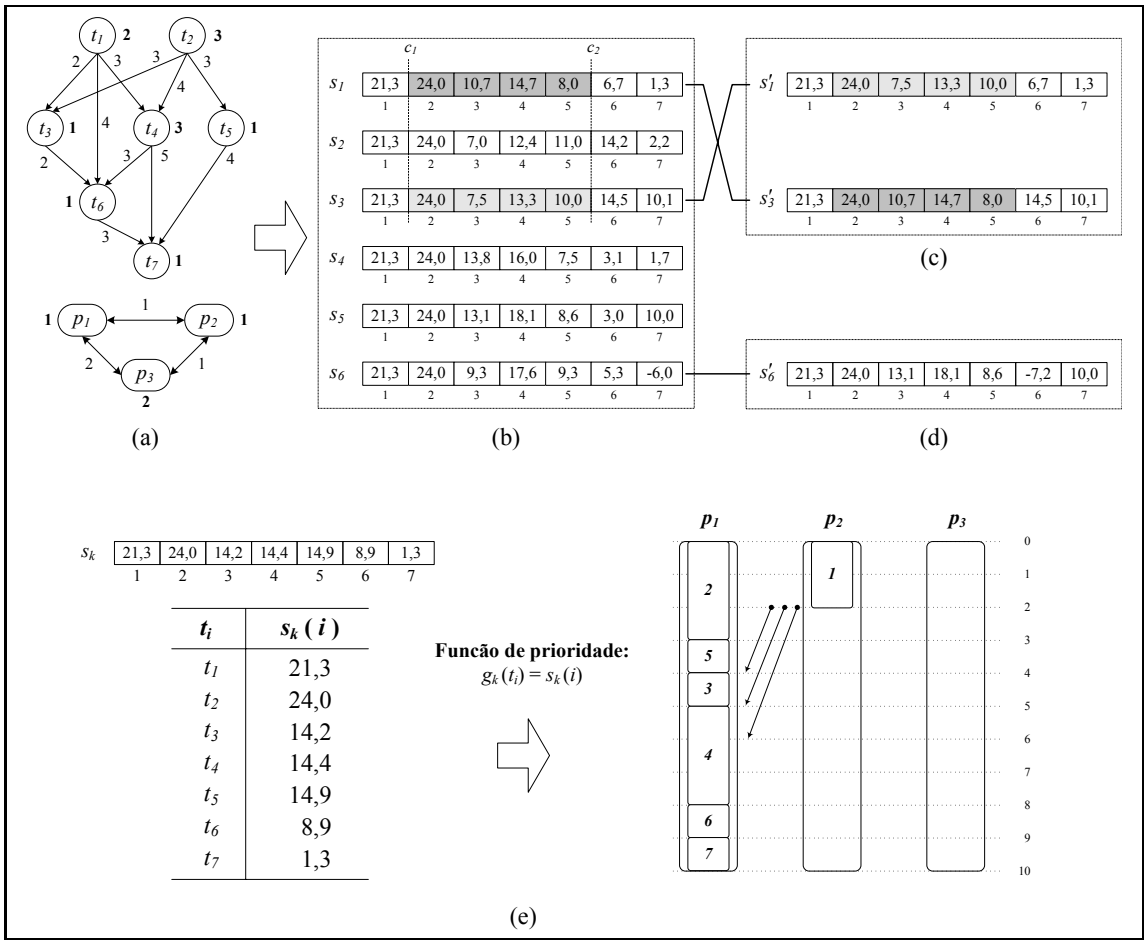


Figura 3.8: O processo de geração, cruzamento, mutação e decodificação de indivíduos efetuado por PSGA.

tarefas em ambientes heterogêneos baseadas nos algoritmos aqui apresentados.

Capítulo 4

Algoritmos Propostos

Conforme discutido anteriormente, a qualidade de um escalonamento depende fortemente de fatores diversos, tais como: a estrutura do GAD, as características do ambiente de computação e das estratégias utilizadas por algoritmos de escalonamento. As heurísticas de construção apresentadas nas seções anteriores, utilizam diferentes estratégias de escalonamento que não garantem a obtenção de uma solução de boa qualidade para qualquer instância. Este comportamento se deve, principalmente, ao fato de que as decisões relativas ao escalonamento de uma tarefa são baseadas em informações locais utilizando critérios gulosos. Quando uma determinada tarefa está para ser escalonada, muitas heurísticas não tem uma visão global sobre o impacto desta tomada de decisão. Desta forma, é provável que a estratégia utilizada por muitos algoritmos heurísticos tradicionais de escalonamento para decidir sobre o melhor destino para uma tarefa, não apresente um mesmo desempenho quando aplicada sobre instâncias com características variadas. Ou seja, as heurísticas de construção tradicionais normalmente são muito especializadas para tratar de determinadas classes de instâncias e, conseqüentemente, seu desempenho pode ser pobre quando submetidas a diferentes classes de problemas testes. Neste contexto, o uso de metaheurísticas para o problema de escalonamento apresenta-se como uma alternativa promissora. Devido a sua natureza genérica, estas técnicas permitem que diferentes estratégias de busca possam ser utilizadas em um mesmo

algoritmo heurístico, melhorando assim as chances de obtenção de uma solução final de qualidade para diferentes classes de instâncias.

Neste capítulo são apresentadas as abordagens propostas neste trabalho para o problema de escalonamento de tarefas em ambientes de computação heterogêneos. A primeira proposta é um algoritmo evolutivo baseado em Algoritmo Genético (AG). A segunda é uma heurística multi-partida para ser utilizada numa estrutura GRASP. Antes da definição de cada proposta, é apresentada uma breve fundamentação sobre as metaheurísticas AG e GRASP.

4.1 Algoritmo Proposto HTSGA

Nesta seção, é apresentado um novo algoritmo evolutivo híbrido para o problema de escalonamento estático de tarefas em ambientes computacionais heterogêneos. O algoritmo aqui denominado HTSGA (*Hybrid Task Scheduling Genetic Algorithm*), combina o uso de conceitos de Algoritmos Genéticos com heurísticas de construção da classe *list scheduling* para a produção de soluções de escalonamento. Basicamente, o algoritmo produz diversas listas de prioridades, que são associadas às tarefas de um GAD, com o objetivo de determinar diferentes seqüências de escalonamento destas tarefas no sistema de computação alvo. Então, novamente, cada indivíduo aqui não representa a solução final, mas um conjunto de informações que podem ser transformadas em soluções. Para produzir os escalonamentos associados às listas de prioridades, o algoritmo utiliza diversas heurísticas de construção distintas, com o objetivo de alcançar um desempenho regular para instâncias de classes variadas.

4.1.1 Representação do Indivíduo

Um indivíduo é representado por uma cadeia de elementos de comprimento fixo onde cada elemento armazena um valor de prioridade associado a uma determinada tarefa do grafo. A idéia por trás desta representação é determinar, baseado em

velores de prioridade, a ordem de escalonamento das tarefas de um GAD. Contudo, este modelo não representa uma solução final para o problema de escalonamento, pois o indivíduo precisa ser ainda submetido a uma **função de decodificação** que, a partir da lista de prioridades, produz um escalonamento final. A escolha desta função é um dos pontos chave para o bom desempenho de HTSGA, e deve ser efetuada tendo em vista os principais fatores que influenciam no desempenho de algoritmos de escalonamento. Como existe uma lista de prioridades envolvida, as heurísticas de escalonamento da classe *list scheduling* são a escolha natural para realizar esta tarefa. A análise dos resultados produzidos por algumas heurísticas desta classe encontradas na literatura [17, 21, 24], mostrou que as características das instâncias influenciam diretamente a qualidade dos resultados produzidos. Com o objetivo de obter um método mais consistente para lidar com instâncias de qualquer natureza, HTSGA foi projetado para utilizar diversas heurísticas de escalonamento diferentes para a decodificação dos indivíduos. Seja $\mathcal{H} = \{H_1(\cdot), \ldots, H_h(\cdot)\}$ o conjunto, com cardinalidade h, de funções de decodificação utilizadas pelo algoritmo. Cada elemento $H_k(\cdot) \in \mathcal{H}$ é uma heurística de construção que produz um escalonamento a partir de um sistema distribuído M, um grafo G e uma função de classificação de prioridades $g(\cdot)$:

$$H_k(M, G, g(\cdot)), \text{ onde } 1 \leq k \leq h \tag{4.1}$$

Assim, em HTSGA, um indivíduo é representado por uma cadeia numérica de tamanho igual ao número de tarefas n mais uma unidade. O primeiro elemento de cadeia é um inteiro k, $1 \leq k \leq h$, que indica a função de decodificação associada ao indivíduo e os n elementos seguintes representam a lista de prioridades associada às tarefas do GAD, onde cada elemento i, $1 \leq i \leq n$, determina a prioridade da tarefa t_i. A Figura 4.1 ilustra esta codificação para uma instância composta de um GAD com sete tarefas e um sistema com três processadores. Neste exemplo, assume-se que HTSGA utiliza três funções de decodificação diferentes. As prioridades associadas às tarefas do GAD são determinadas por $g(\cdot)$.

Na implementação de HTSGA realizada neste trabalho, foram utilizadas versões de três heurísticas de construção definidas anteriormente, a saber: EFT

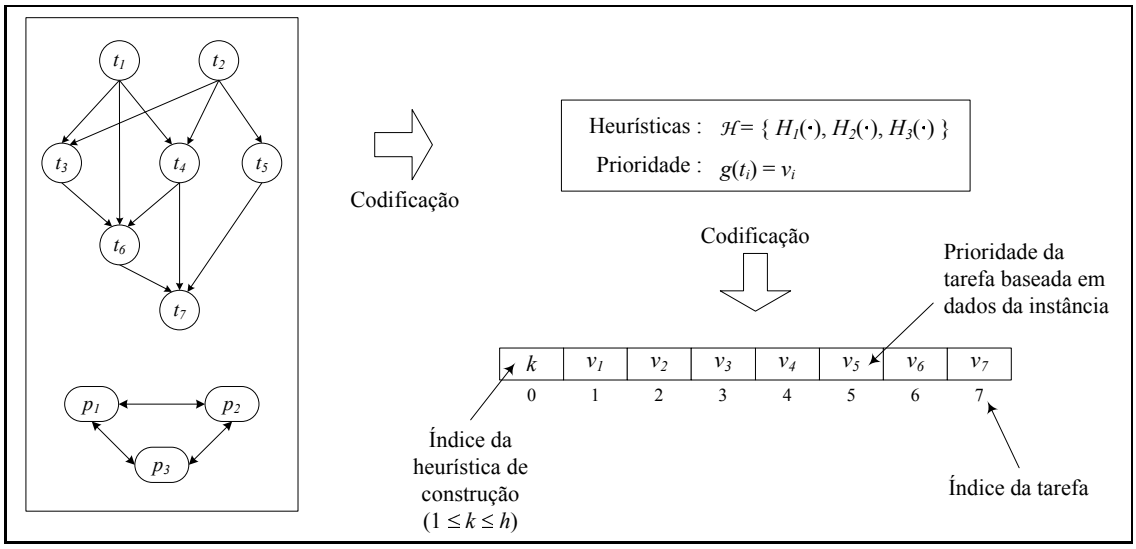


Figura 4.1: Codificação de um indivíduo efetuada por HTSGA.

(*Earliest Finish Time*) [17], HEFT (*Heterogeneous Earliest Finish Time*) [21] e ETF (*Earliest Task First*) [24], esta última adaptada para sistemas heterogêneos. Resumidamente, estas heurísticas apresentam as seguintes características:

EFT: A cada iteração, escalona a tarefa no processador que minimiza seu *tempo de fim*;

HEFT: A cada iteração, escalona a tarefa no processador que minimiza seu *tempo de fim*, considerando os espaços ociosos entre tarefas já escalonadas;

ETF: A cada iteração, seleciona o par tarefa-processador que minimiza o *tempo de fim* da tarefa.

As versões destas heurísticas embutidas em HTSGA, diferem ligeiramente das originais em relação à forma como as tarefas são ordenadas para escalonamento. Enquanto nas heurísticas EFT, HEFT e ETF esta ordenação é definida segundo critérios específicos de cada algoritmo, as versões implementadas para HTSGA — denotadas por $\mathrm{EFT}_{\mathrm{AG}}$, $\mathrm{HEFT}_{\mathrm{AG}}$ e $\mathrm{ETF}_{\mathrm{AG}}$, respectivamente — recebem do AG a seqüência em que as tarefas da aplicação devem ser escalonadas. Assim, para a implementação de HTSGA realizada neste trabalho, o conjunto de funções de decodificação foi definido como $\mathcal{H} = \{\mathrm{EFT}_{\mathrm{AG}}(\cdot), \mathrm{HEFT}_{\mathrm{AG}}(\cdot), \mathrm{ETF}_{\mathrm{AG}}(\cdot)\}$.

4.1.2 População Inicial

Na população inicial, com tamanho max_pop, os indivíduos são gerados através da atribuição de prioridades segundo valores relacionados com o *nível* e *co-nível* de cada tarefa do grafo. Os membros da população são divididos em $h = |\mathcal{H}|$ grupos, de forma que cada grupo de indivíduos utiliza uma das funções de decodificação definidas em $\mathcal{H}$. Cada gene i de um indivíduo s_k, denotado como $s_k(i)$, é gerado de acordo com a seguinte expressão:

$$s_k(i) = \begin{cases} (k \bmod h) + 1 & \text{, se } i = 0 \text{ e } 1 \leq k \leq max\_pop \\ nível(t_i) & \text{, se } i > 0 \text{ e } k = 1 \\ nível(t_i) + rand(-\frac{co\text{-}nível(t_i)}{2}, +\frac{co\text{-}nível(t_i)}{2}) & \text{, se } i > 0 \text{ e } 1 < k \leq max\_pop \end{cases} \quad (4.2)$$

onde $s_k(i)$, com $1 \leq i \leq n$, denota o valor da prioridade associado à tarefa t_i e $rand(x, y)$, com $x, y \in \mathbb{R}$, é uma função que gera um número aleatório entre x e y, inclusive. O termo $(k \bmod h) + 1$, armazenado pelo gene $s_k(0)$, garante a formação dos h grupos de indivíduos que utilizam cada uma das funções de decodificação presentes em $\mathcal{H}$, onde $k \bmod h$ representa o resto da divisão inteira entre k e h. Então, na própria geração da população, a abordagem aqui proposta atribui de antemão a heurística de construção a ser utilizada posteriormente no processo de decodificação.

Como cada indivíduo precisa ser decodificado para produzir uma solução válida para o problema, para simplificar a referência à função de decodificação associada a um indivíduo s_k, defina:

$$\begin{aligned} decod(s_k) &= H_j(M, G, g_k(\cdot)) \\ j &= s_k(0) \\ g_k(t_i) &= s_k(i) \end{aligned} \quad (4.3)$$

onde M e G representam, respectivamente, o sistema de computação e o GAD do problema, H_j, com $j = s_k(0)$ é a heurística de construção associada ao indivíduo s_k e $g_k(t_i)$, com $1 \leq i \leq n$, a função que determina a prioridade associada à tarefa t_i para o indivíduo s_k. A Figura 4.2 ilustra uma população gerada por HTSGA a partir de uma instância composta por um GAD e um sistema distribuído.

s_1	1	18,3	20,7	9,0	12,7	6,7	5,7	1,3
s_2	2	18,3	20,7	11,7	14,5	6,1	6,4	9,1
s_3	3	18,3	20,7	8,8	9,2	9,8	7,1	-6,4
s_4	1	18,3	20,7	8,6	11,9	6,7	-0,8	-3,7
s_5	2	18,3	20,7	6,1	11,5	7,2	12,7	-6,6
s_6	3	18,3	20,7	12,4	12,6	6,0	5,2	2,6

Figura 4.2: População de seis indivíduos gerada por HTSGA.

4.1.3 Função de Aptidão

Em problemas de escalonamento, a qualidade de uma solução é medida em relação ao valor do *makespan* obtido. No entanto, em sistemas multitarefa, onde os processadores são recursos disputados, outro fator importante que deve ser considerado na medida de qualidade de um escalonamento é o número de processadores efetivamente utilizados. Para considerar estes parâmetros na medida de qualidade de um indivíduo, HTSGA utiliza uma função de aptidão (*fitness*) que considera, para o escalonamento asssociado a um indivíduo s_k, o *makespan* e o número de processadores com tarefas atribuídas, expresso como:

$$fitness(s_k) = makespan(decod(s_k)) + np(decod(s_k)) \cdot 10^{-(\lfloor \log_{10} m \rfloor + 1)} \quad (4.4)$$

onde m representa o número total de processadores do sistema e $np(decod(s_k))$ especifica o número de processadores com pelo menos uma tarefa atribuída no escalonamento associado ao indivíduo s_k. O expoente da potência que figura como coeficiente de $np(\cdot)$ computa o número máximo de dígitos decimais necessários para representar o número m. Desta forma, a aptidão de um indivíduo é determinada por um número real onde a parte inteira representa o *makespan* e a parte fracionária o número de processadores utilizados pelo escalonamento associado. Suponha, por exemplo, que para um ambiente distribuído com 100 processadores, dois indivíduos s_i e s_j apresentem $fitness(s_i) = 63,004$ e $fitness(s_j) = 63,005$. Isto indica

que o indivíduo s_i, com *makespan* 63 utilizando 4 processadores, é mais apto que o indivíduo s_j, que utiliza 5 processadores para obter o mesmo *makespan*, já que o escalonamento associado ao primeiro indivíduo utiliza menos recursos.

Tradicionalmente, os Algoritmos Genéticos procuram maximizar a função de aptidão dos indivíduos. Contudo, o escalonamento de tarefas aqui estudado é um problema de minimização que procura obter um escalonamento com o menor *makespan* possível utilizando o mínimo de recursos do sistema. Na modelagem da função de aptidão de HTSGA, quanto menor o valor do $fitness(\cdot)$ de um indivíduo, melhor é a solução a ele associada. Desta forma, como função objetivo, HTSGA procura a cada iteração *minimizar* o valor da função de aptidão dos indivíduos.

A Figura 4.3 ilustra o processo de decodificação necessário para a avaliação de um indivíduo. No exemplo é utilizado o indivíduo s_5 da população apresentada na Figura 4.2. Abaixo, em (a), à esquerda, é apresentada uma tabela com os valores de prioridades que este indivíduo associa a cada tarefa e, à direita, as funções de prioridade, decodificação e de aptidão. Em (b) pode ser visto o escalonamento associado a este indivíduo produzido pela respectiva função de decodificação.

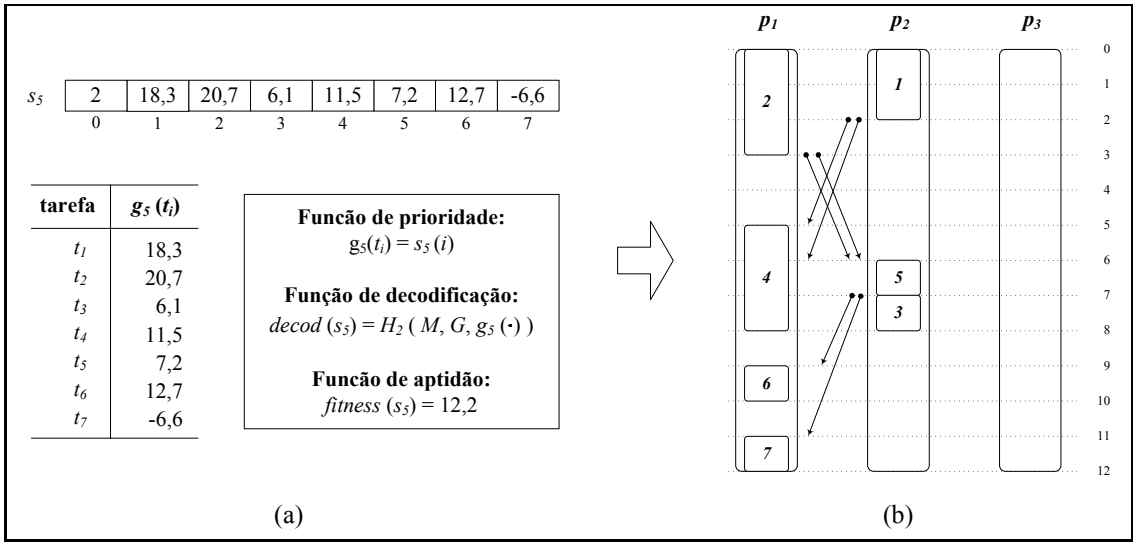


Figura 4.3: Processo de decodificação de um indivíduo realizado por HTSGA.

4.1.4 Seleção

No método de seleção utilizado por HTSGA, cada indivíduo possui probabilidade de ser selecionado proporcional ao valor de sua função de aptidão em relação aos demais membros da população e, conseqüentemente, de transmitirem suas características para gerações futuras. Baseada no *método da roleta* [106], a técnica de seleção implementada em HTSGA leva em consideração que os indivíduos mais aptos possuem *menor* valor da função de aptidão. Desta forma, a probabilidade de seleção de cada indivíduo s_k em uma população S é definida como:

$$prob(s_k) = \frac{fitness_{max} - fitness(s_k)}{\sum_{i=1}^{max\_pop}(fitness_{max} - fitness(s_i))} \quad (4.5)$$

$$fitness_{max} = \max_{s_k \in S}\{fitness(s_k)\} + 1 \quad (4.6)$$

onde $fitness_{max}$ é o maior valor de aptidão dos indivíduos da população S. Observe, na Expressão 4.6, que uma unidade foi adicionada ao maior valor de aptidão da população. Este artifício é usado para garantir que o pior indivíduo da população — aquele com maior valor da função $fitness(\cdot)$ — tenha oportunidade, mesmo que pequena, de participar da seleção.

O Algoritmo 4.1 mostra o pseudocódigo da implementação do método da roleta. Inicialmente, um segmento de comprimento d_i é associado a cada indivíduo s_i da população *pop* (linha 4) e o somatório de todos os d_i é calculado e armazenado em *soma*. Um valor inteiro aleatório r é então selecionado entre 0 e *soma*, inclusive (linha 7). Finalmente, o algoritmo identifica e retorna o indivíduo associado ao segmento que contém o valor de r sorteado no passo anterior (linhas 10 a 16).

4.1.5 Mecanismo de Diversificação

Em uma primeira análise, experimentos realizados mostraram que as soluções produzidas por HTSGA convergem rapidamente, já nas primeiras gerações, para a melhor solução encontrada pelo algoritmo. O principal motivo desta convergência prematura deve-se ao uso de heurísticas de construção que produzem soluções de boa

```
1  função seleção(pop)
2      soma = 0;
3      para cada s_i ∈ pop faça
4          d_i = fitness_max − fitness(s_i);
5          soma = soma + d_i;
6      fim
7      r = rand(0, soma);
8      sp = 0;
9      max = |pop|;
10     para i = 1 até max faça
11         se sp > r então
12             retorna s_i;
13         fim
14         sp = sp + d_i;
15     fim
16     retorna s_max;
17 fim
```

Algoritmo 4.1: Seleção pelo método da roleta realizado por HTSGA.

qualidade. Com o objetivo de aumentar a diversidade dos indivíduos da população, HTSGA incorpora um mecanismo de diversificação da população que, quando uma busca se mostra infrutífera em determinada região, permite ao algoritmo deslocar a busca para regiões ainda pouco exploradas. Esta estratégia é aplicada após um número max_div (parâmetro de entrada) de gerações sem atualização do melhor indivíduo (campeão). Uma característica importante da etapa de diversificação é a utilização de diferentes critérios para a atribuição de prioridades aos genes de um indivíduo. A cada operação de diversificação, um critério diferente de prioridade base e de pertubação são definidos. Um critério de prioridade base é um dado relacionado com a instância de entrada que determina o valor básico de prioridade para a geração de um indivíduo. Um critério de pertubação é um valor que modifica um critério de prioridade base. Por exemplo, considere o esquema de atribuição de prioridades descrito na geração da população inicial (Expressão 4.2). Neste caso, o critério de prioridade utilizado é o valor do $nível(\cdot)$ de uma tarefa e o critério de pertubação é um valor aleatório gerado por $rand(-co\text{-}nível(\cdot)/2, +co\text{-}nível(\cdot)/2)$. Na diversificação, toda a população corrente, com exceção do campeão, é descartada

e novos indivíduos são gerados utilizando um novo critério para geração de prioridades. Desta forma, seja $\mathcal{C} = \{(base_1(\cdot), pertub_1(\cdot)), \ldots, (base_c(\cdot), pertub_c(\cdot))\}$ o conjunto de tuplas que determina os c critérios de geração de prioridades utilizados por HTSGA. As funções $base_j(i)$ e $pertub_j(i)$, com $1 \leq j \leq c$ e $1 \leq i \leq n$, indicam, respectivamente, os critérios de geração de prioridade base e de pertubação para uma tarefa t_i do GAD. Se j, $1 \leq j \leq c$, é o critério de prioridade correntemente utilizado por HTSGA, a geração de um indivíduo s_k numa operação de diversificação é realizada de acordo com a seguinte expressão:

$$s_k(i) = \begin{cases} (k \bmod h) + 1 & \text{, se } i = 0 \text{ e } 1 \leq k \leq max\_pop \\ base_j(i) & \text{, se } i > 0 \text{ e } k = 1 \\ base_j(i) + pertub_j(i) & \text{, se } i > 0 \text{ e } 1 \leq k \leq max\_pop \end{cases} \tag{4.7}$$

Na implementação de HTSGA realizada neste trabalho, foi utilizado um conjunto de quatro critérios de geração de prioridades definidos de acordo com a Tabela 4.1 apresentada a seguir:

j	$base_j(t_i)$	$pertub_j(t_i)$
1	$nível(t_i)$	$rand(\frac{-co\text{-}nível(t_i)}{2}, \frac{+co\text{-}nível(t_i)}{2})$
2	$co\text{-}nível(t_i)$	$rand(\frac{-nível(t_i)}{2}, \frac{+nível(t_i)}{2})$
3	$nível(t_i) + co\text{-}nível(t_i)$	$rand(\frac{-co\text{-}nível(t_i)}{2}, \frac{+co\text{-}nível(t_i)}{2})$
4	$nível(t_i) + co\text{-}nível(t_i)$	$rand(\frac{-nível(t_i)}{2}, \frac{+nível(t_i)}{2})$

Tabela 4.1: Critérios de geração de prioridades de HTSGA.

Pode-se verificar que o critério de geração da população inicial (Expressão 4.2) é, na realidade, o critério utilizado quando $j = 1$.

O Algoritmo 4.2 apresenta o mecanismo de diversificação realizado por HTSGA. Inicialmente, o indivíduo campeão (s^*) é passado para a nova população (linha 2). O número de indivíduos que devem ser gerados para a nova população é então determinado (linha 3). Esta etapa é necessária porque este procedimento também é utilizado para gerar a população inicial. Neste caso, quando não existe ainda um melhor indivíduo definido, a população é inicializada como vazio. Em seguida, todos os indivíduos da nova população são criados segundo um critério de geração

de prioridades definidos por j. Finalmente, o algoritmo retorna a nova população gerada.

```
1  função pop_coloniza(j, s*)
2      pop = {s*};
3      max_indivs = max_pop − |pop|;
4      para k = 1 até max_indivs faça
5          s(0) = (k mod h) + 1;
6          para i = 1 até n faça
7              s(i) = base_j(i);
8              se i > 1 então
9                  s(i) = s(i) + pertub_j(i);
10             fim
11         fim
12         pop = pop ∪ {s};
13     fim
14     retorna pop;
15 fim
```

Algoritmo 4.2: Procedimento de diversificação de uma população implementado por HTSGA.

4.1.6 Cruzamento

Dados dois indivíduos s_i e s_j, selecionados da população pelo método da roleta, a operação de cruzamento inicia com a escolha aleatória de dois pontos de corte c_1 e c_2, tal que $0 \leq c_1 < c_2 \leq n$. Os pontos de cruzamento são escolhidos de tal forma que nunca $c_1 = 0$ e $c_2 = n$ simultaneamente. Em seguida, o segmento dos indivíduos s_i e s_j situado entre estes pontos é permutado, gerando, nesta operação, dois novos indivíduos f_1 e f_2. Este operador é aplicado com probabilidade $prob\_cruz$ durante o processo de reprodução. A Figura 4.4 exibe um exemplo de uma operação de cruzamento para dois indivíduos selecionados da população apresentada na Figura 4.2.

Observe que este mecanismo de cruzamento sempre produz soluções factíveis, pois as informações permutadas durante a operação determinam apenas as

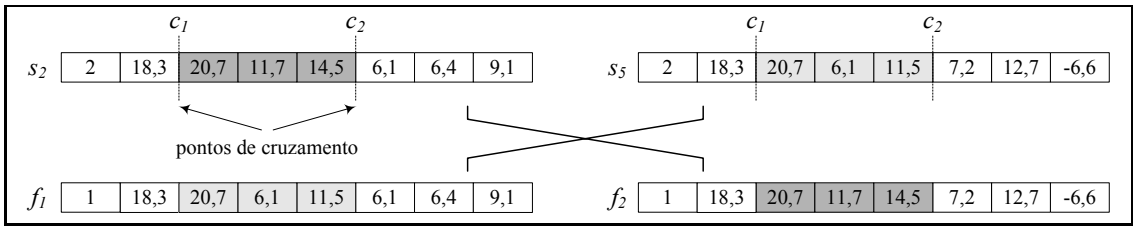


Figura 4.4: Operação de cruzamento efetuada por HTSGA.

prioridades associadas a cada tarefa da aplicação.

O Algoritmo 4.3 apresenta o pseudocódigo do operador de cruzamento implementado por HTSGA. A função $prob\_evento(prob\_cruz)$ (linha 2) é uma função que determina, baseada na probabilidade $prob\_cruz$, a ocorrência do evento cruzamento. Se a operação for realizada, o algoritmo seleciona aleatoriamente os pontos de corte c_1 e c_2 tal que $c_1 < c_2$ e nunca ocorre que $c_1 = 0$ e $c_2 = n$ simultaneamente (linha 3). Em seguida dois novo indivíduos $filho_1$ e $filho_2$ são gerados (linhas 4 a 11). Caso a função $prob\_evento(\cdot)$ determine a não ocorrência do evento, os indivíduos $filho_1$ e $filho_2$ gerados são cópias dos pais (linhas 14 e 15).

```
1  procedimento cruzamento(prob_cruz, pai1, pai2, filho1, filho2)
2      se prob_evento(prob_cruz) então
3          Selecione c1 e c2 aleatoriamente, tal que 0 ≤ c1 < c2 ≤ n;
4          para i = 0 até n faça
5              se (i ≥ c1) e (i ≤ c2) então
6                  filho1(i) = pai2(i);
7                  filho2(i) = pai1(i);
8              senão
9                  filho1(i) = pai1(i);
10                 filho2(i) = pai2(i);
11             fim
12         fim
13     senão
14         filho1 = pai1;
15         filho2 = pai2;
16     fim
17     retorna filho1, filho2;
18 fim
```

Algoritmo 4.3: Operador de cruzamento de HTSGA.

4.1.7 Mutação

O operador de mutação, aplicado com probabilidade *prob_mut*, é realizado através da pertubação dos valores associados aos genes de cada indivíduo de uma população. Se s_k é um indivíduo que vai sofrer mutação, e j indica o critério de geração de prioridades utilizado correntemente, a pertubação a ser efetuada é definida por:

$$s_k(i) = \begin{cases} s_k(i) & \text{, se } i = 0 \\ s_k(i) + pertub_j(i) & \text{, se } i > 0 \end{cases} \tag{4.8}$$

A Figura 4.5 ilustra o processo de mutação sofrido pelo indivíduo s_6 da população apresentada na Figura 4.2. Neste exemplo, o critério de pertubação aplicado foi para $j = 1$ de acordo com a Tabela 4.1.

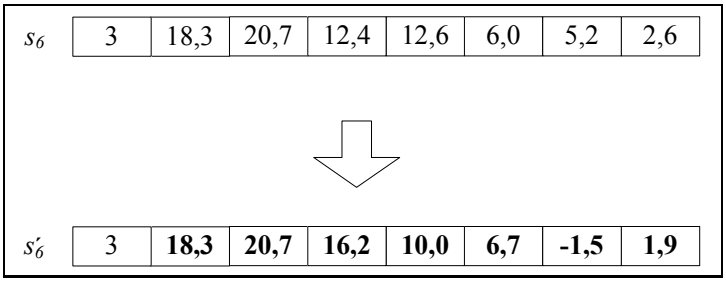


Figura 4.5: Operação de mutação realizada por HTSGA.

O Algoritmo 4.4 mostra o pseudocódigo do operador de mutação. Inicialmente, o algoritmo verifica se a operação será realizada baseado na probabilidade *prob_mut* (linha 2). No caso da ocorrência da operação, os genes do indivíduo s são pertubados segundo o critério de geração de prioridades corrente (indicado por j) gerando nesta operação um novo indivíduo m (linhas de 3 a 6). Caso a função de probabilidade determine a não ocorrência da operação, o próprio indivíduo s é retornado como resultado (linha 8).

4.1.8 Critério de Parada

Usando um critério elitista, HTSGA sempre migra para a geração seguinte a melhor solução da população corrente. Esta operação é responsável pela manutenção

```
1  função mutação(prob_mut, j, s)
2      se prob_evento(prob_mut) então
3          m(0) = s(0);
4          para i = 1 até n faça
5              m(i) = s(i) + pertub_j(i);
6          fim
7      senão
8          m = s;
9      fim
10     retorna m;
11 fim
```

Algoritmo 4.4: Operação de mutação de HTSGA.

da melhor solução encontrada ao longo das gerações. O algoritmo termina após atingir max_gen gerações produzidas, onde max_gen é um parâmetro de entrada.

O Algoritmo 4.5 apresenta o pseudocódigo de HTSGA. Inicialmente o algoritmo incializa as variáveis na e j (linhas 2 e 3) que controlam, respectivamente, o número de gerações sem atualização do campeão e o critério de geração de prioridades utilizado a cada iteração. A função $pop\_coloniza(j, s)$ (linhas 4 e 24) é utilizada para gerar uma nova população a partir de um critério de geração de prioridades c. Além disso, esta função garante a inclusão do indivíduo s na população gerada, visando preservar uma solução produzida anteriormente. Esta função é utilizada tanto para a geração da população inicial como para o procedimento de diversificação implementado pelo algoritmo. Na geração da população inicial (linha 4), como ainda não existem indivíduos produzidos, s é vazio. Na linha 5, a função $pop\_campeão(\cdot)$ retorna o indivíduo de melhor aptidão encontrado na população inicial. No laço principal (linhas 6 a 29), são executadas as iterações (gerações) do AG. A cada geração, uma população S_i é criada a partir da reprodução dos indivíduos da população anterior. Durante o processo de reprodução, o melhor indivíduo encontrado é preservado (linha 7) e operações de cruzamento e mutação são realizadas observando suas respectivas probabilidades de ocorrência (linhas 8 a 15). Em seguida, o indivíduo mais apto da nova população é identificado (linha 16) e, se for o caso, uma atualização do campeão é efetuada (linhas 17 a 19). Caso o melhor

indivíduo encontrado até o momento não seja atualizado, o algoritmo registra o fato incrementando o número de gerações sem melhora do campeão (linha 21). Após um número max_div de gerações sem incremento da melhor solução, o algoritmo muda a estratégia de geração de probabilidades (linha 23) e promove a diversificação da população atual através da geração de uma nova população, onde somente o campeão é preservado (linha 24). Após a execução de todas as gerações, o escalonamento associado ao melhor indivíduo produzido é retornado pelo algoritmo (linha 30).

4.2 Algoritmo Proposto HTSG

Neste trabalho é proposta uma heurística híbrida denotada *Hybrid Task Scheduling GRASP*, ou HTSG, que utiliza conceitos da metaheurística GRASP para o problema de escalonamento estático de tarefas em ambientes de computação heterogêneos. A natureza híbrida do algoritmo é determinada pelo uso de diferentes heurísticas durante a fase de construção, com o objetivo de possibilitar que o algoritmo apresente um comportamento estável quando submetido a instâncias com características variadas.

Seja $\mathcal{H} = \{H_1(\cdot), \ldots, H_h(\cdot)\}$ o conjunto, com cardinalidade h, de heurísticas de construção utilizadas por HTSG. Cada heurística $H_k(M, G, \alpha) \in \mathcal{H}$ gera um escalonamento S a partir das tarefas do grafo G no sistema distribuído M, e um fator α. O fator $\alpha \in [0, 1]$ determina o grau de aleatoriedade utilizado por $H_k(\cdot)$ na seleção de uma tarefa para escalonamento. A cada iteração, HTSG utiliza uma heurística de construção $H_k(\cdot) \in \mathcal{H}$, com $1 \leq k \leq h$, e um procedimento de busca local para produzir uma solução. Após executadas todas as iterações, o algoritmo retorna a melhor solução encontrada.

4.2.1 Representação e Avaliação da Solução

Uma solução S é representada por um escalonamento das tarefas de um GAD nos processadores de um sistema de computação heterogêneo, onde cada tarefa

```
1  algoritmo HTSGA(max_gen, max_pop, max_div, prob_cruz, prob_mut)
2      na = 0;
3      j = 1;
4      pop_0 = pop_coloniza(j, ∅);
5      s* = pop_campeão(S_0);
6      para i = 1 até max_gen faça
7          S_i = {s*};
8          para k = 1 até max_pop/2 faça
9              pai_1 = sel_roleta(S_{i-1});
10             pai_2 = sel_roleta(S_{i-1});
11             cruzamento(prob_cruz, pai_1, pai_2, filho_1, filho_2);
12             filho_1 = mutação(prob_mut, j, filho_1);
13             filho_2 = mutação(prob_mut, j, filho_2);
14             S_i = S_i ∪ {filho_1, filho_2};
15         fim
16         s' = pop_campeão(S_i);
17         se fitness(s') < fitness(s*) então
18             s* = s';
19             na = 0;
20         senão
21             na = na + 1;
22             se na = max_div então
23                 j = (j mod c) + 1;
24                 S_i = pop_coloniza(j, s*);
25                 s* = pop_campeão(S_i);
26                 na = 0;
27             fim
28         fim
29     fim
30     retorna decod(s*);
31 fim
```

Algoritmo 4.5: HTSGA - *Hybrid Task Scheduling Genetic Algorithm.*

do grafo é um elemento componente da solução. Uma solução é avaliada segundo o *makespan* e o número de processadores utilizados pelo escalonamento gerado, conforme definido a seguir:

$$aval(S) \quad = \quad makespan(S) + np(S) \cdot 10^{-(\lfloor \log_{10} m \rfloor + 1)} \tag{4.9}$$

onde S é um escalonamento, m representa a quantidade de processadores do sistema e $np(S)$ indica o número de processadores com pelo menos uma tarefa atribuída no escalonamento S. A função $aval(S)$ retorna, para o escalonamento S, um valor real onde a parte inteira indica o *makespan* e a parte fracionária o número de processadores efetivamente utilizados. A Figura 4.6 apresenta uma solução gerada por HTSG e sua respectiva avaliação. No exemplo, a solução apresenta valor de $aval(\cdot)$ igual a 10,2 indicando um escalonamento com *makespan* igual a 10 e número de processadores utilizados igual a 2.

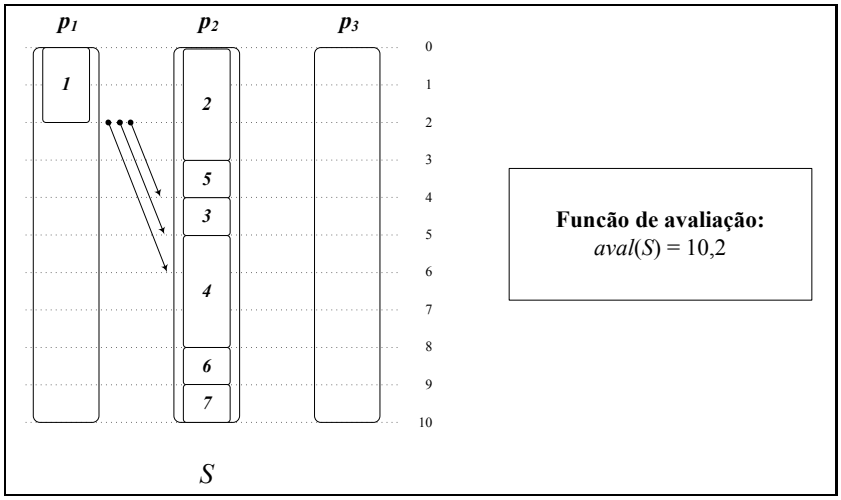


Figura 4.6: Uma solução e sua avaliação realizada por HTSG.

4.2.2 Fase de Construção

Conforme discutido, a construção de uma solução em um GRASP é realizada através de duas etapas: *seleção de elemento* e *atribuição de elemento*. Na primeira etapa, um elemento é selecionado aleatoriamente de uma Lista Restrita de Candidatos (LRC) construída de acordo com uma função gulosa $g(\cdot)$, que avalia o benefício de incorporar cada elemento. Na segunda etapa, o elemento selecionado é adicionado à solução que está sendo construída. Uma analogia deste processo com o método de construção de escalonamentos efetuados por heurísticas da classe *list scheduling*, leva à conclusão de que estes procedimentos são semelhantes. De fato, a etapa de *seleção de elemento* pode ser associada à fase de *priorização de tarefas*,

onde a função gulosa $g(\cdot)$ assume o papel da função de classificação de prioridades dos algoritmos *list scheduling*. Por sua vez, a etapa de *atribuição de elemento* pode ser relacionada com a fase de seleção de processador. No entanto, numa análise mais acurada, percebe-se que a fase de priorização de tarefas é o único fator que impede que heurísticas *list scheduling* possam ser utilizadas como métodos de construção em um GRASP. Isto porque o processo de priorização de tarefas é totalmente guloso, ou seja, a cada passo, um algoritmo *list scheduling* seleciona a tarefa de maior prioridade associada, caracterizando um algoritmo determinístico. Portanto, para que um algoritmo *list scheduling* possa atuar como procedimento de construção de um GRASP, a heurística deve incorporar um fator aleatório durante a fase de priorização de tarefas. Seja $\alpha \in [0, 1]$ este fator, que determina o grau de aleatoriedade na construção do escalonamento de algoritmo *list scheduling* randômico. Este parâmetro determina como uma heurística *list scheduling* seleciona uma tarefa livre para escalonamento. Por exemplo, um valor de $\alpha = 0$ indica que o algoritmo deve adotar um comportamento guloso, como faz normalmente. Já um valor de $\alpha = 1$ indica que, em cada iteração da heurística, a tarefa candidata a escalonamento deve ser selecionada aleatoriamente dentre todas as tarefas livres naquele instante. Este procedimento permite que estas heurísticas possam gerar diferentes seqüências de seleção de tarefas e, conseqüentemente, construir soluções não-determinísticas, adaptando-se perfeitamente como procedimentos de construção para HTSG.

Heurísticas de Construção Randomizadas

Na implementação de HTSG realizada para este trabalho, utilizamos uma variação das heurísticas EFT, HEFT e ETF, descritas no Capítulo 2, que foram adaptadas para funcionarem como heurísticas de construção randomizadas, batizadas, respectivamente, de EFT_{GR}, HEFT_{GR} e ETF_{GR}. A adaptação consistiu, basicamente, na introdução do parâmetro $\alpha \in [0, 1]$ que determina o grau de aleatoriedade do algoritmo no momento da decisão referente ao escalonamento de uma tarefa. Nas heurísticas EFT_{GR} e HEFT_{GR}, onde as prioridades associadas às tarefas são determinadas estaticamente, o parâmetro α exerce sua influência no momento

da seleção de uma tarefa livre para escalonamento. Nestas heurísticas, a cada iteração, uma Lista Restrita de Tarefas Candidatas (LRC) é criada a partir da lista de tarefas livres naquele instante. A LRC, cujo conteúdo é determinado em função de α, contém as tarefas com as maiores prioridades em cada iteração. Desta forma, o comprimento da LRC é determinado em função de α, já que dependendo deste parâmetro mais ou menos elementos serão utilizados em sua composição. A adaptação da heurística EFT foi ligeiramente diferente, já que neste algoritmo, as prioridades são determinadas dinamicamente. Esta heurística, procura, a cada iteração, o par tarefa-processador que apresenta o menor *tempo de fim* de uma tarefa pronta em um processador livre. Nesta fase, pode ocorrer que vários pares candidatos apresentem-se na mesma situação. Neste momento, o parâmetro α é utilizado para criar uma LRC de onde um par tarefa-processador pode ser escolhido aleatoriamente.

O Algoritmo 4.6 apresenta o pseudocódigo de EFT_{GR}. O algoritmo recebe como entrada um sistema distribuído M, um grafo G e um fator de aleatoriedade α. Em relação à heurística original (EFT), este algoritmo difere apenas no processo de seleção de tarefas para escalonamento (linhas 6 a 9). Para efetuar a seleção, uma Lista Restrita de Tarefas Candidatas (LRC) é criada em função de α e $g(\cdot)$ (linhas 6 a 8). Em seguida, a função $sel\_rand(LRC)$ (linha 9) seleciona aleatoriamente uma tarefa de LRC que é então escalonada no processador que minimiza seu *tempo de fim*. O algoritmo retorna o escalonamento produzido.

O pseudocódigo da versão randomizada de HEFT é mostrado no Algoritmo 4.7. O algoritmo HEFT_{GR} funciona de maneira similar à versão original, diferindo apenas no processo de seleção de uma tarefa para escalonamento. O processo de seleção é efetuado a partir da seleção aleatória (linha 9) de uma tarefa da LRC, construída em função de α e do $nível(\cdot)$ (linhas 6 a 8). A tarefa selecionada é então atribuída ao processador onde seu *tempo de fim* é minizado, levando em consideração os espaços ociosos entre tarefas alocadas em um mesmo processador (linha 10).

O Algoritmo 4.8 apresenta o pseudocódigo de ETF_{GR}. O princípio de funcionamento do algoritmo é o mesmo da heurística original, onde o diferencial reside na forma de desempate utilizada quando mais de um par tarefa-processador são

```
1  algoritmo EFT_GR(M, G, g(·), α)
2      Atribua prioridades às tarefas de T ∈ G;
3      S = {};
4      L = {t' ∈ T | PRED(t') = ∅};
5      enquanto |L| > 0 faça
6          g_min = min {g(t') | t' ∈ L};
7          g_max = max {g(t') | t' ∈ L};
8          LRC = {t' ∈ L | g(t') ≥ g_min + α · (g_max − g_min)};
9          t̂ = sel_rand(LRC);
10         p̂ = {p' ∈ P | f(t̂, p') = min{f(t̂, p'') | p'' ∈ P}};
11         S = S ∪ {(t̂, p̂, s(t̂, p̂))};
12         L = L − {t̂};
13         para cada t' ∈ SUCC(t̂) faça
14             se nesc(t') = 0 então L = L ∪ {t'};
15         fim
16     fim
17     retorna S;
18 end
```

Algoritmo 4.6: EFT_{GR} - *Earliest Finishing Time Randomized.*

```
1  algoritmo HEFT_GR(M, G, α)
2      Calcule o valor de nível para cada tarefa de G;
3      S = {};
4      L = {t_i ∈ T | PRED(t_i) = ∅};
5      enquanto |L| > 0 faça
6          n_min = min {nível(t') | t' ∈ L};
7          n_max = max {nível(t') | t' ∈ L};
8          LRC = {t' ∈ L | nível(t') ≥ n_min + α · (n_max − n_min)};
9          t̂ = sel_rand(LRC);
10         p̂ = {p_i ∈ P | f_ins(S, t̂, p_i) = min{f_ins(S, t̂, p_k) | p_k ∈ P}};
11         S = S ∪ {(t̂, p̂, f_ins(S, t̂, p̂) − e(t̂, p̂))} ;
12         L = L − {t̂};
13         para cada t' ∈ SUCC(t̂) faça
14             se nesc(t') = 0 então  L = L ∪ {t'};
15         fim
16     fim
17     retorna S;
18 fim
```

Algoritmo 4.7: HEFT_{GR} - *Heterogeneous Earliest Finish Time Randomized.*

candidatos à seleção. O algoritmo cria uma lista C de pares tarefa-processador que apresentam o menor *tempo de fim* possível entre as tarefas prontas e os processadores livres em um determinado instante (linha 8). Uma Lista Restrita de Candidatos é então construída a partir das tarefas contidas nos pares presentes em C que apresentam os maiores valores de *nível* (linhas 9 a 11). Em seguida, um par $(\hat{t}, \hat{p})$ é selecionado de LRC e o algoritmo procede como na versão original, retornando no final, o escalonamento gerado.

```
1  algoritmo ETF_GR(M, G, α)
2      S = {};
3      I = P;
4      L = {t' ∈ T | PRED(t') = ∅};
5      Q = ∅;   CM = 0;   NM = ∞;
6      enquanto |Q| < n faça
7          enquanto |I| > 0 e |L| > 0 faça
8              C = {(t', p') | f(t', p') = min{f(t'', p'') | t'' ∈ L e p'' ∈ I}};
9              n_min = min{nível(t') | (t', p') ∈ C};
10             n_max = max{nível(t') | (t', p') ∈ C};
11             LRC = {(t', p') ∈ C | nível(t') ≥ n_min + α · (n_max − n_min)};
12             (t̂, p̂) = sel_rand(LRC);
13             ŝ = max{CM, f(t̂, p̂)};
14             se ŝ ≤ NM então
15                 S = S ∪ {(t̂, p̂, s(t̂, p̂))};
16                 L = L − {t̂};   I = I − {p̂};   Q = Q ∪ {t̂};
17                 se f(t̂) ≤ NM então  NM = f(t̂);
18             senão
19                 Abandone o laço;
20             fim
21         fim
22         CM = NM;
23         NM = min_{(t',p',m)∈S} {f(t', p') | f(t', p') > CM};
24         para cada t' ∈ Q e p' ∈ P | f(t', p') = CM faça
25             I = I ∪ {p'};
26             se nesc(t') = 0 então  L = L ∪ {t'};
27         fim
28     fim
29     retorna S;
30 fim
```

Algoritmo 4.8: ETF_{GR} - *Earliest Task First Randomized.*

4.2.3 Busca Local

O HTSG pode utilizar diversos procedimentos de busca local para explorar a vizinhança de uma solução gerada na fase de construção. Diferentemente do procedimento de construção, não existe a necessidade da aplicação de um fator aleatório para efetuar um procedimento de busca, assim pode ser utilizado qualquer algoritmo que gere um escalonamento a partir de outro previamente determinado. Na implementação aqui realizada, foi utilizado o procedimento de busca local definido como TASK [59], descrito no capítulo anterior.

O Algoritmo 4.9 apresenta o pseudocódigo de HTSG. A variável i, com $1 \leq i \leq h$, indica a heurística de construção utilizada. A cada iteração, uma solução é construída com a heurística $h_i(\cdot)$ e o escalonamento obtido é submetido à busca local TASK. A melhor solução obtida é armazenada em S^*. Antes do início da próxima iteração um novo procedimento de construção é selecionado e o processo é repetido até que todas as iterações sejam executadas.

4.2.4 Fase de Treinamento

Conforme já comentado, a função do parâmetro $\alpha \in [0, 1]$ é determinar se a heurística de construção do GRASP deve adotar um comportamento guloso ou aleatório. Dependendo da natureza da instância para qual uma solução estiver sendo construída, um ou outro comportamento pode ser adequado.

Desta forma, foi incorporado um mecanismo de treinamento em duas fases executado durante as iterações de HTSG. Na fase inicial, o algoritmo reserva a primeira metade das iterações para analisar qual valor do parâmetro $\alpha \in [0, 1]$ está associado à melhor solução obtida nesta etapa. Na fase seguinte, o valor de α determinado na fase anterior é fixado e utilizado nas próximas iterações. O objetivo deste procedimento é permitir que a heurística se adapte às características específicas de cada instância, determinando assim se deve utilizar uma estratégia de construção de soluções gulosa ou aleatória.

```
1  algoritmo HTSG(M, G, α, max_iters)
2      S* = {};
3      i = 1;
4      para k = 1 até max_iters faça
5          S = H_i(M, G, α);
6          S = TASK(M, G, S);
7          se aval(S) < aval(S*) então
8              S* = S;
9          fim
10         i = i + 1;
11         se i > h então
12             i = 1;
13         fim
14     fim
15     retorna S*;
16 fim
```

Algoritmo 4.9: HTSG - *Hybrid Task Scheduling GRASP*.

4.3 Resumo

Neste capítulo foram apresentadas duas abordagens baseadas em metaheurísticas GRASP e Algoritmo Genético. As abordagens propostas são algoritmos híbridos que podem incorporar diversas heurísticas para produzir soluções para o problema de escalonamento de tarefas. No próximo capítulo, será estudado o desempenho destas heurísticas quando submetidas a instâncias de aplicações e sistemas distribuídos com características variadas.

Capítulo 5

Resultados Experimentais

Neste capítulo são apresentados os resultados computacionais obtidos pelas heurísticas *Hybrid Task Scheduling Genetic Algorithm* (HTSGA) e *Hybrid Task Scheduling GRASP* (HTSG), propostas neste trabalho. Além da análise destes resultados, o desempenho dos algoritmos aqui propostos é comparado com três outros algoritmos encontrados na literatura, sendo duas heurísticas pertencentes à classe *list scheduling* (EFT [24] e HEFT [21]) e um algoritmo genético (PSGA [17]).

5.1 Metodologia da Análise

A introdução de fatores de heterogeneidade nos processadores e nos canais de comunicação de um sistema de computação, colabora para aumentar a complexidade do problema de escalonamento de tarefas [107]. Na tentativa de identificar quais os fatores que influenciam no desempenho das heurísticas propostas, o estudo dos resultados deste trabalho se concentra em grupos específicos de GADs que são classificados de acordo com a topologia e a granularidade. A classificação dos GADs oferece a possibilidade de examinar a robustez das estratégias para as diferentes classes de topologias, pois, geralmente, as heurísticas de construção produzem soluções de escalonamento eficientes para classes específicas de grafos. Já a classificação em

relação à granularidade possibilita a análise de desempenho dos algoritmos quando submetidos a instâncias com uma mesma topologia, mas com fatores de computação e/ou comunicação variados. Além da análise em função da topologia e granularidade, também foi estudado o comportamento das heurísticas no processamento de grafos que representam aplicações reais e de instâncias de GADs extraídas da literatura.

Neste estudo, foram utilizadas diversas instâncias de aplicações paralelas e de ambientes computacionais para analisar o desempenho das heurísticas propostas. As seções seguintes descrevem a metodologia utilizada na análise dos resultados.

5.1.1 Instâncias de Aplicações Paralelas

A análise dos resultados obtidos se concentrou em dois conjuntos de grafos. O primeiro conjunto, denominado GADs *unitários*, é composto por um grupo de grafos conhecido como *Unit Execution Transfer and Unit Data Transfer* (UET-UDT), extraído de [43]. Neste grupo, onde as instâncias possuem pesos de computação e comunicação unitários, os grafos apresentam-se com topologias regulares (árvores binárias, árvores binárias invertidas e diamantes) e irregulares (gerados randomicamente). No segundo conjunto, denominado de GADs *não unitários*, os grafos apresentam topologias e pesos de computação e comunicação variados. As instâncias deste conjunto são formadas pelo grupo de grafos *Peer Set Graphs* (PSG), composto por GADs extraídos da literatura [10] e por um grupo de grafos que representam o cálculo da eliminação de Gauss [108]. A relação completa das instâncias de GADs utilizadas neste trabalho ser encontrada no Apêndice B.

Um dos aspectos estudados neste trabalho foi a variação da granularidade dos GADs unitários. Para este fim, foram definidos dois fatores multiplicativos, $me \in \mathbb{N}$ e $mc \in \mathbb{N}$, que modificam os pesos de execução das tarefas e de comunicação das arestas de um grafo. Desta forma, ao aplicar-se os fatores (me, mc) a um grafo $G = (T, E)$, o peso de execução de toda tarefa $t_i \in T$ passa a ser dada por $me \cdot \varepsilon(t_i)$ e o peso de comunicação associado a toda aresta $(t_i, t_j) \in E$ passa a ser computado como $mc \cdot \omega(t_i, t_j)$.

5.1.2 Instâncias de Ambientes Computacionais

Para a representação dos sistemas computacionais utilizados nos experimentos, foi definido um tipo de ambiente denominado *dual*. Neste ambiente, os processadores do sistema são divididos em duas classes, de acordo com o grau de heterogeneidade $h(\cdot)$, classificando estes processadores como rápidos, quando $h(\cdot) = 1$, e lentos, quando $h(\cdot) = 2$. Em relação aos canais de comunicação, todos os ambientes gerados apresentam um mesmo fator de latência $\lambda(\cdot) = 1$. Esta modelagem, permite observar o comportamento das heurísticas em relação ao uso de processadores rápidos e lentos durante o escalonamento de uma aplicação. A relação completa das instâncias de ambientes computacionais utilizadas neste trabalho pode ser vista no Apêndice C.

Fazendo uma analogia com sistemas reais, o ambiente dual poderia representar um cluster de computadores composto por máquinas com dois tipos de processadores, um deles, de última geração, com processadores mais rápidos e o outro com processadores mais lentos. Neste cluster, onde os computadores seriam pertencentes a uma rede local, as máquinas teriam um mesmo tipo de interface de comunicação, sendo conectadas através de um *switch*.

5.1.3 Execução dos Experimentos

Os algoritmos aqui apresentados foram implementados em linguagem C++ e compilados com o *GNU C++ Compiler* versão 3.3.3 sobre plataforma Linux com kernel 2.6.5. Os conjuntos de testes foram executados em um microcomputador Intel Pentium 4 com 2,8 GHz de freqüência e 512MB de RAM instalada.

Em todos os experimentos, os Algoritmos Genéticos PSGA e HTSGA foram executados utilizando-se os mesmos parâmetros de configuração para o número de gerações ($max\_gen = 120$), tamanho da população ($max\_pop = 20$) e probabilidades de cruzamento ($prob\_cruz = 80\%$) e mutação ($prob\_mut = 5\%$). Estes valores foram determinados empiricamente pela observação da qualidade das soluções geradas em relação à variação destes parâmetros. Tradicionalmente, os Algoritmos

Genéticos utilizam um número maior de gerações na busca por uma melhor solução. Nas aplicações aqui apresentadas, a qualidade das soluções geradas pelas heurísticas de construção embutidas (EFT_{AG}, ETF_{AG} e $HEFT_{AG}$), leva a uma rápida convergência para uma solução sub ótima, isto é, o AG obtém sua melhor solução já nas primeiras iterações. Por este motivo, estas aplicações utilizam um pequeno número de iterações (max_gen) e de indivíduos na população (max_pop), o que, no final, representa um ganho significativo no tempo total de execução da implementação desta heurística.

Para os experimentos com o GRASP HTSG, foi aplicado um número de iterações $max\_iter = 120$ e valores de $\alpha \in \{0,00; 0,10; 0,15; 0,20; 0,40; 0,80; 1,00\}$ para serem utilizados durante a fase de treinamento. Desta forma, durante a primeira metade das iterações do GRASP, o programa seleciona qual valor de α está associado à melhor solução obtida, para depois executar as iterações seguintes com o α determinado na etapa anterior.

Cada algoritmo analisado foi executado cinco vezes e o melhor resultado obtido foi selecionado. Os testes se concentraram em dois conjuntos básicos de instâncias de ambientes computacionais, considerando o número de processadores envolvidos: *restritos* e *não restritos*. Para cada conjunto, foram analisados os resultados produzidos pelas metaheurísticas propostas e pelas heurísticas da literatura, conforme será visto nas próximas seções.

5.2 Ambientes Computacionais Não Restritos

Numa análise inicial, os testes foram realizados sobre ambientes computacionais de forma que a quantidade de processadores disponíveis fosse suficiente para o escalonamento das tarefas da maioria dos GADs submetidos aos ambientes ditos não restritos. Este tipo de ambiente permite que as heurísticas de escalonamento possam explorar o paralelismo da aplicação quando não existe uma limitação no número de processadores disponíveis. Através deste experimento, é possível observar a eficiência das heurísticas tanto na minimização do *makespan* quanto do número de

processadores efetivamente utilizados pelos escalonamentos produzidos. É correto afirmar que não se está otimizando o número processadores, mas o *makespan* dos escalonamentos. Contudo, o mecanismo de avaliação implementado permite que quando duas soluções apresentam o mesmo valor de *makespan* o algoritmo opte pela que utiliza menos processadores.

Para a geração das instâncias utilizadas neste estudo, foram fixadas três dimensões de ambiente (m), cada uma contendo um total de 20, 40 e 60 processadores, respectivamente. Em cada ambiente, existem q processadores rápidos, com fator de heterogeneidade $h(\cdot) = 1$, e $m - q$ processadores lentos, com $h(\cdot) = 2$. Em todos os ambientes criados, o valor de q varia de $\frac{1}{4}m$ a m em incrementos de 25% de m. Por exemplo, para as instâncias com dimensão de 20 processadores, foi gerado um grupo de 4 instâncias, onde em cada instância existem 5, 10, 15 e 20 processadores rápidos, respectivamente. Desta forma, foram construídas 4 instâncias para cada uma das 3 dimensões citadas, totalizando 12 ambientes computacionais distintos.

Para facilitar a referência a ambientes do tipo dual, fica convencionado que o termo *sistema dual* m/q indica um sistema distribuído dual com um total de m processadores onde os q primeiros são mais rápidos que os demais. Por exemplo: *sistema dual 60/15*, indica um ambiente computacional distribuído com 60 processadores onde os 15 primeiros são mais rápidos que os 45 demais.

5.2.1 GADS Unitários

Os GADs unitários formam um grupo de grafos com topologias do tipo diamante, árvore binária (*out-tree*), árvore binária invertida (*in-tree*) e randômicos, totalizando 42 instâncias (ver Apêndice B). A análise dos GADs unitários é conveniente pois permite que a granularidade dos grafos seja controlada através do par de fatores (me, mc), conforme discutido na Seção 5.1.1. Os experimentos executados para estas instâncias consideraram grafos com granularidade grossa, obtidos pela aplicação dos fatores (10,1), (5,1), (1,1) e com granularidade fina, com os fatores (1,5) e (1,10). Cada um dos grafos unitários foi submetido aos ambientes compu-

tacionais não restritos descritos anteriormente, produzindo um total de 2.520 casos diferentes.

Diamantes

Foi estudado o desempenho dos algoritmos para grafos do tipo diamante com número de tarefas variando de 25 a 1024, num total de 9 instâncias. Cada grafo foi submetido aos 12 ambientes não restritos previamente definidos, onde, para cada ambiente, a granularidade dos GADs foi variada de acordo com os cinco pares de fatores (me, mc) já definidos, totalizando 540 casos.

Numa primeira análise, para cada uma das heurísticas estudadas, foram contados os casos em que os *makespans* produzidos apresentaram resultados melhores, iguais ou piores em relação aos *makespans* obtidos pelas demais heurísticas. O resultado pode ser visto na Tabela 5.1, onde cada grupo de três linhas mostra, em valores absolutos e percentuais, o número de casos em que uma heurística apresentou resultados piores (>), iguais (=) e melhores (<) quando comparados com cada uma das outras heurísticas confrontadas.

		ETF		**HEFT**		**PSGA**		**HTSGA**		**HTSG**	
	>			46	8,5%	246	45,6%	260	48,1%	252	46,7%
ETF	=			140	25,9%	97	18,0%	280	51,9%	288	53,3%
	<			354	65,6%	197	36,5%	0	0,0%	0	0,0%
	>	354	65,6%			444	82,2%	456	84,4%	448	83,0%
HEFT	=	140	25,9%			96	17,8%	84	15,6%	92	17,0%
	<	46	8,5%			0	0,0%	0	0,0%	0	0,0%
	>	197	36,5%	0	0,0%			423	78,3%	376	69,6%
PSGA	=	97	18,0%	96	17,8%			117	21,7%	155	28,7%
	<	246	45,6%	444	82,2%			0	0,0%	9	1,7%
	>	0	0,0%	0	0,0%	0	0,0%			29	5,4%
HTSGA	=	280	51,9%	84	15,6%	117	21,7%			306	56,7%
	<	260	48,1%	456	84,4%	423	78,3%			205	38,0%
	>	0	0,0%	0	0,0%	9	1,7%	205	38,0%		
HTSG	=	288	53,3%	92	17,0%	155	28,7%	306	56,7%		
	<	252	46,7%	448	83,0%	376	69,6%	29	5,4%		

Tabela 5.1: Comparação de desempenho em termos de *makespans* piores (>), iguais (=) e melhores (<) para escalonamentos gerados em *ambientes não restritos* a partir de GADs *diamante* para um total de 540 casos.

Pela Tabela 5.1, verifica-se que quando comparados com a heurística PSGA, os algoritmos HTSGA e HTSG apresentam desempenho igual ou superior em praticamente todos os casos. Este resultado foi possível devido à robustez das heurísticas propostas ao aplicarem diversos métodos para a construção de soluções. Conforme descrito no Capítulo 4, os algoritmos HTSGA e HTSG são heurísticas híbridas, pois incorporam em seus mecanismos de busca diversas algoritmos de construção de soluções.

Na implementação realizada neste trabalho, o Algoritmo Genético HTSGA utiliza versões das heurísticas EFT, ETF e HEFT para experimentar diversas seqüências de escalonamento durante a execução de suas iterações. Para realizar buscas através do mecanismo de construção de cada um destes algoritmos, o número total de iterações do AG é dividido em três blocos, onde em cada bloco é aplicada uma das heurísticas embutidas (EFT_{AG}, ETF_{AG} e $HEFT_{AG}$). Da mesma forma, a versão do GRASP HTSG implementada, divide suas iterações eqüitativamente entre as versões randomizadas das heurísticas da literatura (EFT_{GR}, ETF_{GR} e $HEFT_{GR}$) durante a fase de construção de soluções. O objetivo desta metodologia é fazer com que HTSGA e HTSG procurem se adaptar às características intrínsecas de cada instância, permitindo efetuar buscas em regiões onde somente cada uma destas heurísticas pode chegar. Isto significa que a variação no uso de técnicas de construção realizada por HTSGA e HTSG, permite uma melhor adaptação para cada caso, pois procura extrair de cada heurística embutida o melhor resultado que ela pode obter.

Um outro resultado interessante na Tabela 5.1 pode ser observado quando é feita uma comparação dos resultados de ETF e HEFT em relação às heurísticas propostas, que apresentam desempenho igual ou superior em 100,0% os casos. Este resultado já era esperado, já que os algoritmos HTSGA e HTSG incorporam versões destas heurísticas em seus mecanismos de busca. No caso de HTSGA, este resultado é atingido devido aos métodos de geração de prioridades utilizados pelo algoritmo. Estes métodos, que determinam as seqüências de escalonamento, podem produzir a mesma ordenação de tarefas utilizadas por ETF e HEFT quando as versões embutidas destes algoritmos são utilizadas como heurísticas de construção de soluções (veja Seção 4.1.2). Já para HTSG, este resultado é alcançado quando uma iteração

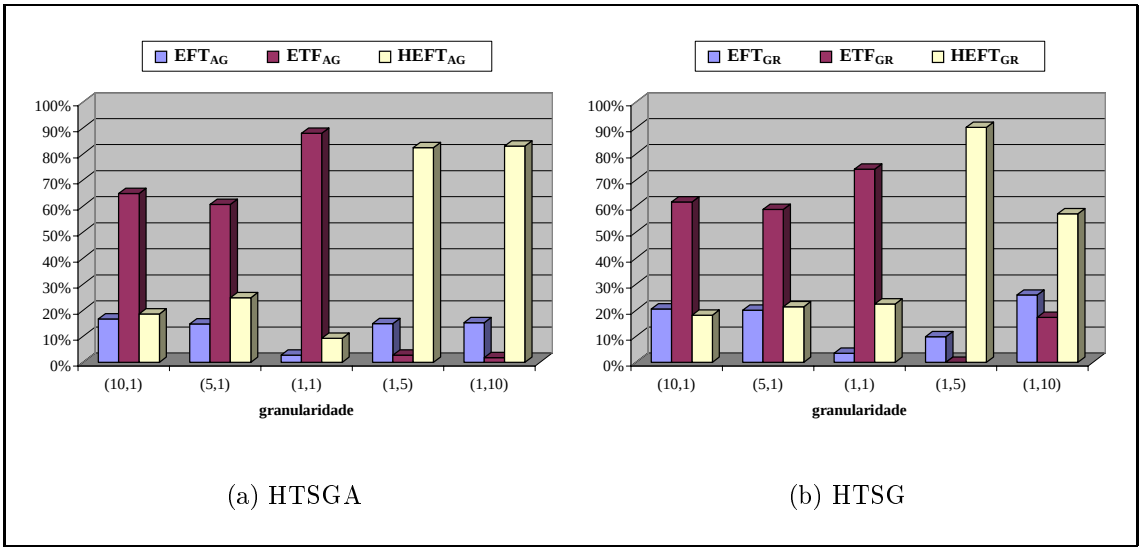
com valor de $\alpha = 0$ determina que o GRASP deve ser guloso na fase de construção das soluções, fazendo com que as respectivas heurísticas de construção embutidas (ETF_{GR} e HEFT_{GR}) assumam um comportamento similar aos de ETF e HEFT (veja Seção 4.2.2).

Uma segunda análise dos resultados foi realizada em termos do ganho percentual em relação ao *makespan* das soluções produzidas por HTSGA e HTSG quando comparado com as demais heurísticas. Neste experimento, observou-se o desempenho dos algoritmos quando submetidos a GADs com diferentes graus de granularidade. Os resultados podem ser vistos na Tabela 5.2, onde a primeira coluna apresenta os pares de fatores (me, mc) utilizados para perturbar, respectivamente, os pesos de execução e de comunicação de cada GAD. O restante da tabela é dividido em duas seções, onde em cada seção são comparados, respectivamente, o desempenho de HTSGA e HTSG com as demais heurísticas. Cada linha da tabela mostra, para o respectivo par de fatores (me, mc), o ganho percentual médio em relação ao *makespan* produzido por HTSGA/HTSG quando comparado ao das demais heurísticas. Finalmente, a última linha apresenta o ganho médio geral obtido por HTSGA/HTSG sobre cada uma das heurísticas confrontadas.

	HTSGA			**HTSG**		
(me,mc)	**ETF**	**HEFT**	**PSGA**	**ETF**	**HEFT**	**PSGA**
(10,1)	6,3%	1,5%	0,5%	6,2%	1,3%	0,3%
(5,1)	6,1%	4,1%	1,6%	5,9%	3,8%	1,3%
(1,1)	0,6%	21,3%	11,8%	0,8%	21,5%	12,1%
(1,5)	22,6%	32,5%	10,1%	19,6%	29,8%	6,6%
(1,10)	27,4%	32,6%	14,0%	22,2%	27,7%	7,8%
média	**12,6%**	**18,4%**	**7,6%**	**10,9%**	**16,8%**	**5,6%**

Tabela 5.2: Comparação do ganho percentual médio em relação ao *makespan* dos escalonamentos gerados para os GADs *diamante* para granularidades grossa e fina.

Um resultado importante que pode ser destacado na Tabela 5.2 é o desempenho superior de HTSGA quando submetido a GADs com granularidade fina, onde os pesos de comunicação dominam os pesos de execução. A responsável por este desempenho é a heurística HEFT_{AG}, já que desponta como a heurística de construção que mais vezes conseguiu produzir a melhor solução, conforme pode ser visto na Fi-



(a) HTSGA (b) HTSG

Figura 5.1: Comparação, em termos proporcionais, do número de vezes em que as heurísticas de construção de (a) HTSGA e (b) HTSG atingem a melhor solução final para GADs com topologia *diamante*.

gura 5.1(a). Nesta figura, é apresentado um gráfico que mostra, em termos relativos, o número de vezes em que cada uma das heurísticas de construção embutidas em HTSGA conseguiu produzir, para cada par de fatores de granularidade aplicados, a melhor solução fornecida pelo algoritmo.

Percebe-se claramente que, para GADs com granularidade fina, a heurística HEFT_{AG} é, na grande maioria dos casos, a heurística de construção utilizada por HTSGA a produzir melhores soluções. Este comportamento, deve-se à política de inserção da heurística HEFT_{AG} combinada com o mecanismo de busca implementado por HTSGA. Neste mecanismo, onde diversas seqüências de escalonamento são experimentadas, os altos custos de comunicação existentes nos grafos com esta granularidade, influenciam diretamente no desempenho de HEFT_{AG}. Isto porque, durante a construção de uma solução, existe uma maior probabilidade de surgirem espaços ociosos entre tarefas previamente escalonadas, espaços estes ocasionados pelos custos de comunicação interprocessadores. Como HEFT_{AG} procura utilizar estes espaços, é possível que algumas tarefas sejam escalonadas nestes locais, aproveitando a ociosidade do processador e, conseqüentemente, deixando de incrementar o *makespan* durante a construção do escalonamento.

Ainda sobre os resultados para granularidade fina, pode ser observado na Tabela 5.2 que os ganhos médios apresentados por HTSG, mostram que esta heurística também tirou proveito da política de inserção implementada por HEFT_{GR}. Esta afirmação pode ser comprovada pelos dados apresentados na Figura 5.1(b), que mostram o método de construção HEFT_{GR} como o que mais vezes conseguiu chegar à melhor solução produzida pelo algoritmo.

Todavia, este mesmo resultado quando confrontado com os ganhos de desempenho da Tabela 5.2, leva a uma constatação interessante: os maiores ganhos médios obtidos tanto por HTSGA quanto por HTSG são justamente sobre a heurística HEFT original. Este resultado mostra que não apenas a política de inserção das versões embutidas de HEFT (HEFT_{AG} e HEFT_{GR}) influencia na construção de bons escalonamentos, mas também a seqüência de atribuição de tarefas empregada. Conforme já discutido no capítulo anterior, o diferencial básico entre HEFT_{AG} /HEFT_{GR} e o HEFT original está relacionado com a forma com que as prioridades associadas a cada tarefa são atribuídas. Enquanto HEFT determina um único conjunto de prioridades relacionado com a instância, as versões embutidas associam diversas seqüências de prioridades através dos mecanismos de busca de suas respectivas metaheurísticas. Ao associar prioridades às tarefas, os algoritmos procuram na realidade determinar a importância de cada tarefa no processo de construção de um escalonamento. Neste contexto, a política de inserção surge como uma oportunidade de reavaliar a influência de cada tarefa durante a construção, permitindo que a ordem de escalonamento seja alterada. A capacidade de HEFT_{AG} e HEFT_{GR} poderem especificar novas seqüências de prioridades, eventualmente não distantes de um conjunto de prioridades ideal, combinada com o mecanismo de reordenação de tarefas possibilitado pela política de inserção, permite que estas heurísticas possam reavaliar a seqüência de escalonamento original, fazendo com que estes algoritmos procurem explorar adequadamente as características paralelas da aplicação e do sistema de computação.

Para ilustrar que a combinação da política de inserção com diferentes seqüências de escalonamento pode levar a bons resultados, são mostradas na Figura 5.2 duas soluções produzidas pelas heurísticas baseadas em HEFT. Os escalonamentos

apresentados foram obtidos a partir do diamante unitário da Figura 5.2(a), com 16 tarefas, submetido a um ambiente dual com 20/5 processadores. Os custos de execução e comunicação do grafo foram pertubados pelos fatores $(me, mc) = (1, 5)$, respectivamente. O escalonamento mostrado na Figura 5.2(b) foi produzido pela versão original de HEFT, que utiliza o *nível* como função de classificação de prioridades na seleção da próxima tarefa a ser escalonada, conforme discutido na Seção 3.2.2. Esta função faz com que as tarefas que possuem o mesmo valor de *nível* sejam selecionadas primeiro. Esta ordenação não permite que a heurística explore adequadamente o paralelismo do sistema, pois os custos de comunicação envolvidos fazem com que HEFT quase sempre escalone a tarefa corrente no mesmo processador em que seu predecessor que termina mais tarde está atribuído. Para uma melhor compreensão deste comportamento, considere um grafo $G = (T, E)$, com $(t_k, t_i) \in E$ onde t_i é a tarefa correntemente sendo escalonada e t_k o predecessor de t_i que termina sua execução mais tarde. HEFT só utilizará um novo processador $p(t_i) \neq p(t_k)$ quando a soma dos custos de execução das tarefas alocadas após t_k no processador $p(t_k)$ for maior que o custo de comunicação envolvido entre t_k e t_i.

Já o esquema de ordenação empregado pelas metaheurísticas propostas através de $\text{HEFT}_{\text{AG}}/\text{HEFT}_{\text{GR}}$ permite a atribuição de diferentes prioridades às tarefas, possibilitando que que outros processadores sejam utilizados, aproveitando a disponibilidade de recursos do sistema para uma maior minimização do *makespan*. Entretanto, é possível que não só a exploração de recursos seja suficiente para produzir um bom resultado, pois, assim como em HEFT, as prioridades de determinadas tarefas podem não ter sido devidamente avaliadas inicialmente. Por exemplo, na Figura 5.2(c), após o escalonamento da tarefa t_{14}, surge um espaço ocioso no processador p_2 entre as tarefas t_{12} e t_{14}. Seguindo a seqüência de atribuição original exibida na Figura 5.2(c) logo abaixo do diagrama, a próxima tarefa a ser escalonada é t_{15}, que seria atribuída logo após t_{14}. Devido à política de inserção, o algoritmo detecta uma locação mais vantajosa para t_{15} no espaço ocioso surgido na etapa anterior, modificando a ordem de escalonamento definida originalmente e, conseqüentemente, não incrementando o *makespan*.

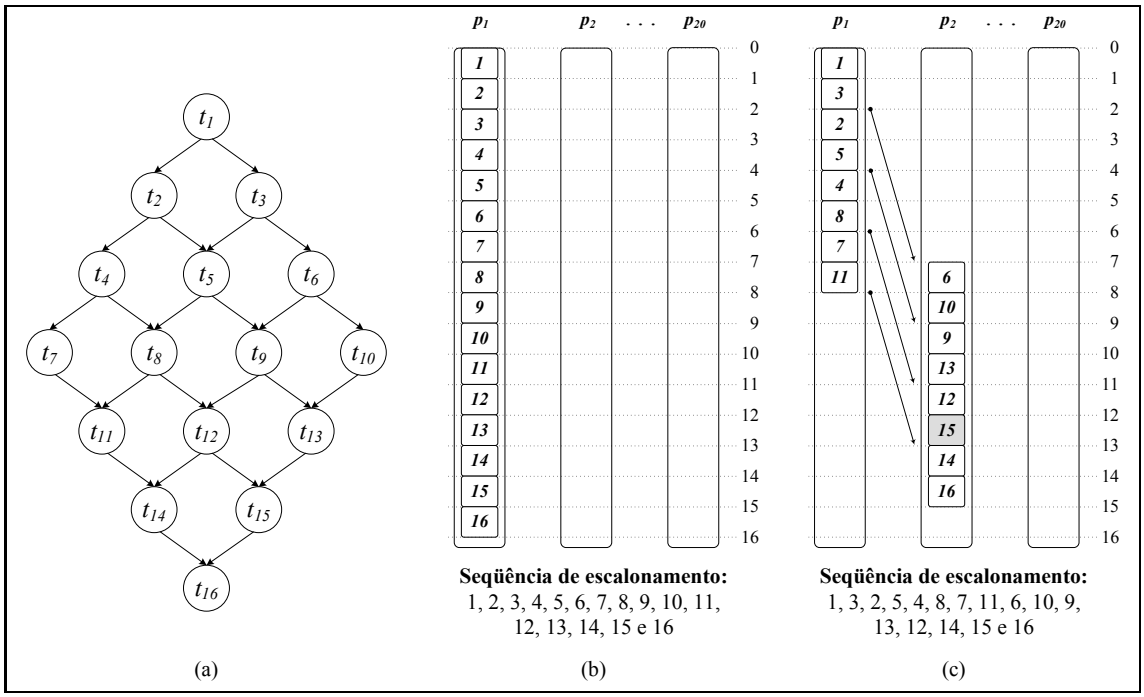


Figura 5.2: Escalonamentos gerados por (b) HEFT e (c) HEFT_{AG} a partir de um (a) diamante com 16 tarefas, perturbado com os fatores $(me, mc) = (1, 5)$, em um ambiente dual com 20/5 processadores.

Analisando agora os dados da Tabela 5.2 para grafos com granularidade grossa, com $(me, mc) = \{(10, 1), (5, 1)\}$, verifica-se que HTSGA e HTSG também conseguem melhorar os resultados, porém com um ganho menor do que quando comparados com os resultados para grafos com granularidade fina. O desempenho médio de HTSGA e HTSG em relação a cada heurística é praticamente o mesmo, com o AG destacando-se ligeiramente.

Em relação ao uso das heurísticas de construção embutidas, os algoritmos novamente apresentam resultados similares, com as respectivas versões da heurística ETF (ETF_{AG} e ETF_{GR}) figurando como as que mais vezes alcançaram os melhores resultados para os grafos de granularidade grossa, como pode ser observado nos gráficos da Figura 5.1. Em GADs com baixos pesos de comunicação, os fatores mais influentes no comprimento do *makespan* são, obviamente, os custos de execução das tarefas. O bom desempenho das versões embutidas de ETF é devido a uma característica dos escalonamentos gerados por estas heurísticas, que aglomeram tarefas de um mesmo caminho do grafo em um mesmo processador. Como será

visto posteriormente, este comportamento também é identificado em outras classes de grafos.

Árvores Binárias Invertidas

O conjunto de grafos com topologia árvore binária invertida (*in-tree*) analisado possui 6 instâncias com número de tarefas variando entre 15 e 511. As instâncias foram escalonadas nos 12 ambientes computacionais não restritos já definidos, onde cada grafo teve seus pesos perturbados de forma a produzir instâncias com granularidades fina e grossa, gerando um total de 360 diferentes casos.

A contagem dos casos em que o *makespan* produzido por cada heurística é menor, igual ou pior que o das demais pode ser vista na Tabela 5.3. Novamente, as heurísticas propostas conseguem se sobressair em relação às heurísticas da literatura, apresentando soluções com *makespan* igual ou superior na quase totalidade dos casos. Quando comparadas com PSGA, a heurística HTSGA obtém resultados iguais ou melhores em 100,0% dos casos, enquanto HTSG atinge a marca de 98,9%.

		ETF		**HEFT**		**PSGA**		**HTSGA**		**HTSG**	
ETF	>			191	53,1%	212	58,9%	225	62,5%	272	75,6%
	=			154	42,8%	148	41,1%	135	37,5%	88	24,4%
	<			15	4,2%	0	0,0%	0	0,0%	0	0,0%
HEFT	>	15	4,2%			185	51,4%	210	58,3%	252	70,0%
	=	154	42,8%			175	48,6%	150	41,7%	108	30,0%
	<	191	53,1%			0	0,0%	0	0,0%	0	0,0%
PSGA	>	0	0,0%	0	0,0%			118	32,8%	128	35,6%
	=	148	41,1%	175	48,6%			242	67,2%	228	63,3%
	<	212	58,9%	185	51,4%			0	0,0%	4	1,1%
HTSGA	>	0	0,0%	0	0,0%	0	0,0%			66	18,3%
	=	135	37,5%	150	41,7%	242	67,2%			210	58,3%
	<	225	62,5%	210	58,3%	118	32,8%			84	23,3%
HTSG	>	0	0,0%	0	0,0%	4	1,1%	84	23,3%		
	=	88	24,4%	108	30,0%	228	63,3%	210	58,3%		
	<	272	75,6%	252	70,0%	128	35,6%	66	18,3%		

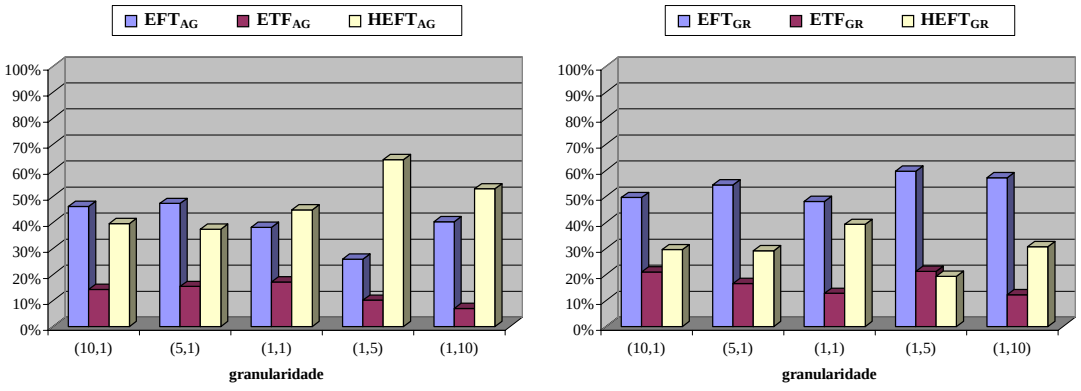
Tabela 5.3: Comparação de desempenho em termos de *makespans* piores (>), iguais (=) e melhores (<) para escalonamentos gerados em *ambientes não restritos* a partir de GADs *árvore binária invertida* para um total de 360 casos.

Com base nos dados expostos na Tabela 5.4, foi realizada a análise do ganho médio dos *makespans* obtidos por cada uma das heurísticas propostas em relação às da literatura. Desta vez, para grafos de granularidade fina, a heurística HTSG obteve ganhos superiores em relação a HTSGA. Neste caso, o principal responsável por este desempenho foi o procedimento de busca local TASK [59] utilizado pelo GRASP. Conforme discutido na Seção 3.2.4, este procedimento procura melhorar a qualidade de um escalonamento movimentando as tarefas já atribuídas entre os processadores do ambiente de computação. O mecanismo de busca utilizado por TASK é baseado no comprimento do caminho crítico escalonado, o qual determina o *makespan* do escalonamento. A cada passo, o algoritmo simula a atribuição de uma tarefa nos processadores do sistema, verificando se tal movimento reduz o comprimento do caminho crítico do escalonamento e, conseqüentemente, o *makespan*.

	HTSGA			**HTSG**		
(me,mc)	**ETF**	**HEFT**	**PSGA**	**ETF**	**HEFT**	**PSGA**
(10,1)	12,0%	1,4%	0,1%	12,2%	1,6%	0,3%
(5,1)	11,6%	1,3%	0,3%	11,7%	1,4%	0,4%
(1,1)	4,4%	3,8%	1,1%	4,1%	3,4%	0,8%
(1,5)	15,1%	8,0%	4,6%	22,0%	14,9%	11,4%
(1,10)	18,2%	8,9%	6,0%	28,4%	18,9%	16,0%
média	**12,3%**	**4,7%**	**2,4%**	**15,7%**	**8,0%**	**5,8%**

Tabela 5.4: Comparação do ganho percentual médio em relação ao *makespan* dos escalonamentos gerados para os GADs *árvore binária invertida* para granularidades grossa e fina.

Para grafos desta topologia, a busca local TASK tem conseguido melhorar de maneira significativa os escalonamentos produzidos pelas heurísticas de construção de HTSG, chegando, em alguns casos, a decrementos da ordem de 50% no valor do *makespan* em relação à solução obtida na fase de construção do algoritmo. Este bom desempenho deve-se principalmente à característica das árvores *in-tree* de apresentarem um número elevado de tarefas livres em relação ao total de tarefas do grafo. Outra caraterística que influencia este desempenho está relacionada com as heurísticas de construção de HTSG, que tendem a escalonar as tarefas livres em processadores distintos. Este comportamento, embora adequado em outros casos, consome rapidamente os recursos do sistema devido ao grande número de ta-



(a) HTSGA (b) HTSG

Figura 5.3: Comparação, em termos proporcionais, do número de vezes em que as heurísticas de construção de (a) HTSGA e (b) HTSG atingem a melhor solução final para GADs com topologia *árvore binária invertida*.

refas disponíveis inicialmente. Observe que os escalonamentos produzidos por TASK apresentam um número reduzido de processadores utilizados em relação ao escalonamento original. Isto porque, a busca local TASK oferece uma nova oportunidade de reavaliar o paralelismo de uma aplicação, trazendo mais tarefas para dentro de espaços ociosos que antes eram consumidos com eventos de comunicação.

A Figura 5.4 ilustra o resultado de uma busca efetuada por TASK. Neste exemplo, foi utilizado um ambiente dual com 20 processadores, onde os cinco primeiros processadores são mais rápidos que os demais. Os escalonamentos apresentados foram gerados a partir do GAD unitário da Figura 5.4(a), onde os pesos de execução e comunicação foram perturbados com os fatores (1,5), respectivamente. O escalonamento na Figura 5.4(b) foi gerado por HTSG utilizando a heurística $\mathrm{EFT_{GR}}$ como método de construção. Observe como a heurística tira proveito da disponibilidade de recursos do sistema, escalonando as tarefa livres em diversos processadores, gerando como conseqüência, diversos eventos de comunicação. A Figura 5.4(c) mostra o escalonamento produzido pela busca local TASK a partir do escalonamento da Figura 5.4(b). A independência de um grande número de tarefas permite que a busca local TASK reavalie o escalonamento atual alocando tarefas nos espaços ociosos gerados pelos eventos de comunicação. Além disso, o fato de alguns processadores estarem

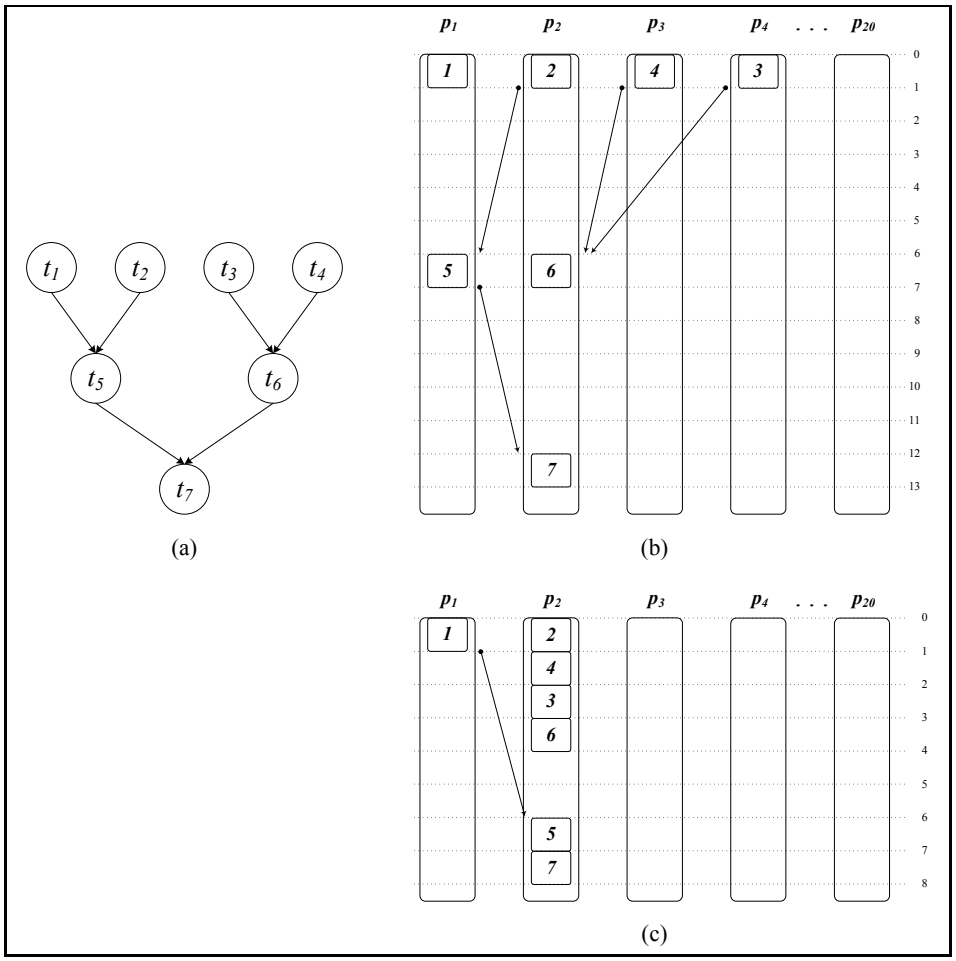


Figura 5.4: (a) Árvore binária invertida com 7 tarefas; (b) escalonamento foi obtido por uma heurística de construção de HTSG sobre um sistema dual com 20/5 processadores com $(me, mc) = (1, 5)$ e (c) escalonamento produzido por TASK a partir de (b).

ocupados somente com tarefas independentes, produz uma mobilidade que permite que busca local reduza o número de processadores utilizados pelo escalonamento. Neste exemplo, TASK reduziu o *makespan* do escalonamento original de 13 para 8 unidades de tempo (38,5%) consumindo a metade do número de processadores originalmente utilizados.

No entanto, uma análise mais atenta no escalonamento da Figura 5.4(c), mostra que o mecanismo de busca de TASK não é sempre eficiente. Apesar de ter reduzido de maneira significativa tanto o *makespan* quanto o número de processadores do escalonamento, verifica-se que neste exemplo, todas as tarefas poderiam

ter sido alocadas em um mesmo processador com um custo total de execução ainda menor. Isto indica que a seqüência com que TASK movimenta as tarefas pode ser importante para o desempenho deste procedimento de busca local.

Árvores Binárias

Foram analisados grafos do tipo árvore binária (*out-tree*) com número de tarefas variando entre 15 e 511, totalizando 6 instâncias. Cada grafo foi submetido aos 12 ambientes computacionais não restritos descritos anteriormente, onde, para cada ambiente a granularidade dos GADs foi variada de acordo com os cinco pares de fatores (me, mc) previamente definidos, totalizando 360 casos.

		ETF		**HEFT**		**PSGA**		**HTSGA**		**HTSG**	
	>			34	9,4%	215	59,7%	216	60,0%	236	65,6%
ETF	=			134	37,2%	107	29,7%	144	40,0%	124	34,4%
	<			192	53,3%	38	10,6%	0	0,0%	0	0,0%
	>	192	53,3%			260	72,2%	275	76,4%	277	76,9%
HEFT	=	134	37,2%			100	27,8%	85	23,6%	83	23,1%
	<	34	9,4%			0	0,0%	0	0,0%	0	0,0%
	>	38	10,6%	0	0,0%			111	30,8%	78	21,7%
PSGA	=	107	29,7%	100	27,8%			249	69,2%	265	73,6%
	<	215	59,7%	260	72,2%			0	0,0%	17	4,7%
	>	0	0,0%	0	0,0%	0	0,0%			37	10,3%
HTSGA	=	144	40,0%	85	23,6%	249	69,2%			253	70,3%
	<	216	60,0%	275	76,4%	111	30,8%			70	19,4%
	>	0	0,0%	0	0,0%	17	4,7%	70	19,4%		
HTSG	=	124	34,4%	83	23,1%	265	73,6%	253	70,3%		
	<	236	65,6%	277	76,9%	78	21,7%	37	10,3%		

Tabela 5.5: Comparação de desempenho em termos de *makespans* piores (>), iguais (=) e melhores (<) para escalonamentos gerados em *ambientes não restritos* a partir de GADs *árvore binária* para um total de 360 casos.

A Tabela 5.5 apresenta a contagem dos resultados melhores, iguais e piores obtidos por cada heurística quando comparada com as demais heurísticas analisadas. Verifica-se que, em relação às heurísticas da literatura, HTSGA e HTSG atingem resultados iguais ou superiores em quase todos os casos. A única exceção ocorre quando o desempenho da heurística HTSG é comparado com o de PSGA, onde esta última obtém maior ganho médio em 4,7% dos casos. Analisando os 17 casos

em que PSGA apresentou-se melhor, foi verificado que, dentre estes, 15 envolviam grafos de granularidade fina. Como pode ser observado na Figura 5.5(b), a heurística EFT_{GR} teve um papel importante no desempenho de HTSG para grafos com esta granularidade. Isto mostra que a busca realizada por PSGA foi mais eficiente que a de HTSG nestes casos, já que PSGA executa um maior número de iterações utilizando a política de escalonamento de EFT, possibilitando o experimento de um maior número de combinações de prioridade.

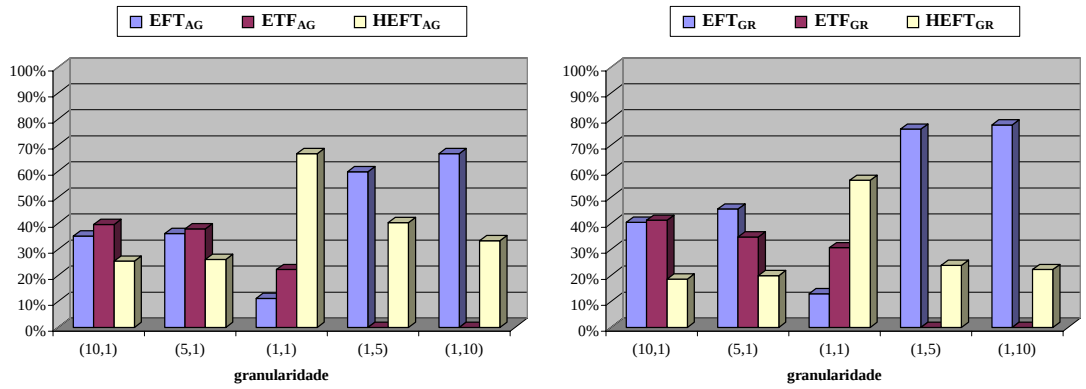
	HTSGA			**HTSG**		
(me,mc)	**ETF**	**HEFT**	**PSGA**	**ETF**	**HEFT**	**PSGA**
(10,1)	2,1%	2,1%	0,3%	2,2%	2,3%	0,5%
(5,1)	2,7%	2,5%	0,4%	2,9%	2,7%	0,6%
(1,1)	2,1%	5,1%	0,4%	2,8%	5,8%	1,2%
(1,5)	19,4%	26,2%	1,7%	18,1%	25,0%	0,1%
(1,10)	26,3%	28,8%	2,7%	23,8%	26,4%	-0,8%
média	**10,5%**	**13,0%**	**1,1%**	**10,0%**	**12,4%**	**0,3%**

Tabela 5.6: Comparação do ganho percentual médio em relação ao *makespan* dos escalonamentos gerados para os GADs *árvore binária* para granularidades grossa e fina.

Para os grafos com granularidade grossa, $(me, mc) \in \{(10,1),(5,1)\}$, todas as heurísticas de construção embutidas apresentam desempenho relativo semelhante em ambas as propostas. A análise nos escalonamentos produzidos mostrou que os métodos de construção estão chegando a soluções onde tarefas pertencentes a um mesmo caminho do GAD são alocadas em um mesmo processador. Entende-se por caminho uma seqüência de tarefas onde a cada duas tarefas, uma apresenta uma relação de dependência com a outra. Por exemplo, no grafo da Figura 5.6(a) as tarefas t_2, t_5 e t_{10} representam um caminho.

Foi observado que as heurísticas baseadas em ETF tendem a aglomerar caminhos mais longos, enquanto as baseadas em EFT e HEFT agrupam caminhos mais curtos. Dependendo do número de processadores rápidos existentes no sistema, cada um destes esquemas pode ser adequado ou não. Num sistema com um grande número de processadores rápidos (em relação ao total de processadores), a alocação de caminhos mais longos mostrou-se mais adequada. Já para sistemas com uma quantidade menor de processadores rápidos, a aglomeração de caminhos mais curtos

gerou melhores soluções.



(a) HTSGA (b) HTSG

Figura 5.5: Comparação, em termos proporcionais, do número de vezes em que as heurísticas de construção de (a) HTSGA e (b) HTSG atingem a melhor solução final para GADs com topologia *árvore binária*.

Numa árvore binária, o agrupamento de tarefas pertencentes a um caminho num mesmo processador, faz com que os descendentes de tarefas deste caminho que estejam alocadas em outros processadores iniciem sua execução mais tarde. Quanto mais longo o caminho, mais tarefas descendentes terão seu início atrasado. Quando os processadores mais lentos começam a ser utilizados, o custo de execução das tarefas aumenta, introduzindo um novo fator de atraso no início de execução das tarefas ainda não alocadas.

A Figura 5.6 procura ilustrar este comportamento. No exemplo, são apresentados escalonamentos produzidos a partir da árvore binária da Figura 5.6(a) utilizando os fatores $(me, mc) = (5, 1)$. O sistema alvo foi um ambiente distribuído dual com 20/5 processadores. Por conveniência, as setas indicando os eventos de comunicação nos escalonamentos foram omitidas para permitir uma melhor visualização dos diagramas.

No escalonamento da Figura 5.6(b), produzido por ETF_{AG}, as tarefas em destaque indicam caminhos do grafo. Observe que esta heurística, seguindo a tendência da versão original, conseguiu aglomerar caminhos mais longos num mesmo

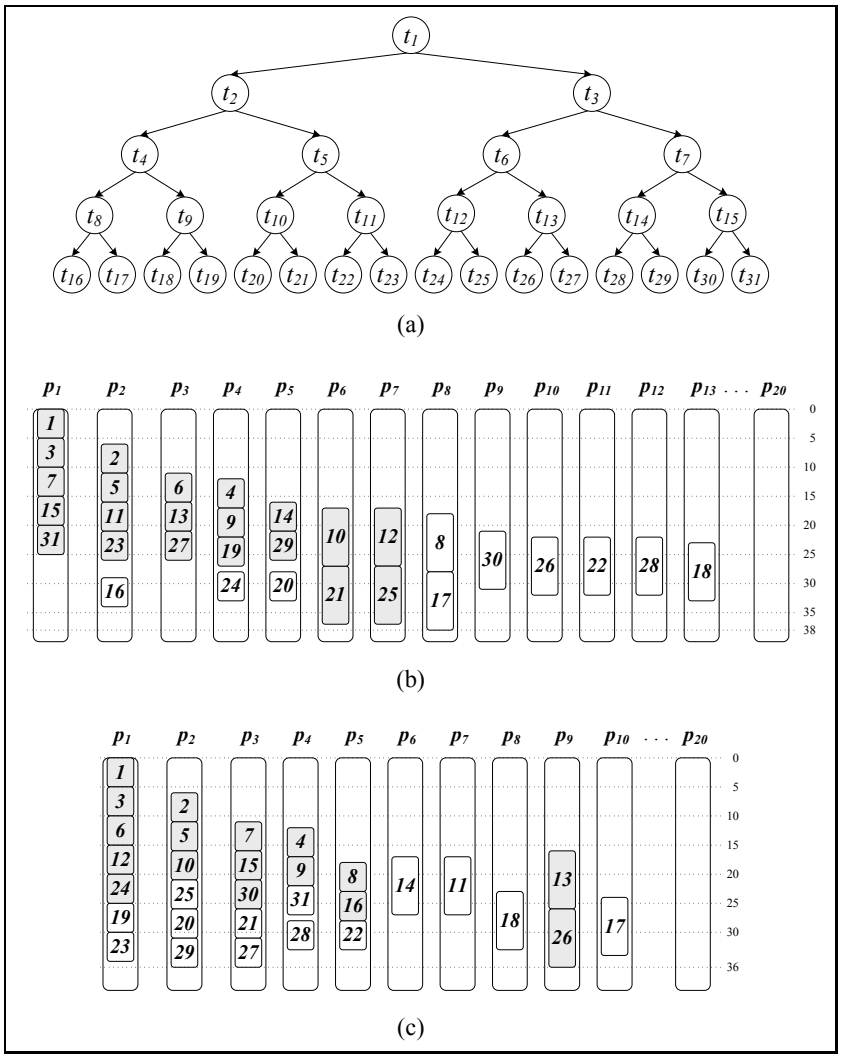


Figura 5.6: (a) Árvore binária com 31 tarefas; (b) escalonamento obtido por $\mathrm{ETF}_{\mathrm{AG}}$ sobre um sistema dual com 20/5 processadores com $(me, mc) = (5, 1)$ e (c) escalonamento obtido por $\mathrm{EFT}_{\mathrm{AG}}$ sob no mesmo sistema.

processador. Contudo isto não foi suficiente para produzir um bom resultado, já que a heurística $\mathrm{EFT}_{\mathrm{AG}}$ conseguiu um escalonamento de melhor qualidade reduzindo o *makespan* e o número de processadores utilizados. Isto porque, os agrupamentos mais curtos de $\mathrm{EFT}_{\mathrm{AG}}$ obtidos durante o escalonamento, permitiram uma alocação mais propícia de tarefas independentes, gerando uma maior sobreposição de comunicação e computação.

Randômicos

Adicionalmente, foram testadas 21 instâncias de grafos gerados aleatoriamente [43] com número de tarefas variando entre 80 e 546. Novamente, cada instância foi submetida aos 12 ambientes computacionais previamente definidos, onde cada grafo teve sua granularidade perturbada pelos fatores (me, mc) conforme já descrito, resultando num total de 1.260 casos avaliados.

A Tabela 5.7 apresenta os resultados da contagem de casos em que o *makespan* de cada heurística foi melhor, igual ou pior quando comparado com o das demais. Mais uma vez os algoritmos propostos apresentam resultados melhores ou equivalentes que os das heurísticas da literatura em praticamente todos os casos. Quando comparadas somente com PSGA, as heurísticas HTSGA e HTSG contabilizam empate ou sucesso em 100,0% e 99,4% dos casos, respectivamente.

		ETF		**HEFT**		**PSGA**		**HTSGA**		**HTSG**	
	>			1028	81,6%	1212	96,2%	1249	99,1%	1259	99,9%
ETF	=			67	5,3%	23	1,8%	11	0,9%	1	0,1%
	<			165	13,1%	25	2,0%	0	0,0%	0	0,0%
	>	165	13,1%			811	64,4%	916	72,7%	943	74,8%
HEFT	=	67	5,3%			379	30,1%	344	27,3%	317	25,2%
	<	1028	81,6%			70	5,6%	0	0,0%	0	0,0%
	>	25	2,0%	70	5,6%			449	35,6%	575	45,6%
PSGA	=	23	1,8%	379	30,1%			811	64,4%	678	53,8%
	<	1212	96,2%	811	64,4%			0	0,0%	7	0,6%
	>	0	0,0%	0	0,0%	0	0,0%			241	19,1%
HTSGA	=	11	0,9%	344	27,3%	811	64,4%			799	63,4%
	<	1249	99,1%	916	72,7%	449	35,6%			220	17,5%
	>	0	0,0%	0	0,0%	7	0,6%	220	17,5%		
HTSG	=	1	0,1%	317	25,2%	678	53,8%	799	63,4%		
	<	1259	99,9%	943	74,8%	575	45,6%	241	19,1%		

Tabela 5.7: Comparação de desempenho em termos de *makespans* piores (>), iguais (=) e melhores (<) para escalonamentos gerados em *ambientes não restritos* a partir de GADs *randômico* para um total de 1260 casos.

Na Tabela 5.8 são apresentadas as vantagens percentuais obtidas pelos *makespans* produzidos pelas propostas em relação às demais heurísticas. Observando o desempenho médio geral das heurísticas propostas em relação a cada heurística da literatura, verifica-se que tanto HTSGA quanto HTSG apresentam desempenhos

semelhantes, com o GRASP se mostrando ligeiramente mais eficiente. Por exemplo, tomando ETF como referência, HTSGA consegue um ganho médio geral de 21,8% sobre os resultados obtidos por esta heurística, enquanto HTSG atinge 22,0% em média, uma diferença de 0,2 ponto percentual. Estendendo esta comparação às outras heurísticas aqui analisadas, verifica-se que esta diferença nas médias nunca ultrapassa 0,3 ponto percentual.

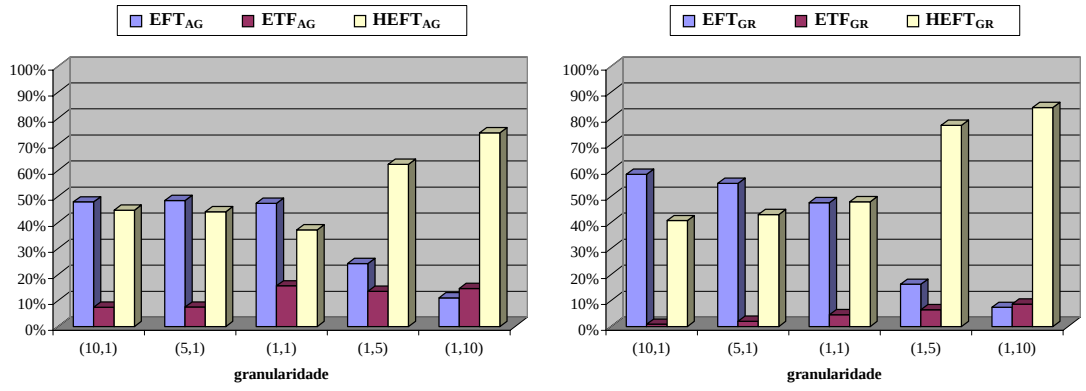
	HTSGA			**HTSG**		
(me,mc)	**ETF**	**HEFT**	**PSGA**	**ETF**	**HEFT**	**PSGA**
(10,1)	21,0%	0,9%	0,1%	21,1%	1,0%	0,2%
(5,1)	21,8%	1,2%	0,1%	21,9%	1,3%	0,2%
(1,1)	26,9%	17,8%	1,0%	28,4%	19,3%	3,0%
(1,5)	22,1%	17,4%	4,7%	22,2%	17,4%	4,7%
(1,10)	17,4%	9,0%	6,9%	16,6%	8,0%	5,9%
média	**21,8%**	**9,2%**	**2,5%**	**22,0%**	**9,4%**	**2,8%**

Tabela 5.8: Comparação do ganho percentual médio em relação ao *makespan* dos escalonamentos gerados para os GADs *randômico* para granularidades grossa e fina.

Uma análise nos resultados obtidos por HTSG, mostra que dos 1.260 casos com GADs randômicos estudados, em mais de 70,0% deles a busca local TASK figura como o recurso que permitiu ao algoritmo produzir sua melhor solução. Este fato indica que, na maioria dos casos, a solução produzida durante a fase de construção de HTSG foi melhorada durante a fase de busca local.

O desempenho de cada heurística de construção embutida em HTSGA e HTSG pode ser observado na Figura 5.7, que mostra em termos relativos, quantas vezes cada heurística conseguiu atingir a melhor solução retornada por seu respectivo algoritmo. Uma comparação entre os dois gráficos, mostra que as heurísticas embutidas apresentam um desempenho similar para todas as granularidades avaliadas.

Focando os casos para GADs de granularidade grossa, os gráficos da Figura 5.7 mostram que, em ambas as heurísticas propostas, as versões embutidas de EFT e HEFT apresentam desempenho relativo semelhante. Este resultado indica que a política de inserção presente nas versões embutidas de HEFT, não teve um impacto importante na construção das soluções, fazendo com que EFT_{AG}/HEFT_{AG} e



(a) HTSGA (b) HTSG

Figura 5.7: Comparação, em termos proporcionais, do número de vezes em que as heurísticas de construção de (a) HTSGA e (b) HTSG atingem a melhor solução final para GADs com topologia *randômica*.

EFT_{GR}/HEFT_{GR} apresentassem um comportamento similar. Isto ocorre possivelmente porque a diferença básica entre estas heurísticas é a política de inserção empregada pelas versões embutidas de HEFT. Um resultado que reforça esta hipótese é o ganho modesto, considerando os grafos com granularidade grossa, das propostas sobre a heurística PSGA (veja Tabela 5.8), que utiliza EFT como heurística de construção. Uma vez que as versões embutidas de EFT e HEFT comportam-se de maneira similar nas heurísticas propostas, a maioria das iterações de cada algoritmo foi executada com uma versão de EFT, onde o resultado foi um desempenho quase equivalente ao de PSGA. Ou seja, para esta granularidade, é como se as heurísticas propostas executassem $\frac{2}{3}$ de suas iterações com as respectivas versões de EFT e o restante com as de ETF, uma vez que a política de inserção parece não influenciar no resultado.

Já para os grafos com granularidade fina, os gráficos da Figura 5.7 destacam as versões embutidas de HEFT em ambas as heurísticas propostas. No caso de HTSGA, este resultado pode ser atribuído à política de inserção empregada por HEFT_{AG} que se favorece dos altos custos de comunicação característicos de GADs desta granularidade. Este fato pode ser comprovado pelo desempenho pouco significativo das heurísticas EFT_{AG} e ETF_{AG}, de acordo com a Figura 5.7(a). Já no caso de

HTSG, o desempenho alcançado foi devido à combinação de HEFT_{AG} com a busca local TASK. De fato, dos 504 casos para grafos de granularidade fina, HEFT_{AG} aparece como responsável pela produção da melhor solução em 463 deles (91,8%), dos quais 290 (57,5%) foram em obtidos em combinação com a busca TASK.

5.2.2 GADS Não Unitários

Para analisar o desempenho das heurísticas propostas quando submetidas a GADs não unitários, foi utilizada uma classe de grafos de dimensões variadas que representam a paralelização do procedimento matemático conhecido como eliminação de Gauss [108].

A princípio, o conjunto de grafos *Peer Set Graphs* (PSG) [10] também havia sido selecionado para representar os GADs não unitários nesta análise. Contudo, o pequeno número de tarefas existentes nestes GADs em relação às dimensões das instâncias de ambientes, faz com que as heurísticas cheguem sempre à mesma solução para todos os sistemas não restritos avaliados. Por este motivo, os grafos PSG serão analisados somente quando submetidos a ambientes restritos, conforme será visto adiante neste capítulo.

Eliminação de Gauss

A eliminação de Gauss é um método utilizado para solucionar sistemas de equações lineares que consiste em transformar um sistema de equações em uma matriz triangular superior equivalente. Em um sistema com n variáveis, o algoritmo seqüencial pode ser paralelizado resultando em um GAD com $\frac{(n^2+n)}{2} - 1$ tarefas, cujo procedimento é descrito em [108].

Uma característica das instâncias desta classe é que os GADs possuem granularidade inerentemente grossa, já que os pesos de execução dominam os pesos de comunicação. Desta forma, os experimentos foram realizados pertubando apenas os pesos de comunicação com os fatores (1,1), (1,5) e (1,10). Os grafos representantes desta classe possuem o número de tarefas variando entre 9 e 1.175, num total de 12

instâncias. Cada GAD foi perturbado com os três pares de fatores (me, mc) descritos e então submetidos a cada um dos 12 ambientes computacionais do tipo dual previamente definidos, totalizando 432 casos.

		ETF		HEFT		PSGA		HTSGA		HTSG	
	>			408	94,4%	420	97,2%	420	97,2%	420	97,2%
ETF	=			12	2,8%	12	2,8%	12	2,8%	12	2,8%
	<			12	2,8%	0	0,0%	0	0,0%	0	0,0%
	>	12	2,8%			228	52,8%	245	56,7%	241	55,8%
HEFT	=	12	2,8%			170	39,4%	187	43,3%	191	44,2%
	<	408	94,4%			34	7,9%	0	0,0%	0	0,0%
	>	0	0,0%	34	7,9%			100	23,1%	79	18,3%
PSGA	=	12	2,8%	170	39,4%			332	76,9%	324	75,0%
	<	420	97,2%	228	52,8%			0	0,0%	29	6,7%
	>	0	0,0%	0	0,0%	0	0,0%			24	5,6%
HTSGA	=	12	2,8%	187	43,3%	332	76,9%			318	73,6%
	<	420	97,2%	245	56,7%	100	23,1%			90	20,8%
	>	0	0,0%	0	0,0%	29	6,7%	90	20,8%		
HTSG	=	12	2,8%	191	44,2%	324	75,0%	318	73,6%		
	<	420	97,2%	241	55,8%	79	18,3%	24	5,6%		

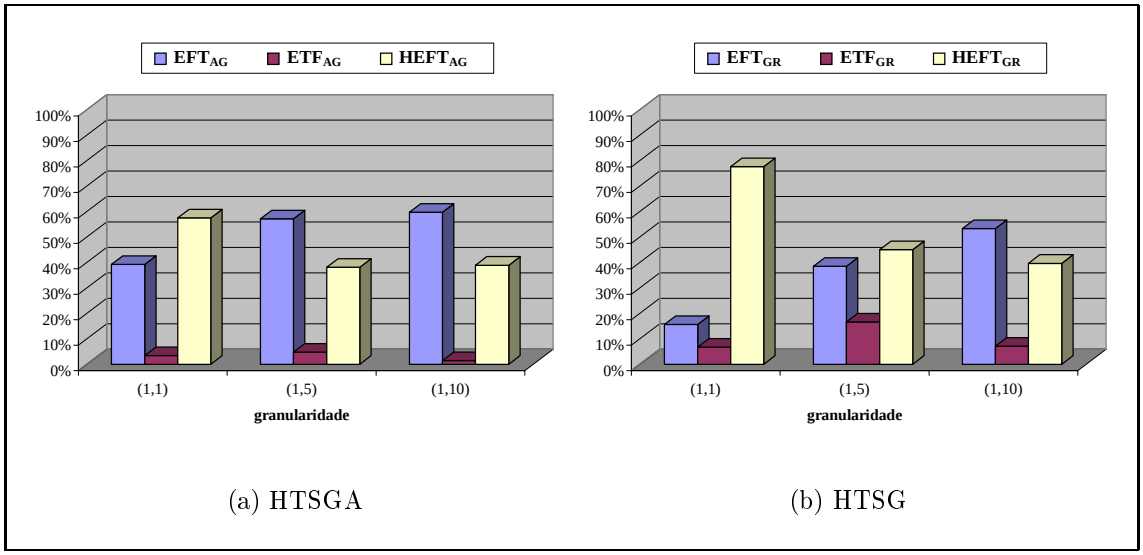
Tabela 5.9: Comparação de desempenho em termos de *makespans* piores (>), iguais (=) e melhores (<) para escalonamentos gerados em *ambientes não restritos* a partir de GADs *eliminação de Gauss* para um total de 432 casos.

Inicialmente foram contabilizados os casos em que as heurísticas propostas atingem valores de *makespan* piores, iguais e melhores que as demais heurísticas da literatura. O resultado pode ser visto na Tabela 5.9 que mostra as heurísticas propostas com resultados iguais ou superiores na grande maioria dos casos. Uma comparação das propostas com a heurística PSGA mostra que HTSGA e HTSG chegam a resultados iguais ou superiores em 100,0% e 93,3% dos casos, respectivamente.

A Tabela 5.10 mostra o ganho sobre o *makespan* obtido pelas heurísticas propostas quando comparadas com as demais heurística analisadas, de acordo com cada fator de granularidade empregado. Uma análise nos ganhos médios gerais de HTSGA e HTSG mostra que estas heurísticas apresentam desempenho médio relativo semelhante em relação a cada heurística confrontada. Os maiores ganhos ocorrem sobre a heurística ETF, com HTSGA atingindo uma redução média de 21,3% do *makespan* gerado por esta heurística, enquanto HTSG reduz em 21,2%

(me,mc)	HTSGA			HTSG		
	ETF	**HEFT**	**PSGA**	**ETF**	**HEFT**	**PSGA**
(1,1)	18,5%	0,1%	0,4%	18,5%	0,1%	0,4%
(1,5)	22,8%	0,8%	0,4%	22,9%	0,8%	0,5%
(1,10)	22,5%	4,3%	0,7%	22,3%	4,0%	0,4%
média	**21,3%**	**1,7%**	**0,5%**	**21,2%**	**1,6%**	**0,4%**

Tabela 5.10: Comparação do ganho percentual médio em relação ao *makespan* dos escalonamentos gerados para os GADs *eliminação de Gauss* para granularidades grossa e fina.



(a) HTSGA (b) HTSG

Figura 5.8: Comparação, em termos proporcionais, do número de vezes em que as heurísticas de construção de (a) HTSGA e (b) HTSG atingem a melhor solução final para GADs *eliminação de Gauss*.

em média. O mau desempenho de ETF para este tipo de GAD também se repete em suas versões embutidas nas heurísticas propostas, conforme pode ser visto na Figura 5.8. Os gráficos mostram que tanto ETF_{AG} quanto ETF_{GR} foram os métodos de construção que menos vezes conseguiram produzir a melhor solução de seus respectivos algoritmos.

Foi observado que ETF não consegue identificar as tarefas que mais influenciam na determinação do *makespan* de um escalonamento. Isto porque, ao escalonar uma tarefa com muitos sucessores, o algoritmo explora esta característica procurando alocar as tarefas livres em processadores distintos, não avaliando adequadamente durante o escalonamento o impacto dos custos de comunicação oriundos

destas tarefas em iterações futuras. A conseqüência deste comportamento é o atraso no início da execução de tarefas sucessoras devido aos custos de comunicação oriundos de tarefas que enviam um número elevado de mensagens.

5.3 Ambientes Computacionais Restritos

Um outro conjunto de testes foi realizado como o objetivo de observar o comportamento das heurísticas propostas quando submetidas a ambientes computacionais com número de processadores restritos. Com este experimento, é possível observar a eficiência das heurísticas na minimização do *makespan* quando existe limitação no número de recursos disponíveis no sistema. Desta forma, foram definidas três dimensões (m) de ambientes computacionais do tipo dual, cada qual com 2, 4 e 6 processadores. Em cada ambiente gerado, existem q processadores rápidos, com fator de heterogeneidade $h() = 1$, e $m - q$ processadores lentos, com $h() = 2$. Para cada dimensão, o valor de q é variado conforme descrito na Tabela C.1, produzindo um total de 9 ambientes computacionais distintos.

Dimensão m	**procs rápidos** q	**procs lentos** $m - q$
2	1	1
2	2	0
4	1	3
4	2	2
4	4	0
6	1	5
6	2	4
6	4	2
6	6	0

Tabela 5.11: Instâncias de ambientes computacionais do tipo dual restrito.

5.3.1 GADS Unitários

Para os ambientes restritos, foram submetidos inicialmente grafos unitários do tipo diamante e randômicos. Para observar o impacto da granularidade dos GADs

de entrada sobre o desempenho das heurísticas, os pesos de computação e execução dos grafos foram pertubados por fatores (me, mc), respectivamente. Os testes foram então executados com grafos de granularidade grossa, utilizando fatores (10,1), (5,1) e (1,1), e granularidade fina, com os fatores (1,5) e (1,10).

Diamantes

A classe de grafos com topologia diamante é composta por 9 instâncias com número de tarefas variando entre 25 e 1024. Cada grafo teve seus pesos de computação e comunicação perturbados pelos fatores (me, mc) já descritos, sendo submetidos a cada um dos 9 ambientes computacionais restritos previamente determinados, gerando um total de 405 casos distintos.

O número de casos em que cada heurística atingiu *makespan* melhor, igual ou pior que as demais é mostrado na Tabela 5.13, onde pode ser observado que as heurísticas propostas continuam igualando ou superando os resultados obtidos pelas demais heurísticas. Em relação a PSGA, HTSGA mostra resultado igual ou superior em todos os casos, enquanto HTSGA atinge a marca de 97,8% de sucesso ou empate. Pode-se destacar também o número de casos em que as propostas foram efetivamente superiores a PSGA, com HTSGA e HTSG produzindo escalonamento melhores em 64,7% e 62,2% dos casos, respectivamente.

A Tabela 5.13 mostra a vantagem média atingida por HTSGA e HTSG em relação ao *makespan* obtido pelas demais heurísticas. Pelo desempenho médio geral, ambas as propostas conseguem melhorar o *makespan* em relação às heurísticas da literatura. Os maiores ganhos obtidos pelas propostas foram sobre os casos que envolvem GADs de granularidade fina, sendo este resultado devido à política de inserção existente nas versões de HEFT embutidas em HTSGA e HTSG. Este fato pode ser constatado por uma análise nos gráficos da Figura 5.9, que mostram, em termos relativos, quantas vezes cada heurística embutida conseguiu atingir a melhor solução produzida por seu respectivo algoritmo.

Tomando como referência os ganhos obtidos pelas propostas sobre a heurística PSGA para grafos com granularidade fina, verifica-se que HTSGA obtém uma

		ETF		HEFT		PSGA		HTSGA		HTSG	
	>			139	34,3%	294	72,6%	344	84,9%	350	86,4%
ETF	=			79	19,5%	48	11,9%	61	15,1%	55	13,6%
	<			187	46,2%	63	15,6%	0	0,0%	0	0,0%
	>	187	46,2%			306	75,6%	339	83,7%	344	84,9%
HEFT	=	79	19,5%			91	22,5%	66	16,3%	61	15,1%
	<	139	34,3%			8	2,0%	0	0,0%	0	0,0%
	>	63	15,6%	8	2,0%			262	64,7%	252	62,2%
PSGA	=	48	11,9%	91	22,5%			143	35,3%	144	35,6%
	<	294	72,6%	306	75,6%			0	0,0%	9	2,2%
	>	0	0,0%	0	0,0%	0	0,0%			75	18,5%
HTSGA	=	61	15,1%	66	16,3%	143	35,3%			142	35,1%
	<	344	84,9%	339	83,7%	262	64,7%			188	46,4%
	>	0	0,0%	0	0,0%	9	2,2%	188	46,4%		
HTSG	=	55	13,6%	61	15,1%	144	35,6%	142	35,1%		
	<	350	86,4%	344	84,9%	252	62,2%	75	18,5%		

Tabela 5.12: Comparação de desempenho em termos de *makespans* piores (>), iguais (=) e melhores (<) para escalonamentos gerados em *ambientes restritos* a partir de GADs *diamante* para um total de 405 casos.

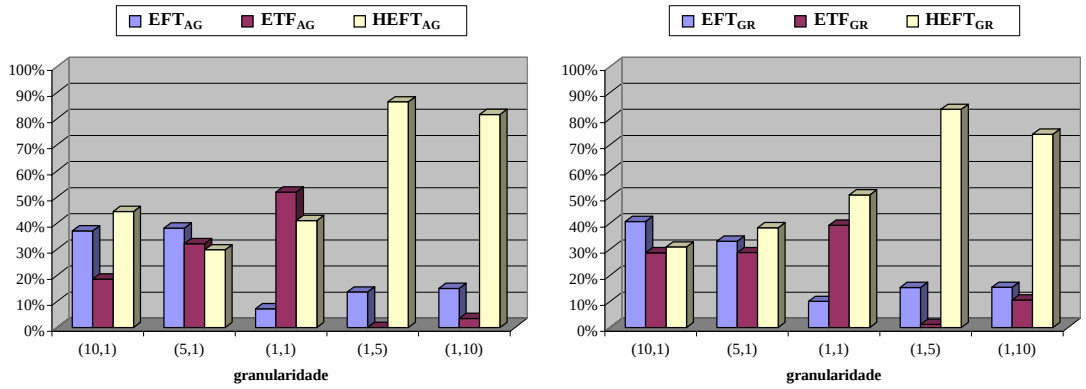
	HTSGA			HTSG		
(me,mc)	**ETF**	**HEFT**	**PSGA**	**ETF**	**HEFT**	**PSGA**
(10,1)	5,8%	2,8%	0,5%	5,8%	2,8%	0,5%
(5,1)	5,7%	2,3%	0,6%	5,6%	2,2%	0,4%
(1,1)	4,6%	10,7%	5,3%	3,5%	9,7%	4,2%
(1,5)	18,8%	25,1%	10,3%	14,3%	21,1%	5,3%
(1,10)	22,7%	27,1%	13,2%	17,4%	22,2%	7,0%
média	**11,5%**	**13,6%**	**6,0%**	**9,3%**	**11,6%**	**3,5%**

Tabela 5.13: Comparação do ganho percentual médio em relação ao *makespan* dos escalonamentos gerados para os GADs *diamante* para granularidades grossa e fina.

vantagem média de até 13,2%, enquanto HTSG atinge a marca de até 7,0%. A diferença entre as vantagens obtidas pelas propostas é devido ao maior número de soluções gerados pelo Algoritmo Genético HTSGA, conforme já discutido anteriormente.

Randômicos

Foram analisados os resultados obtidos para 21 grafos gerados aleatoriamente [43] com número de tarefas variando entre 80 e 546. A granularidade dos GADs foi perturbada como nos testes anteriores e submetidas aos 9 ambientes com-



(a) HTSGA (b) HTSG

Figura 5.9: Comparação, em termos proporcionais, do número de vezes em que as heurísticas de construção de (a) HTSGA e (b) HTSG atingem a melhor solução final para GADs *diamante*.

putacionais restritos definidos, resultando num total de 945 casos avaliados.

A comparação dos casos onde os *makespans* obtidos por cada heurística foram melhores, iguais ou piores pode ser visto na Tabela 5.14. As heurísticas propostas empatam ou superam as heurísticas da literatura em praticamente todos os casos. Quando comparados somente com PSGA, as propostas HTSGA e HTSG atingem, respectivamente, casos de sucesso ou empate em 100,0% e 99,9% dos casos.

A comparação dos ganhos sobre o *makespan* obtidos por HTSGA e HTSG em relação às demais heurísticas da literatura é mostrado na Tabela 5.15. De uma forma geral, os ganhos obtidos pelas propostas para GADs de granularidade grossa foram modestos. Este desempenho melhora quando são considerados os grafos com granularidade fina, principalmente sobre a heurística ETF. No entanto, pode-se destacar que HTSG, mesmo produzindo um menor número de soluções durante suas iterações, apresentou ganhos semelhantes a HTSGA na maioria dos casos. Este desempenho pode ser atribuído à busca local TASK, já que figura em 85,3% dos casos como uma das responsáveis pelo melhor resultado obtido por esta heurística.

		ETF		HEFT		PSGA		HTSGA		HTSG	
	>			818	86,6%	809	85,6%	827	87,5%	833	88,1%
ETF	=			122	12,9%	124	13,1%	118	12,5%	112	11,9%
	<			5	0,5%	12	1,3%	0	0,0%	0	0,0%
	>	5	0,5%			113	12,0%	265	28,0%	312	33,0%
HEFT	=	122	12,9%			649	68,7%	680	72,0%	633	67,0%
	<	818	86,6%			183	19,4%	0	0,0%	0	0,0%
	>	12	1,3%	183	19,4%			281	29,7%	329	34,8%
PSGA	=	124	13,1%	649	68,7%			664	70,3%	615	65,1%
	<	809	85,6%	113	12,0%			0	0,0%	1	0,1%
	>	0	0,0%	0	0,0%	0	0,0%			94	9,9%
HTSGA	=	118	12,5%	680	72,0%	664	70,3%			778	82,3%
	<	827	87,5%	265	28,0%	281	29,7%			73	7,7%
	>	0	0,0%	0	0,0%	1	0,1%	73	7,7%		
HTSG	=	112	11,9%	633	67,0%	615	65,1%	778	82,3%		
	<	833	88,1%	312	33,0%	329	34,8%	94	9,9%		

Tabela 5.14: Comparação de desempenho em termos de *makespans* piores (>), iguais (=) e melhores (<) para escalonamentos gerados em *ambientes restritos* a partir de GADs *randômico* para um total de 945 casos.

	HTSGA			HTSG		
(me,mc)	**ETF**	**HEFT**	**PSGA**	**ETF**	**HEFT**	**PSGA**
(10,1)	2,4%	0,1%	0,0%	2,5%	0,1%	0,0%
(5,1)	2,6%	0,1%	0,0%	2,7%	0,1%	0,1%
(1,1)	4,7%	0,5%	0,3%	4,8%	0,5%	0,3%
(1,5)	13,4%	3,7%	3,0%	13,4%	3,5%	2,9%
(1,10)	17,7%	4,8%	8,6%	17,3%	4,2%	8,1%
média	**8,2%**	**1,8%**	**2,4%**	**8,1%**	**1,7%**	**2,3%**

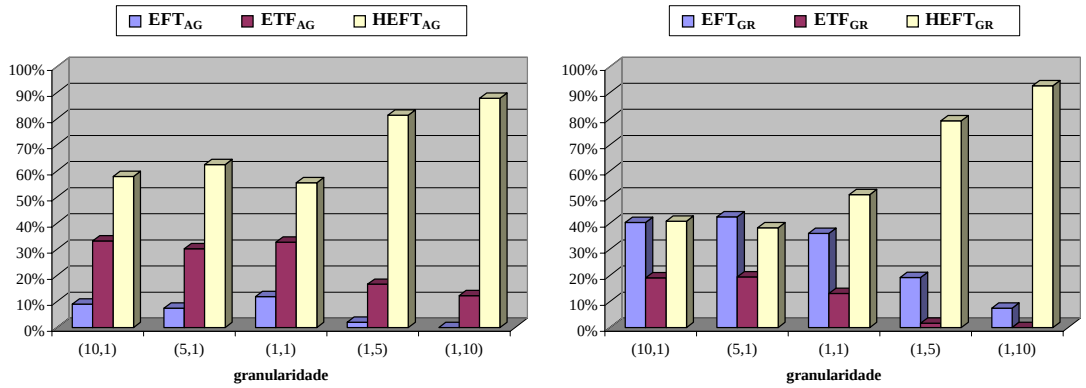
Tabela 5.15: Comparação do ganho percentual médio em relação ao *makespan* dos escalonamentos gerados para os GADs *randômico* para granularidades grossa e fina.

5.3.2 GADS Não Unitários

Foram realizados testes com ambientes restritos para GADs com pesos não unitários. Os grafos examinados possuem um número variado de tarefas e são pertencentes a dois grupos distintos: grafos PSG [10], coletados da literatura, e grafos resultantes do cálculo da eliminação de Gauss [108].

Peer Set Graphs

O *Peer Set Graphs* (PSG) é um conjunto composto por GADs que foram utilizados como exemplos em diversas publicações na área de escalonamento de



(a) HTSGA (b) HTSG

Figura 5.10: Comparação, em termos proporcionais, do número de vezes em que as heurísticas de construção de (a) HTSGA e (b) HTSG atingem a melhor solução final para GADs *randômicos*.

tarefas. Coletados por Kwok *et al* [10], este conjunto de 11 instâncias é formando por grafos não unitários e de topologia irregular com um pequeno número de tarefas. Para este conjunto, os experimentos foram executados com todos os 9 ambientes restritos do tipo dual previamente descritos, produzindo um total de 99 casos.

Os ganhos médios obtidos pelas propostas sobre os *makespans* produzidos pelas demais heurísticas, são apresentados na Tabela 5.16. As duas primeiras colunas apresentam a relação dos grafos do conjunto PSG e seus respectivos números de tarefas. O restante da tabela, dividida em duas seções, apresenta o ganho médio obtido por cada uma das propostas sobre os *makespans* produzidos pelas demais heurísticas. Finalmente, a última linha apresenta o ganho médio geral de cada heurística proposta sobre as demais.

De um modo geral, os algoritmos propostos produziram bons resultados para os grafos irregulares em PSG, conforme pode ser observado na Tabela 5.16. Na média geral, a proposta HTSGA foi sempre melhor que as demais heurísticas da literatura, com ganhos variando entre 1,0% e 17,7%. O mesmo ocorre com HTSG que em média conseguiu um desempenho similar ao de HTSGA, obtendo ganhos na faixa de 1,1% e 17,8%.

Instância	n	HTSGA			HTSG		
		ETF	**HEFT**	**PSGA**	**ETF**	**HEFT**	**PSGA**
al-maasarani	16	10,9%	5,6%	0,8%	11,0%	5,6%	0,8%
al-mouhamed	17	16,7%	5,3%	0,7%	17,1%	5,6%	1,2%
g4	18	28,6%	5,1%	0,0%	28,3%	4,6%	-0,6%
kruatrachue-fig8	11	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
lctd-eg1	11	26,6%	11,4%	1,3%	26,6%	11,4%	1,3%
lwb	9	8,0%	0,0%	0,0%	8,0%	0,0%	0,0%
mccreary1	9	13,6%	11,5%	4,2%	13,6%	11,5%	4,2%
pbsa-ipps95	13	19,7%	10,9%	2,7%	18,7%	9,3%	1,1%
ranka	11	22,8%	9,4%	0,0%	22,8%	9,4%	0,0%
tyang1	7	28,8%	1,7%	0,0%	30,1%	4,0%	2,3%
tyang4	7	19,2%	13,7%	1,1%	20,2%	14,7%	2,2%
média		**17,7%**	**6,8%**	**1,0%**	**17,8%**	**6,9%**	**1,1%**

Tabela 5.16: Comparação do ganho percentual médio em relação ao *makespan* dos escalonamentos gerados para os GADs *PSG*.

Para a maior parte dos grafos, tanto HTSGA quanto HTSG produziram escalonamentos melhores que as demais heurísticas da literatura. A principal exceção foi para o grafo "g4", onde HTSG empatou com PSGA em 8 casos e perdeu em 1, levando seu ganho médio para $-0,6\%$. Uma instância em que nenhuma das heurísticas conseguiu se sobressair foi o grafo "kruatrachue-fig8", onde todas os algoritmos chegaram ao mesmo *makespan*. Um outro caso que merece destaque ocorreu com o grafo "mccreary1", cujo valor do escalonamento ótimo é conhecido [10]. Para este grafo ambas as heurísticas propostas conseguiram atingir o escalonamento ótimo em 66,7% das ocorrências envolvendo esta instância.

5.4 Tempos Computacionais

Neste trabalho, são implementadas diferentes heurísticas de construção (EFT [17], HEFT [21] e ETF [24]) e uma busca local (TASK [59]). As heurísticas de construção foram implementadas em sua forma original (gulosa) e também adaptadas para funcionarem como métodos de construção recebendo diferentes listas de prioridades de HTSGA. A adequação para a estrutura da metaheurística GRASP das heurísticas EFT, ETF e HEFT consistiu na introdução de elementos aleatórios de forma a torná-las adaptativas, randômicas e semi-gulosas.

Portanto, pode-se dizer que tanto o Algoritmo Genético HTSGA quanto o GRASP HTSG utilizam a cada iteração uma ou mais das heurísticas EFT, ETF e HEFT. Com isso, os algoritmos propostos apresentam um tempo computacional bem maior que o utilizado pelas heurísticas gulosas adaptadas. Por exemplo, para escalonar um grafo com topologia randômico com 256 tarefas em um ambiente dual com 6/2 processadores, as heurísticas de construção ETF e HEFT necessitam de aproximadamente 2 milisegundos, enquanto PSGA, HTSGA e HTSG consomem 3.295, 3.497 e 314 milisegundos, respectivamente. Este valores são proporcionais ao número de soluções construídas durante as iterações de cada algoritmo.

Na prática, os tempos computacionais médios exigidos por HTSGA e HTSG nos experimentos realizados foram muito pequenos, ficando, em média, na ordem de alguns segundos. Mais informações sobre tempos computacionais exigidos pelas heurísticas estudadas, podem ser observados na análise probabilística de resultados descrita a seguir.

5.5 Análise Probabilística dos Resultados

Uma metodologia para análise de desempenho de metaheurísticas foi proposta por Aiex, Resende e Ribeiro em [109]. Nesta abordagem, é analisada a probabilidade de um algoritmo atingir determinada solução alvo em função do tempo de processamento. Esta análise é interessante pois permite, através de elementos probabilísticos, fornecer indicações quanto a tendência de convergência de um algoritmo em direção à soluções de boa qualidade.

Para efetuar esta análise, o passo inicial foi determinar um conjunto de instâncias representantes de algumas classes de GADs estudadas neste trabalho. Para isto, foi selecionada uma instância de cada classe diamante, árvore binária, árvore binária invertida e randômico. Cada uma destas instâncias teve seus pesos perturbados por $(me, mc) = (1, 10)$ sendo então submetida a um ambiente computacional dual restrito com 6 processadores, sendo 2 processadores mais rápidos que os demais. Em seguida, cada GAD foi escalonado 200 vezes no sistema alvo por

cada uma das heurísticas PSGA, HTSGA e HTSG, onde os respectivos valores de *makespan* produzidos foram armazenados. Este procedimento inicial foi executado com o objetivo de determinar os valores de *makespan* a serem utilizados como valores alvo na análise probabilística posteriormente realizada. Para este estudo, foram estipulados dois tipos de alvos: fácil e difícil. Para cada instância selecionada, foi calculada a média aritmética dos *makespan* obtidos por cada uma das heurísticas PSGA, HTSGA e HTSG nas 200 execuções efetuadas. O alvo fácil foi determinado com sendo o maior valor médio de *makespan* obtido nas execuções. Já o alvo difícil foi fixado com sendo o menor valor médio dos *makespans* obtidos nestas mesmas execuções. Este processo foi repetido para cada uma das instâncias selecionadas.

Após a determinação dos alvos, as heurísticas PSGA, HTSGA e HTSG foram novamente executadas. Cada heurística escalonou 200 vezes cada uma das instâncias selecionadas, de forma que quando um algoritmo obtinha uma solução menor ou igual ao alvo estipulado, sua execução era interrompida e o tempo gasto no processamento registrado. Este procedimento foi executado tanto para os alvos fáceis quanto para os difíceis. Em seguida, os tempos de processamento obtidos foram dispostos em ordem crescente em um conjunto $K =< k_1, k_2, \ldots, k_{200} >$, onde para cada $k_i \in K$ existe um valor de probabilidade associado definido por $prob(k_i) = (i - \frac{1}{2})/200$. Finalmente, uma distribuição de probabilidades gerada por cada heurística foi plotada em um plano cartesiano $K \times [0, 1]$, gerando um gráfico formado por pontos do tipo $(k_i, prob(k_i))$.

Conforme já mencionado, foram selecionadas instâncias que representam as classes de grafos estudadas neste trabalho. Por questões práticas, o critério de seleção, tanto dos grafos quanto do sistema de computação, foi orientado pelos tempos de execução gastos pelos algoritmos. Isto porque o grande número de repetições necessárias para avaliar cada instância demanda um tempo de processamento considerável. As instâncias selecionadas e seus respectivos alvos estão relacionadas na Tabela 5.17.

Classe	Instância	No. de tarefas	Alvo Fácil	Alvo Difícil
diamante	di256	256	177	159
árvores binárias invertidas	intree255	255	85	79
árvores binárias	outtree255	255	79	76
randômicos	rand140	140	64	56

Tabela 5.17: Instâncias utilizadas para a análise probabilística de resultados de PSGA, HTSGA e HTSG.

Diamante

Para a instância di256 selecionada como representante da classe diamante, os alvos fácil e difícil foram fixados em 177 e 159, respectivamente. A Figura 5.11 apresenta a distribuição de probabilidades de PSGA, HTSGA e HTSG atingirem um *makespan* alvo *fácil* no valor de 177. O eixo vertical do gráfico indica os valores de probabilidade e o eixo horizontal o tempo de execução decorrido em milisegundos. Vale ressaltar que a escala utilizada no eixo horizontal (tempo) não é linear, e sim logarítmica.

De acordo com a Figura 5.11, das três heurísticas analisadas, HTSG é o algoritmo que apresenta uma convergência mais rápida para o valor alvo (curva mais à esquerda no gráfico). Nesta figura, pode ser observado que num tempo menor que 10 milisegundos, os algoritmos propostos apresentam uma probabilidade de convergência de 100%, enquanto o Algoritmo Genético da literatura (PSGA) mostra uma probabilidade de convergência de no máximo 5%. Comparando os dois algoritmos propostos, observa-se que para probabilidades de convergência mais baixas (menor que 70%), a heurística HTSG se mostra mais rápido que HTSGA.

Como alvo *difícil* para a instância di256 foi empregado o valor 159 onde a distribuição de probabilidades correspondente pode ser vista na Figura 5.12. As heurísticas propostas sempre atingem o alvo estipulado, com HTSGA necessitando de no máximo 8 milisegundos de processamento para antingí-lo. Desta vez, a heurística HTSG leva um pouco mais de tempo para obter uma probabilidade de convergência de 100%, necessitando de pelo menos 28 milisegundos. Já a heurística PSGA só consegue atingir o alvo em raras ocasiões e mesmo assim necessitando entre 1.300 e

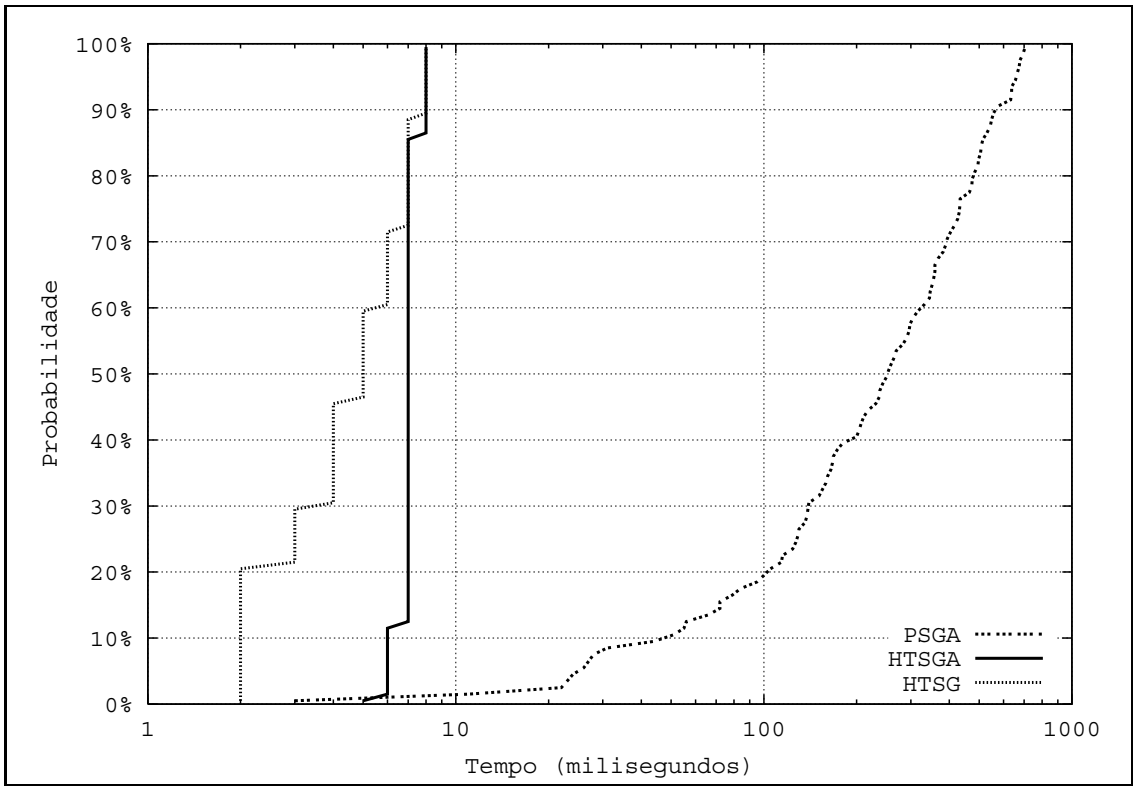


Figura 5.11: Distribuição empírica de probabilidade do tempo para PSGA, HTSGA e HTSG atingirem um *makespan* alvo *fácil* fixado em 177 para a instância *diamante* di256 num ambiente distribuído dual com 6/2 processadores.

3.032 milisegundos para obter o *makespan* especificado. Ou seja, trabalhando com um alvo difícil, PSGA, na maioria das 200 execuções, não conseguiu atingir o valor alvo num tempo máximo pré-fixado.

A rápida convergência das heurísticas propostas deve-se principalmente à política de inserção das versões embutidas da heurística HEFT. Conforme já demonstrado, esta política obteve bons resultados com grafos de granularidade fina, como é o caso do diamante utilizado no exemplo.

Árvore Binária Invertida

Foi selecionada a instância intree255 para representar a classe de grafos árvore binária invertida. De acordo com a Figura 5.13, todos os algoritmos sempre atingem o alvo *fácil*, fixado em 85. Novamente, HTSGA é a heurística que converge mais rapidamente para o alvo, necessitando de pouco menos de 100 milisegundos

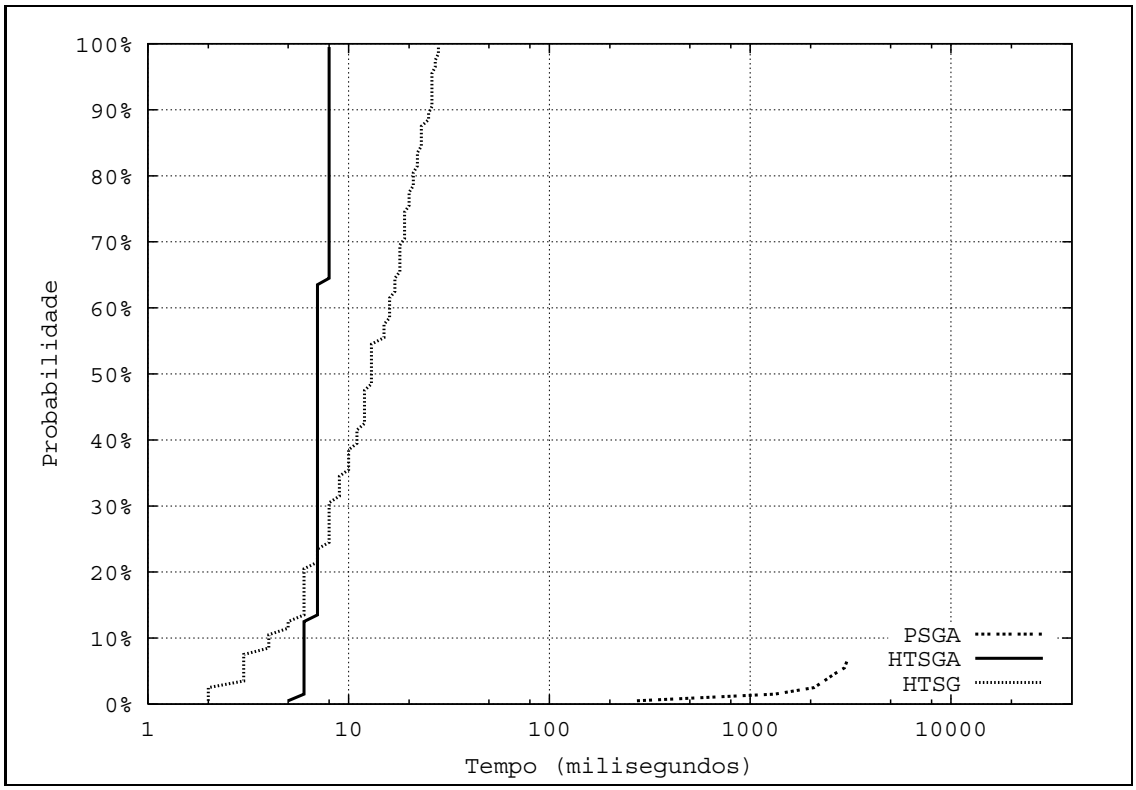


Figura 5.12: Distribuição empírica de probabilidade do tempo para PSGA, HTSGA e HTSG atingirem um *makespan* alvo *difícil* fixado em 159 para a instância *diamante* di256 num ambiente distribuído dual com 6/2 processadores.

de processamento para obter uma probabilidade de convergência de 100%. Para atingir o alvo com esta mesma probabilidade, HTSG já necessita de pouco mais de 300 milisegundos e PSGA não alcança este resultado antes de 1.900 milisegundos.

Para o alvo *difícil* da instância intree255 foi estabelecido o valor 79 com todas as heurísticas sempre conseguindo atingir o alvo determinado, conforme mostrado na Figura 5.14. Para obter com uma probabilidade de 100% uma solução pelo menos tão boa quanto o *makespan* fixado, HTSGA necessita de até 153 ms de processamento, enquanto HTSG utiliza não menos de 391 ms. Já a heurística PSGA necessita de muito mais tempo, ultrapassando a marca dos 11.000 ms para atingir o alvo com uma probabilidade de 100%.

Os gráficos mostram que as heurísticas apresentam desempenho relativo semelhante em relação ao tempo de processamento necessário para atingir os respectivos alvos. Novamente, a política de inserção empregada na construção de soluções

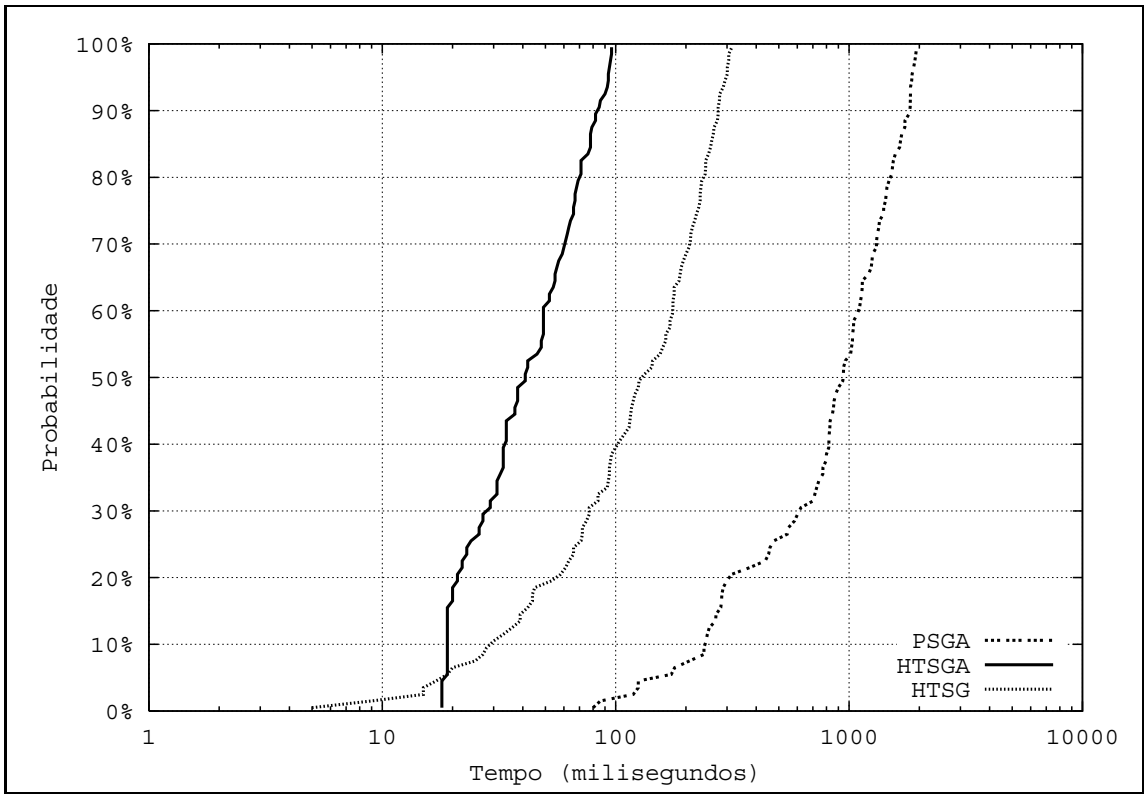


Figura 5.13: Distribuição empírica de probabilidade do tempo para PSGA, HTSGA e HTSG atingirem um *makespan* alvo *fácil* fixado em 85 para a instância *árvore binária invertida* intree255 num ambiente dual com 6/2 processadores.

por HTSGA e HTSG tem ajudado estes algoritmos a convergir mais rapidamente para bons resultados. Outra característica que merece ser mencionada é que HTSG, apesar de gerar menos soluções durante suas iterações que HTSGA e PSGA, consegue chegar a bons resultados num tempo razoável. Este desempenho vem confirmar o fato já comentado sobre a influência de TASK nos resultados obtidos por esta heurística para GADs com esta topologia.

Uma comparação entre os gráficos das Figuras 5.13 e 5.14 mostra a tendência dos algoritmos de consumirem mais tempo no processamentos dos alvos mais difícies.

Árvore Binária

A Figura 5.15 mostra a distribuição de probabilidades das heurísticas atingirem um alvo *fácil*, fixado em 79, para a instância outtree255. Para um tempo

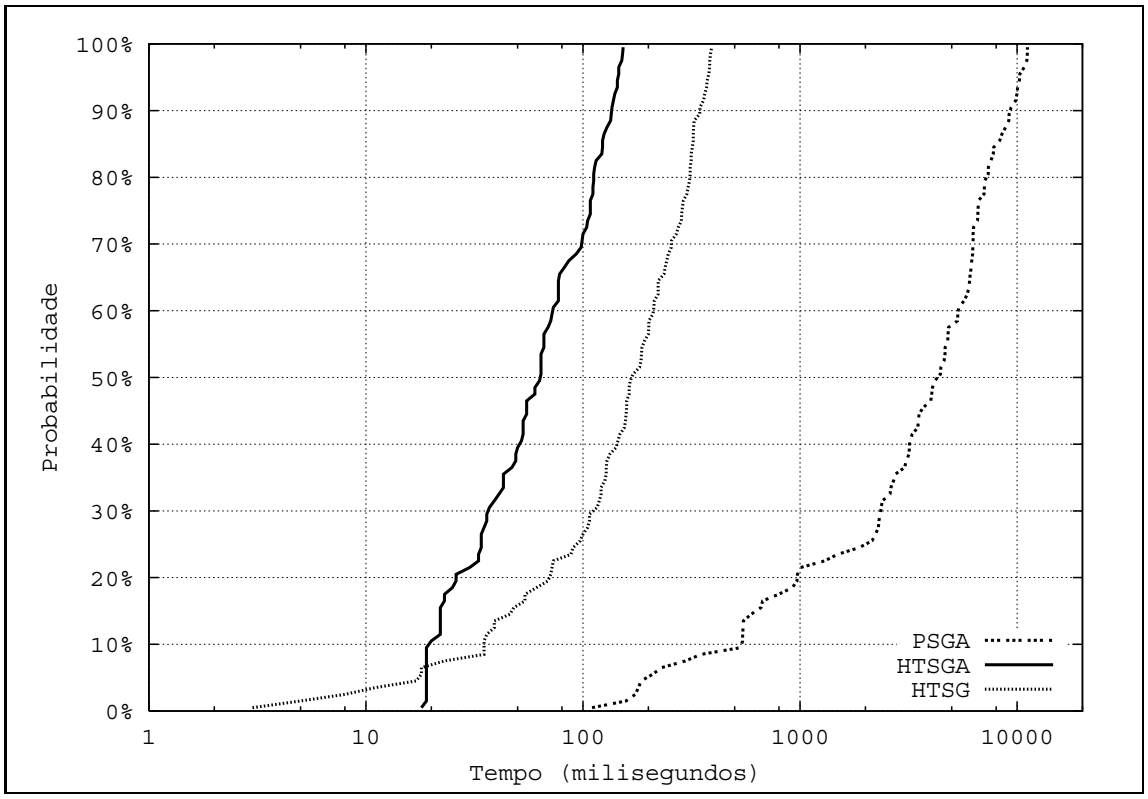


Figura 5.14: Distribuição empírica de probabilidade do tempo para PSGA, HTSGA e HTSG atingirem um *makespan* alvo *difícil* fixado em 79 para a instância *árvore binária invertida* intree255 num ambiente distribuído dual com 6/2 processadores.

máximo de execução de 10 ms, HTSGA consegue com probabilidade de 100,0% obter uma solução pelo menos tão boa quanto o alvo. Para este mesmo tempo, a heurística PSGA tem uma chance de 85,5% de obter uma solução melhor ou igual ao alvo, enquanto HTSG não ultrapassa os 17,5% de probabilidade para atingir este *makespan*.

A rápida convergência tanto de HTSGA quanto de PSGA em direção ao alvo fácil, indica que uma solução de qualidade igual ou superior ao alvo é atingida logo nas primeiras iterações, com possibilidade de ser obtida ainda na geração da população inicial destes Algoritmos Genéticos. Já HTSG leva mais tempo que HTSGA e PSGA para obter, com probabilidade maior que 16%, uma solução pelo menos tão boa quanto a estipulada pelo alvo.

Para o alvo *difícil*, estipulado em 74, o comportamento da heurística PSGA se modifica, necessitando de mais iterações para obter uma solução pelo menos tão

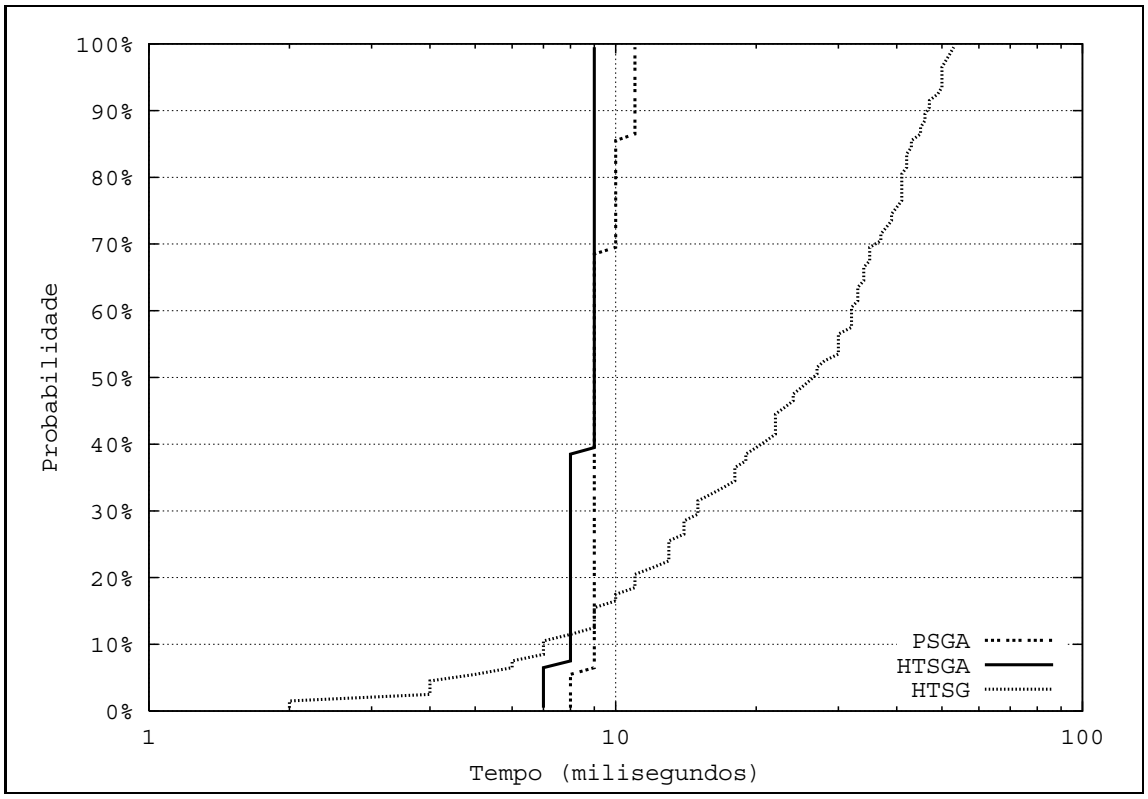


Figura 5.15: Distribuição empírica de probabilidade do tempo para PSGA, HTSGA e HTSG atingirem um *makespan* alvo *fácil* fixado em 79 para a instância *árvore binária* outtree255 num ambiente distribuído dual com 6/2 processadores.

boa quanto o alvo, enquanto HTSGA permanece sendo o que mais rapidamente converge para o *makespan* fixado, conforme pode ser visto na Figura 5.16. A heurística HTSG continua sendo a que mais tempo leva para obter uma solução pelo menos equivalente ao alvo determinado.

Fixada a probabilidade mínima de 90% de chance para as heurísticas produzirem uma solução de melhor ou igual qualidade que a determinada pelo alvo difícil, o Algoritmo Genético HTSGA necessita de no mínimo 11 e no máximo 12 milisegundos de processamento. Já as heurísticas PSGA e HTSG consomem pelo menos de 1.703 e 5.519 segundos, respectivamente.

Tanto para o alvo fácil quanto para o difícil, pode-se observar que HTSG converge lentamente em direção ao alvo quando comparado com PSGA e HTSGA. Isto porque, em GADs com topologia de árvore binária, existem inicialmente poucas tarefas livres para serem selecionadas durante a fase de construção de uma solução.

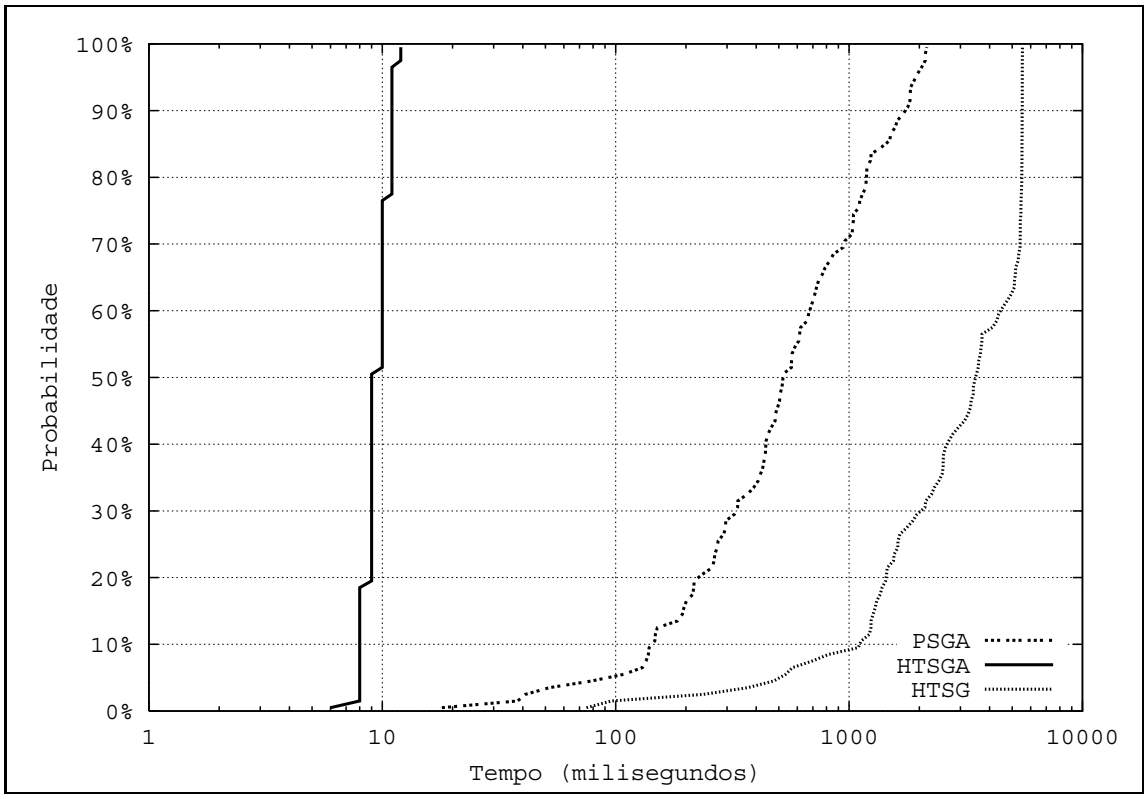


Figura 5.16: Distribuição empírica de probabilidade do tempo para PSGA, HTSGA e HTSG atingirem um *makespan* alvo *difícil* fixado em 74 para a instância *árvore binária* outtree255 num ambiente distribuído dual com 6/2 processadores.

Conforme já discutido, HTSG constrói uma Lista Restrita de Candidatos (LRC) a partir das tarefas que estão livres em cada iteração da construção, para depois, selecionar aleatoriamente da LRC a tarefa que será efetivamente escalonada. Como existem poucas tarefas livres inicialmente, o algoritmo tem poucas opções de escolha, necessitando executar várias iterações até exista um número de tarefas livres que possibilite que o mecanismo de construção possa sofrer influência do fator de aleatoriedade existente no momento da seleção de uma tarefa para escalonamento. Em suma, a topologia deste tipo de grafo faz com que os métodos de construção embutidos sofram pouca influência do fator de aleatoriedade existente, já que este fator permite que o algoritmo produza soluções em diversas áreas do espaço de busca do problema onde, eventualmente, podem ser encontradas melhores soluções.

Randômicos

A instância randômica ran140 teve seu alvo *fácil* fixado em 64. A Figura 5.17 mostra que as heurísticas propostas convergem muito rapidamente para uma solução alvo. O GRASP HTSG, em alguns casos, necessita de menos de 1 ms para gerar uma solução adequada. Isto indica que esta heurística está obtendo a solução desejada já na fase de construção da primeira iteração. Comportamento semelhante ocorre com HTSGA que necessita de no mínimo 3 e no máximo 4 ms para chegar ao alvo com uma probabilidade de convergência de 100%. A análise das soluções geradas mostrou que estas heurísticas utilizaram as respectivas versões de HEFT na produção de soluções pelo menos tão boas quanto o alvo. Isto justifica o fato de PSGA necessitar de no mínimo 917 ms para chegar ao mesmo resultado, já que não pode contar com o mecanismo de escalonamento de HEFT para construir suas soluções.

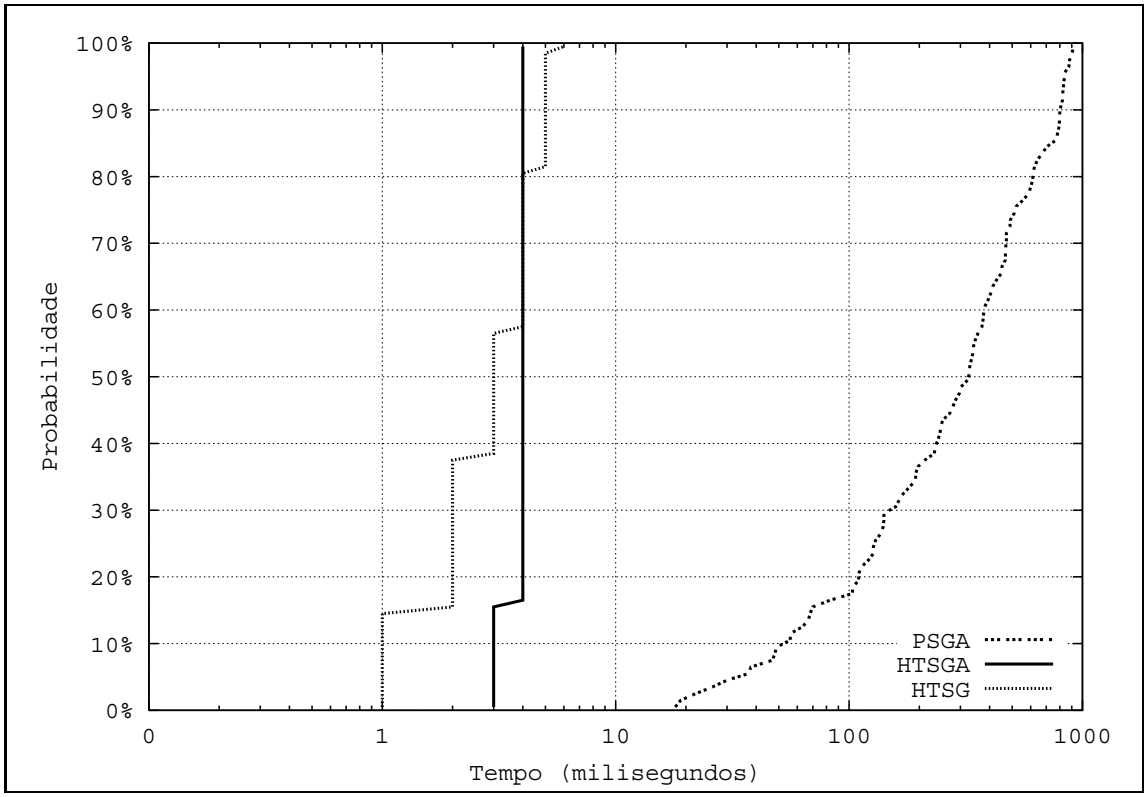


Figura 5.17: Distribuição empírica de probabilidade do tempo para PSGA, HTSGA e HTSG atingirem um *makespan* alvo *fácil* fixado em 64 para a instância *randômica* ran140 num ambiente distribuído dual com 6/2 processadores.

O alvo *difícil* da instância ran140 foi estipulado como 53 para o sistema distribuído alvo. A Figura 5.18 mostra que a heurística PSGA nunca consegue atingir este alvo, motivo pelo qual não figura no gráfico. Já as propostas HTSGA e HTSG sempre chegam a uma solução compatível, com HTSGA novamente convergindo mais rapidamente em direção ao alvo. Se for fixado um tempo máximo de execução de 1.000 ms, a heurística HTSG apresenta probabilidade até 30,5% de atingir uma solução pelo menos de mesma qualidade que do alvo. Já HTSGA sempre produz uma solução igual ou melhor que o alvo, necessitando para tanto de no máximo 380 ms.

A ausência de PSGA no gráfico mostra que o esquema de construção empregado por esta heurística não é adequado em alguns casos. A versão da heurística EFT embutida em PSGA não consegue produzir uma solução com *makespan* menor ou igual ao alvo, apesar da heurística ter sido executada por pelo menos 400 segundos. Este comportamento vem mostrar um fato já argumentado neste trabalho, de que algumas heurísticas de construção apresentam comportamentos diferenciados dependendo das instâncias a qual são submetidas. No exemplo apresentado, fica claro que PSGA não atinge a solução alvo determinada devido a uma deficiência da heurística de construção EFT embutida.

Já foi comentado que tanto HTSGA quanto PSGA produzem mais soluções durante suas iterações que HTSG. Este fato, em alguns casos, tem contribuído para que os AGs cheguem mais rapidamente a uma solução especificada. No entanto, em relação a HTSGA, um outro fator que também pode influenciar este comportamento é a forma com que as tarefas são ordenadas para escalonamento pelas heurísticas embutidas nestes algoritmos. Conforme comentado no capítulo anterior, o mecanismo de geração de prioridades de HTSGA é sempre baseado em dados da instância, isto é, o algoritmo calcula e atribui prioridades às tarefas em função de propriedades como o *nível* e *co-nível*, por exemplo. No caso de HTSG, o mecanismo de construção não considera características das instâncias, selecionando aleatoriamente a próxima tarefa a ser escalonanda durante a construção de soluções. Os resultados apresentados até agora mostram que a ordenação totalmente aleatória de prioridades parece não ser um mecanismo adequado para alguns casos.

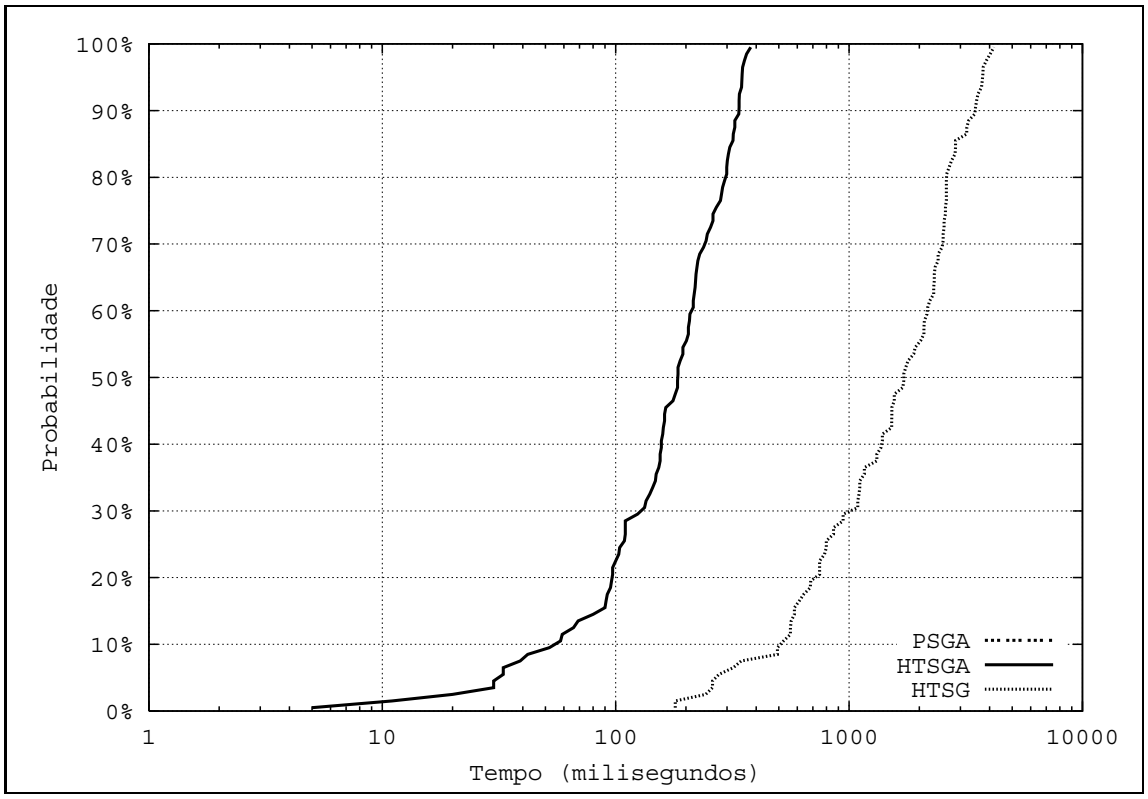


Figura 5.18: Distribuição empírica de probabilidade do tempo para PSGA, HTSGA e HTSG atingirem um *makespan* alvo *difícil* fixado em 53 para a instância *randômica* ran140 num ambiente distribuído dual com 6/2 processadores.

A análise probabilística mostra como principal resultado empírico que, em todos os testes, os algoritmos propostos HTSGA e HTSG se mostraram bastante robustos, convergindo sempre para o valor alvo pré-definido, ao contrário do Algoritmo Genético PSGA da literatura. Este resultado é importante pois muitas das heurísticas e algumas metaheurísticas apresentam um comportamento muito heterogêneo na prática, não obtendo um desempenho estável quando submetidos a instâncias com características variadas.

Uma comparação entre o desempenho dos algoritmos propostos, mostra que nos testes realizados, HTSGA se mostrou, em média, mais rápido que HTSG em algumas classes de instâncias. No entanto, o bom desempenho da metaheurística GRASP em diferentes aplicações, mostra que HTSG pode ser a base para uma versão mais aperfeiçoada de um algoritmo para o problema aqui estudado.

5.6 Resumo

Nesta seção foi apresentada a análise experimental dos resultados obtidos pelas abordagens híbridas propostas neste trabalho. Também foi realizada uma análise, baseada em distribuição de probabilidades, comparando o desempenho em função do tempo das propostas e de uma heurística da literatura.

Em relação aos resultados obtidos pelos algoritmos propostos, os testes realizados mostram que a estrutura de uma metaheurística pode melhorar bastante o desempenho de algoritmos tradicionais de construção, embora dispendendo um tempo de computacional mais elevado. Numa comparação entre metaheurísticas, a análise probabilísitca mostra o grau de robustez destes algoritmos, mostrando em conseqüência, sua confiabilidade de uso em aplicações reais. No próximo capítulo, são expostas as conclusões e sugestões de trabalhos futuros.

Capítulo 6

Conclusões e Propostas de Trabalhos Futuros

Neste trabalho, foi estudada a viabilidade da integração de heurísticas de escalonamento da classe *list scheduling* e mecanismos de busca local baseados em metaheurísticas para a solução do problema de escalonamento de tarefas em ambientes de computação distribuídos. Neste sentido, heurísticas tradicionais da literatura foram selecionadas, adaptadas e incorporadas a duas propostas baseadas em um Algoritmo Genético e um GRASP.

Heurísticas de escalonamento *list scheduling* são algoritmos construtivos bastante explorados pela literatura especializada devido, provavelmente, a sua simplicidade e baixa complexidade quando comparadas a outras classes de heurísticas [8]. No entanto, a característica inerentemente gulosa destes algoritmos pode conduzí-los a um comportamento míope durante a produção de escalonamentos. Este comportamento, pode influenciar negativamente as heurísticas, não permitindo uma avaliação adequada da influência das tarefas na construção do escalonamento. Partindo desta premissa, foi proposto neste trabalho um mecanismo que procurasse suprir esta deficiência e que, ao mesmo tempo, pudesse conduzir a um algoritmo que se adaptasse bem a diversas classes de instâncias, tendo como conseqüência a pro-

dução de escalonamentos de qualidade. Baseado no trabalho de Ahmad *et al* [17], tal mecanismo consiste basicamente na geração, baseada em dados da instância, de diversas seqüências de prioridades que são posteriormente utilizadas por heurísticas *list scheduling* para a ordenação de tarefas durante a construção de um escalonamento. Assim, o conjunto destas seqüências de prioridades formam um espaço de busca que é explorado pelas heurísticas aqui propostas, denominadas de HTSGA (Algoritmo Genético) e HTSG (GRASP). Estas propostas incorporam conceitos de algoritmos *list scheduling* tradicionais como EFT [17], HEFT [21] e ETF [24] para mapear as seqüências de prioridades em escalonamentos orientados para ambientes distribuídos com o objetivo de minimizar o *makespan* de uma aplicação paralela.

Como forma de validar o estudo realizado neste trabalho, foram realizados experimentos onde os algoritmos propostos foram submetidos a instâncias representantes de diferentes classes de aplicações paralelas. Foram utilizadas instâncias com pesos de computação e comunicação unitários [43] com topologias regulares (diamantes, árvores binárias, árvores binárias invertidas) e irregulares (gerados randomicamente). Os experimentos também consideraram GADs com pesos arbitrários como os grafos PSG [10] e instâncias representantes da paralelização do cálculo da eliminação de Gauss [108].

Para representar sistemas de computação distribuídos, foi definido um tipo de sistema de computação denominado *dual*, onde os processadores são classificados como rápidos ou lentos. Foram definidas diversas dimensões de ambientes duais, onde, para cada ambiente gerado, o número de processadores rápidos foi variado enquanto o número total de processadores era mantido constante. Os testes se concentraram sobre dois tipos de ambientes duais. No primeiro, denominado ambiente não restrito, as instâncias foram dimensionadas de tal forma que o número total de processadores fosse suficiente para o escalonamento das tarefas da maioria das aplicações paralelas submetidas. No segundo tipo, denominado ambiente restrito, as instâncias foram dimensionadas com um número reduzido de processadores.

Inicialmente, foi analisado o desempenho das heurísticas propostas quando submetidas a ambientes não restritos. O objetivo deste experimento foi avaliar o

comportamento destas heurísticas no que diz respeito à exploração do paralelismo da aplicação quando não há restrição no número de processadores disponíveis. Desta forma, foi possível observar o comportamento das heurísticas tanto na minimização do *makespan* quanto na redução do número de processadores efetivamente utilizados pelos escalonamentos obtidos. Para comparação dos resultados foram utilizadas duas heurísticas bastante referenciadas na literatura, ETF [24] e HEFT [21], além do Algoritmo Genético PSGA [17], que serviu de base para este trabalho. Foram executados 2.520 experimentos para ambientes não restritos com GADs unitários e não unitários, onde a análise dos resultados foi organizada de acordo com as topologias dos grafos.

De maneira geral, os resultados obtidos pelas propostas foram bastante promissores, com destaque para a heurística HTSGA que se mostrou mais eficiente que HTSG em alguns casos. Em relação a ETF e HEFT, os métodos aqui propostos sempre conseguiram produzir soluções de melhor ou igual qualidade que as obtidas por estas heurísticas. Quando comparadas com o Algoritmo Genético PSGA, HTSGA conseguiu produzir soluções melhores ou iguais em todos os casos, enquanto HTSG perde somente em alguns poucos casos. Contudo, é importante enfatizar que nos experimentos realizados a heurística HTSG produz um número menor de soluções por iteração que os Algoritmos Genéticos HTSGA e PSGA, uma característica intrínseca das metaheurísticas GRASP. Esta atitude em relação à heurística HTSG foi intencional, com o objetivo de observar o comportamento da busca local TASK embutida no algoritmo. Os resultados mostraram que TASK não apresentou bons resultados para maioria dos casos, com exceção para árvores binárias invertidas onde obteve melhorias consideráveis, chegando em alguns casos a melhorias da ordem de 50% do valor do *makespan*.

Num segundo experimento, foi analisado o comportamento das heurísticas quando submetidas a ambientes restritos. Nestes experimentos, foram submetidos GADs unitários de topologia diamante e randômica além de grafos do conjunto PSG [10]. As heurísticas propostas repetiram os desempenhos obtidos em ambientes restritos, com HTSGA obtendo soluções melhores ou iguais que as heurísticas da literatura implementadas neste trabalho.

Como conclusões deste trabalho verificou-se que a atribuição de prioridades é um fator importante na determinação da qualidade das soluções obtidas por heurísticas *list scheduling*. Contudo, não somente este fator influencia na qualidade final das soluções produzidas por estes algoritmos, mas também a política de escalonamento empregada. A variação tanto na ordenação de tarefas quanto nas técnicas de escalonamento implementadas, forneceram às heurísticas de busca aqui propostas mecanismos para escapar de ótimos locais e produzir boas soluções dentro das limitações dos algoritmos de construção utilizados nos experimentos.

Versões híbridas de Algoritmos Genéticos e GRASP, utilizando diferentes heurísticas de construção, se mostram mais robustos e confiáveis que heurísticas do tipo *list scheduling* e metaheurísticas evolutivas da literatura. Este resultados são muito importantes para a aplicação de tais heurísticas em ambientes distribuídos atuais, onde a heterogeneidade e a limitação dos processadores são características sempre presentes.

Como continuação deste estudo, algumas possibilidades que foram identificadas e que merecem ser investigadas são:

- Analisar mais detalhadamente o comportamento da busca local TASK, observando principalmente o uso de novas estratégias para a movimentação de tarefas entre processadores, investindo na ordenação de tarefas em um mesmo processador.
- Estudar a aplicabilidade das propostas aqui apresentadas em outros modelos arquiteturais mais reais, como $LogP$ [30] por exemplo;
- Investigar o uso de heurísticas de construção embutidas que apliquem técnicas de replicação de tarefas;
- O uso de outras metaheurísticas como Busca Tabu [87] e VNS [90].

Apêndice A

Notações

Notação	**Descrição**
$PRED(t_i)$	Conjunto de tarefas predecessoras de t_i
$SUCC(t_i)$	Conjunto de tarefas sucessoras de t_i
$\lambda(p_i, p_j)$	Latência do canal de comunicação que liga os processadores p_i e p_j
$h(p_i)$	Fator de heterogeneidade associado ao processador p_i
$\varepsilon(t_i)$	Peso de computação associado à tarefa t_i
$\omega(t_i, t_j)$	Peso de comunicação entre as tarefas t_i e t_j
$\overline{\lambda}$	Latência média dos canais de comunicação de uma máquina alvo
$\overline{\varepsilon}(t_i)$	Custo médio de execução da tarefa t_i nos processadores de uma máquina alvo
$\overline{\omega}(t_i, t_j)$	Custo médio de comunicação entre as tarefas t_i e t_j em uma máquina alvo
$\mathit{nível}(t_i)$	Nível da tarefa t_i em um GAD
$\mathit{co\text{-}nível}(t_i)$	Co-nível da tarefa t_i em um GAD
$\mathit{nível}_e(t_i)$	Nível escalonado da tarefa t_i em um GAD
$\mathit{co\text{-}nível}_e(t_i)$	Co-nível escalonado da tarefa t_i em um GAD
$p(t_i)$	Processador atribuído à tarefa t_i após o escalonamento
$c(t_i, t_j)$	Custo de transmissão de dados entre as tarefas t_i e t_j após o escalonamento
$s(t_i, p_j)$	Tempo de início da execução da tarefa t_i no processador p_j
$f(t_i, p_j)$	Tempo de término da execução da tarefa t_i no processador p_j
$e(t_i, p_j)$	Custo de execução da tarefa t_i no processador p_j
$\mathit{makespan}(S)$	Tempo total de execução do escalonamento S
$\mathit{gran}(G, M)$	Granularidade do grafo G em relação à máquina M

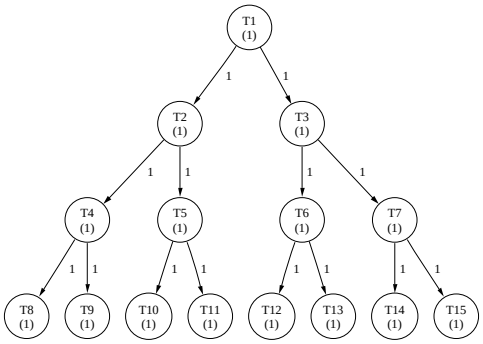
Apêndice B

Instâncias de Aplicações Paralelas

Neste apêndice, estão relacionadas as instâncias de GADs unitários e não unitários utilizadas nos experimentos. Também são mostradas diagramas das instâncias pertencentes ao conjunto *PSG* [10].

B.1 GADs Unitários

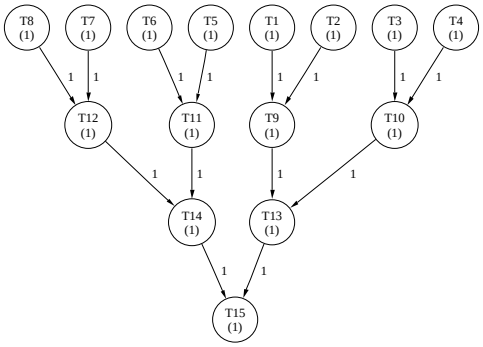
Árvore Binária



Nome da Instância	No. de Tarefas
outtree015	15
outtree031	31
outtree063	63
outtree127	127
outtree255	255
outtree511	511

Figura B.1: (a) Exemplo de um grafo unitário com topologia *árvore binária* com 15 tarefas; (b) Relação de instâncias com esta topologia utilizadas nos experimentos.

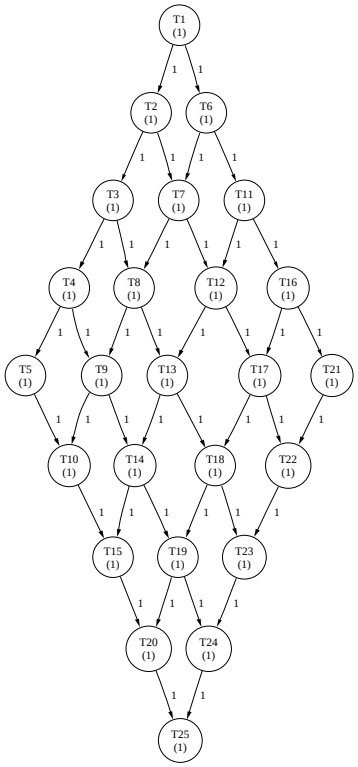
Árvore Binária Invertida



Nome da Instância	No. de Tarefas
intree015	15
intree031	31
intree063	63
intree127	127
intree255	255
intree511	511

Figura B.2: (a) Exemplo de um grafo unitário com topologia *árvore binária invertida* com 15 tarefas; (b) Relação de instâncias com esta topologia utilizadas nos experimentos.

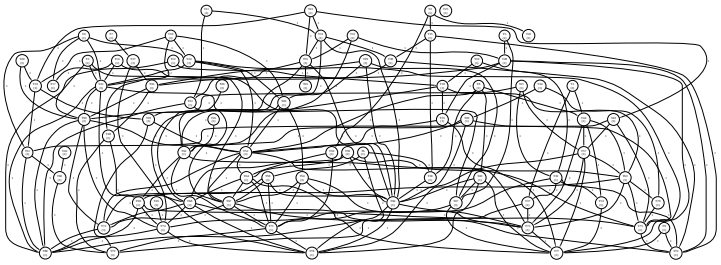
Diamante



Nome da Instância	No. de Tarefas
di0025	25
di0036	36
di0064	64
di0100	100
di0144	144
di0225	225
di0256	256
di0400	400
di1024	1024

Figura B.3: (a) Exemplo de um grafo unitário com topologia *diamante* com 25 tarefas; (b) Relação de instâncias com esta topologia utilizadas nos experimentos.

Randômico



Nome da Instância	No. de Tarefas
ran080	80
ran098	98
ran108	108
ran124	124
ran135	135
ran140	140
ran152	152
ran153	153
ran154	154
ran170	170
ran186	186
ran223	223
ran234	234
ran256	256
ran286	286
ran298	298
ran310	310
ran357	357
ran364	364
ran510	510
ran546	546

Figura B.4: (a) Exemplo de um grafo unitário com topologia *randômica* com 80 tarefas; (b) Relação de instâncias com esta topologia utilizadas nos experimentos.

B.2 GADs não unitários

Eliminação de Gauss

Nome da Instância	No. de Tarefas
gauss0009	9
gauss0020	20
gauss0035	35
gauss0054	54
gauss0104	104
gauss0170	170
gauss0209	209
gauss0275	275
gauss0324	324
gauss0594	594
gauss1034	1034

Tabela B.1: Instâncias de GADs com topologia *eliminação de Gauss*.

PSG - Peer Set Graphs

Nome da Instância	No. de Tarefas
al-maasarani	16
al-mouhamed	17
g4	18
kruatrachue-fig8	11
lctd_eg1	11
lwb	9
mccreary1	9
pbsa-ipps95	13
ranka	11
tyang1	7
tyang4	7

Tabela B.2: Instâncias de GADs pertencentes ao conjunto *Peer Set Graphs - PSG*.

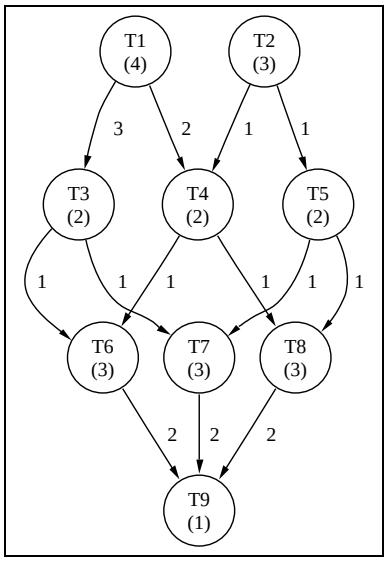


Figura B.5: Grafo lwb

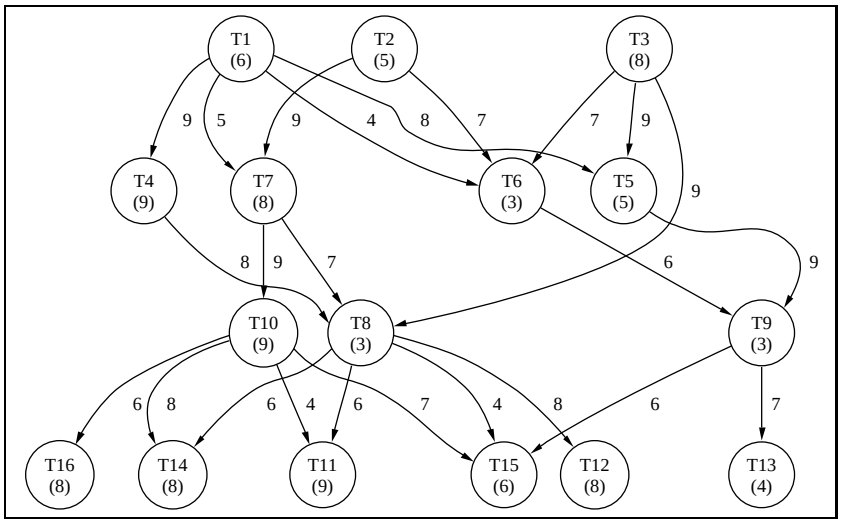


Figura B.6: Grafo al-maasarani

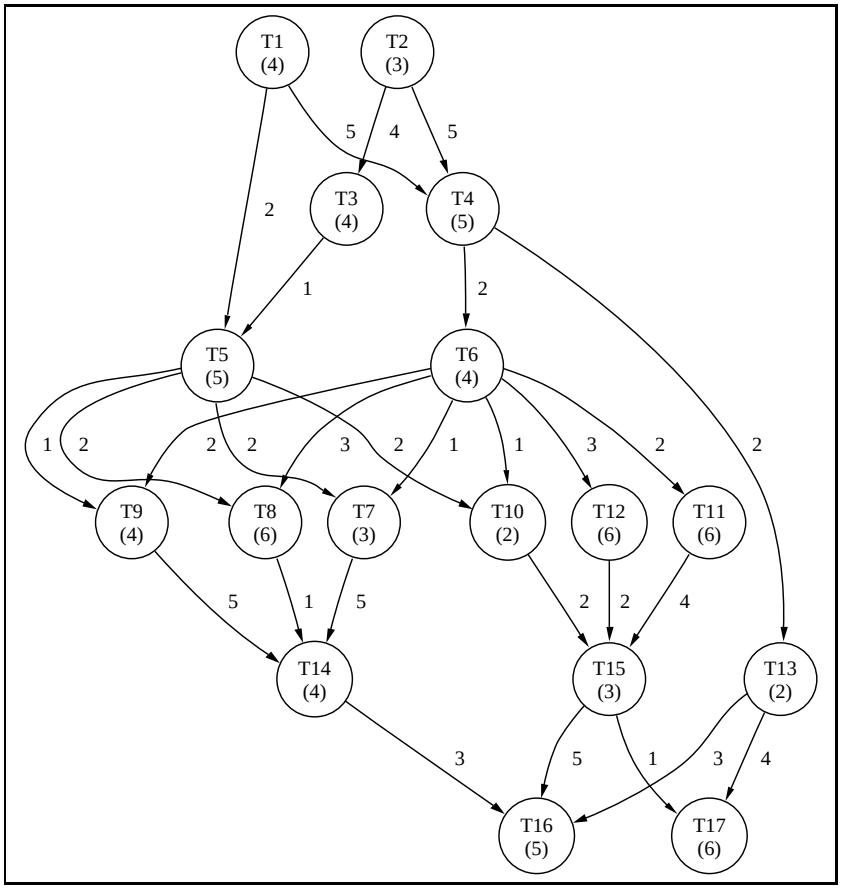


Figura B.7: Grafo al-mouhamend

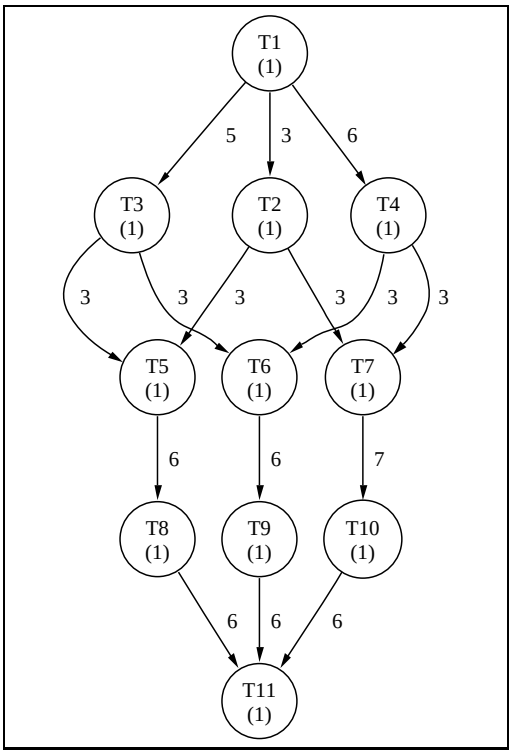


Figura B.8: Grafo kruatrachue-fig8

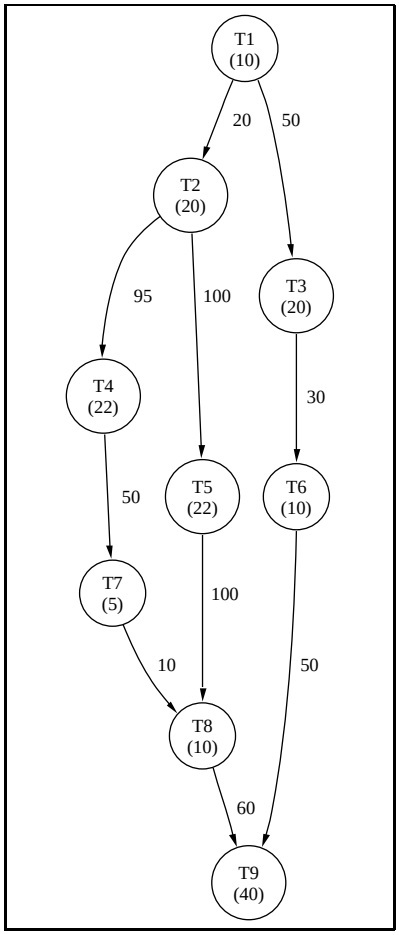


Figura B.9: Grafo mccreary1

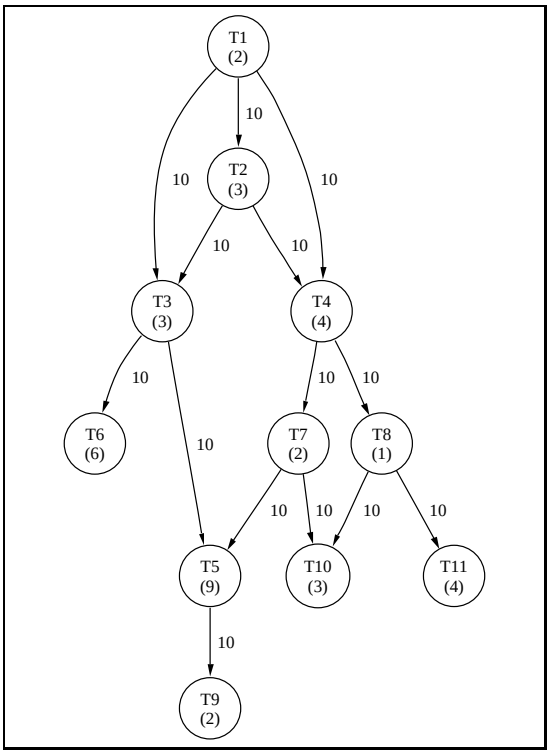


Figura B.10: Grafo lctd-eg1

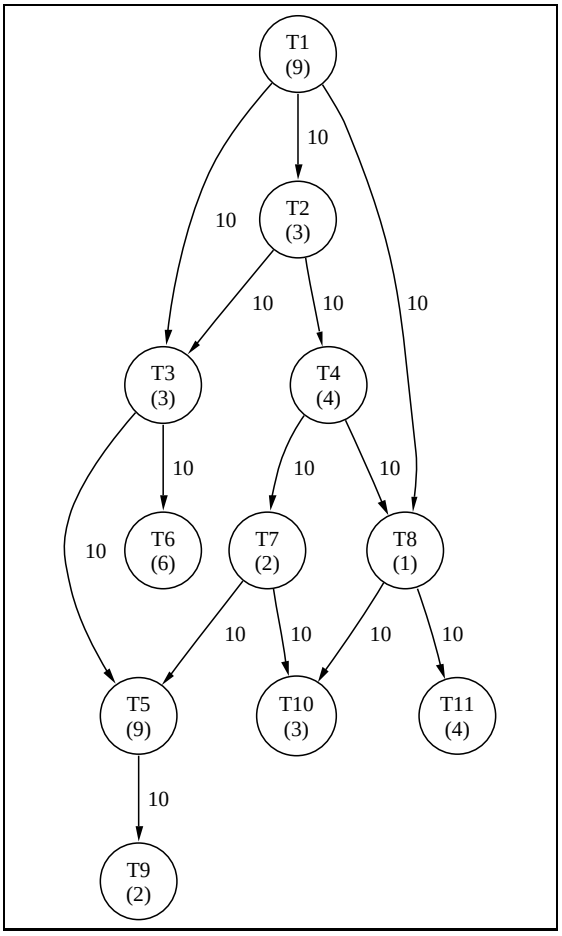


Figura B.11: Grafo ranka

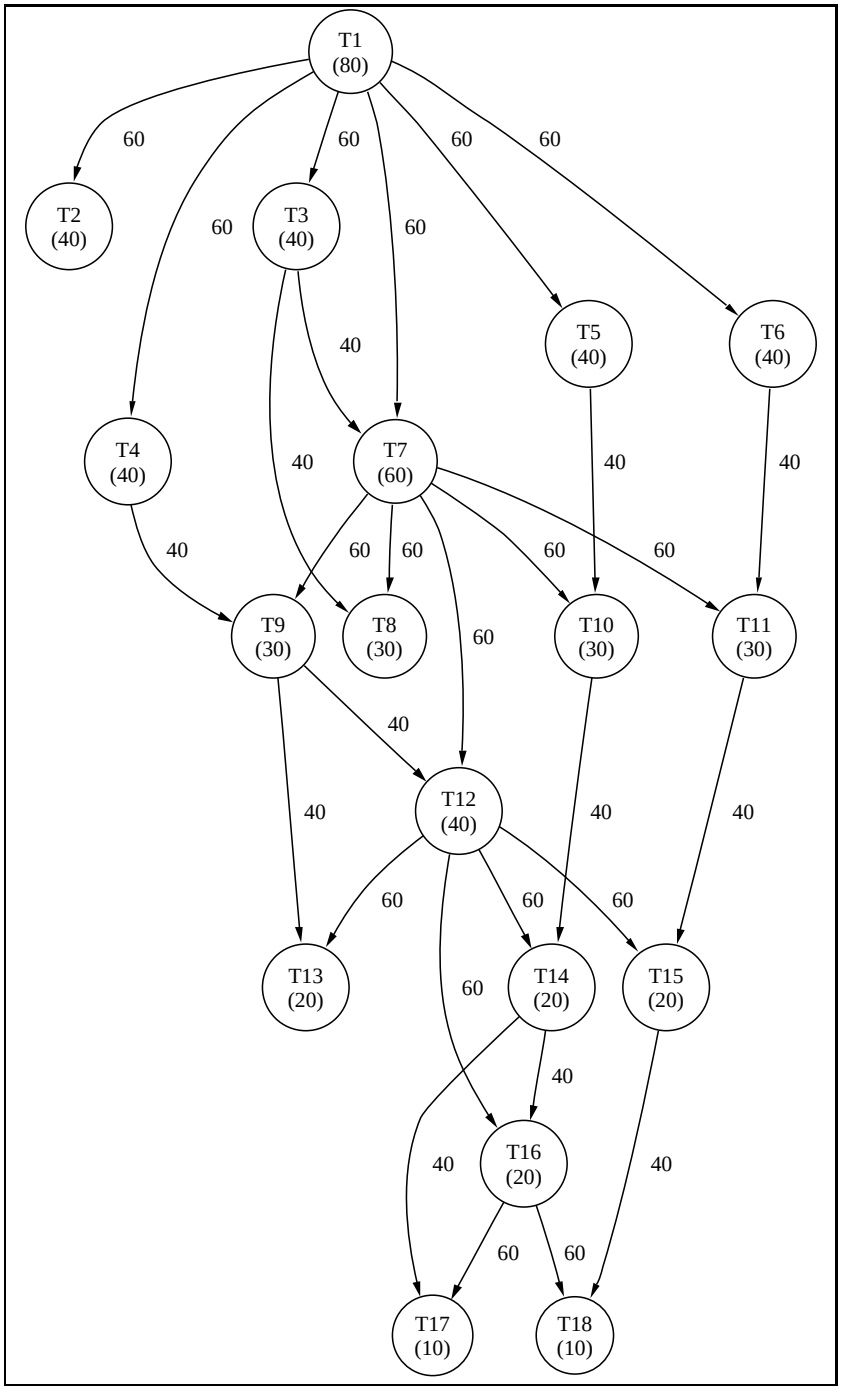


Figura B.12: Grafo g4

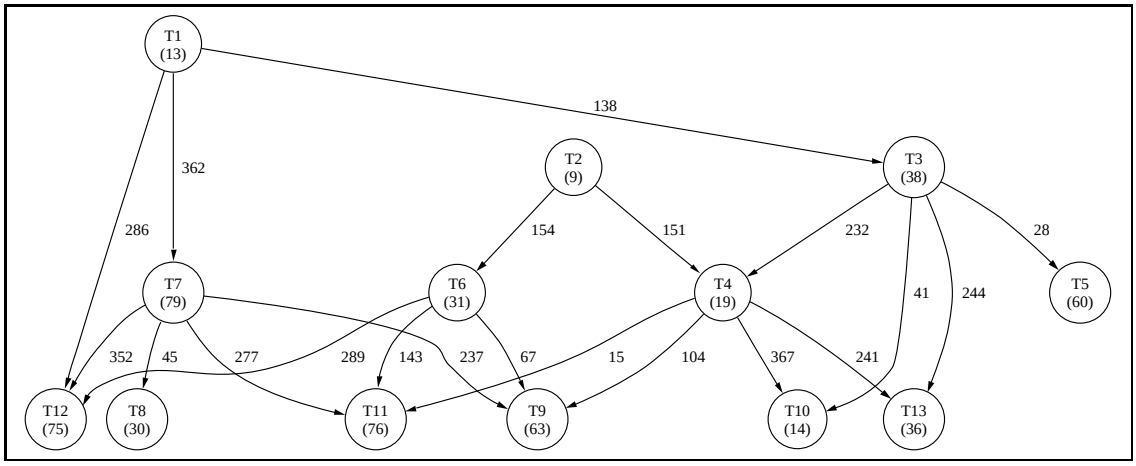


Figura B.13: Grafo pbsa-ipps95

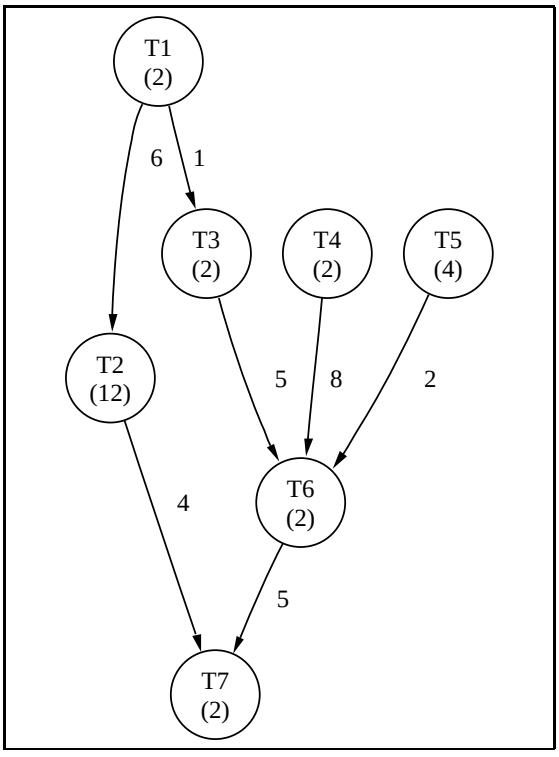


Figura B.14: Grafo tyang1

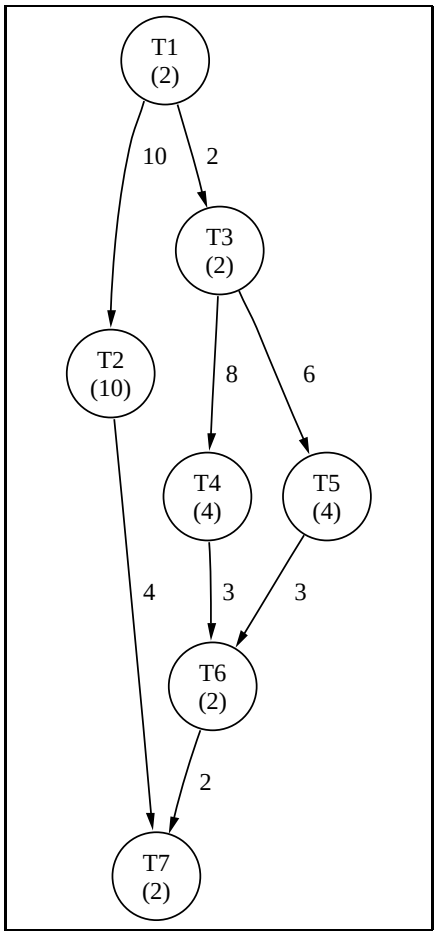


Figura B.15: Grafo tyang4

Apêndice C

Instâncias de Ambientes Computacionais

Esta apêndice relaciona as instâncias de sistemas computacionais do tipo dual utilizados nos experimentos.

Nome da Instância	**Dimensão**	**No. Procs Rápidos**	**No. Procs Lentos**
sis-r2-1	2	1	1
sis-r2-2	2	2	0
sis-r4-1	4	1	3
sis-r4-2	4	2	2
sis-r4-4	4	4	0
sis-r6-1	6	1	5
sis-r6-2	6	2	4
sis-r6-4	6	4	2
sis-r6-6	6	6	0

Tabela C.1: Instâncias de ambientes computacionais do tipo dual restrito utilizadas nos experimentos.

Nome da Instância	Dimensão	No. Procs Rápidos	No. Procs Lentos
sis-nr20-05	20	5	15
sis-nr20-10	20	10	10
sis-nr20-15	20	15	5
sis-nr20-20	20	20	0
sis-nr40-10	40	10	30
sis-nr40-20	40	20	20
sis-nr40-30	40	30	10
sis-nr40-40	40	40	0
sis-nr60-15	60	15	45
sis-nr60-30	60	30	30
sis-nr60-45	60	45	15
sis-nr60-60	60	60	0

Tabela C.2: Instâncias de ambientes computacionais do tipo dual não restrito utilizadas nos experimentos.

Apêndice D

Resultados para Ambientes Não Restritos

Este apêndice contém alguns dos resultados obtidos pelas heurísticas analisadas neste trabalho. Em cada tabela, as colunas **nome** e **n** indicam, respectivamente, o nome e o número de tarefas do GAD escalonado. As colunas **ms/pu** representam, respectivamente, o *makespan* obtido e o número de processadores utilizados pelo escalonamento produzido pela heurística indicada. As colunas **te** mostram o tempo de execução do algoritmo em milisegundos. As colunas **pm** indicam o percentual de melhora do *makespan* obtido pelas heurísticas propostas HTSGA e HTSG em relação ao melhor *makespan* produzido por ETF, HEFT e PSGA.

grafo		**ETF**	**HEFT**	**PSGA**		**HTSGA**		**HTSG**	
nome	***n***	***ms/pu***	***ms/pu***	***ms/pu***	***te***	***ms/pu***	***te***	***ms/pu***	***te***
gauss0009	9	23/1	23/1	23/1	72	23/1	69	23/1	11
gauss0014	14	42/2	46/3	40/2	98	39/3	109	39/3	16
gauss0035	35	125/6	107/5	97/5	283	97/5	430	97/6	45
gauss0044	44	168/7	132/5	119/7	365	119/5	523	119/7	60
gauss0077	77	321/11	214/7	198/9	723	198/9	1510	198/9	122
gauss0170	170	735/17	401/13	396/15	1837	396/15	5360	396/15	357
gauss0252	252	1060/20	573/17	561/20	2971	561/20	10120	562/20	599
gauss0324	324	1300/20	714/20	701/20	4094	701/20	15029	705/20	910
gauss0464	464	1860/20	1031/20	1071/20	6619	1031/20	25329	1031/20	1524
gauss0819	819	3918/20	2209/20	2195/20	13943	2115/20	57567	2195/20	3415
gauss1080	1080	5514/20	3118/20	3117/20	20905	3070/20	88361	3117/20	5033
gauss1175	1175	5991/20	3493/20	3508/20	23708	3464/20	98796	3493/20	5864

Tabela D.1: Resultados obtidos para GADs com topologia *eliminação de Gauss* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (1, 10)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	*n*	*ms/pu*	*ms/pu*	*ms/pu*	*te*	*ms/pu*	*te*	*ms/pu*	*te*
gauss0009	9	22/2	20/2	20/2	61	20/2	77	20/2	10
gauss0014	14	36/3	33/3	33/4	95	33/4	124	33/2	16
gauss0035	35	104/7	78/5	77/6	276	77/5	415	77/6	45
gauss0044	44	142/8	97/5	96/5	361	96/5	666	96/8	59
gauss0077	77	280/11	163/7	159/10	700	159/7	1436	159/10	122
gauss0170	170	664/17	343/13	339/16	1850	339/16	5417	339/16	348
gauss0252	252	974/20	503/17	499/19	2993	499/17	9906	500/20	596
gauss0324	324	1199/20	644/20	646/20	4137	644/20	15074	641/20	884
gauss0464	464	1739/20	1001/20	1000/20	6627	996/20	26039	988/20	1505
gauss0819	819	3747/20	2148/20	2094/20	14358	2080/20	60809	2094/20	3394
gauss1080	1080	5313/20	3049/20	3047/20	20714	2994/20	69057	3026/20	5036
gauss1175	1175	5785/20	3410/20	3437/20	23154	3340/20	103459	3373/20	5850

Tabela D.2: Resultados obtidos para GADs com topologia *eliminação de Gauss* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (1, 5)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	*n*	*ms/pu*	*ms/pu*	*ms/pu*	*te*	*ms/pu*	*te*	*ms/pu*	*te*
gauss0009	9	17/3	15/3	15/3	63	15/3	78	15/3	9
gauss0014	14	27/4	24/4	24/4	96	24/4	122	24/4	15
gauss0035	35	83/7	63/5	63/5	285	63/5	431	63/5	43
gauss0044	44	117/8	80/5	80/6	378	80/5	598	80/5	58
gauss0077	77	243/11	143/7	143/9	724	143/10	1491	143/7	118
gauss0170	170	603/17	323/13	323/15	1922	323/13	5455	323/13	334
gauss0252	252	901/20	483/17	483/19	3127	483/17	9794	483/17	590
gauss0324	324	1114/20	624/20	625/20	4316	624/20	15269	624/20	890
gauss0464	464	1638/20	955/20	957/20	6862	943/20	26522	949/20	1489
gauss0819	819	3606/20	2073/20	2039/20	14714	1992/20	59513	2019/20	3360
gauss1080	1080	5148/20	2978/20	3012/20	21261	2930/20	69714	2919/20	4921
gauss1175	1175	5616/20	3324/20	3347/20	24368	3280/20	83433	3276/20	5836

Tabela D.3: Resultados obtidos para GADs com topologia *eliminação de Gauss* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (1, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	*n*	*ms/pu*	*ms/pu*	*ms/pu*	*te*	*ms/pu*	*te*	*ms/pu*	*te*
di0025	25	25/1	25/1	25/1	187	25/1	203	25/1	32
di0036	36	36/1	36/1	36/1	267	31/2	354	35/2	49
di0064	64	64/1	64/1	53/2	518	49/2	690	55/2	99
di0100	100	100/1	100/1	77/3	831	67/2	1275	74/2	176
di0144	144	133/2	143/2	100/3	1249	86/3	2580	97/2	253
di0225	225	175/2	209/2	142/6	2042	114/3	2394	126/4	438
di0256	256	191/2	231/2	153/5	2374	127/3	4231	151/4	525
di0400	400	265/2	319/2	219/9	3806	172/5	4854	179/5	954
di1024	1024	496/4	583/3	402/13	10875	303/6	17123	320/7	3766

Tabela D.4: Resultados obtidos para GADs com topologia *árvore binária invertida* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (1, 10)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
intree015	15	34/8	25/5	24/5	88	16/5	197	16/5	19
intree031	31	47/16	36/11	28/11	214	27/11	953	27/12	56
intree063	63	49/20	40/20	38/20	582	37/20	1001	38/20	162
intree127	127	53/20	51/20	49/20	1769	44/20	9526	49/20	511
intree255	255	70/20	60/20	60/20	6433	55/20	11101	58/20	1794
intree511	511	95/20	81/20	81/20	23969	74/20	44561	79/20	6763

Tabela D.5: Resultados obtidos para GADs com topologia *árvore binária* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (1, 10)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
outtree015	15	15/1	15/1	14/2	91	14/2	177	14/2	19
outtree031	31	26/3	26/2	19/5	213	19/3	399	20/3	44
outtree063	63	36/5	37/4	25/6	545	25/6	928	27/5	106
outtree127	127	46/9	49/11	33/11	1554	33/11	2190	33/11	264
outtree255	255	56/17	62/20	44/20	4832	44/20	4451	44/20	689
outtree511	511	73/20	83/20	63/20	16953	62/20	14481	63/20	2189

Tabela D.6: Resultados obtidos para GADs com topologia *randômico* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (1, 10)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
ran080	80	60/20	45/20	46/20	1184	40/20	2051	43/20	204
ran098	98	54/20	49/19	52/20	1575	49/19	3511	49/19	240
ran108	108	69/20	55/20	54/20	1714	49/20	2450	52/20	268
ran124	124	26/20	28/20	27/20	2263	26/20	13112	24/20	482
ran135	135	64/20	51/20	51/20	2365	48/20	5281	50/20	389
ran140	140	52/20	51/20	52/20	2434	50/20	5572	50/20	378
ran152	152	56/20	52/20	53/20	2912	49/20	5000	51/20	418
ran153	153	62/20	61/20	59/20	2814	51/20	7393	53/20	440
ran154	154	49/20	40/20	40/20	3090	39/20	6945	39/20	582
ran170	170	37/20	28/20	26/20	4336	26/20	26949	26/20	877
ran186	186	57/20	37/20	39/20	4218	34/20	17414	36/20	738
ran223	223	43/20	32/20	31/20	6584	29/20	27896	30/20	1275
ran234	234	69/20	67/20	63/20	5790	54/20	15118	57/20	915
ran256	256	53/20	46/20	43/20	8073	41/20	31025	41/20	1481
ran286	286	101/20	70/20	78/20	7360	68/20	7956	68/20	1193
ran298	298	53/20	48/20	47/20	10030	42/20	12475	43/20	1689
ran310	310	78/20	84/20	74/20	8009	63/20	21475	65/20	1424
ran357	357	55/20	46/20	46/20	16299	41/20	73524	41/20	3018
ran364	364	107/20	64/20	65/20	11383	57/20	21325	59/20	1918
ran510	510	99/20	97/20	88/20	20927	76/20	21469	76/20	3150
ran546	546	158/20	86/20	91/20	22486	81/20	49374	80/20	3487

Tabela D.7: Resultados obtidos para GADs com topologia *randômico* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (1, 10)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
di0025	25	25/1	25/1	22/2	187	22/2	200	22/2	32
di0036	36	36/2	36/1	28/2	268	26/2	402	28/2	45
di0064	64	51/2	60/2	42/3	507	40/2	634	42/3	86
di0100	100	70/2	84/2	60/6	831	55/3	1041	58/4	142
di0144	144	89/3	108/2	78/6	1223	67/3	1580	73/4	226
di0225	225	116/4	144/3	105/6	1984	89/6	2718	94/5	405
di0256	256	132/4	156/3	114/7	2346	100/4	3988	104/5	476
di0400	400	172/5	204/4	151/10	3844	131/5	7178	138/5	902
di1024	1024	316/12	348/6	276/18	10991	234/8	29706	246/9	3523

Tabela D.8: Resultados obtidos para GADs com topologia *árvore binária invertida* em um sistema dual não restrito com $20/5$ processadores onde $(me, mc) = (1, 5)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
intree015	15	19/8	15/5	14/5	88	14/5	264	13/4	20
intree031	31	27/16	21/11	18/11	210	17/11	732	17/9	56
intree063	63	29/20	25/20	24/20	570	22/20	2891	24/20	162
intree127	127	39/20	32/20	31/20	1807	29/20	9563	30/20	516
intree255	255	47/20	43/20	41/20	6244	39/20	9545	40/20	1808
intree511	511	66/20	63/20	62/20	24327	58/20	41364	60/20	6739

Tabela D.9: Resultados obtidos para GADs com topologia *árvore binária* em um sistema dual não restrito com $20/5$ processadores onde $(me, mc) = (1, 5)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
outtree015	15	13/3	13/2	10/3	92	10/3	152	10/3	18
outtree031	31	18/5	19/4	14/5	210	14/4	406	14/5	40
outtree063	63	23/8	26/6	20/7	533	19/6	672	20/7	95
outtree127	127	29/19	33/17	25/20	1455	25/14	1609	25/20	246
outtree255	255	37/20	44/20	35/20	4856	34/20	4276	35/20	636
outtree511	511	57/20	64/20	56/20	16686	54/20	14891	56/20	1943

Tabela D.10: Resultados obtidos para GADs com topologia *randômico* em um sistema dual não restrito com $20/5$ processadores onde $(me, mc) = (1, 5)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
ran080	80	33/20	29/20	27/20	1200	27/20	2013	27/20	201
ran098	98	38/20	35/19	31/20	1582	30/20	2765	31/20	234
ran108	108	42/20	36/20	33/20	1694	31/20	1968	32/20	263
ran124	124	21/20	18/20	17/20	2372	15/20	9106	16/20	480
ran135	135	40/20	35/20	32/20	2382	30/20	5363	31/20	381
ran140	140	34/20	33/20	34/20	2448	33/20	6520	31/20	376
ran152	152	37/20	34/20	34/20	2914	31/20	5199	32/20	407
ran153	153	41/20	44/20	37/20	2865	32/20	6621	34/20	423
ran154	154	31/20	28/20	25/20	3043	24/20	13834	24/20	571
ran170	170	30/20	20/20	18/20	4210	17/20	27422	17/20	861
ran186	186	29/20	27/20	24/20	4282	22/20	18073	23/20	734
ran223	223	33/20	25/20	22/20	6611	20/20	37095	21/20	1260
ran234	234	52/20	44/20	39/20	5676	36/20	14342	36/20	901
ran256	256	36/20	34/20	29/20	7939	26/20	40923	27/20	1456
ran286	286	58/20	48/20	48/20	7350	45/20	17882	46/20	1153
ran298	298	40/20	34/20	32/20	9899	29/20	45977	30/20	1680
ran310	310	61/20	58/20	46/20	8053	43/20	30948	44/20	1396
ran357	357	41/20	34/20	32/20	16035	30/20	67028	30/20	2982
ran364	364	74/20	51/20	43/20	11263	39/20	33928	40/20	1907
ran510	510	78/20	63/20	60/20	20796	53/20	46616	53/20	3137
ran546	546	85/20	61/20	61/20	23330	55/20	80866	56/20	3522

Tabela D.11: Resultados obtidos para GADs com topologia *randômico* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (1, 5)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
di0025	25	13/3	16/3	14/3	188	13/3	219	13/3	26
di0036	36	16/4	20/3	18/4	276	16/3	333	16/4	39
di0064	64	22/5	28/4	25/6	519	22/5	690	22/5	74
di0100	100	35/6	36/5	33/6	839	33/6	1051	30/6	126
di0144	144	41/6	44/6	42/7	1262	39/7	1630	37/7	199
di0225	225	52/7	59/7	55/9	2028	52/7	2900	50/8	363
di0256	256	57/8	65/7	60/11	2340	57/8	3201	55/9	437
di0400	400	75/11	89/9	78/14	3878	74/11	5998	74/14	803
di1024	1024	135/19	161/20	140/20	11288	132/20	19137	135/20	2813

Tabela D.12: Resultados obtidos para GADs com topologia *árvore binária invertida* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (1, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
intree015	15	7/8	7/5	7/5	90	7/5	252	7/5	21
intree031	31	11/16	10/11	9/11	212	9/11	627	9/11	58
intree063	63	13/20	13/20	12/20	588	12/20	2518	12/20	164
intree127	127	20/20	18/20	17/20	1787	17/20	13356	17/20	522
intree255	255	29/20	29/20	27/20	6298	27/20	8581	27/20	1845
intree511	511	49/20	49/20	48/20	24590	47/20	93831	48/20	6902

Tabela D.13: Resultados obtidos para GADs com topologia *árvore binária* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (1, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	*n*	*ms/pu*	*ms/pu*	*ms/pu*	*te*	*ms/pu*	*te*	*ms/pu*	*te*
outtree015	15	7/6	7/4	7/5	93	7/3	102	7/4	16
outtree031	31	10/11	10/6	9/8	212	9/5	266	9/5	40
outtree063	63	13/18	13/17	12/17	533	11/17	644	12/15	97
outtree127	127	17/20	19/20	17/20	1517	17/20	1670	16/20	255
outtree255	255	27/20	29/20	27/20	4567	26/20	3833	26/20	695
outtree511	511	47/20	49/20	49/20	16606	47/20	12183	47/20	2086

Tabela D.14: Resultados obtidos para GADs com topologia *randômico* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (1, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	*n*	*ms/pu*	*ms/pu*	*ms/pu*	*te*	*ms/pu*	*te*	*ms/pu*	*te*
ran080	80	17/20	16/20	14/20	1219	14/20	2034	14/20	205
ran098	98	20/20	18/19	15/20	1589	15/20	2626	15/20	227
ran108	108	24/20	20/20	16/20	1753	16/20	2339	15/20	262
ran124	124	14/20	11/20	11/20	2253	10/20	13256	10/20	501
ran135	135	26/20	18/20	15/20	2417	15/20	6578	15/20	388
ran140	140	24/20	20/20	17/20	2529	17/20	4755	16/20	381
ran152	152	23/20	19/20	17/20	3026	17/20	3857	16/20	419
ran153	153	26/20	20/20	17/20	2888	17/20	5513	17/20	431
ran154	154	23/20	15/20	14/20	3045	13/20	10029	14/20	584
ran170	170	21/20	14/20	14/20	4181	14/20	27120	14/20	897
ran186	186	23/20	16/20	16/20	4283	16/20	13134	16/20	762
ran223	223	24/20	19/20	19/20	6665	18/20	18656	18/20	1312
ran234	234	28/20	24/20	22/20	5954	22/20	10171	21/20	956
ran256	256	27/20	22/20	22/20	7991	22/20	40494	22/20	1528
ran286	286	37/20	29/20	28/20	7444	27/20	7733	27/20	1233
ran298	298	32/20	25/20	25/20	10029	25/20	17205	24/20	1790
ran310	310	37/20	30/20	29/20	8135	28/20	15437	28/20	1467
ran357	357	35/20	30/20	30/20	16213	30/20	64575	30/20	3139
ran364	364	37/20	32/20	31/20	11589	31/20	11775	31/20	2030
ran510	510	49/20	44/20	43/20	21699	43/20	30287	42/20	3282
ran546	546	60/20	46/20	46/20	23495	46/20	28171	45/20	3634

Tabela D.15: Resultados obtidos para GADs com topologia *randômico* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (1, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	*n*	*ms/pu*	*ms/pu*	*ms/pu*	*te*	*ms/pu*	*te*	*ms/pu*	*te*
di0025	25	49/5	49/5	49/5	189	49/5	210	49/5	24
di0036	36	90/6	60/5	60/5	276	60/5	360	60/5	36
di0064	64	130/6	88/5	87/6	523	87/6	644	87/6	71
di0100	100	171/7	124/5	120/9	830	119/8	1063	120/9	123
di0144	144	212/8	167/6	155/10	1232	153/9	1668	157/12	200
di0225	225	274/10	233/9	211/13	2040	206/13	2806	211/13	372
di0256	256	294/10	255/10	229/13	2386	225/13	3623	229/13	456
di0400	400	376/12	343/14	307/19	3925	299/18	6249	307/19	854
di1024	1024	629/20	621/20	589/20	11312	569/20	19235	581/20	3091

Tabela D.16: Resultados obtidos para GADs com topologia *árvore binária invertida* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (5, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
intree015	15	40/8	26/5	26/5	90	26/5	271	26/5	21
intree031	31	47/16	36/11	35/13	212	35/13	984	35/13	58
intree063	63	61/20	51/20	48/20	573	48/20	3746	48/20	164
intree127	127	100/20	75/20	73/20	1828	73/20	11515	73/20	506
intree255	255	141/20	126/20	124/20	6267	124/20	15333	124/20	1795
intree511	511	250/20	230/20	227/20	24684	227/20	40754	227/20	6790

Tabela D.17: Resultados obtidos para GADs com topologia *árvore binária* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (5, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
outtree015	15	28/8	27/5	26/5	95	26/5	137	26/5	16
outtree031	31	38/13	38/9	35/14	208	35/13	223	35/14	38
outtree063	63	48/20	52/20	47/20	543	47/20	533	47/20	88
outtree127	127	76/20	76/20	73/20	1512	73/20	1269	72/20	221
outtree255	255	126/20	128/20	126/20	4557	126/20	3996	123/20	597
outtree511	511	227/20	230/20	230/20	20660	227/20	11807	227/20	1883

Tabela D.18: Resultados obtidos para GADs com topologia *randômico* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (5, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
ran080	80	71/20	55/20	54/20	1192	54/20	2601	54/20	200
ran098	98	81/20	67/20	58/20	1611	58/20	2237	60/20	224
ran108	108	98/20	71/20	64/20	1709	64/20	3453	64/20	255
ran124	124	71/20	51/20	51/20	2285	51/20	12512	51/20	486
ran135	135	98/20	72/20	66/20	2351	65/20	5050	64/20	383
ran140	140	93/20	77/20	72/20	2444	72/20	4768	72/20	370
ran152	152	91/20	78/20	73/20	2932	73/20	4764	72/20	406
ran153	153	86/20	81/20	73/20	2888	73/20	5255	74/20	424
ran154	154	89/20	67/20	65/20	3076	65/20	10670	66/20	566
ran170	170	91/20	70/20	70/20	4303	70/20	26223	70/20	872
ran186	186	104/20	80/20	80/20	4270	80/20	16023	80/20	737
ran223	223	111/20	91/20	91/20	6668	91/20	31255	91/20	1283
ran234	234	133/20	106/20	102/20	5874	102/20	10187	101/20	926
ran256	256	131/20	110/20	110/20	7963	110/20	30234	110/20	1502
ran286	286	164/20	131/20	127/20	7441	127/20	13306	124/20	1193
ran298	298	150/20	121/20	121/20	10160	121/20	19801	121/20	1751
ran310	310	172/20	137/20	135/20	8233	135/20	18856	134/20	1404
ran357	357	159/20	150/20	150/20	16344	150/20	42398	150/20	3042
ran364	364	179/20	152/20	152/20	11484	152/20	12914	151/20	1969
ran510	510	231/20	210/20	210/20	21665	210/20	24995	210/20	3218
ran546	546	264/20	222/20	222/20	24250	222/20	25867	221/20	3591

Tabela D.19: Resultados obtidos para GADs com topologia *randômico* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (5, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	*n*	*ms/pu*	*ms/pu*	*ms/pu*	*te*	*ms/pu*	*te*	*ms/pu*	*te*
di0025	25	94/5	94/5	94/5	191	94/5	226	94/5	23
di0036	36	175/6	120/5	120/5	275	120/5	359	120/5	35
di0064	64	256/7	173/5	172/6	510	172/6	666	172/6	71
di0100	100	337/8	244/5	233/8	835	231/9	1120	233/8	124
di0144	144	418/9	324/7	301/12	1242	296/10	1723	301/12	199
di0225	225	539/10	444/10	403/13	2065	403/13	3171	403/13	374
di0256	256	580/11	490/10	445/14	2351	436/14	3367	445/14	464
di0400	400	741/12	654/14	590/17	3906	585/18	5700	590/17	860
di1024	1024	1239/20	1184/20	1158/20	11551	1122/20	20329	1130/20	3254

Tabela D.20: Resultados obtidos para GADs com topologia *árvore binária invertida* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (10, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	*n*	*ms/pu*	*ms/pu*	*ms/pu*	*te*	*ms/pu*	*te*	*ms/pu*	*te*
intree015	15	80/8	51/5	51/5	90	51/5	272	51/5	21
intree031	31	92/16	71/11	70/13	214	70/13	1017	70/13	58
intree063	63	121/20	101/20	92/20	588	92/20	3543	92/20	164
intree127	127	200/20	150/20	143/20	1841	143/20	11996	143/20	507
intree255	255	281/20	251/20	250/20	6264	250/20	26909	243/20	1807
intree511	511	500/20	460/20	453/20	23910	453/20	80992	452/20	6799

Tabela D.21: Resultados obtidos para GADs com topologia *árvore binária* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (10, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	*n*	*ms/pu*	*ms/pu*	*ms/pu*	*te*	*ms/pu*	*te*	*ms/pu*	*te*
outtree015	15	53/8	52/5	51/5	93	51/5	142	51/5	16
outtree031	31	73/13	73/9	70/13	211	70/13	235	70/14	38
outtree063	63	93/20	102/20	92/20	535	92/20	551	92/20	88
outtree127	127	151/20	151/20	143/20	1515	143/20	1339	142/20	219
outtree255	255	251/20	253/20	252/20	4791	251/20	3580	243/20	601
outtree511	511	452/20	460/20	454/20	21304	452/20	12094	452/20	1894

Tabela D.22: Resultados obtidos para GADs com topologia *randômico* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (10, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	***n***	***ms/pu***	***ms/pu***	***ms/pu***	***te***	***ms/pu***	***te***	***ms/pu***	***te***
ran080	80	141/20	105/20	104/20	1173	104/20	1993	104/20	200
ran098	98	161/20	132/20	115/20	1599	114/20	2702	115/20	228
ran108	108	193/20	141/20	125/20	1693	124/20	3979	125/20	254
ran124	124	141/20	101/20	101/20	2289	101/20	12820	101/20	489
ran135	135	193/20	142/20	124/20	2389	124/20	6838	124/20	377
ran140	140	183/20	152/20	141/20	2431	141/20	5679	135/20	374
ran152	152	181/20	153/20	143/20	2918	143/20	5381	143/20	406
ran153	153	171/20	161/20	144/20	2856	144/20	5280	144/20	424
ran154	154	174/20	132/20	131/20	3081	131/20	13264	131/20	576
ran170	170	181/20	140/20	140/20	4286	140/20	26740	140/20	876
ran186	186	204/20	160/20	160/20	4275	160/20	12824	160/20	745
ran223	223	221/20	181/20	181/20	6771	181/20	26723	181/20	1283
ran234	234	258/20	210/20	202/20	5763	202/20	9807	201/20	916
ran256	256	261/20	220/20	220/20	8010	220/20	31427	220/20	1511
ran286	286	324/20	261/20	252/20	7411	252/20	13671	246/20	1192
ran298	298	300/20	241/20	241/20	10097	241/20	23554	241/20	1725
ran310	310	342/20	272/20	271/20	8108	271/20	21627	263/20	1418
ran357	357	314/20	300/20	300/20	16049	300/20	48742	300/20	3048
ran364	364	354/20	302/20	301/20	11765	301/20	32484	301/20	1949
ran510	510	461/20	420/20	420/20	22451	420/20	21990	420/20	3155
ran546	546	524/20	442/20	442/20	25112	442/20	50742	441/20	3558

Tabela D.23: Resultados obtidos para GADs com topologia *randômico* em um sistema dual não restrito com 20/5 processadores onde $(me, mc) = (10, 1)$.

Apêndice E

Resultados para Ambientes Restritos

Este apêndice contém alguns dos resultados obtidos pelas heurísticas analisadas neste trabalho. Em cada tabela, as colunas **nome** e **n** indicam, respectivamente, o nome e o número de tarefas do GAD escalonado. As colunas **ms/pu** representam, respectivamente, o *makespan* obtido e o número de processadores utilizados pelo escalonamento produzido pela heurística indicada. As colunas **te** mostram o tempo de execução do algoritmo em milisegundos. As colunas **pm** indicam o percentual de melhora do *makespan* obtido pelas heurísticas propostas HTSGA e HTSG em relação ao melhor *makespan* produzido por ETF, HEFT e PSGA.

grafo		**ETF**	**HEFT**	**PSGA**		**HTSGA**		**HTSG**	
nome	***n***	***ms/pu***	***ms/pu***	***ms/pu***	***te***	***ms/pu***	***te***	***ms/pu***	***te***
gauss0009	9	23/1	23/1	23/1	30	23/1	32	23/1	6
gauss0014	14	42/2	40/2	40/2	46	39/2	50	39/2	9
gauss0035	35	150/6	107/5	96/6	135	96/5	233	96/6	25
gauss0044	44	183/6	132/6	119/6	179	117/6	298	119/6	34
gauss0077	77	349/6	223/6	215/6	354	208/6	620	210/6	66
gauss0170	170	864/6	590/6	583/6	1000	568/6	1419	582/6	186
gauss0252	252	1458/6	1006/6	975/6	1692	975/6	2962	975/6	329
gauss0324	324	1986/6	1420/6	1405/6	2392	1405/6	3705	1405/6	500
gauss0464	464	3225/6	2381/6	2351/6	3904	2351/6	8103	2358/6	903
gauss0819	819	6953/6	5461/6	5443/6	8731	5443/6	14338	5443/6	2325
gauss1080	1080	10207/6	8227/6	8232/6	13338	8227/6	28203	8227/6	3798
gauss1175	1175	11474/6	9358/6	9332/6	15895	9332/6	28610	9332/6	4441

Tabela E.1: Resultados obtidos para GADs com topologia *eliminação de Gauss* em um sistema dual restrito com $6/2$ processadores onde $(me, mc) = (1, 10)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
gauss0009	9	22/2	20/2	20/2	29	20/2	40	20/2	6
gauss0014	14	44/3	33/2	33/2	48	33/2	69	32/3	9
gauss0035	35	122/6	78/6	78/6	134	78/6	227	78/6	25
gauss0044	44	152/6	102/6	98/6	179	98/6	306	98/6	33
gauss0077	77	307/6	197/6	189/6	349	189/6	615	189/6	67
gauss0170	170	802/6	555/6	546/6	987	543/6	1658	550/6	186
gauss0252	252	1376/6	959/6	953/6	1693	953/6	3250	953/6	329
gauss0324	324	1894/6	1377/6	1368/6	2358	1368/6	4273	1368/6	498
gauss0464	464	3113/6	2330/6	2320/6	3910	2320/6	4829	2320/6	897
gauss0819	819	6806/6	5423/6	5419/6	8916	5419/6	16543	5419/6	2314
gauss1080	1080	10035/6	8203/6	8197/6	13605	8197/6	20239	8203/6	3800
gauss1175	1175	11297/6	9315/6	9308/6	15589	9308/6	31156	9303/6	4453

Tabela E.2: Resultados obtidos para GADs com topologia *eliminação de Gauss* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 5)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
gauss0009	9	23/3	15/2	15/2	38	15/2	49	15/2	6
gauss0014	14	35/4	24/3	24/3	61	24/3	78	24/3	9
gauss0035	35	105/6	63/6	63/6	168	63/6	238	63/6	26
gauss0044	44	131/6	83/6	83/6	219	82/6	320	83/6	34
gauss0077	77	278/6	186/6	173/6	437	173/6	666	173/6	68
gauss0170	170	757/6	533/6	523/6	1230	523/6	2009	523/6	193
gauss0252	252	1315/6	936/6	929/6	2069	929/6	2592	929/6	346
gauss0324	324	1825/6	1354/6	1351/6	2976	1351/6	4920	1349/6	531
gauss0464	464	3028/6	2309/6	2304/6	4754	2304/6	7707	2304/6	960
gauss0819	819	6693/6	5399/6	5401/6	10556	5399/6	19344	5398/6	2526
gauss1080	1080	9902/6	8188/6	8184/6	16052	8184/6	34296	8184/6	4132
gauss1175	1175	11160/6	9296/6	9289/6	18783	9289/6	35332	9289/6	4841

Tabela E.3: Resultados obtidos para GADs com topologia *eliminação de Gauss* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
di0025	25	25/1	25/1	25/1	113	25/1	129	25/1	19
di0036	36	36/1	36/1	34/2	162	31/2	186	34/2	29
di0064	64	64/1	64/1	53/2	316	47/2	409	52/2	62
di0100	100	100/1	100/1	79/3	510	68/2	681	75/2	109
di0144	144	133/2	143/2	107/4	766	89/3	847	100/2	183
di0225	225	175/2	209/2	156/6	1272	126/3	1922	139/5	337
di0256	256	191/2	231/2	173/6	1488	134/3	1868	146/4	408
di0400	400	265/2	319/2	236/6	2454	188/6	4311	200/6	878
di1024	1024	493/6	583/4	487/6	7905	364/6	11763	419/6	4028

Tabela E.4: Resultados obtidos para GADs com topologia *árvore binária invertida* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 10)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	***n***	***ms/pu***	***ms/pu***	***ms/pu***	***te***	***ms/pu***	***te***	***ms/pu***	***te***
intree015	15	36/6	25/6	25/6	61	25/6	113	17/4	11
intree031	31	48/6	29/6	29/6	159	27/6	260	28/6	27
intree063	63	60/6	37/6	37/6	465	34/6	909	36/6	72
intree127	127	76/6	54/6	52/6	1560	49/6	1754	52/6	219
intree255	255	98/6	86/6	82/6	5722	81/6	6465	82/6	763
intree511	511	172/6	150/6	148/6	22646	145/6	24753	144/6	2988

Tabela E.5: Resultados obtidos para GADs com topologia *árvore binária* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 10)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	***n***	***ms/pu***	***ms/pu***	***ms/pu***	***te***	***ms/pu***	***te***	***ms/pu***	***te***
outtree015	15	15/1	15/1	14/2	64	14/2	78	14/2	11
outtree031	31	27/4	26/2	20/5	146	19/4	208	20/5	26
outtree063	63	36/6	38/6	29/6	435	27/6	443	29/6	60
outtree127	127	50/6	54/6	43/6	1261	42/6	1275	46/6	146
outtree255	255	81/6	86/6	76/6	3978	74/6	3354	76/6	474
outtree511	511	145/6	150/6	140/6	14620	139/6	13105	140/6	1565

Tabela E.6: Resultados obtidos para GADs com topologia *randômico* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 10)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	***n***	***ms/pu***	***ms/pu***	***ms/pu***	***te***	***ms/pu***	***te***	***ms/pu***	***te***
ran080	80	50/6	43/6	47/6	808	43/6	1169	41/6	103
ran098	98	75/6	51/6	51/6	1097	48/6	1093	48/6	130
ran108	108	66/6	58/6	60/6	1135	51/6	1571	50/6	148
ran124	124	46/6	34/6	36/6	2017	32/6	2100	32/6	229
ran135	135	72/6	56/6	57/6	1772	50/6	1780	51/6	215
ran140	140	75/6	64/6	66/6	1749	52/6	1790	54/6	217
ran152	152	70/6	61/6	62/6	2165	52/6	2193	55/6	254
ran153	153	74/6	60/6	68/6	2084	54/6	2303	57/6	255
ran154	154	60/6	44/6	50/6	2600	43/6	2610	44/6	300
ran170	170	55/6	43/6	46/6	3852	43/6	5571	43/6	420
ran186	186	58/6	52/6	56/6	3699	47/6	3560	48/6	396
ran223	223	68/6	56/6	59/6	6155	56/6	8086	56/6	640
ran234	234	86/6	75/6	84/6	4784	69/6	4554	69/6	532
ran256	256	91/6	65/6	69/6	7289	64/6	7456	64/6	785
ran286	286	109/6	88/6	99/6	6190	85/6	5793	86/6	732
ran298	298	93/6	78/6	79/6	9110	75/6	9088	76/6	936
ran310	310	116/6	94/6	103/6	7126	89/6	6873	90/6	865
ran357	357	108/6	90/6	90/6	15304	90/6	19837	90/6	1584
ran364	364	119/6	101/6	104/6	10555	97/6	9892	97/6	1158
ran510	510	159/6	138/6	141/6	18927	134/6	17608	134/6	2075
ran546	546	172/6	145/6	151/6	21674	145/6	21250	141/6	2253

Tabela E.7: Resultados obtidos para GADs com topologia *randômico* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 10)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
di0025	25	25/1	25/1	22/2	114	22/2	130	22/2	18
di0036	36	36/2	36/1	28/2	161	26/2	195	29/2	27
di0064	64	51/2	60/2	45/3	311	40/2	475	45/2	54
di0100	100	70/2	84/2	62/4	506	57/3	800	59/3	100
di0144	144	98/4	108/2	86/5	765	74/3	1201	77/5	167
di0225	225	136/6	144/3	117/6	1286	102/5	2159	108/6	314
di0256	256	140/6	156/4	130/6	1468	109/5	3139	119/6	385
di0400	400	199/6	204/5	180/6	2533	148/6	4004	164/6	735
di1024	1024	356/6	356/6	366/6	8095	301/6	11618	324/6	3321

Tabela E.8: Resultados obtidos para GADs com topologia *árvore binária invertida* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 5)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
intree015	15	21/6	15/6	15/6	62	15/6	100	15/5	11
intree031	31	28/6	19/6	18/6	157	17/6	237	18/6	27
intree063	63	34/6	28/6	27/6	472	25/6	530	26/6	72
intree127	127	50/6	44/6	43/6	1583	41/6	1800	42/6	219
intree255	255	82/6	76/6	74/6	5762	72/6	6744	73/6	766
intree511	511	146/6	140/6	138/6	22720	135/6	25601	137/6	2991

Tabela E.9: Resultados obtidos para GADs com topologia *árvore binária* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 5)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
outtree015	15	13/3	13/2	11/2	64	10/4	80	11/2	10
outtree031	31	19/6	20/4	16/6	151	15/6	184	16/6	24
outtree063	63	26/6	28/6	23/6	416	23/6	406	23/6	56
outtree127	127	41/6	44/6	39/6	1259	39/6	1135	39/6	147
outtree255	255	74/6	76/6	72/6	4008	71/6	3517	72/6	453
outtree511	511	138/6	140/6	136/6	15653	135/6	10581	137/6	1606

Tabela E.10: Resultados obtidos para GADs com topologia *randômico* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 5)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	***n***	***ms/pu***	***ms/pu***	***ms/pu***	***te***	***ms/pu***	***te***	***ms/pu***	***te***
ran080	80	36/6	34/6	31/6	812	29/6	1254	29/6	102
ran098	98	49/6	40/6	35/6	1064	33/6	1280	34/6	126
ran108	108	43/6	41/6	40/6	1153	36/6	1589	36/6	145
ran124	124	37/6	32/6	32/6	2041	32/6	2164	32/6	229
ran135	135	50/6	39/6	39/6	1798	37/6	1713	37/6	215
ran140	140	55/6	48/6	43/6	1783	40/6	2034	40/6	212
ran152	152	50/6	47/6	46/6	2134	44/6	2271	43/6	244
ran153	153	51/6	48/6	48/6	2132	42/6	2268	43/6	245
ran154	154	51/6	40/6	40/6	2577	39/6	3229	39/6	293
ran170	170	52/6	43/6	44/6	3775	43/6	8075	43/6	419
ran186	186	60/6	47/6	48/6	3787	47/6	4871	47/6	402
ran223	223	66/6	56/6	58/6	6129	56/6	7095	56/6	637
ran234	234	75/6	65/6	64/6	4626	60/6	5238	61/6	533
ran256	256	76/6	65/6	65/6	7325	64/6	7304	64/6	765
ran286	286	85/6	79/6	78/6	6304	76/6	5693	75/6	729
ran298	298	80/6	75/6	76/6	9222	75/6	9466	75/6	945
ran310	310	91/6	85/6	82/6	7137	81/6	8097	81/6	826
ran357	357	104/6	90/6	90/6	15068	90/6	15774	90/6	1529
ran364	364	104/6	93/6	92/6	10324	92/6	13540	92/6	1178
ran510	510	137/6	128/6	128/6	18845	128/6	16752	128/6	2061
ran546	546	152/6	137/6	138/6	21539	137/6	20810	137/6	2264

Tabela E.11: Resultados obtidos para GADs com topologia *randômico* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 5)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	***n***	***ms/pu***	***ms/pu***	***ms/pu***	***te***	***ms/pu***	***te***	***ms/pu***	***te***
di0025	25	17/3	16/3	15/3	110	14/3	126	14/3	16
di0036	36	20/3	21/3	18/4	161	18/3	200	18/4	24
di0064	64	30/5	34/3	29/6	303	27/5	352	29/5	48
di0100	100	40/6	46/5	39/6	493	37/6	656	39/6	82
di0144	144	52/6	58/6	51/6	753	49/6	1035	50/6	136
di0225	225	72/6	79/6	74/6	1282	72/6	1528	69/6	252
di0256	256	78/6	87/6	82/6	1496	78/6	1678	77/6	311
di0400	400	116/6	123/6	118/6	2704	113/6	2847	113/6	621
di1024	1024	272/6	279/6	277/6	8815	270/6	12646	269/6	3046

Tabela E.12: Resultados obtidos para GADs com topologia *árvore binária invertida* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	***n***	***ms/pu***	***ms/pu***	***ms/pu***	***te***	***ms/pu***	***te***	***ms/pu***	***te***
intree015	15	9/6	8/6	7/6	62	7/6	95	7/5	11
intree031	31	13/6	12/6	11/6	156	11/6	321	11/6	27
intree063	63	21/6	20/6	19/6	457	19/6	648	19/6	70
intree127	127	37/6	36/6	35/6	1545	35/6	2170	35/6	217
intree255	255	69/6	68/6	67/6	5588	67/6	7122	67/6	770
intree511	511	133/6	132/6	131/6	22745	131/6	33144	131/6	3041

Tabela E.13: Resultados obtidos para GADs com topologia *árvore binária* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	*n*	*ms/pu*	*ms/pu*	*ms/pu*	*te*	*ms/pu*	*te*	*ms/pu*	*te*
outtree015	15	8/5	8/4	7/6	61	7/4	63	7/4	10
outtree031	31	11/6	12/6	11/6	154	11/6	156	11/6	23
outtree063	63	19/6	20/6	20/6	401	19/6	406	19/6	55
outtree127	127	35/6	36/6	36/6	1418	35/6	1137	35/6	153
outtree255	255	67/6	68/6	68/6	5363	67/6	3429	67/6	471
outtree511	511	131/6	132/6	132/6	18607	131/6	10592	131/6	1632

Tabela E.14: Resultados obtidos para GADs com topologia *randômico* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	*n*	*ms/pu*	*ms/pu*	*ms/pu*	*te*	*ms/pu*	*te*	*ms/pu*	*te*
ran080	80	28/6	22/6	21/6	825	21/6	1072	21/6	100
ran098	98	27/6	27/6	26/6	1066	25/6	1223	25/6	126
ran108	108	31/6	30/6	29/6	1164	29/6	1263	29/6	143
ran124	124	34/6	32/6	32/6	2164	32/6	2821	32/6	232
ran135	135	38/6	34/6	34/6	1793	34/6	2140	34/6	209
ran140	140	39/6	36/6	36/6	1764	36/6	1976	36/6	213
ran152	152	47/6	38/6	38/6	2182	38/6	2115	38/6	246
ran153	153	40/6	39/6	39/6	2213	39/6	2252	39/6	250
ran154	154	44/6	39/6	39/6	2694	39/6	3797	39/6	298
ran170	170	47/6	43/6	43/6	3949	43/6	4212	43/6	429
ran186	186	52/6	47/6	48/6	3648	47/6	3753	47/6	408
ran223	223	59/6	56/6	56/6	6158	56/6	6464	56/6	654
ran234	234	61/6	59/6	59/6	4883	59/6	5358	59/6	539
ran256	256	68/6	64/6	64/6	7421	64/6	9010	64/6	772
ran286	286	76/6	72/6	72/6	6294	72/6	6715	72/6	718
ran298	298	76/6	75/6	75/6	9655	75/6	9616	75/6	987
ran310	310	83/6	78/6	78/6	7547	78/6	8297	78/6	865
ran357	357	93/6	90/6	90/6	16045	90/6	22094	90/6	1624
ran364	364	95/6	92/6	92/6	11094	92/6	13165	92/6	1245
ran510	510	129/6	128/6	128/6	20391	128/6	20141	128/6	2116
ran546	546	141/6	137/6	137/6	22978	137/6	25037	137/6	2399

Tabela E.15: Resultados obtidos para GADs com topologia *randômico* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (1, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	*n*	*ms/pu*	*ms/pu*	*ms/pu*	*te*	*ms/pu*	*te*	*ms/pu*	*te*
di0025	25	83/4	70/2	60/4	111	60/4	130	60/4	16
di0036	36	103/4	90/3	78/5	162	78/5	192	78/5	24
di0064	64	144/5	130/5	117/6	305	115/6	361	121/6	47
di0100	100	186/6	176/6	166/6	499	165/6	615	166/6	83
di0144	144	236/6	230/6	224/6	766	221/6	1039	221/6	136
di0225	225	343/6	331/6	328/6	1371	328/6	2014	322/6	260
di0256	256	371/6	370/6	367/6	1599	361/6	2034	361/6	329
di0400	400	556/6	550/6	549/6	2787	549/6	3581	541/6	690
di1024	1024	1335/6	1330/6	1330/6	9214	1330/6	14279	1321/6	3538

Tabela E.16: Resultados obtidos para GADs com topologia *árvore binária invertida* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (5, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
intree015	15	37/6	31/6	31/6	60	31/6	82	31/5	11
intree031	31	61/6	51/6	51/6	154	51/6	326	51/6	26
intree063	63	97/6	91/6	91/6	458	91/6	697	91/6	69
intree127	127	181/6	171/6	171/6	1544	171/6	2385	171/6	219
intree255	255	337/6	331/6	331/6	5768	331/6	13961	331/6	801
intree511	511	661/6	651/6	651/6	23844	651/6	41919	651/6	3105

Tabela E.17: Resultados obtidos para GADs com topologia *árvore binária* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (5, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
outtree015	15	32/6	32/6	31/6	62	31/5	67	31/5	9
outtree031	31	52/6	52/6	51/6	149	51/6	152	51/6	22
outtree063	63	92/6	92/6	91/6	429	91/6	372	91/6	53
outtree127	127	172/6	172/6	172/6	1433	172/6	948	171/6	148
outtree255	255	332/6	332/6	331/6	5093	331/6	3104	331/6	470
outtree511	511	652/6	652/6	652/6	18884	652/6	10212	651/6	1720

Tabela E.18: Resultados obtidos para GADs com topologia *randômico* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (5, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
ran080	80	123/6	104/6	101/6	828	101/6	951	101/6	98
ran098	98	141/6	126/6	125/6	1086	125/6	1232	125/6	125
ran108	108	151/6	141/6	140/6	1142	140/6	1440	140/6	142
ran124	124	161/6	160/6	160/6	2033	160/6	2756	160/6	230
ran135	135	181/6	170/6	170/6	1739	170/6	2231	170/6	208
ran140	140	182/6	180/6	180/6	1776	180/6	1976	180/6	213
ran152	152	212/6	191/6	191/6	2183	191/6	2152	191/6	245
ran153	153	195/6	195/6	195/6	2190	195/6	2142	195/6	247
ran154	154	203/6	195/6	195/6	2624	195/6	4467	195/6	295
ran170	170	231/6	215/6	215/6	3915	215/6	7412	215/6	424
ran186	186	245/6	236/6	236/6	3755	236/6	5079	235/6	397
ran223	223	291/6	281/6	281/6	6146	281/6	10772	280/6	649
ran234	234	303/6	295/6	295/6	4753	295/6	5568	295/6	518
ran256	256	330/6	320/6	320/6	7593	320/6	11984	320/6	776
ran286	286	371/6	360/6	360/6	6276	360/6	6653	360/6	721
ran298	298	375/6	375/6	375/6	9592	375/6	13746	375/6	979
ran310	310	402/6	390/6	390/6	7589	390/6	10426	390/6	860
ran357	357	456/6	450/6	450/6	16433	450/6	27652	450/6	1604
ran364	364	461/6	460/6	460/6	11446	460/6	13607	460/6	1223
ran510	510	650/6	640/6	640/6	20286	640/6	20594	640/6	2144
ran546	546	692/6	686/6	685/6	23344	685/6	23880	685/6	2438

Tabela E.19: Resultados obtidos para GADs com topologia *randômico* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (5, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
di0025	25	163/4	140/2	116/5	115	115/5	134	115/5	17
di0036	36	203/4	180/3	154/6	165	154/5	191	154/6	25
di0064	64	284/5	260/5	226/6	302	226/6	354	226/6	47
di0100	100	366/6	351/6	326/6	497	322/6	600	326/6	83
di0144	144	465/6	460/6	447/6	779	437/6	933	436/6	135
di0225	225	665/6	660/6	651/6	1371	651/6	1759	636/6	258
di0256	256	736/6	740/6	725/6	1610	725/6	2227	717/6	334
di0400	400	1096/6	1100/6	1087/6	2864	1087/6	5226	1078/6	702
di1024	1024	2656/6	2660/6	2651/6	9158	2651/6	28042	2638/6	3635

Tabela E.20: Resultados obtidos para GADs com topologia *árvore binária invertida* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (10, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
intree015	15	72/6	61/6	61/6	60	61/6	98	61/5	10
intree031	31	121/6	101/6	101/6	150	101/6	282	101/6	26
intree063	63	192/6	181/6	181/6	461	181/6	704	181/6	70
intree127	127	361/6	341/6	341/6	1580	341/6	2942	341/6	220
intree255	255	672/6	661/6	661/6	5857	661/6	13472	661/6	806
intree511	511	1321/6	1301/6	1301/6	23281	1301/6	51644	1301/6	3077

Tabela E.21: Resultados obtidos para GADs com topologia *árvore binária* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (10, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	n	ms/pu	ms/pu	ms/pu	te	ms/pu	te	ms/pu	te
outtree015	15	62/6	62/6	61/6	61	61/5	61	61/5	9
outtree031	31	102/6	102/6	101/6	146	101/6	162	101/6	21
outtree063	63	182/6	182/6	181/6	435	181/6	359	181/6	53
outtree127	127	342/6	342/6	342/6	1512	342/6	1010	341/6	147
outtree255	255	662/6	662/6	662/6	5235	662/6	3502	661/6	472
outtree511	511	1302/6	1302/6	1302/6	19646	1302/6	10694	1301/6	1702

Tabela E.22: Resultados obtidos para GADs com topologia *randômico* em um sistema dual restrito com 6/2 processadores onde $(me, mc) = (10, 1)$.

grafo		ETF	HEFT	PSGA		HTSGA		HTSG	
nome	***n***	***ms/pu***	***ms/pu***	***ms/pu***	***te***	***ms/pu***	***te***	***ms/pu***	***te***
ran080	80	243/6	204/6	201/6	831	201/6	943	202/6	99
ran098	98	281/6	251/6	250/6	1068	250/6	1209	250/6	126
ran108	108	301/6	281/6	281/6	1147	281/6	1450	280/6	142
ran124	124	321/6	320/6	320/6	2070	320/6	3693	320/6	226
ran135	135	361/6	340/6	340/6	1795	340/6	2007	340/6	206
ran140	140	362/6	360/6	360/6	1795	360/6	2126	360/6	211
ran152	152	422/6	381/6	381/6	2188	381/6	2074	380/6	247
ran153	153	390/6	390/6	390/6	2192	390/6	2219	390/6	247
ran154	154	403/6	390/6	390/6	2661	390/6	4103	390/6	294
ran170	170	461/6	430/6	430/6	3897	430/6	7035	430/6	428
ran186	186	485/6	471/6	471/6	3823	471/6	5781	470/6	401
ran223	223	581/6	561/6	560/6	6251	560/6	8978	560/6	634
ran234	234	603/6	590/6	590/6	4793	590/6	5815	590/6	525
ran256	256	660/6	640/6	640/6	7536	640/6	13163	640/6	775
ran286	286	741/6	720/6	720/6	6244	720/6	7186	720/6	755
ran298	298	750/6	750/6	750/6	9575	750/6	12162	750/6	979
ran310	310	802/6	780/6	780/6	7675	780/6	9946	780/6	868
ran357	357	911/6	900/6	900/6	16472	900/6	23900	900/6	1612
ran364	364	921/6	920/6	920/6	11406	920/6	14110	920/6	1239
ran510	510	1300/6	1280/6	1280/6	20520	1280/6	21485	1280/6	2109
ran546	546	1382/6	1371/6	1370/6	23625	1370/6	22600	1370/6	2396

Tabela E.23: Resultados obtidos para GADs com topologia *randômico* em um sistema dual restrito com $6/2$ processadores onde $(me, mc) = (10, 1)$.

Referências Bibliográficas

[1] KUMAR, V. et al. *Introduction to parallel computing: design and analysis of algorithms*. [S.l.]: Benjamin-Cummings Publishing Co., Inc., 1994. ISBN 0-8053-3170-0.

[2] FOSTER, I. *Internet Computing and the Emerging Grid*. Nature Web Matters, December 2000. Disponível em: <http://www.nature.com/nature/webmatters/grid/grid.html>. Acesso em: 11/09/2004.

[3] FOSTER, I. The grid: A new infrastructure for 21st century science. *Physics Today*, v. 2, n. 55, p. 42–47, 2002. Disponível em: <http://www.physicstoday.org/vol-55/iss-2/p42.html>. Acesso em: 22/11/2004.

[4] BUYYA, R. *High Performance Cluster Computing: Programming and Applications*. [S.l.]: Prentice Hall PTR, 1999. ISBN 0130137855.

[5] FOSTER, I. The anatomy of the grid: enabling scalable virtual organizations. In: *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*. [S.l.]: IEEE Computer Society, 2001. p. 6. ISBN 0-7695-1010-8.

[6] FOSTER, I.; KESSELMAN, C. *The grid: blueprint for a new computing infrastructure*. [S.l.]: Morgan Kaufmann Publishers Inc., 1999. ISBN 1-55860-475-8.

[7] ANDERSON, D. P. et al. Setihome: an experiment in public-resource computing. *Communications of the ACM*, ACM Press, v. 45, n. 11, p. 56–61, 2002. ISSN 0001-0782.

[8] FILHO, J. V. *Estratégia para Escalonamento de Tarefas em Dois Estágios para Ambientes Heterogêneos*. Dissertação (Mestrado) — Universidade Federal Fluminense, Brasil, Outubro 2004.

[9] GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. [S.l.]: W. H. Freeman & Co., 1979. ISBN 0716710447.

[10] KWOK, Y.-K.; AHMAD, I. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, v. 59, n. 3, p. 381–422, December 1999.

[11] KWOK, Y.-K.; AHMAD, I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, ACM Press, v. 31, n. 4, p. 406–471, 1999. ISSN 0360-0300.

[12] AGGARWAL, A.; CHANDRA, A.; SNIR, M. *On communications latency in PRAM computations*. Yorktown Heights, NY, USA, September 1989.

[13] FORTUNE, S.; WYLLIE, J. Parallelism in random access machines. In: *Proceedings of the 10th Annual ACM Symposium of Theory of Computing*. [S.l.]: ACM Press, 1978. p. 114–118.

[14] HOU, E.; ANSARI, N.; REN, H. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel Distributed Systems*, IEEE Press, v. 5, n. 2, p. 113–120, 1994.

[15] PAPADIMITRIOU, C. H.; YANNAKAKIS, M. Towards an architecture-independent analysis of parallel algorithms. *SIAM Journal on Computing*, v. 19, p. 322–328, 1996.

[16] CORRêA, R.; FERREIRA, A.; REBREYEND, P. Scheduling multiprocessor tasks with genetic algorithms. *IEEE Transactions on Parallel and Distributed Systems*, v. 10, n. 8, p. 825–837, August 1999.

[17] DHODHI, M. et al. An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, v. 62, p. 1338–1361, 2002.

[18] KWOK, Y.-K.; AHMAD, I. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *Parallel and Distributed Computing*, Academic Press, v. 47, p. 58–77, 1997.

[19] WANG, L. et al. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of Parallel Distributed Computing*, Academic Press, Inc., v. 47, n. 1, p. 8–22, 1997. ISSN 0743-7315.

[20] DOGAN, A.; ÖZGüNER, F. LDBS: A duplication based scheduling algorithm for heterogeneous computing systems. In: *ICPP '02: Proceedings of the 2002 International Conference on Parallel Processing (ICPP'02)*. [S.l.]: IEEE Computer Society, 2002. p. 352–359. ISBN 0-7695-1677-7.

[21] TOPCUOGLU, H.; HARIRI, S.; WU, M.-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, v. 13, n. 3, p. 260–274, March 2002.

[22] OH, H.; HA, S. A static scheduling heuristic for heterogeneous processors. In: *Proceedings of the Second International Euro-Par Conference on Parallel Processing-Volume II*. [S.l.]: Springer-Verlag, 1996. p. 573–577. ISBN 3-540-61627-6.

[23] SIH, G. C.; LEE, E. A. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel Distrib. Syst.*, IEEE Press, v. 4, n. 2, p. 175–187, 1993. ISSN 1045-9219.

[24] HWANG, J.-J. et al. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal on Computing*, Society for Industrial and Applied Mathematics, v. 18, n. 2, p. 244–257, April 1989.

[25] ALI, H.; EL-REWINI, H. The time complexity of scheduling interval orders with communications is polynomial. *Parallel Processing Letters*, v. 3, n. 1, p. 53–58, 1993.

[26] COFFMAN, E.; GRAHAM, R. Optimal scheduling for two-processor systems. *Acta Informatica*, v. 1, p. 200–213, 1972.

[27] FERNANDEZ, E. B.; BUSSELL, B. Bounds on the number of processor and time for multiprocessor optimal schedules. *IEEE Transactions on Computers*, C-22, n. 8, p. 745–751, 1973.

[28] HU, T. Parallel sequencing and assembly line problems. *Operations Research*, v. 19, n. 6, p. 841–848, November 1961.

[29] ALEXANDROV, A. et al. LogGP: incorporating long messages into the LogP model for parallel computation. *Journal of Parallel and Distributed Computing*, Academic Press, Inc., v. 44, n. 1, p. 71–79, 1997. ISSN 0743-7315.

[30] CULLER, D. et al. LogP: towards a realistic model of parallel computation. In: *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*. [S.l.]: ACM Press, 1993. p. 1–12. ISBN 0-89791-589-5.

[31] INO, F.; FUJIMOTO, N.; HAGIHARA, K. LogGPS: a parallel computational model for synchronization analysis. In: *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*. [S.l.]: ACM Press, 2001. p. 133–142. ISBN 1-58113-346-4.

[32] VALIANT, L. G. A bridging model for parallel computation. *Communications of the ACM*, ACM Press, v. 33, n. 8, p. 103–111, 1990. ISSN 0001-0782.

[33] ESTEVES, M. A. N. *Rumo a uma Heurística Rápida para Geração de Escalonamentos Eficientes em Grids Computacionais*. Dissertação (Mestrado) — Universidade Federal Fluminense, Brasil, Junho 2003.

[34] ALEXANDROV, A. et al. LogGP: incorporating long messages into the LogP model: one step closer towards a realistic model for parallel computation. In:

Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures. [S.l.]: ACM Press, 1995. p. 95–105. ISBN 0-89791-717-0.

[35] BEAUMONT, O.; BOUDET, V.; ROBERT, Y. A realistic model and an efficient heuristic for scheduling with heterogeneous processors. In: *Proceedings of the 16th International Parallel and Distributed Processing Symposium*. [S.l.]: IEEE Computer Society, 2002. p. 37. ISBN 0-7695-1573-8.

[36] GIBBONS, P. B. A more practical PRAM model. In: *Proceedings of the first annual ACM symposium on Parallel algorithms and architectures*. [S.l.]: ACM Press, 1989. p. 158–168. ISBN 0-89791-323-X.

[37] ALMOND, J.; SNELLING, D. Unicore: Uniform access to supercomputing as an element of electronic commerce. *Future Generation Computer Systems*, NH-Elsevier, v. 15, n. 5, p. 539–548, 1999.

[38] LöWE, W.; ZIMMERMANN, W. Scheduling balanced task-graphs to LogP-machines. *Parallel Comput.*, Elsevier Science Publishers B. V., v. 26, n. 9, p. 1083–1108, 2000. ISSN 0167-8191.

[39] MARTIN, R. P. et al. Effects of communication latency, overhead, and bandwidth in a cluster architecture. In: *Proceedings of the 24th annual international symposium on Computer architecture*. [S.l.]: ACM Press, 1997. p. 85–97.

[40] OCHI, L. S.; DRUMMOND, L.; VIANNA, L. S. Um algoritmo grasp para o problema de escalonamento de tarefas em múltiplos processadores utilizando o modelo LogP. In: *XXXII Simpósio Brasileiro de Pesquisa Operacional (XXXII SBPO)*. [S.l.: s.n.], 2000. v. 1, p. 200–218.

[41] ROMBERG, M. The unicore architecture: Seamless access to distributed resources. In: *Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing*. [S.l.]: IEEE Computer Society, 1999. p. 287–293. ISBN 0-7695-0287-3.

[42] TAM, A.; WANG, C. A realistic communication model for parallel computing on cluster. In: *Proceedings of the I IEEE International Workshop on Cluster Computing*. [S.l.]: IEEE Computer Society Press, 1999. p. 92–101.

[43] BOERES, C.; REBELLO, V. E. F. A versatile cost modelling approach for multicomputer task scheduling. *Parallel Computing*, v. 25, n. 1, p. 63–86, 1999.

[44] CORPORATION, I. *IBM Parallel enviromment for AIX, MPL: programming and subroutine reference*. [S.l.], 1995.

[45] MPI, F. *MPI: A Message-Passing Interface Standard*. [S.l.], May 1994.

[46] AL-MOUHAMED, M.; AL-MAASARANI, A. Performance evaluation of scheduling precedence-constrained computations on message-passing systems. *IEEE Trans. Parallel Distrib. Syst.*, IEEE Press, v. 5, n. 12, p. 1317–1321, 1994.

[47] BAJAJ, R.; AGRAWAL, D. Improving scheduling of tasks in a heterogeneous environment. *IEEE Transactions on Parallel and Distributed Systems*, v. 15, n. 2, p. 107–118, 2004.

[48] BARADA, H.; SAIT, S. M.; BAIG, N. A simulated evolution approach to task matching and scheduling in heterogeneous computing environments. *Engineering Applications to Artificial Intelligence*, Elsevier Science Ltd., v. 15, p. 491–500, 2002.

[49] BOERES, C.; REBELLO, V. E. F. Towards optimal task scheduling for realistic machine models: Theory and practice. *The International Journal of High Performance Computing Applications*, v. 17, n. 2, p. 173–190, 2003.

[50] BRAUN, T. D. et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel Distributed Computing*, Academic Press, Inc., v. 61, n. 6, p. 810–837, 2001.

[51] KWOK, Y.-K. Parallel program execution on a heterogeneous PC cluster using task duplication. In: *Proceedings of the 9th Heterogeneous Computing Workshop*. [S.l.]: IEEE Computer Society, 2000. p. 364–374. ISBN 0-7695-0556-2.

[52] KWOK, Y.-K.; AHMAD, I. Dynamic critical-path scheduling: an effective technique for allocation task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, v. 7, n. 5, p. 506–521, May 1996.

[53] SINNEN, O.; SOUSA, L. List scheduling: Extension for contention awareness and evaluation of node priorities for heterogeneous cluster architectures. *Parallel Computing*, v. 30, n. 1, p. 81–101, January 2004.

[54] TSUCHIYA, T.; OSADA, T.; KIKUNO, T. Genetic-based multiprocessor scheduling using task duplication. *Microprocessors and Microsystems*, v. 22, p. 197–207, 1998.

[55] YAO, W.; YOU, J.; LI, B. Main sequences genetic scheduling for multiprocessors systems using task duplication. *Microprocessors and Microsystems*, v. 28, n. 10, p. 85–94, March 2004.

[56] ZHONG, Y.-W.; YANG, J.-G. A genetic algorithm for tasks scheduling in parallel multiprocessor systems. In: *Proceedings of the 2nd International Conference on Machine Learning and Cybernetics*. [S.l.: s.n.], 2003. p. 1785–1790.

[57] BARBOSA, V. C. *An Introduction to Distributed Algorithms*. [S.l.]: The MIT Press, 1996. ISBN 0262024128.

[58] CASAVANT, T. L.; KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, IEEE Press, v. 14, n. 2, p. 141–154, 1988. ISSN 0098-5589.

[59] HU, M.-Y.; SHU, W.; GU, J. Efficient local search for DAG scheduling. *IEEE Transactions on Parallel and Distributed Systems*, v. 12, n. 6, p. 617–627, June 2001.

[60] KHAN, A.; MCCREARY, C.; JONES, M. Comparison of multiprocessor scheduling heuristics. In: TAI, K. (Ed.). *Proceedings of 8th International Parallel Procesing*. Cancun, Mexico: IEEE Computer Society Press, 1994. v. 2, p. 243–250.

[61] SARKAR, V. *Partitioning and scheduling parallel programs for execution on multiprocessors*. Tese (Doutorado) — Stanford University, 1987.

[62] NASCIMENTO, A. de P. *Aglomeração de Tarefas em Máquinas Paralelas com Memória Distribuída*. Dissertação (Mestrado) — Universidade Federal Fluminense, Brasil, Outubro 1999.

[63] ADAM, T.; CHANDY, K.; DICKSON, J. A comparison of list schedules for parallel processing systems. *Communications of the ACM*, ACM Press, v. 17, n. 12, p. 685–690, 1974. ISSN 0001-0782.

[64] Coffman, Jr., E. *Computer and Job Shop Scheduling Theory*. New York: John Wiley & Sons Inc, 1976. 312 p. ISBN 0471163198.

[65] GRAHAM, R. et al. Optimization and approximation in deterministic sequencing and scheduling: A survey. In: *Annals of Discrete Mathematics*. [S.l.: s.n.], 1979. v. 5, p. 287–326.

[66] LIAO, G. et al. A comparative study of multiprocessor list scheduling heuristics. In: *Proceedings of 27th Annual Hawaii International Conference on System Sciences*. [S.l.: s.n.], 1994. p. 68–77.

[67] CHO, T. W.; PYO, S.; HEATH, J. Parallex: a parallel approach to switchbox routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 13, n. 6, p. 684–693, June 1994.

[68] WU, M.; GADJSKI, D. Hypertool: a programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, v. 1, n. 3, p. 101–119, July 1990.

[69] YANG, T.; GERASOULIS, A. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, IEEE Press, v. 5, n. 9, p. 951–967, 1994. ISSN 1045-9219.

[70] CIROU, B.; JEANNOT, E. Triplet: A clustering scheduling algorithm for heterogeneous systems. In: *Proceedings of the 2001 International Conference on Parallel Processing Workshops*. [S.l.]: IEEE Computer Society, 2001. p. 231–236.

[71] BOERES, A. P. N. C.; REBELLO, V. E. F. Cluster-based task scheduling for the LogP model. *International Journal of Foundations of Computer Science*, World Scientific Publishing Co, v. 10, n. 4, p. 405–424, 1999.

[72] KIM, S.; BROWNE, J. A general approach to mapping of parallel computation upon multiprocessor architectures. In: *Proceedings of the 1988 IEEE International Conference on Parallel Processing*. [S.l.: s.n.], 1988. v. 3, p. 1–8.

[73] FRIESEN, D. K. Tighter bounds for LPT scheduling on uniform processors. *SIAM Journal on Computing*, Society for Industrial and Applied Mathematics, v. 16, n. 3, p. 554–560, 1987. ISSN 0097-5397.

[74] Gonzalez, Jr., M. J. Deterministic processor scheduling. *ACM Computing Surveys*, ACM Press, v. 9, n. 3, p. 173–204, 1977. ISSN 0360-0300.

[75] MCCREARY, C.; CLEVELAND, M.; KHAN, A. *The problem with critical path scheduling algorithm*. Alburn, AL: [s.n.], 1996.

[76] GERASOULIS, A.; YANG, T. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors. *Journal of Parallel and Distributed Systems*, v. 16, p. 276–291, 1992.

[77] YU, W. H. *LU Decomposition on a Multiprocessing System with Communication Delay*. Tese (Doutorado) — Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1984.

[78] BOERES, C.; REBELLO, V. E. F. Cluster-based static scheduling: theory and practice. In: *Proceedings of the 14th Computer Architecture and High Performance Computing*. [S.l.]: IEEE Press, 2002. p. 133–140.

[79] BOERES, C.; REBELLO, V. E. F. A cluster-based task scheduling methodology for parallel machines. In: *New Trends in Scheduling for Parallel and Distributed Systems (J. Blazewicz, Ed Coffman, K. Ecker and D. Trystram, Organizers), CIRM*. [S.l.: s.n.], 2001. p. 1–5.

[80] BOERES, C.; REBELLO, V. E. F. Task clustering for logp-type models. In: *Advanced Algorithmic Techniques for Parallel Computation with Application, Applied Optimization Series*. [S.l.]: Kluwer, 2001.

[81] GERASOULIS, A.; VENUGOPAL, S.; YANG, T. Clustering task graphs for message passing architectures. In: *Proceedings of the 4th international conference on Supercomputing*. [S.l.]: ACM Press, 1990. p. 447–456. ISBN 0-89791-369-8.

[82] PALIS, M.; LIOU, J.-C.; WEI, D. Task clustering and scheduling for distributed memory parallel architectures. *IEEE Transactions on Parallel and Distributed Systems*, v. 7, n. 1, p. 46–55, 1996.

[83] SAKELLARIOU, R.; ZHAO, H. A hybrid heuristic for DAG scheduling on heterogeneous systems. In: *Proceedings of the 18th International Parallel and Distributed Processing Symposium*. [S.l.]: ACM Press, 2004. p. 111–123.

[84] BOERES, C.; LIMA, A.; REBELLO, V. E. F. Hybrid task scheduling: Integrating static and dynamic heuristics. In: *Proceeding of 15th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'03)*. [S.l.: s.n.], 2003. p. 119.

[85] HOLLAND, J. H. *Adaptation in natural and artificial systems*. [S.l.]: University of Michigan Press, 1975.

[86] FEO, T.; RESENDE, M. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–133, 1995.

[87] GLOVER, F. Tabu Search - Part I. *ORSA Journal on Computing*, v. 1, p. 190–206, 1989.

[88] GLOVER, F. Tabu Search - Part II. *ORSA Journal on Computing*, v. 2, p. 4–32, 1990.

[89] KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, v. 220, 4598, p. 671–680, 1983. Disponível em: <citeseer.nj.nec.com/kirkpatrick83optimization.html>.

[90] HANSEN, P.; MLADENOVIé, N. *A Tutorial on Variable Neighborhood Search.* [S.l.], Julho 2003.

[91] PORTO, S. C. S.; KITAJIMA, J. P.; RIBEIRO, C. C. Performance evaluation of a parallel tabu search task scheduling algorithm. *Parallel Computing*, v. 26, n. 1, p. 73–90, January 2000.

[92] AUYEUNG, A.; GONDRA, I.; DAI, H. K. Multi-heuristic list scheduling genetic algorithm for task scheduling. In: *Proceedings of the 2003 ACM symposium on Applied computing.* [S.l.]: ACM Press, 2003. p. 721–724.

[93] CHENG, W.; MIN, L. A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence in Engineering*, v. 13, p. 399–403, 1999.

[94] ZOMAYA, A.; WARD, C.; MACEY, B. Genetic scheduling for parallel processor systems: comparative studies and performance issues. *IEEE Transactions on Parallel and Distributed Systems*, v. 10, n. 8, p. 795–812, August 1998.

[95] WANG, L.; SIEGEL, H. J.; ROYCHOWDHURY, V. P. A genetic-algorithm-based approach for task matching and scheduling in heterogeneous computing environments. In: *Proceedings of the Heterogeneous Computing Workshop.* [S.l.: s.n.], 1993.

[96] AHMAD, I.; KWOK, Y.-K. On exploiting task duplication in parallel program scheduling. *IEEE Transactions on Parallel and Distribuited Systems*, IEEE Press, v. 9, n. 9, p. 872–892, 1998. ISSN 1045-9219.

[97] BANSAL, S.; KUMAR, P.; SINGH, K. An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, IEEE Computer Society, v. 14, n. 6, p. 533–544, Junho 2003.

[98] BOERES, C.; REBELLO, V. E. F.; SKILLICORN, D. B. Static scheduling using task replication for LogP and BSP models. In: *Proceedings of the 4th In-*

ternational Euro-Par Conference on Parallel Processing. [S.l.]: Springer-Verlag, 1998. p. 337–346. ISBN 3-540-64952-2.

[99] BOERES, C.; REBELLO, V. E. F. Escalonamento estático com replicação de tarefas em modelos universais. In: *Workshop de Parelelismo em Otimização Combinatória, X Simpósio Brasileiro de Arquitetura de Computadores e Processamento de Alto Desempenho (WPOC/SBAC-PAD'98)*. [S.l.: s.n.], 1998.

[100] DARBHA, S.; AGRAWAL, D. P. Optimal scheduling algorithm for distributed-memory machines. *IEEE Transactions on Parallel and Distributed Systems*, IEEE Press, v. 9, n. 1, p. 87–95, 1998. ISSN 1045-9219.

[101] KWOK, Y.-K. A new duplication-based approach for scheduling tasks to a heterogeneous workstation cluster. In: *Proceedings of the 6th Annual Australasian COmference on Parallel and Real-Time Systems (PART'99)*. Melbourne, Australia: [s.n.], 1999.

[102] LUQUE, E. et al. Impact of task duplication on static-scheduling performance in multiprocessor systems with variable execution-time tasks. In: *Proceedings of the 4th international conference on Supercomputing*. [S.l.]: ACM Press, 1990. p. 439–446. ISBN 0-89791-369-8.

[103] BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, ACM Press, v. 35, n. 3, p. 268–308, 2003. ISSN 0360-0300.

[104] PAPADIMITRIOU, C.; STEIGLITZ, K. *Combinatorial Optimization: Algorithms and Complexity*. [S.l.]: Dover Publications, 1998. ISBN 0486402584.

[105] HEITKOTTER, J.; BEASLEY, D. *The Hitch-Hiker's Guide to Evolutionary Computation: A list of Frequently Asked Questions*. March 2000. Disponível em: <http://www.cs.bham.ac.uk/Mirrors/ftp.de.uu.net/EC/clife/www>. Acesso em: 18/07/2004.

[106] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675.

[107] BEAUMONT, O. et al. *Heterogeneity Considered Harmful to Algorithm Designers*. ENS Lyon, France, June 2000.

[108] COSNARD, M.; TRYSTRAM, D. *Parallel Algorithms and Architecture*. [S.l.]: International Thomson Computer Press, 1995. ISBN 1850321256.

[109] AIEX, R. M.; RESENDE, M. G. C.; RIBEIRO, C. C. Probability distribution of solution time in grasp: An experimental investigation. *Journal of Heuristics*, Kluwer Academic Publishers, v. 8, n. 3, p. 343–373, 2002. ISSN 1381-1231.