Bruno Cardoso Coutinho

Proposta de Algoritmo Híbrido para o Problema de Escalonamento de Tarefas em Ambientes Distribuídos Homogêneos

Vitória - ES, Brasil

28 de março de 2008

Bruno Cardoso Coutinho

Proposta de Algoritmo Híbrido para o Problema de Escalonamento de Tarefas em Ambientes Distribuídos Homogêneos

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Centro Tecnológio da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Mestre em Informática.

Orientador:
Prof. Dr. Elias Silva de Oliveira

MESTRADO EM INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA
CENTRO TECNOLÓGICO
UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Vitória - ES, Brasil

28 de março de 2008

Dissertação de Mestrado sob o título "*Proposta de Algoritmo Híbrido para o Problema de Escalonamento de Tarefas em Ambientes Distribuídos Homogêneos*", defendida por Bruno Cardoso Coutinho e aprovada em 28 de março de 2008, em Vitória, Estado do Espírito Santo, pela banca examinadora constituída pelos professores:

Prof. Dr. Elias Silva de Oliveira
Orientador

Profª. Drª. Cristina Boeres
Universidade Federal Fluminense

Profª. Drª. Lúcia Catabriga
Universidade Federal do Espírito Santo

Dedicatória

Dedico este trabalho aos meus pais Emanoel e Lili.

Agradecimentos

Agradeço primeiramente a Deus por me sustentar ao longo desses anos de intenso estudo e muitas horas de trabalho.

A minha família por todo apoio e compreensão pelos vários momentos que não pude estar com eles porque tinha muitos experimentos a fazer, em especial, a minha irmã Fabiane por ceder seu computador para alguns experimentos!

Ao professor Elias pela amizade, orientação e dicas passadas em momentos complicados.

A todo o pessoal do LCAD, Felipe, Fernando, Francisco, Márcia, Patrick e demais por estarem sempre à disposição.

Às irmãs Boeres (Cristina e Cláudia), se é que posso chamar assim, por me sugerirem materiais importantíssmos para a condução de minhas pesquisas.

Ao professor e colega Sérgio Nery pelas orientações na realização dos testes no *Cluster*.

Ao "sempre prestativo" professor (de fato e de verdade) Arlindo Gomes, sempre atencioso com o andamento do meu curso.

Finalmente, agradeço a todos aqueles que torceram pelo meu sucesso e também por todos aqueles que torceram pelo meu fracasso. Todos eles me ajudaram de alguma forma.

Sumário

Lista de Tabelas

Lista de Figuras

Resumo

Problemas de Escalonamento de Tarefas estão entre os mais difíceis da Otimização Combinatória. Por se tratar de um elemento importante em áreas como Gerência de Projetos e Arquitetura de Sistemas Distribuídos, torna-se fator de motivação para pesquisas, visando maior eficiência desse processo de seqüenciamento. Neste trabalho é feita a proposta de um algoritmo híbrido para o escalonamento de tarefas em ambientes distribuídos homogêneos, na tentativa de encontrar seqüenciamentos ótimos ou bem próximos do ótimo. Este algoritmo utiliza-se de formulações matemáticas e algoritmos genéticos, através do particionamento do grafo de precedências por níveis de altura das tarefas, dividindo o problema do escalonamento em subproblemas menores e mais fáceis de resolver. Foi possível obter uma economia de até 25% no tempo de execução das aplicações utilizadas nos experimentos. Com o seqüenciamento obtido pelo algoritmo híbrido e, também, com a realização de experimentos em um *Cluster* Real, pôde-se confirmar a importância de um Processo Escalonador no aumento da vazão de processamento de tarefas em um Ambiente Computacional Distribuído.

Abstract

Tasks Scheduling Problems are among the most difficult of Optimization Combinatorics. This is an important element in areas such as Management of Projects and Architecture for Distributed Systems, it is factor of motivation for searching, seeking greater efficiency of this process of sequencing. In this paper it's make a proposal for a hybrid algorithm for tasks scheduling in distributed environments homogeneous, in the attempt to find optimal schedules or good coming of the great. This algorithm uses mathematical formulations and genetic algorithms, through the graph partitioning of precedence by levels of height of the tasks, dividing the problem of scaling in sub-minor problems and easier to solve. It was possible to obtain a saving of 25% at the time of execution of the applications used in the experiments. With the sequencing obtained by the hybrid algorithm and also conduct tests in a Real *Cluster* confirming the importance of a scheduler process on increasing the flow of processing tasks in a Computational Distributed Environment.

1 Introdução

“Porque os caminhos do homem estão perante os
olhos do Senhor e ele considera
todas as suas veredas.”
Prov. 5:21

O planejamento faz parte da vida de qualquer ser humano e em qualquer situação. Um objetivo é traçado, o projeto é formulado e dividido em tarefas para melhor organização.

Quando uma pessoa sai para trabalhar, ele planeja por exemplo sua rota até o serviço. O objetivo dele é chegar ao seu local de trabalho, seu projeto consiste em traçar um caminho que o permita chegar ao trabalho no horário, então ele pode dividir este projeto em tarefas: entrar no carro, colocar o cinto de segurança, ligar o carro, sair da garagem, etc.

Dividir um projeto em tarefas permite maior organização de nossas atividades. Um Gerente de Projetos, que precisa administrar várias obras ao mesmo tempo, costuma dividir projetos em sub-projetos e estes em tarefas, delegando cada tarefa a um ou mais empregados e determinando uma estimativa de tempo para a realização de cada tarefa.

A máxima da computação é *"dividir para conquistar"*. Desde o início de um curso de graduação em computação, o aluno é orientado a "quebrar" problemas maiores em sub-problemas menores, para facilitar o estudo, implementação e manutenção de determinado programa de computador, por exemplo.

Com a necessidade atual de se economizar tempo, pois "tempo é dinheiro", exige-se a realização de tarefas, muitas vezes complexas, otimizando tempo e custo. Na computação não é diferente. Cada vez mais é exigido maior poder de processamento de nossas máquinas, rapidez na execução de programas, economia de recursos computacionais. A implementação de algoritmos de otimização de processos tem-se tornado fator de estudos e pesquisas com o objetivo de economizar "tempo".

Se problemas podem ser divididos em tarefas, tarefas possuem seus custos de execução e os recursos disponíveis são escassos, podemos trabalhar técnicas para "planejar" o melhor seqüenciamento dessas tarefas, respeitando suas prioridades, para se alcançar os objetivos finais no menor tempo possível.

Neste Capítulo são apresentados a descrição do problema a ser trabalhado, o objetivo desta dissertação e a estrutura da mesma.

1.1 Objetivo deste trabalho

O objetivo do escalonamento é encontrar uma forma de associar e seqüenciar o uso de recursos compartilhados, satisfazendo a certas restrições e minimizando os custos e tempo de processamento (CAVALCANTE, 1998; MONACI, 2001).

Um bom escalonamento de tarefas, por exemplo, pode representar para um Engenheiro Civil, terminar a construção de um prédio no tempo estimado, com a utilização mínima de recursos e com o reconhecimento por parte de seus clientes (AMARANTE, 2001; STRADAL; CACHA, 1982), ou mesmo implicar em uma eficiente gerência de projetos de uma grande empresa de desenvolvimento de software que precisa lidar com vários projetos ao mesmo tempo, respeitando regras de qualidade definidas no PMBOK (*Project Management Body of Knowledge*) (PMI, 2001). Pode representar também, economia de tempo e recursos computacionais em um ambiente de processamento distribuído utilizado por vários alunos e cientistas de uma determinada universidade (TOPCUOGLU; HARIRI; WU, 2002).

Por se tratar de uma fase importante na Gerência de Projetos ou mesmo na aplicação de Sistemas Distribuídos, o assunto "Escalonamento de Tarefas" ou simplesmente, do inglês, *Scheduling*, vem sendo pesquisado por vários grupos de pesquisas importantes ao longo dos anos, visando uma maior produtividade por parte de institutos, empresas e universidades. Diversas técnicas de escalonamento, determinísticas ou aproximadas, para ambientes computacionais distribuídos ou não, surgem a todo momento, como podemos observar em (CAVALCANTE, 1998; MONACI, 2001; CRAUWLES et al., 2005; TOPCUOGLU; HARIRI; WU, 2002).

1.1.1 O Problema

A maioria dos problemas de Escalonamento envolve diversos fatores, como a priorização de tarefas, restrições de custos, disponibilidade e capacidade de recursos, entre outras. Muitos destes problemas não são facilmente resolvidos e a determinação da melhor solução depende de métodos aproximativos e heurísticos, na tentativa de obtermos uma aproximação do ótimo.

Um grande conjunto de aplicações de otimização combinatória está relacionado ao gerenciamento e uso eficiente de recursos caros e escassos com o objetivo de aumentar a produtividade, que é de interesse de qualquer setor do mercado.

Encontrar uma solução para um problema de escalonamento envolve dois tipos de decisões (CAVALCANTE, 1998):

- *Decisões de Alocação*: que recursos serão destinados para a execução de cada tarefa;
- *Decisões de Seqüenciamento*: quando cada tarefa será executada.

Devido à natureza combinatória dos problemas de escalonamento, a escolha de uma estratégia para obtenção de soluções deve considerar dois aspectos: a complexidade computacional do problema e o tempo que se dispõe para resolvê-lo.

Com exceção de casos muito especiais (recursos abundantes, por exemplo, grande quantidade de máquinas ou mão-de-obra disponíveis), problemas de escalonamento são NP-Hard (CAVALCANTE, 1998; TOPCUOGLU; HARIRI; WU, 2002; BALAS, 1969), ou seja, ainda não se conhece um algoritmo determinístico polinomial para encontrar a solução ótima para o problema, levando em consideração todas as suas possíveis instâncias. Logo, usualmente são aplicadas outras técnicas como: heurísticas e meta-heurísticas.

Este projeto visa construir algoritmos computacionais para a escolha de seqüenciamento eficiente de tarefas a serem executadas em um dado sistema, que visem minimizar o tempo total de execução de todas as tarefas (denominado por *makespan*) de uma aplicação.

Como caso de estudo, será analisado o escalonamento de processos em um ambiente distribuído homogêneo, neste caso, o *Cluster* do Laboratório de Computação de Alto Desempenho que se encontra no Departamento de Informática da UFES.

1.1.2 Metodologia

Neste trabalho são analisados algoritmos determinísticos, como o branch-and-bound (CAVALCANTE, 1998; CRAUWLES et al., 2005) e Modelagem Matemática (MACULAN et al., 1999) para a obtenção de soluções ótimas ou relaxações e, por outro lado, métodos aproximados e heurísticos, como algoritmos Genéticos (GOLDEBERG, 1989) para obtenção de limitantes superiores.

Com o aprendizado e implementação dos algoritmos descritos acima, é desenvolvido um novo algoritmo híbrido, ou seja, compilando as características das técnicas determinísticas e heurísticas e, também, utilizando a fixação de variáveis do modelo matemático utilizado, trabalhar o particionamento do grafo de precedências das tarefas, transformando a aplicação a ser otimizada em um conjunto de grafos reduzidos.

Foram utilizadas instâncias com quantidades variadas de tarefas e processadores disponíveis para avaliação dos algoritmos desenvolvidos e também para experimentos reais no *Cluster*.

1.2 Estrutura da dissertação

No Capítulo 2 são apresentadas algumas abordagens de problemas de escalonamento de tarefas, bem como as principais técnicas de resolução encontradas na literatura. Também são descritas algumas das arquiteturas distribuídas mais conhecidas e utilizadas, *Cluster*s e *Grids*, apresentando algoritmos de resolução do problema de escalonamento de tarefas nesses ambientes.

No Capítulo 3 é proposto um algoritmo híbrido, utilizando formulação MIP (*Mixed Integer Programming* - Programação Inteira Mista (NEMHAUSER; WOLSEY, 1999)) e heurística para resolução do problema, juntamente com fixações de variáveis do modelo matemático nas iterações e reduções do grafo de precedências original, identificando a vantagem de se utilizar algoritmos na tentativa de melhorar a vazão de tarefas processadas por unidade de tempo em um ambiente distribuído real.

No Capítulo 4 são apresentados os resultados obtidos com instâncias do problema, construídas pelos mais variados métodos de geração de relações entre tarefas, tentando aproximar o melhor possível de um ambiente real.

No Capítulo 5 são descritas as conclusões bem como propostas de trabalhos futuros.

2 *Problemas de Escalonamento*

"A resposta branda desvia o furor, mas a palavra dura suscita a ira."
Prov. 15:11

Os problemas de escalonamento estão agrupados entre os mais difíceis problemas da computação e, especificamente, da área de otimização combinatória (PINEDO, 1995). Por serem modelados em diversas situações reais, se tornaram objeto de estudo de várias instituições de pesquisa e desenvolvimento ao longo dos anos.

Este Capítulo apresenta alguns modelos de problemas de escalonamento juntamente com as técnicas mais utilizadas de resolução. Foram pesquisadas algumas abordagens de problemas de escalonamento para analisar qual deles se aproximaria mais do objetivo deste projeto. É abordado o Problema de Escalonamento de Tarefas em Processadores Não-Relacionados Sob Restrições de Precedências, que será a modelagem utilizada neste trabalho. É feita uma descrição de alguns tipos de arquiteturas distribuídas: homogêneas, como os *Clusters* e heterogêneas, como os *Grids*. Por último, é feita uma descrição de uma meta-heurística que, de acordo com o artigo de Topcuoglu, Hariri e Wu (2002), pode ser considerada uma "boa" técnica aproximada para o problema de escalonamento em ambientes distribuídos homogêneos.

2.1 SPLC - Problema de Escalonamento com Restrição de Mão-de-Obra

O Problema de Escalonamento com Restrição de Mão-de-Obra, ou do inglês *Scheduling Problem under Labour Constraints* é muito utilizado principalmente na implementação de modelagens para Indústrias na otimização de seu processo produtivo, como apresentado por Cavalcante (1998), formalizando a descrição do problema da seguinte forma:

Seja I um conjunto de pedidos ($|I| = m$), onde cada pedido está associado a uma máquina e é composto por n_i jobs idênticos. O conjunto de todos os jobs é J ($|J| = \sum_{i=1}^{m} n_i$). Cada job é composto por um conjunto de p_j tarefas de duração 1. As tarefas de um mesmo job devem ser executadas imediatamente uma após a outra. A tupla $(l_{j1}, l_{j2}, ..., l_{jp_j})$ representa a necessidade de mão-de-obra do job j, com j_{js} denotando o número de trabalhadores necessários para a execução da s-ésima tarefa do job j. Existirá um limitante L que indicará o total de trabalhadores disponíveis em cada instante no horizonte de planejamento. Todos os jobs de um mesmo pedido devem ser executados na mesma máquina e não é permitida a preempção (ou interrupção) de pedidos. As relações de precedências entre os jobs são representadas através de um grafo direcionado acíclico (como geralmente acontece na maioria dos problemas de escalonamento de tarefas) $G = (V, A)$ onde o conjunto V representa os jobs e A a relação de precedência entre eles.

S é dito ser um escalonamento viável do problema se:

- *(1)*: $\forall (i, j) \in A$, o job j não começar antes do término do job i;
- *(2)*: o total de trabalhadores solicitados por todos os jobs com processamento no instante t não exceder L $\forall t$ no horizonte de planejamento.

O objetivo do SPLC é encontrar o *makespan*, ou seja, uma solução viável do problema onde todos os jobs são executados e finalizados no tempo mais cedo possível.

O problema pode ser tratado através de heurísticas baseadas em regras de prioridade (CAVALCANTE, 1998), onde cada job recebe um valor associado representando a sua prioridade em relação aos demais. Estas regras podem ser definidas por maior necessidade de recursos de cada job ou por menor tempo de finalização mais tarde, por exemplo. No mesmo trabalho, formulações MIP são desenvolvidas para obter limitantes inferiores e, por outro lado, meta-heurísticas, como a *Busca Tabu*, para Limitantes Superiores e um algoritmo híbrido de *Branch-and-Bound* como técnica determinística.

Em proposta mais recente é apresentada uma técnica de Relaxação Lagrangeana para melhorar a obtenção de limitantes inferiores para o problema (MUHRING et al., 2000), que se torna importante na tentativa de se conseguir melhores escalonamentos em tempo aceitável em instâncias com um maior número de jobs.

2.2 O Problema de Escalonamento Job-Shop

O Problema do Job-Shop tem sido exaustivamente estudado ao longo de algumas décadas por grandes pesquisadores da otimização combinatória como Egon Balas, que publicou um artigo clássico no final da década de sessenta. Neste trabalho (BALAS, 1969), Balas trabalha com a seguinte formulação do JobShop:

Encontrar uma sequência ótima para processar *m* itens em *q* máquinas, onde o tempo de completude de cada tarefa é conhecida e:

- *(i)*: uma dada operação, ou job, em um dado item, ou pedido, tem de ser executada em uma máquina específica;
- *(ii)*: as operações pertinentes a um dado item devem ser carregadas em uma sequência tecnologicamente prescrita;
- *(iii)*: existe uma certa liberdade de escolha da sequência das operações para cada máquina;
- *(iv)*: deve-se buscar uma sequência que minimize o tempo total de execução de todas as operações em todos os itens.

O objetivo da técnica proposta é encontrar um caminho *mini-maximal* no grafo disjuntivo. Considera-se o mesmo grafo de precedência enunciado na Seção 2.1, mas agora com arestas a mais, que são chamadas de arestas disjuntas ou arcos disjuntos. Dois arcos são considerados *pares disjuntos* se qualquer caminho no grafo é permitido ligar ao menos um deles (BALAS, 1969). Então trabalha-se a permutação destes arcos até encontrar o menor *caminho crítico*, isto é, o maior caminho no grafo de precedência. Parte-se do princípio que o grafo gerado pela escolha e fixação de um dos arcos disjuntos para cada par que tenha o menor maior caminho crítico, possui o melhor seqüenciamento. Estudos posteriores desenvolvem essa idéia, trazendo bons resultados para o JobShop, conforme Barker e McMahon (1985), Lageweg, Lenstra e Kan (1977).

Esta modelagem de escalonamento também pode ser utilizada em problemas de seqüenciamento de trens em uma malha ferroviária, podendo ser descrito como um caso específico de

JobShop com algumas adaptações (OLIVEIRA, 2001), onde conflitos podem ocorrer quando dois trens ocupam a mesma seção da ferrovia ao mesmo tempo. Propostas de programação por restrições, formulações matemáticas e meta-heurísticas são empregadas para a resolução do problema nesta tese.

Manne (1960), Applegate e Cook (1991) utilizam-se de formulações MIP, em programação matemática, na tentativa de obter soluções ótimas ou mesmo "boas" relaxações. Eles descrevem um modelo matemático que tem como objetivo minimizar o tempo de atraso da execução das tarefas, gerando, por conseqüência, a diminuição do *makespan*.

Estratégias aproximadas, apresentadas por Carlier e Pinson (1989), utilizam o melhor *makespan* encontrado na literatura, como *upper bound* para inicar seu processo de busca da solução, particionando em sub-problemas do JobShop em uma máquina, considerando escalonamentos preemptivos.

Melhorias das regras de inferência, baseadas na seleção imediata de disjunções, empregadas em (CARLIER; PINSON, 1989), são realizadas para determinar se certa operação, não sequenciada, poderia ser ordenada antes ou depois de um conjunto fixado de operações a serem executadas em uma mesma máquina, conforme (APPLEGATE; COOK, 1991).

Estratégias de obtenção de *lower bound*, que utilizam técnicas de relaxações, como redução de restrições ou particionamento do problema original em sub-problemas menores, para conseguir bons limitantes inferiores, são consideradas nos trabalhos de Applegate e Cook (1991), Carlier e Pinson (1989), Sourd e Nuijten (2000).

2.3 Problema de Escalonamento de tarefas em Processadores não-relacionados

Dentre os Modelos de Problemas de Escalonamento de Tarefas pesquisados, pode-se destacar o *Scheduling Unrelated Processors under Precedence Constraints*, por possuir uma maior proximidade (exigindo menos adaptações) com o problema de escalonamento em ambientes distribuídos homogêneos. Seja a seguinte descrição do Problema, proposta por Maculan et al. (1999):

Considere $T = \{t_1, ..., t_n\}$ como o conjunto de tarefas parcialmente ordenadas e $P = \{p_1, ..., p_m\}$ um conjunto de processadores. Tem-se um grafo direcionado acíclico representando as regras de precedência entre as tarefas, denominado $G = [T, A]$, tal que $(t_i, t_j) \in A$ se e somente se a tarefa t_i deva ser executada antes de t_j. Cada tarefa deverá ser associada a exatamente um pro-

cessador e sem preempção (interrupção). Para cada tarefa $t_i \in T$ e cada processador $p_k \in P$, denota-se por d_{jk} o tempo de processamento total da tarefa t_i associada ao processador p_k. Três situações podem ocorrer:

- *processadores idênticos*: $d_{j,k_1} = d_{j,k_2}, \forall t_j \in T, \forall p_{k_1}, p_{k_2} \in P$;
- *processadores uniformes*: $d_{j_1,k_1}/d_{j_1,k_2} = d_{j_2,k_1}/d_{j_2,k_2}, \forall t_{j_1}, t_{j_2} \in T, \forall p_{k_1}, p_{k_2} \in P$;
- *processadores não-relacionados*: tempos de processamento arbitrários $d_{j,k}, \forall t_j \in T, \forall p_k \in P$.

O objetivo do problema é encontrar uma associação ótima das tarefas em T nos processadores de P, minimizando o *makespan*.

Destacam-se como vantagens desta modelagem matemática proposta (i) não exigir que os tempos de processamento das tarefas sejam valores inteiros e (ii) uso de uma quantidade polinomial de variáveis binárias.

Define-se o conjunto de índices $N = \{1,...,n\}$ e $M = \{1,...,m\}$ associados respectivamente com os conjuntos T de tarefas e P de processadores. Considere também a seguinte classe de variáveis 0-1:

$$y_{jk}^s = \begin{cases} 1, & \text{se a tarefa } t_j \text{ é a s-ésima tarefa executada pelo processador } p_k, \\ 0, & \text{se caso contrário,} \end{cases}$$

para todo $j \in N, k \in M$, e $s = 1,...,n$. Para toda tarefa $t_j \in T$, denota-se por $w_j \geq 0$ o tempo de início de suas execuções. Seja $\Gamma(j)$ o conjunto dos índices dos predecessores imediatos da tarefa t_j, ou seja, $\Gamma(j) = \{i \in N | (t_i, t_j) \in A\}$. O problema consiste em minimizar o *makespan* C_{max} sob restrições de precedências, podendo ser formulado como abaixo:

$$\textit{minimizar} \quad C_{max} \tag{2.1}$$

$$s.a.:$$

$$\sum_{k=1}^{m} \sum_{s=1}^{n} y_{jk}^s = 1, \forall j \in N \tag{2.2}$$

$$\sum_{j=1}^{n} y_{jk}^1 \leq 1, \forall k \in M \tag{2.3}$$

$$\sum_{j=1}^{n} y_{jk}^s \leq \sum_{j=1}^{n} y_{jk}^{s-1}, \tag{2.4}$$

$$\forall k \in M, \forall s = 2,...,n$$

$$w_j \geq w_i + \sum_{k=1}^{m}\sum_{s=1}^{n} d_{ik}.y_{ik}^{s}, \quad (2.5)$$
$$\forall i \in \Gamma(j), \forall j \in N$$
$$w_j \geq w_i + d_{ik} - \alpha.[2 - (y_{ik}^{s} + \sum_{r=s+1}^{n} y_{jk}^{r})], \quad (2.6)$$
$$\forall k \in M, \forall s = 1,...,n-1, \forall (i,j) \in N \times N$$
$$C_{max} \geq w_j + \sum_{k=1}^{m}\sum_{s=1}^{n} d_{jk}.y_{jk}^{s}, \forall j \in N \quad (2.7)$$
$$y_{jk}^{s} \in \{0,1\}, \forall j \in N, \forall k \in M, \forall s = 1,...,n$$
$$w_j \geq 0, \forall j \in N,$$
$$\alpha = \sum_{j=1}^{n} d_{jk} \forall k \in M.$$

A restrição (2.2) garante que cada tarefa seja associada a exatamente um processador. As inequações (2.3-2.4) determinam que nenhum processador poderá ser utilizado simultaneamente por mais de uma tarefa. A expressão (2.5) indica as restrições de precedência, ou seja, nenhuma tarefa pode iniciar sua execução antes que todas as suas predecessoras tenham terminado suas execuções por completo. A restrição (2.7) define o *makespan*, ou seja, o tempo de completude máximo. A restrição (2.6), à primeira vista, a mais complicada, define a sequência dos tempos de início do conjunto de tarefas associadas ao mesmo processador, expressando o fato de que a tarefa t_j deve iniciar ao menos d_{ik} unidades de tempo após o início da tarefa t_i, quando for executada após a tarefa t_i, no mesmo processador p_k, i.e. $y_{ik}^{s} = \sum_{r=s+1}^{n} y_{jk}^{r} = 1$ para algum $s = 1,...,n-1$.

Essa modelagem proposta servirá para avaliação dos resultados conseguidos com a Busca Tabu implementada para este problema, porém considerando alguns processadores com velocidades maiores e sem custos de comunicação, conforme (PORTO; RIBEIRO, 1995).

Com a formulação matemática apresentada, pode ser aplicada uma redução de restrições, utilizando uma meta-heurística de Busca Variável como *upper-bound* do modelo e também relaxação linear, considerando sistemas distribuídos homogêneos (DAVIDOVIC et al., 2004).

2.4 Ambientes Computacionais Distribuídos

Uma das grandes motivações para se pesquisar técnicas de escalonamento é trabalhar um melhor seqüenciamento dos processos, sendo executados em um ambiente de processamento distribuído, onde a eficiência e economia na utilização de recursos computacionais é de grande importância, já que este tipo de ambiente é configurado e implementado com o propósito de

fornecer grandes capacidades de computação a um custo relativamente barato.

Na tentativa de adquirir maior poder de computação, nem sempre se faz necessário gastar milhões de dólares com supercomputadores. Na maioria das vezes, uma maior taxa de execução de instruções em relação ao tempo pode ser conseguida através da junção de várias máquinas menores, por custos bem mais razoáveis.

Os arquitetos dos sistemas computacionais têm sempre buscado o Eldorado do projeto de computadores: criar máquinas poderosas a partir da conexão pura e simples de um conjunto de máquinas menores (BOOKMAN, 2003; PATTERSON; HENNESSY, 2000).

Dentre as arquiteturas distribuídas mais conhecidas destacam-se os *Grids* (heterogêneos) (FOSTER; KESSELMAN, 2003) e *Cluster*s (homogêneos ou quase-homogêneos) (BUYYA, 2001).

No caso dos *Grids* tem-se uma plataforma para execução de aplicações paralelas amplamente distribuída geograficamente, heterogênea, compartilhada, sem controle central e com múltiplos domínios administrativos. Pode-se citar como um possível exemplo de utilização de *Grids* a interconexão de várias redes de filiais de uma empresa multinacional, onde poderão coexistir plataformas de hardware e estruturas de banco de dados diferentes com a necessidade de conversar entre si. A construção de *Grids* demanda tempo de planejamento da interconexão, interação entre dados heterogêneos, implantação de softwares especializados, etc. Contudo, ainda assim pode ser um projeto viável em comparação à compra de vários supercomputadores com processamento mais centralizado.

Os *Cluster*s requerem projetos de construção e implantação relativamente mais simples que os *Grids*. Um *Cluster* pode ser entendido como um agrupamento de computadores com compartilhamento de recursos que trabalham com estreita cooperação. Entre as vantagens da utilização desse tipo de arquitetura, em relação aos ambientes centralizados e fortemente acoplados, citam-se:

- *Disponibilidade e Escalabilidade*: considerando que um *Cluster* é um sistema composto de máquinas independentes, ligadas por meio de uma rede local, é muito mais fácil substituir uma máquina sem derrubar o sistema. Esta facilidade de isolamento de uma máquina também facilita a expansão do sistema, sem que seja necessário afetar o funcionamento de determinada aplicação que estiver rodando no *Cluster*;
- *Custo menor*: o preço para uma estrutura com potência computacional equivalente a de um *Cluster* é bem maior.

E como desvantagens (BOOKMAN, 2003; PATTERSON; HENNESSY, 2000; BUYYA, 2001):

- *Custo de Administração*: o custo de administração de um sistema composto por N máquinas é equivalente ao da administração de N máquinas independentes, ao passo que, no caso de máquina processadora com espaço de memória compartilhado com N processadores, seu custo de administração equivale ao de uma única máquina independente;
- *Conexão entre as Máquinas*: enquanto *Cluster*s são em geral conectados usando-se barramento de entrada/saída de um computador, multiprocessadores são ligados ao barramento de memória, este possuindo uma largura de banda passante mais alta, permitindo que os multiprocessadores usem o link da rede a uma velocidade mais alta e com menos conflitos de tráfego com aplicações com uso intensivo de entrada/saída;
- *Divisão da Memória*: um *Cluster* de N máquinas possui N memórias independentes e N cópias do sistema operacional, mas uma máquina multiprocessadora com memória compartilhada permite que um único programa use quase toda a memória do computador. Portanto, um programa sequencial rodando em um *Cluster* tem 1/N-ésimo da memória disponível para ele, quando comparado a um programa sequencial rodando em um multiprocessador simétrico.

Os *Cluster*s são muito usados pelos provedores de serviço Web, devido à sua alta disponibilidade e à rapidez com que tais sistemas podem ser expandidos, além da possibilidade da expansão incremental (PATTERSON; HENNESSY, 2000).

Por se tratar de um ambiente computacional com crescente aumento da implantação, requer o estudo e implementação de softwares específicos de gerência para poder executar programas em paralelo. Dentre os controladores de um *Cluster*, pode-se citar aqueles responsáveis por balancear a carga de tarefas a serem executadas neste ambiente.

Como observado no Capítulo 1, seqüenciar tarefas computacionais em ambientes distribuídos não é um problema simples de resolver, principalmente quando existem restrições de precedência entre tarefas e recursos computacionais, como processadores, memória, dentre outros. Com isso, faz-se necessária a pesquisa de algoritmos cada vez mais eficientes para tratar deste tipo de escalonamento, como heurísticas e meta-heurísticas. Muitas técnicas de otimização são propostas, tanto para escalonamento de tarefas em *Grids* quanto em *Clusters*. Dentre as heurísticas recomendadas como "eficientes" para ambientes distribuídos, cita-se a utilização de Algoritmos Genéticos (GA) na obtenção de "bons escalonamentos" (TOPCUOGLU; HARIRI; WU, 2002).

2.5 Algoritmos Genéticos (GA)

O GA (GOLDEBERG, 1989) é um procedimento de busca baseado nos mecanismos de seleção natural e genético natural. A estrutura básica de um GA consiste de: um conjunto de crossomos, uma função de mutação e um operador de cruzamento. Um cromossomo no contexto de um GA é uma representação codificada de uma solução. O alvo de um GA é produzir soluções próximas do ótimo a partir de uma população de soluções aleatórias – cromossomos, que serão manipulados por uma seqüência de operações unárias e binárias – mutação e cruzamento (*crossover*), respectivamente. O processo inteiro é governado por um esquema de seleção que conduza o conjunto das soluções para soluções de melhor qualidade. O objetivo do operador do cruzamento é combinar implicitamente propriedades boas de dois cromossomos diferentes escolhidos pelo operador de seleção e, conseqüentemente, transfere esperançosamente estas propriedades a sua prole. Visto que o processo de mutação empurra aleatoriamente o processo da busca para uma área ligeiramente mais larga da busca, mudando um código do cromossomo para aumentar as possibilidades de encontrar soluções boas. O processo de executar o cruzamento, a mutação e selecionar os elementos do sobrevivente da população é repetido então até que os critérios de parada sejam encontrados.

2.5.1 Conceitos

No contexto do problema de escalonamento de processos sob restrições de precedências, será utilizado como material base o artigo de Edwin, Nirwan e Hong (1994). Antes de considerar a descrição propriamente dita do algoritmo, serão definidas algumas notações iniciais.

Levando em consideração a formulação do problema descrito na Seção 2.3, descrevem-se os conceitos de *tarefas predecessoras* e *tarefas sucessoras* no grafo de precedências $G = [T, A]$.

t_i é dita ser uma tarefa predecessora de t_j e t_j é tarefa sucessora de t_i se $(t_i, t_j) \in A$.

$Pred(t_i)$ é o conjunto das tarefas predecessoras de t_i.

$Succ(t_i)$ é o conjunto das tarefas sucessoras de t_i.

$$height(t_i) = \begin{cases} 0, & \text{se } Pred(t_i) = \emptyset, \\ 1 + max_{t_j \in Pred(t_i)} height(t_j), & \text{caso contrário} \end{cases}$$

Na Figura 2.1 tem-se um exemplo de aplicação na forma de um grafo de 8 tarefas, com os atributos de tempo de execução e suas alturas. Como exemplo, a tarefa 4 possui tempo estimado de execução igual a 3 e altura 1.

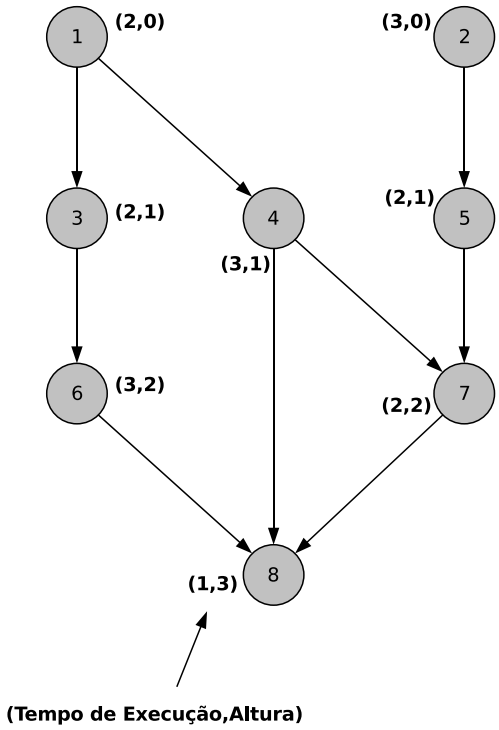

Figura 2.1: Um Grafo de Tarefas

O primeiro item de definição antes de começar a descrever os operadores do algoritmo genético, é identificar uma boa representação para os indivíduos (escalonamento) e que esta seja única. Neste contexto, a representação deverá atender a algumas condições:

- As relações de precedências entre as tarefas deverão ser satisfeitas;
- Toda tarefa deverá aparecer uma única vez no escalonamento.

Na Figura 2.2 tem-se a representação, baseada numa lista de tarefas, onde cada lista representa um processador. Esta representação elimina a necessidade de considerarmos as relações de precedências de tarefas escalonadas em processadores diferentes, considerando as relações somente entre tarefas que estão escalonadas para o mesmo processador.

2.5.2 Operações

Para respeitar a restrição de precedências entre as tarefas do grafo, será imposta uma condição de ordem de alturas das tarefas em todos os escalonamentos gerados, ou seja, *a lista*

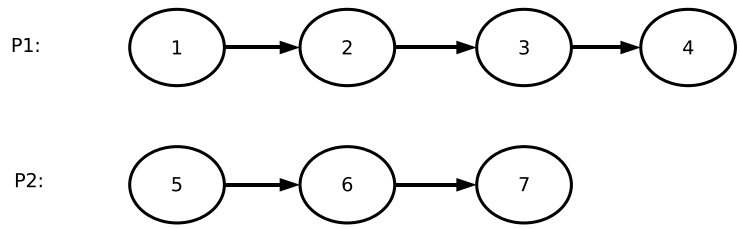

Figura 2.2: Representação do Escalonamento no GA

de tarefas em cada processador do escalonamento é ordenada em ordem ascendente de suas alturas.

O primeiro operador que será descrito é a geração randômica da população inicial. O tamanho desta população é obtido empiramente, conforme resultados dos experimentos.

Desde que a condição de ordem por alturas seja somente uma condição necessária, o escalonamento ótimo pode não satisfazê-la. Para reduzir as chances disso acontecer, será feita uma pequena modificação na definição das alturas das tarefas. Uma nova propriedade "altura", denotada por $height'$ será considerada no algoritmo, da seguinte forma: $height'$ de uma tarefa t_j será um número randômico entre $(max \quad height(t_i)+1)$ e $(min \quad height(t_k)-1)$ sobre toda tarefa $t_i \in Pred(t_j)$ e $t_k \in Succ(T_j)$. No Algoritmo 1 tem-se a descrição do método de geração da população inicial.

1.1 *Calcular height' de todas as tarefas do grafo* $G[T,A]$;
1.2 *Separar as tarefas em grupos de acordo com o valor de height'*;
1.3 **for** $i \leftarrow 1$ **to** $p-1$ *processador* **do**
1.4 | *Para cada conjunto* $G(h)$*, representando as tarefas de altura h, randomicamente escolha um valor r entre 0 e o tamanho máximo de* $G(h)$;
1.5 | *Remova r tarefas de* $G(h)$ *e associe ao processador i*;
1.6 **end**
1.7 *Associe as tarefas que sobraram ao último processador p*

Algorithm 1: GA - Gerador de Escalonamento

Na linha 1.1 é feito o cálculo da nova "altura" para todas as tarefas do grafo. Na linha 1.2 agrupam-se as tarefas do grafo por suas alturas, associando as tarefas de altura 0 na lista $G(0)$,

as tarefas de altura 1 na lista $G(1)$ e assim sucessivamente. No *loop* da linha 1.3, para cada processador, são retiradas randomicamente r tarefas de cada lista G e associada a um único processador, conforme linha 1.5. Após as iterações para cada processador, as tarefas que ainda estiverem sem processador associado, serão alocadas no último processador disponível, o que é feito na linha 1.7.

O número de execuções desse algoritmo dependerá do tamanho da população inicial. Com a definição de uma função para geração de indivíduos, faz-se necessária a implementação de uma função de custo que servirá como qualificadora dos melhores indivíduos de uma população. Denomina-se função *fitness* ou função de custo. A função de custo para este algoritmo genético para o problema de escalonamento de tarefas, será definida sobre o *makespan* de cada processador. Dado S um escalonamento, tem-se:

$$FT(S) = max_{p_j} ftp(p_j),$$

onde $ftp(p_j)$ é o tempo de finalização da última tarefa associada ao processador p_j. Como o objetivo dos operadores de reprodução do algoritmo genético é de maximizar a função de custo, será necessário considerar uma alteração na fórmula anterior visando a minimização do *makespan* em cada processador. Com isso, será definido o valor *fitness* para um escalonamento S da seguinte forma:

$$C_{max} - FT(S),$$

onde C_{max} é o tempo de finalização máximo obtido até o momento.

Um dos operadores de reprodução é a função de *crossover* que tenta buscar as melhores características de dois indivíduos para formar um novo indivíduo, neste caso, um novo escalonamento. O ponto de *crossover* é calculado levando em consideração duas restrições importantes para garantir a legalidade de novos indivíduos:

- A altura das tarefas próximas aos pontos de *crossover* deverão ser diferentes;
- A altura de todas as tarefas imediatamente a frente dos pontos de *crossover* são as mesmas.

Na Figura 2.3 tem-se a aplicação do operador de *crossover* em dois escalonamentos viáveis, baseados no grafo da Figura 2.1. Nota-se que no escalonamento A, com valor da função *fitness* igual a 12, os pontos de cortes estão respeitando as restrições de legalidade do operador, pois

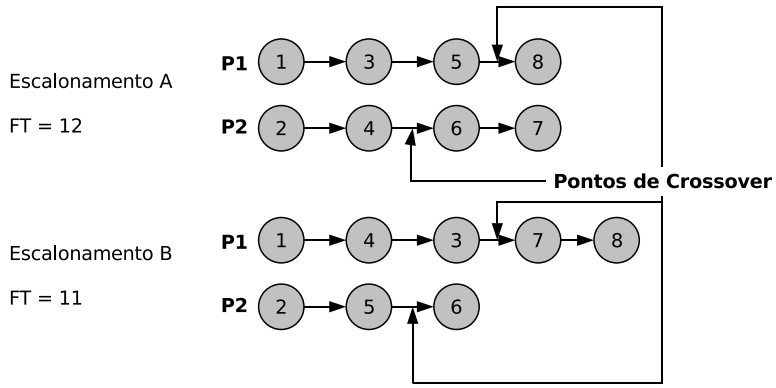

Figura 2.3: Exemplo da Aplicação do Operador de Cruzamento em 2 Escalonamentos Viáveis

as tarefas 4 e 5 possuem a mesma altura e, de forma semelhante, para o escalonamento B, as tarefas 3 e 5 também possuem a mesma altura. Por outro lado, as tarefas 8 e 6 possuem alturas diferentes das alturas das tarefas 5 e 4, o mesmo acontecendo para as tarefas 7 e 6 no escalonamento B.

```
2.1 Randomicamente escolha um valor c entre 0 e a altura máxima de G[T,A];
2.2 foreach processador p_i nos indivíduos A e B do
2.3     Encontrar a última tarefa t_ji no processador p_i com altura c e t_ki é a tarefa
        seguinte a t_ji, ou seja, c = height'(t_ji) < height'(t_ki) e height'(t_ji) é a mesma ∀i;
2.4     foreach processador p_i nos indivíduos A e B do
2.5         Utilizando os pontos de crossover selecionados, faça a troca das partes entre
            os indivíduos para cada processador p_i;
2.6     end
2.7 end
```

Algorithm 2: GA - Crossover

A execução do Algoritmo 2 de *crossover* é baseada em probabilidade que será definida empiricamente. Na linha 2.1 é feita a escolha randômica de uma altura que será utilizada na escolha dos pontos de cortes em cada processador de cada indivíduo. Na linha 2.3, são definidos os pontos de *crossover*, respeitando as regras de legalidade exemplificadas na Figura 2.3. Por fim, na linha 2.5, as tarefas selecionadas são trocadas entre os indivíduos do cruzamento, conforme exemplo da Figura 2.4, gerando os novos escalonamentos viáveis C e D. Os melhores indivíduos encontrados poderão ser incluídos na próxima geração, conforme Algoritmo 3 de reprodução.

Na linhas 3.1 e 3.2 do Algoritmo 3 é feita a definição dos atributos da população, como a quantidade de indivíduos da população que é definida empiricamente. Os valores de *fitness* servem como qualificadores e selecionadores dos melhores indivíduos de cada população, como observado na linha 3.3. Será utilizada uma escolha randômica para selecionar o indivíduo que

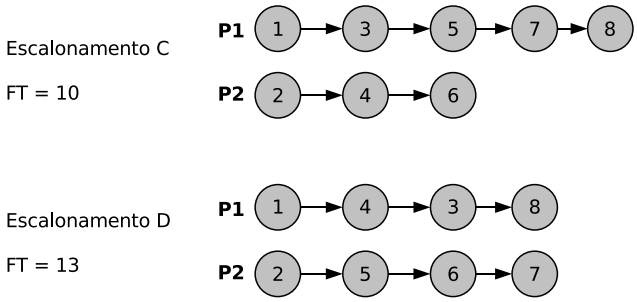

Figura 2.4: Novos Escalonamentos Viáveis gerados a partir da Operação de Cruzamento

passará para a nova população gerada (linha 3.5), bem como, também, o melhor escalonamento será migrado para essa nova população, conforme linha 3.8.

3.1 *Seja NPOP o tamanho da população e POP a população corrente*;
3.2 *NPOP* ← número de indivíduos em *POP* ;
3.3 *Seja NSUM a soma de todos valores fitness dos indivíduos de POP, forme NSUM agrupamentos e associe os indivíduos a esses agrupamentos de acordo com seus valores fitness* ;
3.4 **repeat**
3.5 *Gerar um número randômico entre 1 e NSUM e utilize-o como índice para encontrar o indivíduo correspondente nos agrupamentos* ;
3.6 *Acrescente este novo indivíduo em NEWPOP*;
3.7 **until** $NPOP - 1$;
3.8 *Acrescente o indivíduo com melhor valor de fitness em NEWPOP*

Algorithm 3: GA - Reprodução

O outro operador de reprodução é a função de Mutação. Esta possui uma probabilidade bem menor de ocorrer quando comparado ao operador de crossover. Neste caso, ele servirá para trocar a posição de tarefas de mesma altura em determinado processador. Pode-se observar no Algoritmo 4, que na linha 4.3 é feita a troca das tarefas de altura definida randomicamente, gerando o novo escalonamento.

4.1 *Randomicamente escolha uma tarefa* t_i;
4.2 *Buscar no indivíduo uma tarefa* t_j *de mesma altura*;
4.3 *Formar um novo indivíduo por trocar* t_i *e* t_j *de posição*

Algorithm 4: GA - Mutação

O Algoritmo 5 genético completo é construído juntando todas as funções consideradas ante-

riormente, lembrando que os valores de probabilidades de mutação e *crossover*, a quantidade de indivíduos da população e o número de gerações são específicos para cada tipo de experimento.

```
5.1  Chame o Gerador de Escalonamento N vezes e armazene os indivíduos criados em
     POP;
5.2  repeat
5.3      Computar os valores fitness de cada indivíduo em POP;
5.4      Executar Reprodução, BESTINDIVIDUO ← indivíduo em POP com melhor
         valor fitness;
5.5      repeat
5.6          Escolher dois indivíduos de NEWPOP e chame operador crossover com
             uma probabilidade PROBCROSSOVER;
5.7          if crossover for executado then
5.8              acrescente os novos indivíduos em TMP
5.9          else
5.10             Colocar os dois indivíduos escolhidos em TMP
5.11         end
5.12         foreach indivíduo em TMP do
5.13             Chamar a Função Mutação com probabilidade PROBMUTACAO;
5.14             if mutação for executado then
5.15                 acrescente novo indivíduo em POP
5.16             else
5.17                 Colocar o indivíduo escolhido em POP
5.18             end
5.19         end
5.20     until NPOP/2 vezes ;
5.21     Trocar o indivíduo de POP de menor valor fitness por BESTINDIVIDUO
5.22 until algoritmo converge ;
```

Algorithm 5: GA - Algoritmo Completo

Na linha 5.1 é executada a rotina de geração dos indivíduos para a geração da população inicial. A quantidade de indivíduos é definida a priori de acordo com os experimentos realizados. Com o espaço de busca inicial gerado, a população inicial, pode-se começar o processo da seleção dos melhores escalonamentos utilizando-se como métrica a função de *fitness*, com os valores calculados na linha 5.3. Tanto a operação de cruzamento quanto a função de mutação só serão executadas se suas respectivas probabilidades forem atingidas pelos valores randômicos, verificados nas linhas 5.6 e 5.13. A essência do algoritmo, que é selecionar os melhores indivíduos para as próximas iterações do processo de busca, é executada na linha 5.21, até que o processo convirja, ou seja, termine de gerar a quantidade definida de populações.

No próximo Capítulo será descrito o algoritmo híbrido proposto neste trabalho, apresentando os seus passos de construção e os métodos utilizados em sua compilação.

3 *Algoritmo Híbrido Proposto*

"Nada se cria, nada se perde, tudo se transforma."
Lavousier

Pode ser observado que a questão de seqüenciar tarefas em processadores torna-se importante para a otimização de recursos computacionais. O escalonador de processos é um módulo indispensável, presente na maioria dos núcleos de sistemas operacionais atuais, permitindo a execução de vários programas em conjunto em um *Desktop* simples, por exemplo. Escalonar processos em um ambiente fracamente acoplado, com máquinas independentes, sistemas operacionais diferentes, com intercomunicação entre processos via troca de mensagens, torna-se muito mais complicado ainda e, ao mesmo tempo, de suma importância para um sistema computacional que foi projetado para fornecer muito mais poder de processamento, em comparação a um PC, como um *Cluster*.

Por ser importante, necessário e complexo ao mesmo tempo, o escalonamento de tarefas necessita utilizar-se de técnicas aprimoradas para trabalhar as muitas combinações de seqüenciamentos que existirão em uma aplicação qualquer, ou um conjunto de aplicações. Como analisado no Capítulo 2, vários são os algoritmos implementados com o objetivo de buscar o melhor seqüenciamento de tarefas, visando a economia de tempo de processamento do sistema. Muitas vezes, utilizar-se de uma combinação de técnicas diferentes é a melhor opção.

Neste Capítulo é feita uma proposta de algoritmo híbrido para o problema de escalonamento de tarefas em ambientes distribuídos homogêneos, a partir da formulação MIP apresentada no Capítulo 2, utilizando como limitante superior para o problema os resultados obtidos pelo GA descrito na Seção 2.5, relaxação linear como limitante inferior e fixações de variáveis do modelo.

3.1 Fixação de Variáveis

Existe certa dificuldade (como gasto excessivo de memória e processamento) em se resolver instâncias do problema utilizando-se apenas uma formulação MIP como a descrita na Seção 2.3 e a ajuda dos valores obtidos com alguma meta-heurística (MACULAN et al., 1999), como o algoritmo genético. Logo, faz-se necessária a utilização de outra técnica para ajudar no processo de busca de soluções para o problema. Neste trabalho são utilizadas fixações de variáveis do modelo.

Analisando o modelo MIP proposto, podem ser identificadas três tipos de variáveis importantes, relativas às tarefas do grafo de precedências: (i) uma é a y^s_{jk} que vai designar a sua ordem de execução s na máquina k; (ii) outra é w_j que define o início de execução de cada tarefa j; (iii) a última é *Cmax* que se trata do *makespan* propriamente dito.

```
6.1  Ler G[T,A], T = [t_1,...,t_n], (t_i,t_j) ∈ A, ∀(t_i,t_j), t_i → t_j;
6.2  Ler P = [p_1,...,p_m], D = d[j,k], ∀j ∈ T, ∀k ∈ P;
6.3  Calcula a altura no grafo de todas as tarefas em T;
6.4  UB_GA = makespanGA(G[T,A],P,D);
6.5  G_1 ← Grafo Reduzido apenas com as tarefas de altura menor ou igual a 1;
6.6  S_1 ← GerarMIP(G_1,P,D);
6.7  S_1^⋆ ← SolverMIP(S_1);
6.8  for a ← 2 to AlturaMaxima(G) do
6.9      G_a ← Grafo Reduzido apenas com as tarefas de altura ≤ a;
6.10     S_a ← GerarMIP(G_a,P,D);
6.11     y_a ← y^⋆_{a-1} ∈ S^⋆_{a-1} ∀j ∈ T_{a-1};
6.12     w_a ← w^⋆_{a-1} ∈ S^⋆_{a-1} ∀j ∈ T_{a-1};
6.13     if a = AlturaMaxima(G) then
6.14         UB_Cmax = UB_GA
6.15     end
6.16     while S_a^⋆ ← SolverMIP(S_a) = INVIAVEL do
6.17         liberar variáveis fixadas em S_a referentes à tarefa de mais alta ordem;
6.18     end
6.19 end
6.20 Retornar S_a^⋆
```

Algorithm 6: Algoritmo Híbrido Proposto

Nas linhas 6.1 - 6.2 do Algoritmo 6 são obtidas as informações principais da instância do problema, informando a quantidade n de tarefas e a quantidade m de processadores disponíveis, bem como as relações de precedências entre as tarefas para formar o GAD $G[T,A]$. Os tempos de processamento de cada tarefa ficam armazenados na matriz D, com os valores estimados para a execução em cada processador. Na linha 6.3 são calculadas as alturas de todas as tarefas da instância, conforme Seção 2.5.1 e, na linha 6.4 é obtido o valor retornado pelo algoritmo

GA passando os dados do problema como parâmetros. Após serem feitas as leituras iniciais e calculado o limite superior da instância, é gerada a solução inicial do problema, trabalhando apenas com as tarefas que possuem altura no máximo igual a um, conforme linhas 6.5 - 6.7. As funções GerarMIP e SolverMIP são referentes ao pacote CPLEX [1], com objetivos de ler e resolver problemas MIP respectivamente. O retorno da Função SolverMIP, denominada aqui por $S_1^\star$ contém os valores das variáveis referentes ao escalonamento ótimo do grafo parcial G_1.

Com os valores iniciais obtidos, parte-se para a fixação de variáveis por níveis do grafo. Na linha 6.8 é feita uma iteração para descer no grafo de tarefas até atingir a última tarefa de altura maior. Nas linhas 6.11 e 6.12 são feitas as fixações das variáveis *y* e *w*, conforme valores obtidos nas iterações anteriores. Caso esteja trabalhando com o grafo completo na iteração corrente, utiliza-se o valor de limitante superior retornado pelo algoritmo genético.

Apesar de ter acontecido poucas vezes nos experimentos (apenas em uma das instâncias), faz-se necessário um controle de exceção que pode ocorrer quando fixa-se um valor de variável que poderá gerar uma incosistência quando se acrescentam mais tarefas ao grafo. Por isso são feitas iterações (linhas 6.16 - 6.18) para liberar variáveis fixadas, seguindo o índice de tarefas na ordem descrescente. No final do algoritmo tem-se a solução encontrada pelo algoritmo em $S_a^\star$.

Uma observação importante a se ressaltar é que o Algoritmo 6 transforma o problema original em outro problema semelhante, com a possibilidade de fixação do início de execução das tarefas diferente dos valores do escalonamento ótimo, obtendo assim limitantes superiores para o problema original.

3.2 Exemplo de Aplicação do Algoritmo

Como exemplo da utilização do algoritmo híbrido, será utilizada uma instância com 14 tarefas relacionadas e 2 processadores homogêneos disponíveis. Na Figura 3.1 tem-se a representação do grafo de precedências entre as tarefas e suas alturas descritas como linhas horizontais. Para facilitar a visualização do exemplo, serão considerados todos os tempos de execução das tarefas como de uma unidade de tempo.

1º Passo) Ler as informações básicas da instância como: quantidade de tarefas, número de processadores disponíveis e as relações de precedências:

$T = \{1,2,3,4,5,6,7,8,9,10,11,12,13,14\}$

[1] http://www.ilog.com/products/cplex

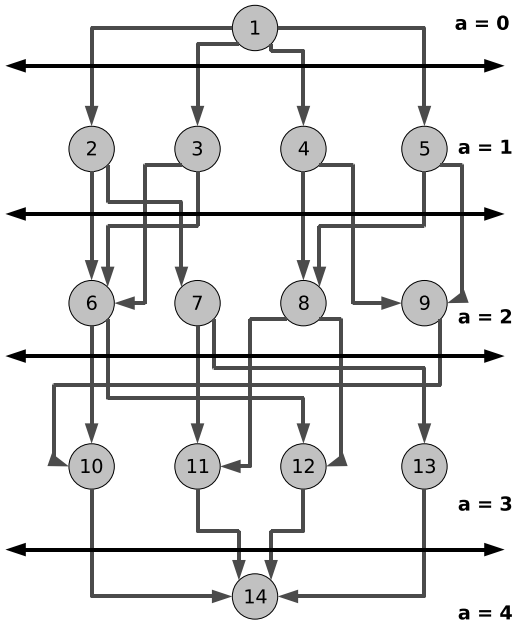

Figura 3.1: Exemplo Prático - 14 instâncias e 2 Processadores disponíveis

$A = \{(1,2),(1,3),(1,4),(1,5),(2,6),...,(6,10),(6,12),...,(12,14),(13,14)\}$

$P = \{1,2\}, d_{jk} = 1, \forall j \in T, \forall k \in P$

2º Passo) Calcular as Alturas de todas as tarefas:

Na Figura 3.1 todas as alturas estão representadas pela letra *a*:

Altura 0: Tarefa 1;
Altura 1: Tarefas 1 à 5;
Altura 2: Tarefas 6 à 9;
Altura 3: Tarefas 10 à 13;
Altura 4: Tarefa 14.

3º Passo) Buscar o *makespan* retornado pelo GA, conforme Seção 2.5.1. Obtem-se como resultado:

$UB_{GA} = 8$

Neste caso, o resultado devolvido pelo GA é exatamente o *makespan* ótimo deste exemplo, ou seja, ele indica que a variável C_{max} é menor ou igual que este valor, o que reduz o espaço de busca do *solver* do CPLEX, eliminando aqueles seqüenciamentos de *makespan* maior.

4º Passo) Gerar o Gráfico Reduzido G_1, conforme Figura 3.2. A partir da geração do grafo reduzido segue-se com a geração do modelo MIP para o problema e a busca da solução com o pacote CPLEX.

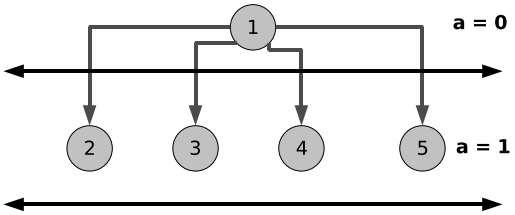

Figura 3.2: Gráfico Reduzido G_1

Para esta instância obtem-se os seguintes resultados:

$$y_1^\star[1,1,1] = 1 : \text{Tarefa 1, primeira a ser escalonada no processador 1;}$$
$$y_1^\star[2,1,3] = 1 : \text{Tarefa 2, terceira a ser escalonada no processador 1;}$$
$$y_1^\star[3,2,2] = 1 : \text{Tarefa 3, segunda a ser escalonada no processador 2;}$$
$$y_1^\star[4,1,2] = 1 : \text{Tarefa 4, segunda a ser escalonada no processador 1;}$$
$$y_1^\star[5,2,1] = 1 : \text{Tarefa 5, primeira a ser escalonada no processador 2;}$$

$$w_1^\star(1) = 0 : \text{Tempo de início de execução da Tarefa 1 é 0;}$$
$$w_1^\star(2) = 2 : \text{Tempo de início de execução da Tarefa 2 é 2;}$$
$$w_1^\star(3) = 2 : \text{Tempo de início de execução da Tarefa 3 é 2;}$$
$$w_1^\star(4) = 1 : \text{Tempo de início de execução da Tarefa 4 é 1;}$$
$$w_1^\star(5) = 1 : \text{Tempo de início de execução da Tarefa 5 é 1;}$$

Na Figura 3.3 observa-se o escalonamento parcial das cinco primeiras tarefas do grafo, gerando um makespan parcial igual a 3.

5º Passo) Iterações do *Loop* Principal:

Na primeira iteração, serão trabalhadas as tarefas que tenham altura menor ou igual a dois, conforme Figura 3.4.

Novamente é gerado o modelo MIP do problema e fixam-se os valores de y_2 e w_2 em $y_1^\star$ e $w_1^\star$ respectivamente. É realizada a busca da solução novamente gerando os valores:

$$y_2^\star[6,2,4] = 1 : \text{Tarefa 6, quarta a ser escalonada no processador 2;}$$

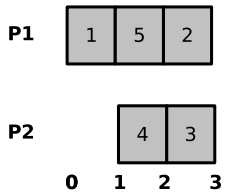

Figura 3.3: Escalonamento em $S_1^\star$

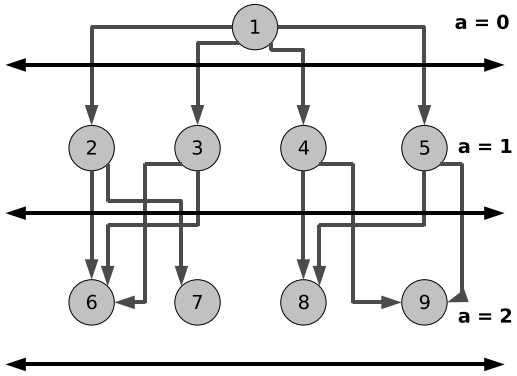

Figura 3.4: Gráfico Reduzido G_2

$y_2^\star[7,1,5] = 1$: Tarefa 7, quinta a ser escalonada no processador 1;
$y_2^\star[8,2,3] = 1$: Tarefa 8, terceira a ser escalonada no processador 2;
$y_2^\star[9,1,4] = 1$: Tarefa 9, quarta a ser escalonada no processador 1;

$w_2^\star(6) = 4$: Tempo de início de execução da Tarefa 6 é 4;
$w_2^\star(7) = 4$: Tempo de início de execução da Tarefa 7 é 4;
$w_2^\star(8) = 3$: Tempo de início de execução da Tarefa 8 é 3;
$w_2^\star(9) = 3$: Tempo de início de execução da Tarefa 9 é 3;

Na Figura 3.5 observa-se o escalonamento parcial das nove primeiras tarefas do grafo, gerando um makespan parcial igual a 5.

Juntando os resultados $y_1^\star$ e $w_1^\star$ e agora $y_2^\star$ e $w_2^\star$, tem-se a solução $S_2^\star$.

Agora, na segunda iteração, com as tarefas de altura menor ou igual a 3, conforme Figura 3.6, tem-se o grafo reduzido G_3. Gera-se o modelo agora fixando os valores de y_3 e w_3 em $y_2^\star$

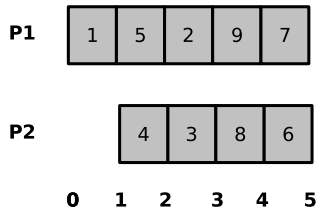

Figura 3.5: Escalonamento em $S_2^\star$

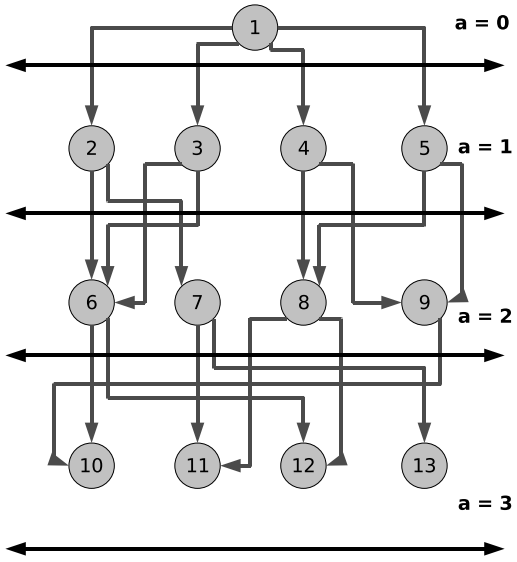

Figura 3.6: Gráfico Reduzido G_3

e $w_2^\star$ respectivamente. Inicia-se o processo de busca pelo CPLEX, obtendo os seguintes valores das variáveis:

$$y_3^\star[10,2,6] = 1 : \text{Tarefa 10, sexta a ser escalonada no processador 2;}$$
$$y_3^\star[11,1,6] = 1 : \text{Tarefa 11, sexta a ser escalonada no processador 1;}$$
$$y_3^\star[12,1,7] = 1 : \text{Tarefa 12, sétima a ser escalonada no processador 1;}$$
$$y_3^\star[13,2,5] = 1 : \text{Tarefa 13, quinta a ser escalonada no processador 2;}$$

$$w_3^\star(10) = 6 : \text{Tempo de início de execução da Tarefa 10 é 6;}$$
$$w_3^\star(11) = 5 : \text{Tempo de início de execução da Tarefa 11 é 5;}$$
$$w_3^\star(12) = 6 : \text{Tempo de início de execução da Tarefa 12 é 6;}$$

$w_3^\star(13) = 5$: Tempo de início de execução da Tarefa 13 é 5;

Na Figura 3.7 observa-se o escalonamento parcial das treze primeiras tarefas do grafo, gerando um makespan parcial igual a 7.

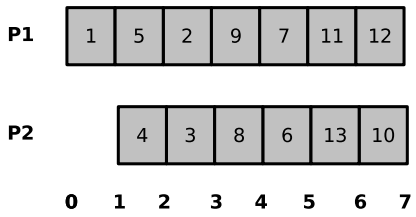

Figura 3.7: Escalonamento em $S_3^\star$

Na última iteração, escalonamos a tarefa 14 restante, gerando o *makespan* representado na Figura 3.8, lembrando que neste ponto será utilizado o valor do *makespan* retornado pelo GA como limitante superior, ou seja,

$C_{max} \leq 8$.

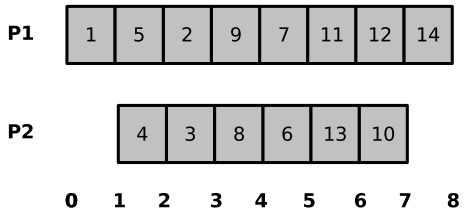

Figura 3.8: *makespan* da instância de 14 tarefas em 2 processadores

Neste exemplo demonstrado, não ocorreu o caso de inviabilidade do modelo, tratado nas linhas 6.16 - 6.18 do algoritmo. Porém, suponha que tivesse ocorrido uma inviabilidade do modelo na segunda iteração do *loop* central. Suponha ainda que as tarefas fixadas até este momento são $1, \ldots, 9$. Com isso, deveríamos, primeiramente, liberar as variáveis relacionadas à tarefa 9, que possui o maior índice no conjunto, ou seja:

$y[9,1,4]$ livre e $w(9)$ livre.

Caso a inviabilidade do modelo ainda continuasse, seria executado o mesmo processo para a tarefa 8 e assim sucessivamente.

Com a utilização deste algoritmo híbrido, espera-se a obtenção de bons resultados de *makespan* para as instâncias utilizadas nos experimentos computacionais.

4 *Resultados Computacionais*

"Só é útil o conhecimento que nos torna melhores"
Sócrates

Com o estudo e implementação das técnicas de resolução do problema de escalonamento de tarefas em ambientes distribuídos homogêneos, nos Capítulos anteriores, pode-se realizar exercícios práticos, com instâncias do problema, para análise de resultados e estudo da eficiência e eficácia dessas ferramentas.

Primeiramente foram pesquisados modos de experimentos em outros trabalhos co-relacionados, ou seja, da área de escalonamento de tarefas. Estes experimentos geralmente são realizados com instâncias do problemas, geradas comumente por métodos matemáticos (Eliminação Gaussiana por exemplo), ou por observação de um ambiente real (algoritmo "dividir e conquistar"). Faz-se importante, em projetos de experimentos, ter em mãos instâncias do problema que se aproximem o máximo de situações reais e, muitas vezes, esses tipos de técnicas nos permitem isso.

Serão apresentadas as instâncias utilizadas para testes dos algoritmos implementados: MIP, Algoritmo Genético e Fixação de Variáveis do modelo matemático. Essas instâncias foram obtidas através de outros trabalhos da área de escalonamento de tarefas em ambientes distribuídos, homogêneos e heterogêneos (TOPCUOGLU; HARIRI; WU, 2002; COLL; RIBEIRO; SOUZA, 1999), geradas pelas mais diversificadas técnicas utilizadas normalmente para se obter simulações bem próximas de situações reais. Descrevem-se os resultados conseguidos passo-a-passo, aplicando uma técnica de cada vez, com o objetivo de se analisar as melhorias conseguidas com a combinação das mesmas no algoritmo híbrido. Por fim, são realizadas simulações de aplicações em um *Cluster* real, validando a eficiência das técnicas implementadas e validando o objetivo deste trabalho de modo prático.

Para a execução da Formulação MIP na resolução das instâncias, foi utilizada a ferramenta CPLEX 9.0 versão Win32 em uma máquina Pentium Duo Core 2.80GHz com 2GB de RAM e, para a execução dos experimentos com o GA, uma máquina Celeron M 360, 1GB de RAM com Sistema Operacional Linux Ubuntu 7.04 e biblioteca GAlib 2.47. Essa utilização de duas máquinas foi necessária por dois motivos: *(i)* todos os algoritmos estavam sendo desenvolvidos no Sistema Operacional Linux, (até a identificação da ineficiência das heurísticas do pacote GLPK [1]), por ser o *Cluster* da UFES todo projetado neste sistema e *(ii)* por não possuirmos uma licença do CPLEX para Linux, apenas para Sistema Windows obtida através da parceria com o Departamento de Engenharia Elétrica da UFES.

4.1 Instâncias do Problema

Algumas das instâncias utilizadas neste trabalho foram obtidas de (COLL; RIBEIRO; SOUZA, 1999). Cada uma delas é composta por um tipo de grafo de precedência onde cada tarefa possui seu custo computacional de execução em cada máquina disponível. O objetivo é buscar instâncias que se aproximem o máximo possível de situações reais.

Diferentes tipos de instâncias foram consideradas, com formas de paralelismo diferentes, conforme Figura 4.1.

Os tipos de técnicas de geração dos grafos direcionados acíclicos (GAD's), representando as relações de precedência entre as tarefas, estão relacionadas abaixo:

- *(d)*: Systolic Computations;
- *(f)*: Fast Fourier Tranform;
- *(g)*: Gaussian Elimination;
- *(i)*: Generic Iterative Algorithm;
- *(v)*: Divide-and-Conquer type Algorithm;
- *(r)*: Random.

Cada grafo é testado com diferentes valores de processadores disponíveis (2, 4 e 8), gerando 28 instâncias no total.

[1] http://www.gnu.org/software/glpk

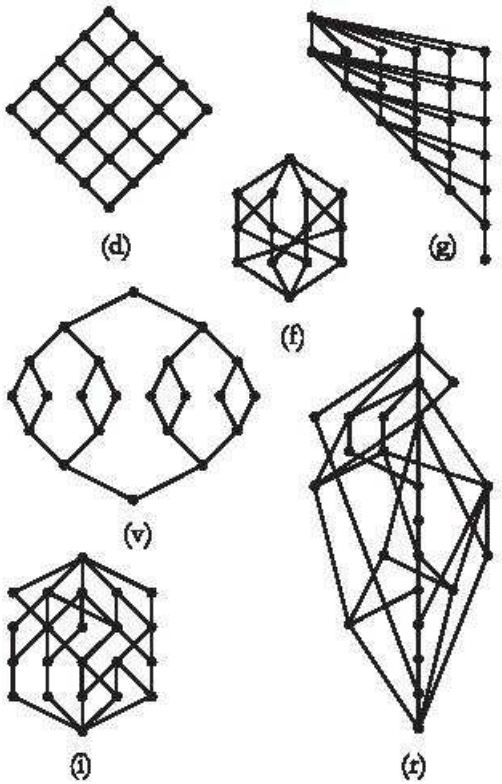

Figura 4.1: Instâncias utilizadas

4.2 Resultados da Formulação MIP

Primeiramente, será utilizada a formulação MIP proposta para o problema sem auxílio de outras técnicas, para avaliar o rendimento do modelo, analisando quanto tempo será gasto para se encontrar a solução ou, pelo menos, o quanto é possível aproximar-se da solução ótima em tempo aceitável.

Na Tabela 4.1 descrevem-se os resultados obtidos com a utilização da formulação matemática apresentada na Seção 2.3. Na primeira coluna tem-se a descrição das instâncias: <modelo do grafo><número de tarefas> <número de processadores disponíveis>. Por exemplo, f14 4 significa que o modelo do grafo foi gerado via *Fast Fourier Tranform*, com 14 tarefas ou vértices do grafo e com 4 processadores disponíveis. Na segunda coluna, os resultados de *makespan* obtidos. Na última coluna, tem-se o tempo de execução do resolvedor do CPLEX para retonar a solução do problema.

Observa-se a dificuldade em resolver problemas já com um número pequeno de tarefas com a modelagem proposta. Para demonstração foram utilizadas instâncias dos modelos f e i, de 14 e 22 tarefas, em 2, 4 e 8 processadores homogêneos disponíveis. Os experimentos foram reali-

Instâncias	Solução	Tempo (seg)
f14 2	??	>4h
f14 4	5	264,61
f14 8	5	0,64
i22 2	??	>4h
i22 4	??	>4h
i22 8	??	>4h

Tabela 4.1: Execução da Instâncias com MIP puro

zados com instâncias de até 34 tarefas que executaram por mais de 4 horas, retornando resultado para apenas duas das instâncias utilizadas. Como o *Solver* do CPLEX não melhorava o makespan encontrado, o *gap* entre os limtes superior e inferior permanecia estático por horas, ou até mesmo dias e, também, o gasto de memória crescia exponensialmente, não foram gastos mais de 4 horas na execução dos experimentos. Com esta dificuldade de obter bons resultados computacionais com apenas a formulação MIP pura, geralmente são consideradas, para as variáveis pertencentes à função objetivo do modelo, soluções retornadas por algoritmos heurísticos para colaborar como limitantes para o modelo, trazendo soluções aproximadas, como observado nos trabalhos de Cavalcante (1998), Maculan et al. (1999), Carlier e Pinson (1989).

4.3 Resultados com o Algoritmo Genético

Considerando o algoritmo genético descrito na Seção 2.5.1, serão resolvidas as instâncias propostas com esta meta-heurística, para analise do fator de aproximação dos resultados retornados pelo algoritmo e, também, quando se torna mais apropriado utilizar o GA como técnica de resolução do problema.

Para a execução do genético foram utilizados os seguintes parâmetros, obtidos empiricamente:

- *Número de Gerações* = 3000;
- *Tamanho da População* = 20;
- *Probabilidade de Mutação* = 10%;
- *Probabilidade de Crossover* = 100%;

Foram obtidos os resultados mostrados na Tabela 4.2. Na primeira coluna tem-se a descrição das instâncias, como na Tabela 4.1. Na segunda coluna, o valor do melhor *makespan*

encontrado. Nas terceira e quarta colunas, estão descritas as soluções do problema encontradas e o tempo de execução do algoritmo genético respectivamente. Na última coluna, a relação de aproximação do resultado retornado pelo GA em comparação com o melhor *makespan* da instância.

Instâncias	SOL	GA	Tempo (seg)	(GA - SOL)/SOL
f14 2	8	8	0,36	0
f14 4	5	6	0,41	0,20
f14 8	5	6	0,29	0,20
i22 2	13	13	0,60	0
i22 4	8	8	0,52	0
i22 8	6	10	0,37	0,67
v22 2	12	12	0,56	0
v22 4	8	9	0,32	0,13
v22 8	7	8	0,36	0,14
g23 2	14	14	0,65	0
g23 4	10	11	0,39	0,10
g23 8	8	9	0,38	0,13
d25 2	14	15	1,04	0,07
d25 4	10	12	0,35	0,20
d25 8	9	12	0,39	0,33
f34 2	20	20	0,81	0
f34 4	11	12	0,45	0,09
f34 8	7	12	0,48	0,71
r48 2	27	28	0,80	0,04
r48 4	18	18	0,57	0
r48 8	16	18	0,62	0,13
i50 2	26	26	2,24	0
i50 4	15	20	0,64	0,33
i50 8	10	19	0,64	0,90
				Média de Aproximação = 0,16

Tabela 4.2: Execução das Instâncias com o GA

Na última linha da tabela é apresentada a aproximação média dos resultados conseguidos com o GA, que foi de 16% em relação ao valor ótimo de makespan conhecido das instâncias. Este "bom" valor de aproximação do ótimo pode colaborar no estudo do quão eficiente é este método em relação ao algoritmo híbrido e, em que situações, fica mais viável utilizar uma ou outra técnica para se resolver determinada instância.

Pode-se identificar melhores soluções quando a proporção $N^{o}Tarefas/N^{o}Processadores$ é maior, ou seja, o fator de aproximação é diretamente proporcional a este valor. Isso se deve ao fato, principalmente, de a operação de crossover necessitar localizar pontos de corte em todos os processadores ao mesmo tempo para executar. Se existem menos tarefas associadas a determinados processadores, ocorrerá menor chance do *crossover* ser realizado. Com isso, quanto mais uniforme a distribuição de tarefas entre os processadores, maiores as chances de realização de cruzamentos entre os indivíduos e, conseqüentemente, maior a probabilidade de geração de indivíduos melhores. Analisando o gráfico da Figura 4.2, pode ser identificado um aumento considerável da aproximação da solução retornada pelo GA em relação ao melhor *makespan* obtido, conforme a quantidade de processadores aumenta. Logo, é possível verificar que os melhores resultados devem ser obtidos para instâncias onde essa relação entre a quantidade de tarefas e quantidade de processadores é maior, como as instâncias f14_2, i22_2, i22_4, v22_2, g23_4, d25_2, f34_4, r48_2, r48_4 e podem ser obtidos piores resultados quando essa relação é menor, como por exemplo em i22_8, d25_4, d25_8, f34_8 e i50_8.

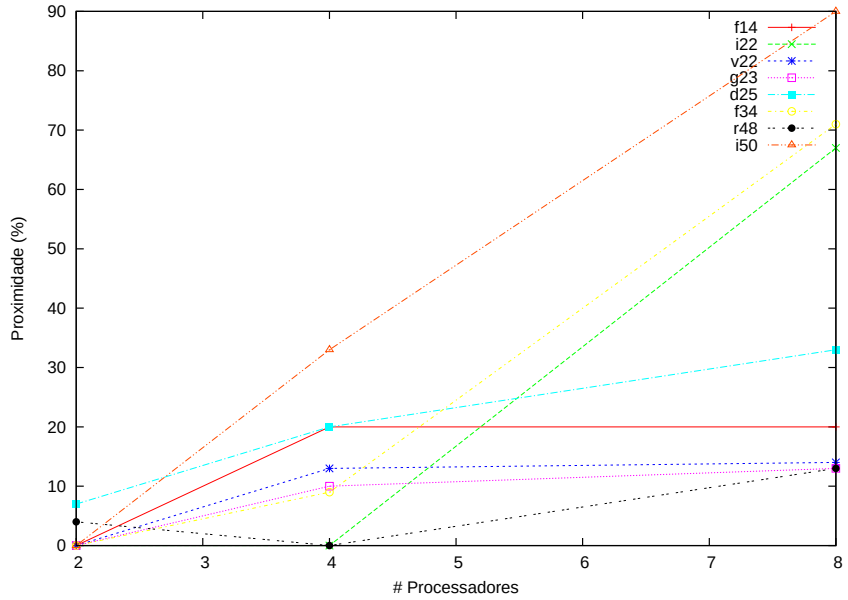

Figura 4.2: Comparativo entre a aproximação obtida com o GA e a Quantidade de Processadores Disponíveis

Uma outra questão que pode causar resultados ruins produzidos pelo GA é a possibilidade de um escalonamento ótimo, de uma determinada instância, não respeitar a ordem ascendente das alturas, ou seja, uma tarefa de uma altura maior ser escalonada antes de uma tarefa de altura menor, sem afetar a ordem de precedência entre as tarefas. Logo, este escalonamento ótimo não estaria no espaço de busca do genético.

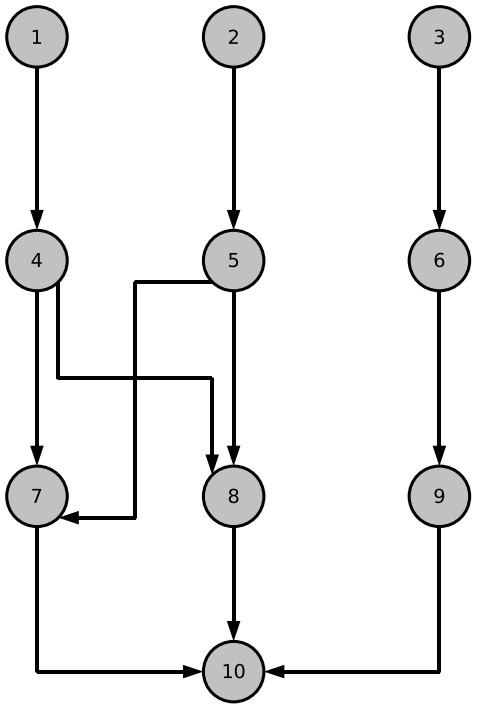

Figura 4.3: Exemplo de GAD que o GA não encontraria o melhor sequênciamento no seu espaço de busca

Será analisado um exemplo com 10 tarefas relacionadas e com 2 processadores disponíveis p_1 e p_2. Baseado na Figura 4.3, tem-se que o escalonamento ótimo associa as tarefas 1, 4, 7 e 10, nesta ordem, ao processador p_1 e as tarefas 2, 5, 8, 3, 6 e 9, nesta ordem, ao processador p_2. O *makespan* do escalonamento é 13. Nota-se que, respeitando a restrição de altura imposta pelo GA, conforme Algoritmo 1 linha 1.2 e Algoritmo 2 linha 2.3, a tarefa 3, de altura 0, é necessariamente escalonada antes das tarefas de altura 1. Neste caso, a tarefa 3 será escalonada antes das tarefas 4 e 5, atrasando o início da tarefa 7, conforme Figura 4.4 e, conseqüentemente aumentando o valor do *makespan*.

Como conclusão desta Seção, destaca-se a questão de quando é "melhor" usar o GA apenas, como resolvedor do problema ou partir para a estratégia híbrida apresentada. Quando a

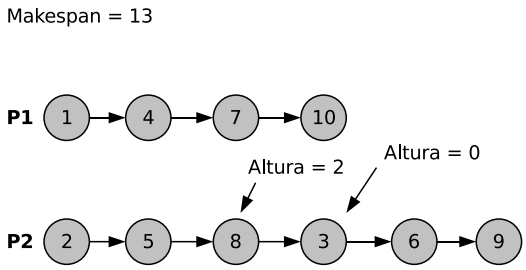

Figura 4.4: Escalonamento Ótimo em 2 processadores que não satisfaz as restrições de altura

situação for uma relação menor de tarefas por processadores disponíveis, ou quando existirem bons valores de *makespan* com combinações de seqüenciamentos em que a restrição de altura não é satisfeita, faz-se melhor estratégia utilizar a "força" do algoritmo híbrido para resolver a instância em questão.

4.4 Resultados Formulação MIP e Limitantes Superiores

Os valores produzidos pelo algoritmo genético servirão de limites superiores para a variável de *makespan* do problema C_{max}. Nota-se que, mesmo com "fator de aproximação zero", ou seja, com a meta-heurística retornando exatamente o valor do melhor makespan obtido, esses valores produzidos ainda colaboram muito pouco para a resolução do problema.

Instâncias	Solução	Tempo (seg)
f14 2	??	>4h
f14 4	5	172,72
f14 8	5	0,62
i22 2	??	>4h
i22 4	??	>4h
i22 8	??	>4h

Tabela 4.3: Execução da Instâncias com MIP e Limitante Superior para o *makespan*

Na Tabela 4.3, são descritos os resultados obtidos com a formulação MIP com a informação adicional do valor do limitante superior obtido executando a instância com o algoritmo

genético. Percebe-se que ainda permanece a dificuldade na obtenção de resultados para instâncias relativamente pequenas com pequenas melhorias no tempo de execução do processo de busca pelo CPLEX. Vale destacar que, pelo fato do algoritmo genético ter tido pouca influência nos resultados da formulação MIP enquanto um produtor de limitantes superiores, não foram realizadas tentativas de melhorias nos resultados obtidos com o GA, já descritos na Seção 4.3.

4.5 Resultados com o Algoritmo Híbrido

Um dos problemas que pode ser verificado nesta modelagem matemática proposta é a grande quantidade de variáveis geradas pelo modelo, mesmo para instâncias com poucas tarefas e processadores. Uma estratégia interessante é tentar reduzir o espaço de busca fixando variáveis do modelo. Combinando técnicas exatas e aproximadas e essa fixação de algumas variáveis do modelo, pode-se reduzir significamente o tempo de procura pela solução. Esta é a proposta do algoritmo híbrido apresentado e aplicado agora na resolução das instâncias.

Na Tabela 4.4 tem-se os resultados obtidos utilizando o algoritmo híbrido descrito na Seção 3.1. Na primeira coluna apresenta-se a descrição da instância e a quantidade de processadores disponíveis, na segunda coluna, o melhor valor do *makespan* obtido, na terceira coluna, o valor obtido com o GA e seu respectivo tempo de execução e, na última coluna, o tempo total de execução do algoritmo híbrido.

Observa-se que a quantidade de tarefas, ou a quantidade de processadores, não são fatores determinantes para o quesito tempo de execução do procedimento, como identificado nas instâncias i50_8 e g23_4. Outra questão importante de se destacar é a maior facilidade de resolução do algoritmo para instâncias com um maior número de processadores, como observado na maioria das instâncias com 8 processadores.

Os resultados completos de todas as execuções, passo-a-passo, podem ser conferidos no Anexo A.

O algoritmo se mostrou bem estável, necessitando passar por tratamento de exceções (desfixação de variáveis) apenas para a instância g23.

Na prática, quando considera-se um procedimento de escalonamento de tarefas em um determinado sistema distribuído computacional, deve-se lembrar que existe o ágio do tempo de execução do otimizador, ou seja, o custo de processamento para que o procedimento retorne o melhor *makespan*, aqui representando o módulo "escalonador de processos" presente em qualquer sistema operacional de qualidade e multiprogramado. Quando são analisados os resultados

Instâncias	SOL	GA (seg)	Tempo Total (seg)
f14 2	8	8 (0,36)	0,78
f14 4	5	6 (0,41)	0,66
f14 8	5	6 (0,29)	0,52
i22 2	13	13 (0,60)	5,26
i22 4	8	8 (0,52)	11,77
i22 8	6	10 (0,37)	1,36
v22 2	12	12 (0,56)	2,30
v22 4	8	9 (0,32)	1,17
v22 8	7	8 (0,36)	1,53
g23 2	14	14 (0,65)	8,92
g23 4	10	11 (0,39)	56,62
g23 8	8	9 (0,38)	1,66
d25 2	14	15 (1,04)	3,09
d25 4	10	12 (0,35)	4,76
d25 8	9	12 (0,39)	1,89
f34 2	20	20 (0,81)	159,77
f34 4	11	12 (0,45)	796,43
f34 8	7	12 (0,48)	7,06
r48 2	27	28 (0,80)	47,76
r48 4	18	18 (0,57)	39,19
r48 8	16	18 (0,62)	46,86
i50 2	26	26 (2,24)	178,02
i50 4	15	20 (0,64)	331,65
i50 8	10	19 (0,64)	29,03

Tabela 4.4: Execução da Instâncias com o Algoritmo Híbrido

de um *makespan* e verifica-se o seu ganho de tempo em relação ao tempo de um *makespan* viável qualquer, deve-se acrescentar o custo de se executar um otimizador para o sistema.

4.6 Experimentos em um *Cluster* Real

Visando validar o objetivo principal deste trabalho, que é construir algoritmos para a otimização de recursos computacionais, especificamente neste caso, processamento, através da obtenção de melhores seqüenciamento das tarefas a serem executadas no ambiente distribuído, serão realizados alguns exercícios práticos em um *Cluster*. Com isso, pode-se abordar problemas mais específicos, como o tempo de carga de um processo para a memória da máquina, caso esse que não foi abordado nos experimentos anteriores.

Para realização dos experimentos em um ambiente distribuído homogêneo real, será uti-

lizado o *Cluster* do Laboratório de Computação de Alto Desempenho (LCAD) da UFES [2]. Trata-se de um conjunto de 64+1 processadores (ATHLON XP 1800+) com performance teórica máxima de 204 Gflops/s. São simuladas algumas instâncias do problema neste *Cluster* para analisar a importância de técnicas de otimização para obtenção de melhores escalonamentos, visando uma maior vazão (*troughput*) de processamento de tarefas pelo sistema.

O *Cluster* da UFES é administrado pelo software SGE (*Sun Grid Engine*) [3] que fornece alguns comandos para gerenciamento de processos no sistema, alguns destes são:

- *qstat*: exibe os processos que estão executando no *Cluster* ou aguardando na fila, bem como algumas características como que usuário submeteu o processo, horário de agendamento e máquina associada;
- *qsub*: permite a submissão de processos no *Cluster*, onde o usuário pode informar o programa que gerará os processos, a data de início de cada processo, em que máquina será executado, etc;
- *qdel*: deleta processos das filas de espera ou mesmo executando.

Nos experimentos, a simulação dos tempos de cada tarefa foi feita através da função *sleep* do linux, ou seja, cada tarefa será um *script bash* com apenas o comando *sleepxm*, onde *x* representa a quantidade de minutos que o processo ficará "dormindo" em alguma máquina do *Cluster*.

Para a simulação das precedência entre as tarefas e a associação de uma tarefa a alguma máquina, utilizou-se o comando *qsub* como exemplificado abaixo:

qsub -a 200802202210 -q tpol12 j1.sge, ou seja, submeta a tarefa 1 às 22:10 horas do dia 20 de fevereiro de 2008, associando a tarefa à fila tpol12. O termo *tpolx* representa a fila de espera de cada máquina que compõe o *Cluster*. O SGE permite a configuração de várias filas de espera de processos, mas no caso do *Cluster* do LCAD, cada máquina foi configurada para ter apenas uma fila de espera, ou seja, quando é referenciada a fila *tpol12* está-se, na verdade, se referindo à máquina 12.

Foi escolhida essa abordagem de agendamento de tarefas nos processadores ao invés de envio de mensagens entre processos, para que ocorresse o mínimo possível de interferência de softwares externos ao SGE nos experimentos.

[2] http://www.lcad.inf.ufes.br
[3] http://www.sun.com/gridware

Um fato que ocorre com certa freqüência no *Cluster* atualmente, apenas utilizando o SGE como escalonador de tarefas nos processadores, é a dificuldade de otimizar a utilização dos recursos quando existem *jobs* seriais e paralelos a serem processados, sendo as aplicações seriais, submetidos por vários usuários, com tempos de processamento relativamente maiores que os tempos das tarefas paralelas. Nesses casos, o que acontece é o SGE alocar as tarefas seriais primeiramente e só depois processar as tarefas paralelas que geralmente tem tempos de execução menores. Como primeiro experimento será utilizado esse caso descrito, para avaliar o algoritmo híbrido proposto neste trabalho.

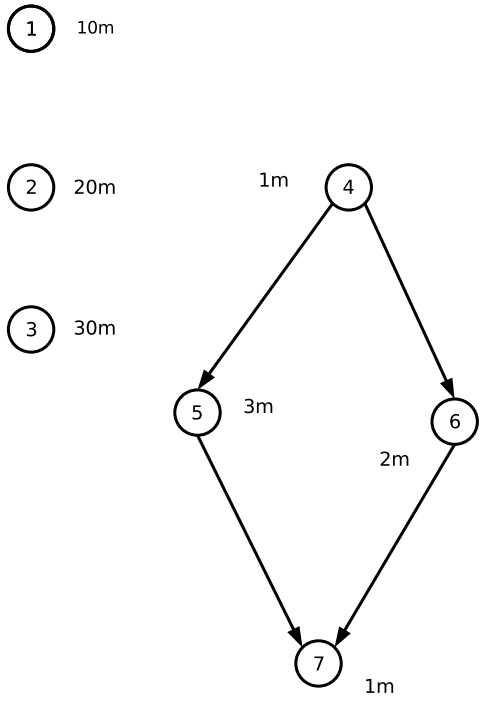

Figura 4.5: Primeiro Experimento em Ambiente Real

Na Figura 4.5 tem-se diagramado o grafo de precedências com 7 tarefas no total, sendo 3 seriais, com tempos de execução de 10, 20 e 30 minutos e, 4 tarefas paralelas, com tempos de 1, 2, 3 minutos de custo de processamento.

Observa-se uma melhoria de 25% do makespan ótimo em relação ao escalonamento viável, conforme Figura 4.6. Como próxima análise será simulada o processamento das 7 tarefas no *Cluster*.

Na Tabela 4.5 estão descritos os horários de início e fim da execução das 7 tarefas, utilizando os dados do escalonamento viável e o otimal. Na primeira coluna tem-se a identificação dos

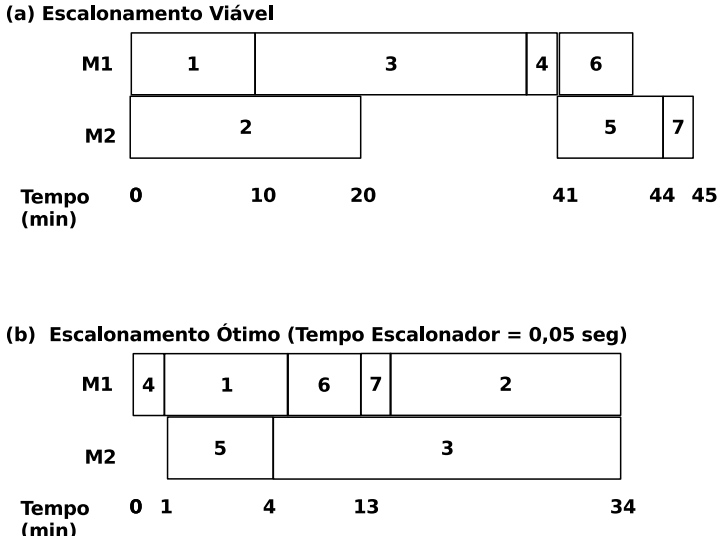

Figura 4.6: Primeiro Experimento em Ambiente Real: (a) Escalonamento Viável e (b) Escalonamento Ótimo

Tarefa	Início (Normal)	Fim (Normal)	Início (Otimal)	Fim (Otimal)
t_1	09:00.10	09:10.10	11:01.05	11:11.05
t_2	09:00.05	09:20.05	11:14.10	11:34.10
t_3	09:10.10	09:40.10	11:04.05	11:34.05
t_4	09:40.15	09:41.15	11:00.13	11:01.13
t_5	09:41.02	09:44.02	11:01.07	11:04.07
t_6	09:41.12	09:43.12	11:11.05	11:13.05
t_7	09:44.04	09:45.04	11:14.20	11:34.20

Tabela 4.5: Primeiro Experimento em Ambiente Real - Horários de início e fim de execução de cada tarefa

índices das tarefas, nas 2^o e 3^o colunas os horários de início e fim da execução das tarefas seguindo o seqüenciamento viável e, nas 4^o e 5^o colunas, os tempos do escalonamento otimal retornado pelo algoritmo híbrido proposto. Analisando os valores, tem-se uma economia de 25% no tempo de execução total das aplicações consideradas.

Como segundo experimento, será utilizada uma instância de 7 tarefas com 2 processadores disponíveis. Na Figura 4.7 observa-se o grafo de precedência entre as tarefas da instância com seus tempos de execução em minutos. Um *makespan* viável, apenas distribuindo as tarefas nos processadores por ordem sequencial de seus índices, é apresentado na Figura 4.8(a). Na Figura 4.8(b) tem-se o escalonamento ótimo das tarefas, com tempo de processamento do otimizador de 0,05 segundos, valor este que deverá ser acrescentado ao *makespan* ótimo.

Na Tabela 4.6 tem-se nas primeira coluna a identificação da tarefa, nas segunda e terceira

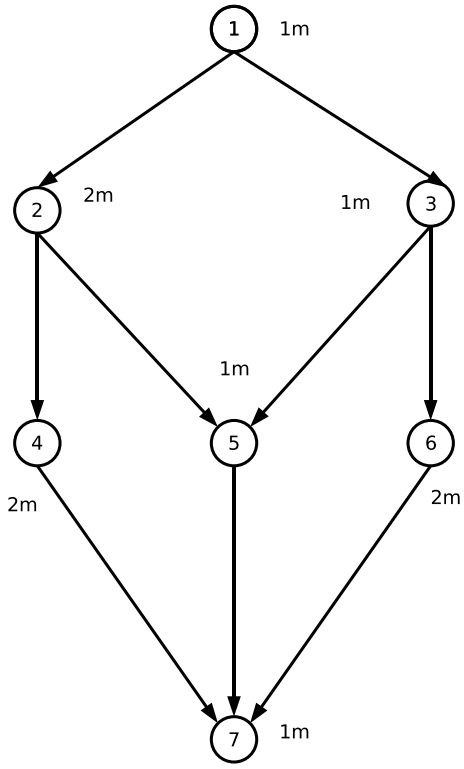

Figura 4.7: Segundo Experimento em Ambiente Real

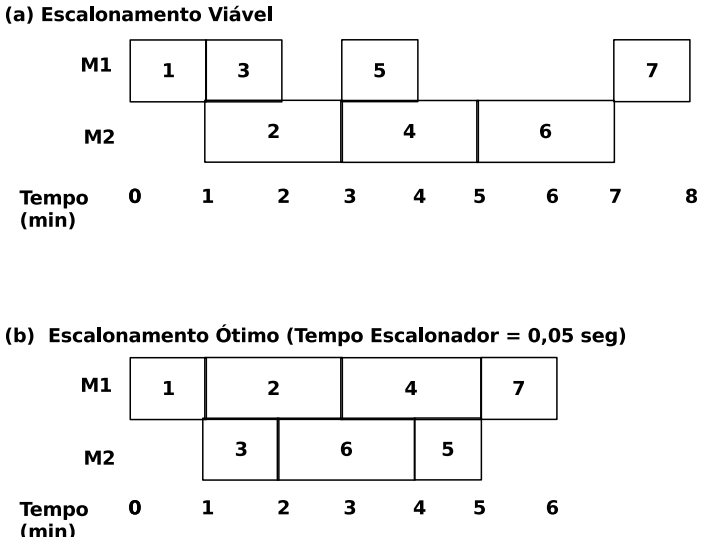

Figura 4.8: Segundo Experimento em Ambiente Real: (a) Escalonamento Viável e (b) Escalonamento Ótimo

colunas a hora de início e fim de sua execução para o seqüenciamento viável e, nas duas últimas colunas, as horas de início e fim de execução para o seqüenciamento ótimo.

Tarefa	Início (Normal)	Fim (Normal)	Início (Otimal)	Fim (Otimal)
t_1	22:10.09	22:11.09	22:30.03	22:31.03
t_2	22:11.10	22:13.10	22:31.18	22:33.18
t_3	22:11.10	22:12.10	22:31.03	22:32.03
t_4	22:13.15	22:15.15	22:33.22	22:35.22
t_5	22:13.01	22:14.01	22:34.07	22:35.07
t_6	22:15.17	22:17.17	22:32.05	22:34.05
t_7	22:17.04	22:18.04	22:35.39	22:36.39

Tabela 4.6: Segundo Experimento em Ambiente Real - Horários de início e fim de execução de cada tarefa

Nota-se um certo atraso em poucos segundos entre o início de execução de uma tarefa t_j e o fim de execução de uma tarefa predecessora t_i, conforme observado nas tarefas t_6 e sua sucessora t_7. Esse atraso se deve ao tempo de carregamento de um processo no processador da máquina, que pode variar em poucos segundos.

Na Figura 4.9 tem-se a relação de quantidade de tarefas ou jobs processados por unidade de tempo para cada tipo de escalonamento utilizado. Neste segundo experimento, as tarefas foram distribuídas sequencialmente entre os processadores disponíveis e, depois, distribuídas conforme o seqüenciamento retornado pelo algoritmo híbrido.

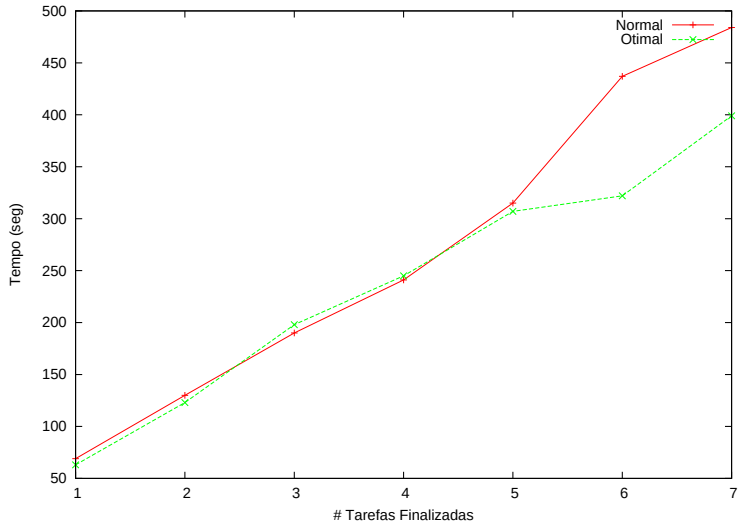

Figura 4.9: Comparativo entre *makespan* Viável e Otimal para 7 Tarefas em 2 Processadores

Verifica-se uma redução no tempo de finalização de todas as tarefas de 484 segundos para 399 segundos com o escalonamento ótimo. Com isso, tem-se um ganho de tempo real aproximado de 18% no tempo total.

Nos dois próximos experimentos serão utilizadas as técnicas de geração de instâncias descritas em (TOPCUOGLU; HARIRI; WU, 2002). O próximo grafo de precedências foi obtido

utilizando eliminação gaussiana. Neste caso serão utilizadas 14 tarefas relacionadas e 3 processadores disponíveis. Na Figura 4.10 tem-se diagramado o grafo de precedência da instância utilizada neste experimento, lembrando que os tempos computacionais de cada tarefa estão em minutos.

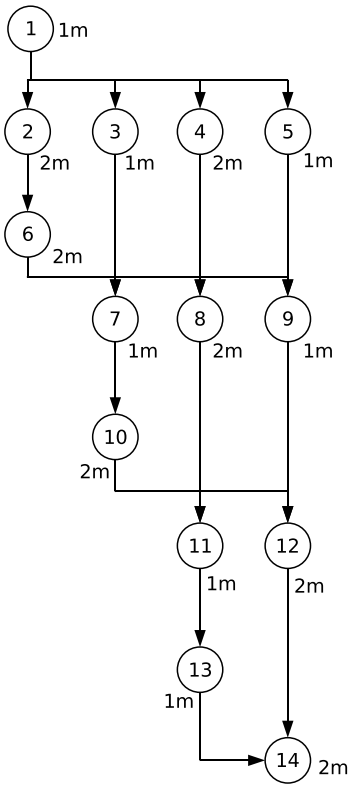

Figura 4.10: Terceiro Experimento em Ambiente Real

Na Figura 4.11 pode ser feito um comparativo entre os dois escalonamentos, com 820 segundos gastos no seqüenciamento viável não-otimal e, para o seqüenciamento otimal, 744 segundos mais o tempo de otimização de 0,13 segundos, totalizando 744,13. Com isso, obteve-se uma economia de tempo de execução de todas as tarefas na faixa de 10%.

Até agora foram utilizadas instâncias com poucas tarefas. Para o próximo experimento foi utilizada uma instância gerada via codificação de dinâmicas moleculares com 41 tarefas em 3 processadores disponíveis. Na Figura 4.12 é apresentado o grafo de precedência, mostrando o alto grau de paralelismo entre as tarefas desta aplicação.

Na Tabela 4.8 são descritos os horários de início e fim obtidos, tanto para o escalonamento viável quanto para o otimal. Percebe-se, como nos dois experimentos anteriores apresentados, que a carga dos processos em memória é um fator a se considerar, principalmente quando são tratadas as restrições de precedência entre as tarefas. Pode-se resolver este "problema" do fator

Tarefa	Início (Normal)	Fim (Normal)	Início (Otimal)	Fim (Otimal)
t_1	15:20.08	15:21.08	15:40.14	15:41.14
t_2	15:21.23	15:23.23	15:41.15	15:43.15
t_3	15:21.08	15:22.08	15:42.32	15:43.32
t_4	15:22.24	15:24.24	15:41.30	15:43.30
t_5	15:21.07	15:22.07	15:41.15	15:42.15
t_6	15:23.10	15:25.10	15:43.31	15:45.31
t_7	15:25.13	15:26.13	15:45.04	15:46.04
t_8	15:26.29	15:28.29	15:45.36	15:47.36
t_9	15:25.13	15:26.13	15:45.04	15:46.04
t_{10}	15:26.29	15:28.29	15:46.06	15:48.06
t_{11}	15:28.00	15:29.00	15:48.08	15:49.08
t_{12}	15:29.17	15:31.17	15:48.08	15:50.08
t_{13}	15:29.02	15:30.02	15:49.08	15:50.08
t_{14}	15:31.04	15:33.04	15:50.24	15:52.24

Tabela 4.7: Terceiro Experimento em Ambiente Real - Horários de início e fim de execução de cada tarefa

de carga, calculando a média de gasto para esta operação e aplicando nas diferenças entre os tempos de início e fim entre tarefas interligadas.

No gráfico da Figura 4.13 é apresentada a relação do *makespan* viável, com as tarefas distribuídas sequencialmente e uniformemente entre os processadores disponíveis. No escalonamento viável não-otimal foi obtido um *makespan* de 5707 segundos enquanto que no escalonamento otimal um *makespan* de 5104 segundos, gerando uma economia de 11% em tempo de processamento.

Com esses experimentos em um ambiente distribuído real foi possível avaliar a qualidade dos escalonamentos gerados pelo algoritmo híbrido, gerando o ótimo para todas as instâncias, bem como medir a economia de recursos com a utilização de um otimizador como procedimento residente em memória.

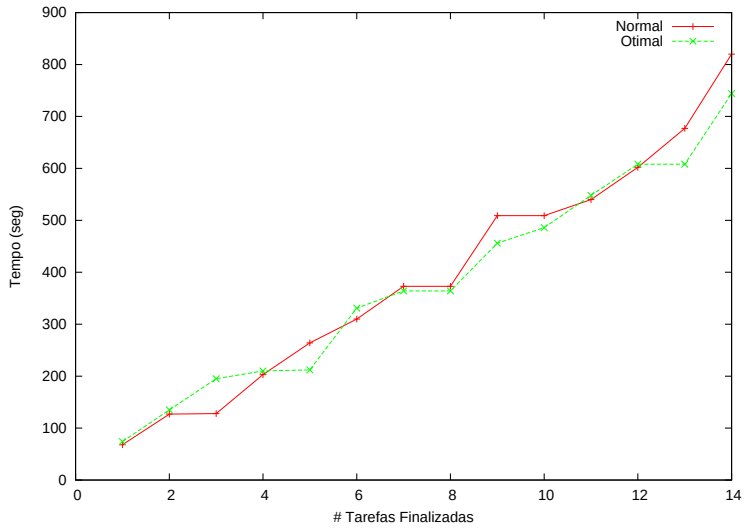

Figura 4.11: Comparativo entre *makespan* Viável e Otimal para 14 Tarefas em 3 Processadores

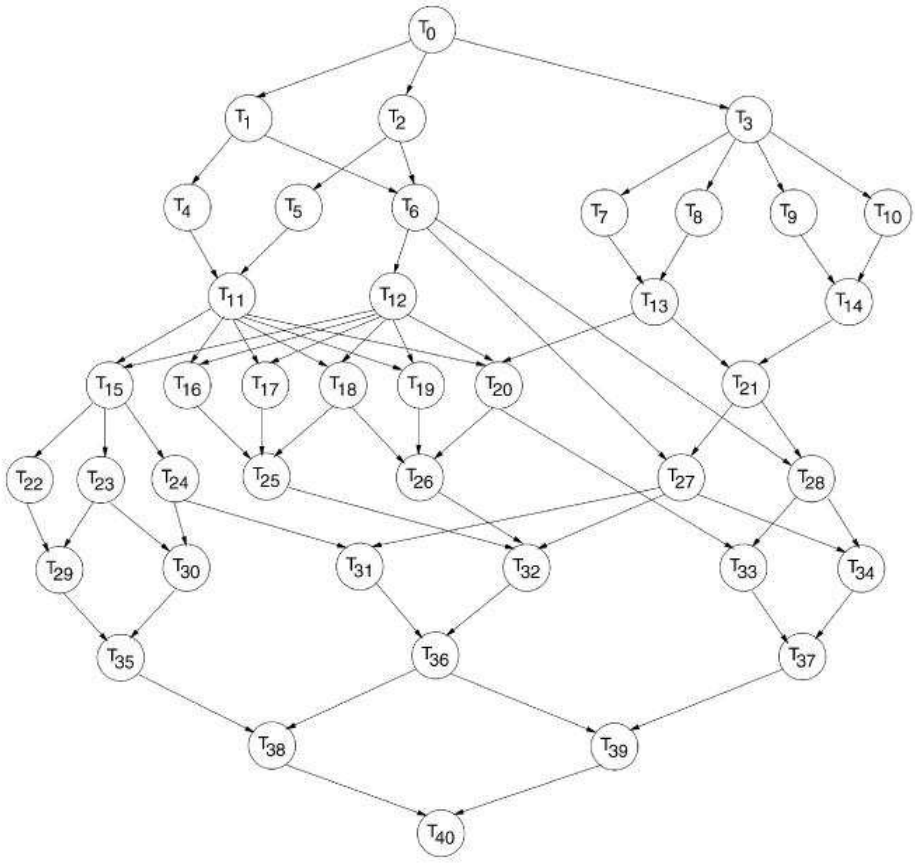

Figura 4.12: Quarto Experimento em Ambiente Real - (TOPCUOGLU; HARIRI; WU, 2002)

Tarefa	Início (Normal)	Fim (Normal)	Início (Otimal)	Fim (Otimal)
t_0	09:00.04	09:05.04	11:00.13	11:05.13
t_1	09:05.09	09:10.09	11:05.19	11:10.19
t_2	09:05.08	09:10.08	11:05.03	11:10.03
t_3	09:10.15	09:15.15	11:05.03	11:10.03
t_4	09:10.15	09:15.15	11:10.08	11:15.08
t_5	09:15.20	09:20.20	11:10.09	11:15.09
t_6	09:10.00	09:15.00	11:10.24	11:15.24
t_7	09:20.25	09:25.25	11:20.34	11:25.34
t_8	09:15.04	09:20.04	11:20.05	11:25.05
t_9	09:25.30	09:30.30	11:15.30	11:20.30
t_{10}	09:30.37	09:35.37	11:20.05	11:25.05
t_{11}	09:20.25	09:25.25	11:30.45	11:35.45
t_{12}	09:15.20	09:20.20	11:25.09	11:30.09
t_{13}	09:35.43	09:40.43	11:25.41	11:30.41
t_{14}	09:35.13	09:40.13	11:25.09	11:30.09
t_{15}	09:25.30	09:30.30	11:40.57	11:45.57
t_{16}	09:25.00	09:30.00	11:40.26	11:45.26
t_{17}	09:40.48	09:45.48	11:35.20	11:40.20
t_{18}	09:30.37	09:35.37	11:40.10	11:45.10
t_{19}	09:45.54	09:50.54	11:35.51	11:40.51
t_{20}	09:30.06	09:35.06	11:35.05	11:40.05
t_{21}	09:40.18	09:45.18	11:30.15	11:35.15
t_{22}	09:45.23	09:50.23	11:45.16	11:50.16
t_{23}	09:35.44	09:40.44	11:45.31	11:50.31
t_{24}	09:40.48	09:45.48	11:55.27	12:00.27
t_{25}	09:45.54	09:50.54	11:55.12	12:00.12
t_{26}	09:50.58	09:55.58	11:46.00	11:51.00
t_{27}	09:56.15	10:01.15	11:55.11	12:00.11
t_{28}	09:50.28	09:55.28	11:50.22	11:55.22
t_{29}	09:55.45	10:00.45	12:00.33	12:05.33
t_{30}	09:50.58	09:55.58	12:05.22	12:10.22
t_{31}	10:01.21	10:06.21	12:05.22	12:10.22
t_{32}	10:00.04	10:05.04	12:05.38	12:10.38
t_{33}	10:05.10	10:10.10	12:00.17	12:05.17
t_{34}	10:06.27	10:11.27	12:00.17	12:05.17
t_{35}	10:11.33	10:16.33	12:10.43	12:15.43
t_{36}	10:05.10	10:10.10	12:10.27	12:15.27
t_{37}	10:10.15	10:15.15	12:10.27	12:15.27
t_{38}	10:20.12	10:25.12	12:15.33	12:20.33
t_{39}	10:25.18	10:30.18	12:15.33	12:20.33
t_{40}	10:30.07	10:35.07	12:20.04	12:25.04

Tabela 4.8: Quarto Experimento em Ambiente Real - Horários de início e fim de execução de cada tarefa

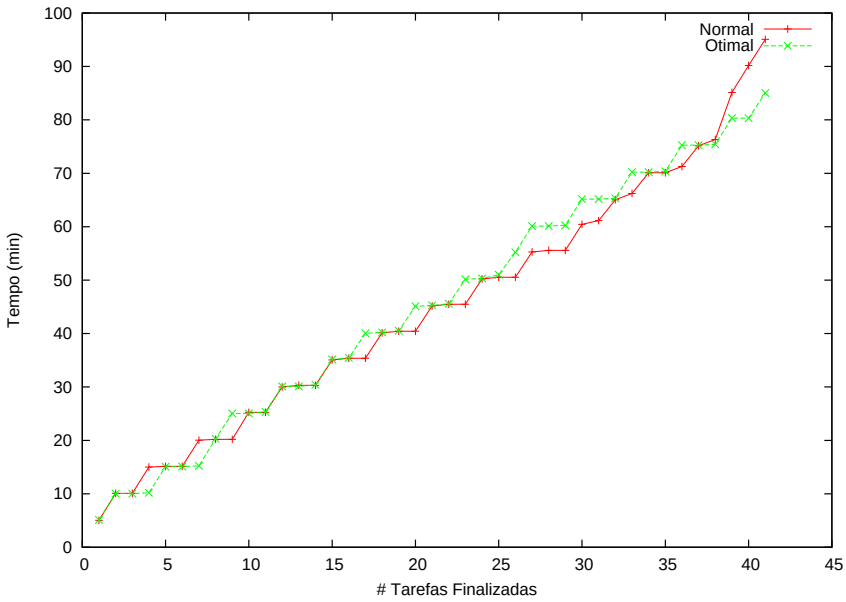

Figura 4.13: Comparativo entre *makespan* Viável e Otimal para 41 Tarefas em 3 Processadores

5 Conclusões e trabalhos futuros

"O mais importante da vida não é saberes onde estás, mas sim para onde vais."

Goethe

Neste Capítulo são apresentadas as conclusões e considerações para trabalhos futuros.

Neste trabalho foi possível estudar alguns conceitos importantes da área de Otimização Combinatória como formulações matemáticas, heurísticas e meta-heurísticas, controle de processos em um Ambiente Distribuído, bem como, também, exercitar a metodologia de pesquisa na identificação de trabalhos correlatos na literatura que pudessem colaborar com os resultados dos experimentos.

Foi apresentada uma proposta de algoritmo híbrido para o problema de escalonamento de tarefas em ambientes computacionais distribuídos homogêneos, passando pelos seguintes estágios:

- *Pesquisa* de modelos matemáticos de problemas de escalonamento na literatura para encontrar aquele que mais se adequace ao objetivo deste trabalho;
- *Identificação* de dificuldades em se resolver instâncias do problema apenas utilizando a formulação MIP devido, principalmente, à grande quantidade de variáveis geradas pelo modelo matemático, mesmo trabalhando com pequenas quantidades de tarefas e processadores disponveis;
- *Construção* de um algoritmo genético para a obtenção de valores que serão utilizados como limitantes superiores para o problema. Contudo, mesmo conseguindo boas aproximações dos valores ótimos para a maioria das instâncias utilizadas, verica-se pouca melhoria nos resultados, principalmente no quesito tempo de processamento;
- *Acréscimo* às técnicas implementadas, da fixação de variáveis do modelo para colaborar no processo de busca do CPLEX, realizando uma varredura no grafo de precedências por níveis, identificando variáveis que poderiam ter seus valores fixados em valores obtidos por soluções ótimas de níveis anteriores.
- *Testes* de eficiência dos escalonamentos retornados pelo algoritmo híbrido em um ambiente distribuído homogêneo real, confirmando o objetivo de melhorar a vazão de processos / tarefas neste tipo de ambiente computacional.

Como trabalhos futuros podem ser identificados:

- *Trabalhar* formas de diminuir a quantidade de variáveis do modelo matemático utilizado;
- *Melhorar* o processo de busca de soluções do algoritmo genético na tentativa de se obter melhores aproximações do ótimo e, assim, por algumas vezes, dispensar a utilização do algoritmo híbrido, identificando situações onde fica mais interessante utilizar somente a meta-heurística para geração do escalonamento;

- *Identificar* formas de se trabalhar um escalonador dinâmico que trabalhe como um processo residente no *Cluster* realizando o balanceamento de carga periodicamente.

Verifica-se o quão difícil é lidar com problemas de ordem não-polinomial, mesmo com toda a tecnologia de hardware e o conseqüente poderio computacional existente. Tratam-se de problemas que ocorrem no dia-a-dia, que afetam qualquer um direta ou indiretamente, mas que ainda requerem grande esforço de pesquisas e construção de técnicas para a solução dos mesmos.

Percebe-se ainda que o SGE algumas vezes não respeita os agendamentos de execução dos processos, podendo existir atrasos de alguns segundos em relação ao instante de início configurado pelo usuário do sistema. Isso deve ser considerado para futuras melhorias da ferramenta, principalmente quando trata-se de uma implantação de um otimizador dinâmico de tarefas no ambiente distribuído.

Referências Bibliográficas

AMARANTE, A. *Planejamento e Controle de Múltiplos Empreendimentos em edificações*. Dissertação (Mestrado) — Florianópolis, 2001.

APPLEGATE, D.; COOK, W. A Computational Study of the Job-Shop Scheduling Problem. *ORSA J. On Comput.*, v. 3, n. 2, p. 149–156, 1991.

BALAS, E. Machine Sequencing via Disjunctive Graphs: an Implicit Enumeration Algorithm. *Operations Research*, v. 17, p. 941–957, 1969.

BARKER, J. R.; MCMAHON, G. B. Scheduling The General Job-Shop. *Management Science*, v. 31, p. 594–598, 1985.

BOOKMAN, C. *Linux Clustering: Building and Maintaining Linux*. [S.l.]: Sams Publishing, 2003.

BUYYA, R. *High Performance Cluster Computing: Architectures and Systems*. [S.l.]: Publishing House of Electronics Industry (PHEL), 2001.

CARLIER, J.; PINSON, E. An Algorithm for Solving the Job-Shop Problem. *Management Science*, v. 35, n. 2, p. 164–176, 1989.

CAVALCANTE, C. C. B. *Escalonamento com restrição de mão-de-obra: heurísticas combinatórias e limitantes inferiores*. Dissertação (Mestrado) — UNICAMP, 1998.

COLL, P. E.; RIBEIRO, C. C.; SOUZA, C. C. *Test instances for scheduling unrelated processors under precedence constraints*. http://www-di.inf.puc-rio.br/celso: [s.n.], 1999. Acesso em: 28 de Março.

CRAUWLES, H. A. et al. Branch-and-Bound Algorithm for Single Machine Scheduling with Batching to Minimize the Number of Late Jobs. *Journal of Scheduling*, v. 8, p. 161–177, 2005.

DAVIDOVIC, T. et al. *Mathematical Programming-Based Approach to Scheduling of Communicating Tasks*. http://citeseer.ist.psu.edu/davidovic04mathematical.html: [s.n.], 2004. Acesso em: 28 de Março.

EDWIN, S. H. H.; NIRWAN, A.; HONG, R. A Genetic Algorithm for Multiprocessor Scheduling. *IEEE Transactions on Parallel and Distributed System*, v. 5, n. 2, p. 113–120, 1994.

FOSTER, I.; KESSELMAN, C. *The Grid 2: Blueprint for a New Computing Infrastructure*. [S.l.]: Morgan Kaufmann, 2003.

GOLDEBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. [S.l.]: Addison-Wesley Publishing Company, 1989.

LAGEWEG, B. J.; LENSTRA, K.; KAN, A. H. G. R. Job-Shop Scheduling by Implicit Enumeration. *Management Science*, v. 4, p. 441–450, 1977.

MACULAN, N. et al. A New formulation for Scheduling Unrelated Processors under Precedence Constraints. *RAIRO Recherche Opérationnelle*, v. 33, p. 87–90, 1999.

MANNE, A. S. On the JobShop Scheduling Problem. *Operations Research*, v. 8, p. 219–223, 1960.

MONACI, M. *Algorithms for Packing and Scheduling Problems*. Tese (Doutorado) — University of Bologna, 2001.

MUHRING, R. H. et al. *Solving Project Scheduling Problems by Minimum Cut Computations*. http://citeseer.ist.psu.edu/305557.html: [s.n.], 2000. Acesso em: 28 de Março.

NEMHAUSER, G. L.; WOLSEY, L. A. *Integer and Combinatorial Optimization*. [S.l.]: Wiley Interscience, 1999.

OLIVEIRA, E. S. *Solving Single-Track Railway Scheduling Problem using Constraint Programming*. Tese (Doutorado) — University of Leeds, 2001.

PATTERSON, D. A.; HENNESSY, J. L. *Organização e projeto de computadores: A Interface Hardware e Software*. [S.l.]: Morgan Kaufmann Publishers, Inc., 2000.

PINEDO, M. *Scheduling: Theory, Algorithms and Systems*. [S.l.]: Prentice Hall, 1995.

PMI. *A Guide to The Project Management Body of Knowledge*. [S.l.]: PMI, 2001.

PORTO, S. C.; RIBEIRO, C. C. A Tabu Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints. *International Journal of High Speed Computing*, v. 7, p. 45–71, 1995.

SOURD, F.; NUIJTEN, W. Multiple-Machine Lower Bounds for Job-Shop Scheduling Problems. *Journal on Computing*, v. 12, n. 4, p. 341–352, 2000.

STRADAL, O.; CACHA, J. Time Space Scheduling Method. *Journal of Construction Division, Proceedings of the American Society of Civil Engineers*, v. 108, n. CO3, p. 445–457, 1982.

TOPCUOGLU, H.; HARIRI, S.; WU, M. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Transactions on Parallel and Distributed Systems*, v. 3, 2002.

ANEXO A – Resultados Completos da Aplicação do Algoritmo Híbrido

O Algoritmo Híbrido Proposto neste trabalho, trabalha com o grafo de precedências particionado por níveis das tarefas, conforme suas alturas. Neste Anexo, são descritos todos os resultados obtidos para as 28 instâncias utilizadas, passo-a-passo, conforme o acréscimo de tarefas no modelo.

Quantidade de Tarefas	Solução	Tempo (seg)
5	3	0,13
9	5	0,09
14	8	0,20
		Tempo Total = 0,42

Tabela A.1: Instância f14 com 2 processadores

Na primeira coluna tem-se a quantidade de tarefas consideradas na iteração do algoritmo, na segunda coluna o valor da solução encontrada até o momento e na última coluna o tempo gasto pelo procedimento para gerar a solução sem considerar o tempo do procedimento GA.

Quantidade de Tarefas	Solução	Tempo (seg)
6	4	1,27
11	7	0,67
16	10	1,08
22	13	1,64
		Tempo Total = 4,66

Tabela A.2: Instância i22 com 2 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
7	4	1,11
11	6	0,09
15	8	0,20
19	10	0,23
22	12	0,11
		Tempo Total = 1,74

Tabela A.3: Instância v22 com 2 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
7	4	7,09
12	7	0,81
16	9	0,14
19	11	0,09
23	14	0,14
		Tempo Total = 8,27

Tabela A.4: Instância g23 com 2 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
6	4	0,28
10	6	0,06
15	9	1,16
19	11	0,28
22	12	0,11
25	14	0,16
		Tempo Total = 2,05

Tabela A.5: Instância d25 com 2 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
6	4	1,27
10	6	0,13
16	9	5,67
18	10	0,05
25	14	148,61
30	17	2,45
34	20	0,78
		Tempo Total = 158,96

Tabela A.6: Instância f34 com 2 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
7	6	0,52
13	9	0,19
19	12	3,78
25	15	5,47
31	18	12,20
37	21	10,36
43	24	11,89
48	27	2,55
		Tempo Total = 46,96

Tabela A.7: Instância r48 com 2 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
7	4	7,25
13	7	4,91
19	10	4,49
25	13	8,17
31	16	11,98
37	19	12,11
43	22	20,03
50	26	106,84
		Tempo Total = 175,78

Tabela A.8: Instância i50 com 2 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
5	2	0,0
9	3	0,02
14	5	0,23
		Tempo Total = 0,25

Tabela A.9: Instância f14 com 4 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
6	3	6,84
11	4	0,05
16	6	3,95
22	8	0,41
		Tempo Total = 11,25

Tabela A.10: Instância i22 com 4 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
7	3	0,14
11	4	0,03
15	5	0,30
19	6	0,16
22	8	0,22
		Tempo Total = 0,85

Tabela A.11: Instância v22 com 4 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
7	3	52,67
12	5	3,06
16	6	0,08
19	7	0,14
23	10	0,28
		Tempo Total = 56,23

Tabela A.12: Instância g23 com 4 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
6	3	0,02
10	4	0,02
15	6	3,63
19	7	0,16
22	8	0,22
25	10	0,36
		Tempo Total = 4,41

Tabela A.13: Instância d25 com 4 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
6	3	6,72
10	4	0,01
16	5	0,20
18	6	0,11
25	8	785,45
30	9	1,41
34	11	2,08
		Tempo Total = 795,98

Tabela A.14: Instância f34 com 4 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
7	6	0,06
13	8	0,19
19	9	2,31
25	11	2,78
31	12	4,64
37	14	6,49
43	16	4,38
48	18	11,28
		Tempo Total = 38,62

Tabela A.15: Instância r48 com 4 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
7	3	52,88
13	5	17,99
19	7	36,52
25	8	0,91
31	10	70,31
37	11	22,94
43	13	120,91
50	15	8,55
		Tempo Total = 331,01

Tabela A.16: Instância i50 com 4 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
5	2	0,02
9	3	0,05
14	5	0,16
		Tempo Total = 0,23

Tabela A.17: Instância f14 com 8 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
6	2	0,02
11	3	0,08
16	4	0,20
22	6	0,69
		Tempo Total = 0,99

Tabela A.18: Instância i22 com 8 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
7	3	0,03
11	4	0,06
15	4	0,25
19	5	0,33
22	7	0,50
		Tempo Total = 1,17

Tabela A.19: Instância v22 com 8 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
7	2	0,03
12	3	0,09
16	4	0,19
19	5	0,28
23	8	0,69
		Tempo Total = 1,28

Tabela A.20: Instância g23 com 8 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
6	3	0,02
10	4	0,05
15	5	0,17
19	6	0,31
22	7	0,50
25	9	0,84
		Tempo Total = 1,89

Tabela A.21: Instância d25 com 8 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
6	2	0,02
10	3	0,05
16	4	0,27
18	4	0,23
25	5	1,16
30	6	1,94
34	7	2,91
		Tempo Total = 6,58

Tabela A.22: Instância f34 com 8 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
7	6	0,19
13	8	0,17
19	9	0,14
25	10	1,50
31	11	2,42
37	12	4,84
43	14	26,47
48	16	10,51
		Tempo Total = 46,24

Tabela A.23: Instância r48 com 8 processadores

Quantidade de Tarefas	Solução	Tempo (seg)
7	2	0,05
13	3	0,14
19	4	0,39
25	5	1,03
31	6	2,11
37	7	4,00
43	8	6,92
50	10	13,75
		Tempo Total = 28,39

Tabela A.24: Instância i50 com 8 processadores

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo