UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RENAN FERRAZ PIRES

Um Estudo sobre Problemas de Escalonamento de Tarefas com Atrasos de Comunicação de Valores Extremos

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Philippe Olivier Alexandre Navaux
Orientador

Prof. Jayme Luiz Szwarcfiter
Co-orientador

Porto Alegre, Dezembro de 2013

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Pires, Renan Ferraz

Um Estudo sobre Problemas de Escalonamento de Tarefas com Atrasos de Comunicação de Valores Extremos / Renan Ferraz Pires. – Porto Alegre: PPGC da UFRGS, 2013.

63 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2013. Orientador: Philippe Olivier Alexandre Navaux; Co-orientador: Jayme Luiz Szwarcfiter.

I. Navaux, Philippe Olivier Alexandre. II. Szwarcfiter, Jayme Luiz. III. Título.

AGRADECIMENTOS

Não há como retribuir meus pais por tudo que me deram, resta-me apenas agradecer. Esta dissertação foi construída com o apoio fundamental que ambos, cada um a sua maneira, me deram.

Esta dissertação também não poderia ser sem os professores que me orientaram. Sou profundamente grato ao Professor Philippe Navaux pela oportunidade de ser aluno da UFRGS e de pertencer ao GPPD. Estas oportunidades me permitiram grande aprendizado, adquirido tanto nas matérias que cursei, como no conhecimento compartilhado no dia a dia pelos integrantes do GPPD, como também nos congressos, workshops e palestras organizadas pelo grupo. A ele também devo a oportunidade de ser co-orientado pelo Professor Jayme Luiz Szwarcfiter, o qual também é parte fundamental na construção desta dissertação. Agradeço profundamente ao Professor Jayme por ter me acolhido com seu aluno, ter compartilhado seu conhecimento e cedido seu tempo para as nossas reuniões. Também sou grato pela compreensão e paciência. Também sou muito agradecido à professora Rosiane de Freitas Rodrigues, que dedicou parte do seu escasso tempo no Rio de Janeiro em inúmeras reuniões inesperadas, para me repassar suas revisões desta dissertação. Reforço que suas revisões foram fundamentais para esta dissertação.

Preciso também agradecer aos meus colegas do GPPD. Todos eles me acolheram quando cheguei a Porto Alegre e mesmo antes quando me ajudaram com informações necessárias para me estabelecer na cidade. Agradeço, em especial, a Daniel Oliveira com quem pude sempre compartilhar atividades e sempre esteve prestativo quando necessitei de sua ajuda.

RESUMO

Esta dissertação de mestrado apresenta um estudo sobre problemas de escalonamento de tarefas com atrasos de comunicação. Mais precisamente, são abordados problemas de escalonar um conjunto de tarefas em um conjunto de máquinas paralelas de número limitado ou não, e tarefas de tempo de processamento unitário, sujeitas a relações de precedência, e com atrasos de comunicação estabelecidos para cada par de tarefas precedentes, assumindo valores extremos, ou seja, podendo ser desprezíveis ou infinitamente grandes, isto com o objetivo de minimizaro o tempo em que a última tarefa escalonada termina seu processamento - minimização do *makespan*. Sendo assim, dois problemas são demostrados serem da classe NP-difícil. Para o primeiro, a quantidade de processadores é indicada a cada instância, sendo este resultado válido ainda que as relações de precedência formem um conjunto de cadeias ($P|chains; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$). O segundo problema admite relações de precedência arbitrárias e é válido para qualquer quantidade fixa de processadores diferente de um ($P_2|prec; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$). Por outro lado, neste trabalho, dois outros problemas são demonstrados serem solúveis em tempo polinomial, ou seja, estarem na classe P, ambos quando uma quantidade ilimitada de processadores está disponível. É visto que, se a ordem de precedência das tarefas é limitada a uma árvore descendente, o problema é polinomial ($P_\infty|tree; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$). O outro caso polinomial demonstrado é válido quando é permitido processar a mesma tarefa em mais de um processador ($P_\infty|prec; c_{i,j} \in \{0, \infty\}; dup; p_j = 1|C_{max}$). Para ambos os casos são apresentados os algoritmos polinomiais. Finalmente, são apresentados resultados para o problema de escalonar tarefas particionadas em conjuntos para os quais todas as tarefas devem ser processadas no mesmo processador. O problema é NP-difícil quando a quantidade de processadores é determinada a cada instância. Esse resultado é válido ainda que a precedência seja restrita a duas cadeias. O problema se torna polinomial quando o conjunto de partições é limitado por constante e as cadeias são restritas em uma das duas formas: pela quantidade delas ou pela quantidade de tarefas em cada uma delas.

Como trabalho futuro, este estudo deixa em aberto a NP-Completude do problema de escalonar sob tais atrasos de comunicação de valores extremos, para uma quantidade fixa de processadores, quando a ordem de precedência é de alguma forma restrita, por exemplo, uma árvore descendente ($P_m|$*out-tree*$; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$).

Palavras-chave: Algoritmos, atraso de comunicação, escalonamento de tarefas, máquinas paralelas, relações de precedência.

A study of scheduling problems subjected to extreme delay values

ABSTRACT

This Master's Thesis presents a study on scheduling problems subject to communication delays. More precisely, this work involves job scheduling problems with a number of parallel machines, limited or not, and where the tasks (or jobs) have unit execution time, and are subject to some precedence relation. Communication delays are imposed at each pair of preceding tasks, taking extreme values, which may be negligible or infinitely large. The objective is minimize the completion time of the latest job to be processed, that is, to get the minimum makespan. Thus, NP-hard results are demonstrated for two cases. For the first, when the number of processors is indicated in the instance of the problem, and this result holds even when the precedence relation is restricted to a set of chains ($P|chains; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$). The second results is valid when arbitrary precedence relations are allowed, and any fixed number of processors (greater than one) is available ($P_2|prec; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$). Two other problems are demonstrated to have polynomial time solutions, both when an unlimited number of processors are available. The first result imposes the precedence relation to be an *out-tree* ($P_\infty|tree; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$). The second result is valid when the execution of the same job on multiples processors are allowed ($P_\infty|prec; c_{i,j} \in \{0, \infty\}; dup; p_j = 1|C_{max}$). For both cases, polynomial algorithms are presented. Finally, results are presented for the problem of job scheduling that are partitioned in sets which must be executed on the same processors. The problem is demonstrated to be NP-hard even if the precedence relation consists of two chains. Also, it is shown that the problem becomes solvable in polynomial time if the number of partitions is limited by a constant and the chains are restricted by a constant on either their number, or the number of tasks that each chain may have.

As future work, this study leaves open whether is NP-hard the case to schedule tasks subject to such communication delays with extreme values, when a fixed number of processors is available, and the precedence relations are some how restricted, for example, by an *out-tree* ($P_m|$*out-tree*$; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$).

Keywords: algorithms, communication delay, parallel machine, precedence relations, task scheduling.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

anc	*Ancestors*
DAG	*Directed Acyclic Graph*
desc	*Decendents*
DLS	*Dynamic Level Scheduling*
ETF	*Earliest Task First*
HLF	*Highest Level First*
HPC	*High Performance Computing*
ISH	*Insertion Scheduling Heuristic*
LCT	*Large Communication Times*
MCP	*Modified Critical Path*
MPI	*Message Passing Interface*
MST	*Minimum Scheduling Time*
PRAM	*Parallel Random Acess Machine*
RAM	*Random Acess Machine*
SCT	*Small Communication Time*
UET	*Unit Execution Time*
UCT	*Unit Communication Time*
VLIW	*Very long Instruction Word*

SUMÁRIO

1 INTRODUÇÃO

Escalonamento de tarefas envolve uma grande classe de problemas com uma teoria cada vez mais consolidada, sendo de grande interesse e estudo em Ciência da Computação e outras áreas, e com aplicações em diversos contextos e segmentos de mercado. Envolve a alocação de recursos na realização de tarefas sob algum critério de otimização, em função do tempo. Os estudos nessa área se iniciaram ainda na década de cinquenta, portanto, pouco após o surgimento dos primeiros computadores, e continuam sendo feitos com intensidade até os dias de hoje.

Problemas de escalonamento estão presentes em diversas áreas, de forma que não é difícil encontrar cenários práticos de aplicação. Tais problemas são encontrados, por exemplo, na aviação comercial, no qual a frota de aviões e a tripulação das empresas são recursos limitados que devem ser alocados em vôos sob diversas restrições a fim de maximizar os retornos financeiros das empresas. Também podem ser encontrados em processos industriais, por exemplo, onde os reatores utilizados para a produção de diferentes linhas de produtos devem ser compartilhados sob restrições, como a necessidade da utilização desses reatores em uma sequência e intervalos de tempos próprios para cada produto. Por último, problemas de escalonamento podem ser utilizados em sistemas computacionais, no qual os processadores são recursos limitados a serem utilizados no execução de rotinas de códigos obedecendo restrições, tais como, de ordem de processamento, de prazo limite para execução das tarefas, etc.

Tais problemas compartilham características entre si, sendo possível criar modelos teóricos genéricos capazes de abranger inúmeros casos de aplicação. Porém, quanto mais genéricas forem essas modelagens, mais difícil se torna encontrar métodos capazes de escalonar otimamente. Muitos problemas em suas formulações genéricas foram provados pertencer à classe de problemas NP-completo, o que torna a existência de soluções eficientes (de tempo polinomial) improváveis. Por esta razão, muitas soluções de problemas de escalonamento são propostas para casos mais restritos, enquanto outras soluções são dadas de forma aproximada.

1.1 Motivação

A vasta abrangência teórica e prática torna os problemas de escalonamento uma importante área de pesquisa na Ciência da Computação. Além de representativos, muitos destes problemas possuem uma grande complexidade de resolução, sendo provados NP-difíceis, e se mantendo NP-difíceis ainda que restrições severas sejam impostas, como a restrição do tempo de processamento de cada uma das tarefas ser idênticos. Por último, há de se ressaltar a peculiaridade do problema de se decidir se há um escalonamento possível para um conjunto de n tarefas, todas de tempo de execução unitários (UET), em

(03) três máquinas paralelas e idênticas, ou seja, $P_3|prec; p = 1|C_{max}$ em notação clássica de escalonamento a ser detalhada posteriormente, seja para encontrar um algoritmo polinomial ou para se provar sua NP-completude, visto que este problema juntamente com isomorfismo de grafos são os dois únicos problemas que permanecem em aberto da famosa lista de problemas em aberto divulgada na primeira versão do livro clássico sobre NP-completude, de Garey e Johnson (GAREY; JOHNSON, 1979). Este problema de escalonamento é o oitavo desta lista, sendo por isto conhecido como "*Open 8*".

O escalonamento de processos em rotinas paralelas de códigos impulsionou ainda mais esta área. Para esta aplicação, os recursos são os processadores e as tarefas são trechos de rotinas de código. Sendo assim, o recente aumento no número de unidades computacionais (*cores* ou *processadores*) em *clusters*, *grids*, computadores pessoais e, até mesmo, em dispositivos móveis, faz crescer a atenção da área acadêmica para essa classe de problemas. Muito do paralelismo em computadores pessoais e dispositivos móveis, até o presente momento, é extraído a partir de *threads* independentes e até mesmo de diferentes processos, e desta forma, se torna trivial o escalonamento. No entanto, em casos de *grids* e *clusters* utilizados em processamento de grande escala, o paralelismo tem de ser extraído a partir de rotinas dependentes que estão submetidas à restrições de precedência entre si, o que pode tornar o escalonamento uma tarefa complexa.

Ainda mais atenção ganha esta área, já que as próximas gerações de supercomputadores (*High Performance Computing - HPC*) apontam um crescimento massivo de paralelismo, principalmente objetivando a redução do custo energético imposta à próxima geração de supercomputadores (PADOIN; NAVAUX, 2011) (BORKAR; CHIEN, 2011). Este crescimento também cria novas perspectivas de aplicações, como em mineração de dados, sistemas multimídia e bioinformática. Neste cenário percebe-se a importância de modelos, ferramentas e algoritmos capazes de extrair o grande potencial computacional das próximas gerações de supercomputadores.

Segundo a literatura, não há ferramentas eficientes para sistemas de processamento paralelo e distribuído, o que deixa claro a dificuldade em se criar tais ferramentas (CAO et al., 2006). Sendo assim, a proposição de modelos de escalonamento que correspondam às arquiteturas atuais visa também atender esta demanda por ferramentas eficientes. Para esta finalidade, considerar atrasos de comunicações é imprescindível, já que o custo de comunicação é um dos principais limitadores da redução do tempo ao aumento de paralelismo. Além dos atrasos, outras considerações podem ser importantes para a eficácia desses modelos. Entre os problemas apontados para as abordagens da literatura, está o fato de que o verdadeiro tempo de processamento das tarefas só é conhecido durante a execução, devido a existência de laços e rotinas condicionais. Além disso, existem as dificuldades quanto a portabilidade eficiente dos algoritmos paralelos nas diversas arquiteturas. A razão disso é a falta de um *framework* capaz de considerar os principais parâmetros arquiteturais sem se tornarem específicos a alguma arquitetura (CAO et al., 2006).

Percebe-se, então, que a importância do estudo no escalonamento de processos não é somente por razões teóricas, mas também pela necessidade de aperfeiçoamento dos modelos vigentes em sistemas paralelos e distribuídos. A relevância prática ganha ainda mais força dado o contexto atual, já citado, do crescimento do paralelismo. Neste sentido, considerações sobre custo de comunicação possibilita uma modelagem mais robusta dos problemas envolvidos.

1.2 Organização da Dissertação

O restante desta dissertação está organizado da seguinte forma. O Capítulo 2 está dividido em três seções que apresentam, respectivamente, conceitos de Teoria do Grafos, Teoria da Complexidade Computacional e Teoria de Escalonamento, utilizados no decorrer desta dissertação. O Capítulo 3 apresenta trabalhos visando descrever o progresso acadêmico-científico em escalonamento de processos computacionais na área de sistemas paralelos e distribuídos, evidenciando o estado da arte de pesquisa nesta área. Para isso, está divido em três seções. Na Seção 3.1, são apresentadas as definições gerais sobre escalonamento de processos computacionais. Na Seção 3.2, são apresentados os principais modelos de escalonamento em sistemas paralelos e distribuídos, enfatizando-se o modelo clássico estendido, usado nos problemas abordados neste trabalho. Na Seção 3.3, é apresentado um resumo sobre trabalhos que consideram problemas de escalonamento para os quais os atrasos de comunicação não são desprezíveis, onde também se enfatiza resultados de aproximações. No Capítulo 4 são apresentados os resultados obtidos nesta dissertação, envolvendo problemas de escalonamento com atrasos de comunicação de valores extremos, infinitamente grande ou desprezíveis. Esta seção de resultados também está dividida em três seções. A primeira, Seção 4.1, apresenta os resultados de NP-completude gerados, evidenciando problemas para os quais soluções polinomiais são improváveis. Assim, os problemas de escalonamento $P|chains; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$ e $P_2|prec; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$ são provados serem NP-difíceis. A segunda seção, Seção 4.2, apresenta soluções obtidas de tempo polinomial para o problema $P_\infty|tree, c_{i,j} \in \{0, \infty\}, p_j = 1|C_{max}$ e para o problema $P_\infty|prec, c_{i,j} \in \{0, \infty\}, dup, p_j = 1|C_{max}$. A última seção deste capítulo, Seção 4.3, apresenta resultados para problemas de escalonamento onde as tarefas estão particionadas em subconjuntos e restritas tais que todas tarefas de um mesmo subconjunto devem ser processadas no mesmo processador. Por fim, na Seção 5, são feitas considerações finais sobre o trabalho e resultados obtidos, indicando-se possíveis trabalhos futuros.

Baseado nos resultados desta dissertação, o artigo entitulado "On parallel machine scheduling with special Communication Delays" (PIRES et al., 2014) foi submetido para a conferência *18th Workshop on Job Scheduling Strategies for Parallel Processing* (JSSPP 2014).

2 PRELIMINARES

Este capítulo introduz alguns conceitos importantes para o tema desta pesquisa, sendo dividido em três seções, as quais apresentam alguns dos conceitos de Teoria dos Grafos, de Complexidade Computacional e de Teoria de Escalonamento, respectivamente.

2.1 Teoria dos Grafos

Um *grafo simples* G é determinado a partir de um conjunto de vértices V e um conjunto de arestas E, cujos elementos são pares não ordenados de elementos de V. As arestas, portanto, são usadas para representar relações definidas entre tais pares de elementos.

A Teoria dos Grafos é a área que estuda esta estrutura matemática robusta, suas propriedades, classes, problemas relacionados e algoritmos, possuindo um vasto material teórico disponível na literatura (BONDY; MURTY, 2008) (DIESTEL, 2000). A seguir, serão apresentados os conceitos necessários para o entendimento dos problemas abordados neste trabalho e dos resultados gerados.

Um *grafo direcionado*, ou somente *digrafo*, é um grafo cujo conjunto de arestas é formado de pares ordenados de elementos de V, ou seja, um conjunto de *arcos*. Desta forma, as arestas (arcos) de um digrafo possuem uma *direção* e podem representar relações não simétricas. As arestas, se convenciona originar ou partir do primeiro vértice do par e incidir no segundo. Em um digrafo, um *caminho* entre os vértices v_0 e v_m, denotado como $path(v_0, v_m)$, é uma seqüência de vértices distintos começando em v_0 e terminando em v_m, para o qual para cada par de vértices consecutivos (v_i, v_{i+1}) existe aresta $(v_i, v_{i+1}) \in E$. Um *ciclo* é similar a um caminho de três ou mais vértices distintos exceto pelo primeiro e último vértices que necessariamente coincidem. Um *digrafo acíclico* (DAG - *Directed Acyclic Graph*) é um digrafo que não possui ciclos. Tais digrafos podem ser usados para representar ordenações (parciais ou totais), pois estabelecem naturalmente uma ordem de precedência entre os vértices. Os ancestrais ou precedentes de v é o conjunto formado pelos vértices u tais que exista $path(u, v)$ em G, assim, u é dito ser *ancestral* ou *predecessor* de v ($u \in ancest(v)$ ou $u \prec v$) e v é dito ser *descendente* ou *sucessor* de u ($v \in descent(u)$). Se existir aresta (u, v), então u é ancestral imediato de v e v descendente imediato de u. O *fecho transitivo* de um digrafo $G = (V, E)$, é um digrafo $G' = (V, E')$ tal que $(u, v) \in E'$ se e somente $u \prec v$ em G. Se não houver caminho nem de u para v ($u \not\prec v$) nem de v para u ($v \not\prec u$), u e v são ditos incomparáveis ($u \sim v$). O *grafo de incomparabilidade* de um digrafo G é o grafo referente ao mesmo conjunto de vértices de G, tal que exista aresta (u, v) se e somente se $u \sim v$ no digrafo. Quando não há nenhum par de vértices incomparáveis em um digrafo acíclico, ele é dito ser uma *ordenação total*, e é dito constituir uma *cadeia* (*chain*) ou sequência se

cada vértice possuir no máximo um predecessor imediato e um sucessor imediato. G é dito ser um conjunto de cadeias, se puder ser particionado em subconjuntos de cadeias tal que não exista dois vértices comparáveis em subconjunto distintos. O grafo G é dito ser uma *out-tree* (respectivamente, *in-tree*) se todos os vértices, exceto um - chamado de raiz (respectivamente, sorvedouro ou *sink*) - possuem exatamente um ancestral (respectivamente, um descendente) imediato. Uma *relação de precedência* é dita formar uma ordem de intervalo quando existe um mapeamento de cada vértice em V para um intervalo em $\mathcal{R}$ tal que $u \prec v$ se e somente se o intervalo referente a u é estritamente anterior ao intervalo referente a v.

2.2 Complexidade Computacional

No decorrer da história, a área da Ciência da Computação se deparou com diversos problemas cujas melhores abordagens de resoluções conhecidas se mostravam ainda muito ineficientes, no sentido de que o tempo de computação dessas soluções se tornava grande demais para maioria das aplicações encontradas. Ainda, havia problemas que nenhum método descritivo finito, por mais exaustivo que fosse, garantia encontrar uma solução. De fato, como foi mostrado posteriormente, há problemas para os quais tais métodos não existem (segundo os modelos computacionais teóricos tratados aqui e a serem definidos posteriormente). A quantidade de tempo e de memória necessária na soluções dos problemas e a existência dessas soluções é estudada pela Teoria da Complexidade Computacional (GAREY; JOHNSON, 1979). Tal teoria tem forte importância prática na medida que pode classificar problemas para os quais a existência de soluções eficientes seja improvável. Muitos desses problemas complexos têm enormes aplicações práticas, e, entres eles, estão diversos problemas de escalonamento. O procedimento de como classificar tais problemas como difíceis ou fáceis se origina de um vasto arcabouço teórico. As subseções seguintes esclarecem alguns termos e, de maneira sucinta, mostram como é realizada tal classificação.

2.2.1 Modelos Computacionais

A definição de problemas computacionalmente solúveis é, de certa forma, abstrata, mas se refere a quais problemas podem ser solucionados de maneira metódica, através de execuções sequenciais de instruções claras. Contudo, há diversas possibilidades sobre quais tipos de instruções devem ser permitidas para essa máquina abstrata. Na tentativa de definir melhor tal conceito foram criados vários modelos computacionais, e entre os primeiros e mais importantes estão a *Máquina de Turing*, *Cálculo-*λ e *Funções Recursivas*, introduzidos respectivamente por Alan Turing, Alonso Church e Kurt Gödel (DIAS; WEBER, 2010). Tais modelos se mostraram semelhantes com relação à capacidade computacional, e ainda, muitos outros modelos os quais aparentavam possuir maior poder de computação, foi mostrado serem equivalentes. O modelo da Máquina de Turing, por exemplo, não admite o que poderia ser associado ao acesso randômico à memória, isso, porém, não diminui seu poder computacional. Assim, o poder computacional desses modelos é considerado ser o poder computacional de qualquer computador hoje conhecido (COPELAND, 2008).

Entre esses modelos o mais conhecido é, certamente, a Máquina de Turing. Tal máquina é uma abstração computacional que possui uma fita de memória a sua disposição e acessa um símbolo em uma determinada posição da fita por vez. Além desta fita de memória, a Máquina de Turing possui um conjunto de estados possíveis e um estado cor-

rente. Ambos, a fita na sua posição atual e o estado corrente, podem ser modificados, juntamente com a posição de acesso da fita em si, em transições discretas predefinidas na Máquina. Essas transições são definidas para cada estado e cada símbolo encontrado. Portanto, ela pode ser definida, de uma maneira informal, a partir de uma posição atual de acesso a fita, um conjunto de estados, um conjunto de símbolos, um conjunto de funções de transições que a partir de um estado e um símbolo decide um símbolo a ser escrito na posição corrente da fita, o próximo estado da máquina, e qual o movimento, à esquerda ou à direita, que a posição de acesso a fita deve fazer. Além disso, um dos estados deve ser definido como estado inicial e um ou mais estados definidos como estados de parada, para os quais a computação deve ser finalizada. Esta definição corresponde à *Máquina de Turing Determinística* (BARKER-PLUMMER, 2013). Um outro possível modelo é permitir que para cada estado da máquina e símbolo encontrado na posição corrente da fita exista não apenas uma, mas, várias funções de transições possíveis. A computação neste modelo é feita de forma paralela nos diversos caminhos de computação. Esta máquina abstrata com poder de paralelização infinita é chamada de *Máquina de Turing Não Determinística*. A Máquina de Turing (determinística ou não) pode ser usada para computação de todo tipo de problema computável, tais como reconhecimento de pertinência de cadeias a linguagens, computação de funções, problemas de otimização, etc.

A Máquina de Turing é utilizada na análise e classificação da complexidade de problemas. Uma das classes mais elementares definidas a partir dela é a *classe* $\mathcal{P}$. À classe $\mathcal{P}$ pertencem os problemas para os quais existe alguma Máquina de Turing Determinística capaz de computar corretamente todas as instâncias do problema em uma quantidade de passos limitada por um polinômio, no tamanho da cadeia de símbolos que descreve a instância do problema.

A classe de problemas $\mathcal{NP}$ considera problemas de decisão, isto se refere à utilização da Máquina de Turing como decisor de pertinência de cadeias de símbolos a uma determinada linguagem. Os problemas nesta classe podem ser resolvidos em uma quantidade polinomial de passos por uma Máquina de Turing Não Determinística. É notável que, se uma Máquina de Turing Não Determinística aceita uma determinada cadeia de símbolos como elemento pertencente a uma linguagem em uma quantidade de passos polinomial no tamanho da cadeia, então, é possível a uma Máquina de Turing Determinística se certificar que tal cadeia pertence a essa linguagem também em uma quantidade de passos polinomial, quando em posse das informações que determinam qual a função de transição deve ser escolhida, a cada transição da Máquina de Turing Não Determinística, a fim de se encontrar o estado de aceitação final. Percebe-se também que essa informação, às vezes referenciada como verificador ou certificador, deve ser polinomial no tamanho da cadeia. Desta forma, esta classe pode ser definida alternativamente como a classe que contém exatamente os problemas de decisão cujas instâncias positivas de aceitação podem ser verificadas em tempo polinomial por uma Máquina de Turing Determinística (IMMERMAN, 2011).

Apesar da Máquina de Turing Determinística e da Não Determinística se mostrarem equivalentes no que diz respeito a quais problemas podiam computar, o tempo de computação necessário à Máquina de Turing Determinística, ao que parecia, era exponencialmente maior a medida que a cadeia de símbolos (a instância do problema) crescia. Essa questão foi colocada em 1971 por Stephen Cook em seu famoso artigo *"The complexity of theorem-proving procedures"* (COOK, 1971). Neste trabalho, Cook se concentrou em utilizar as Máquinas de Turing como dispositivo decisório de pertinência de cadeias de símbolos a linguagens. Cook mostrou que qualquer linguagem que pudesse ser deci-

dida em uma quantidade de passos polinomial no tamanho da palavra em uma Máquina de Turing Não Determinística (problemas $\mathcal{NP}$) poderia ser decidida também em tempo polinomial em uma Máquina de Turing Determinística, na existência de algoritmo polinomial para o problema da satisfatibilidade de formulas lógicas. Em outras palavras, se tal algoritmo existir, então, as duas classes P e $\mathcal{NP}$ definem e contém exatamente os mesmos problemas. Contudo, a existência de tal algoritmo está em aberto, sendo que a maioria dos estudiosos acreditam que tal algoritmo não exista, especialmente em razão das consequências da existência de tal algoritmo. Desta forma, esta questão, $\mathcal{P} = \mathcal{NP}?$, constitui o principal problema da Ciência da Computação e um dos problemas matemáticos mais importantes ainda em aberto.

Ainda que não se saiba se tal algoritmo existe ou não, a demonstração de que alguns problemas possuem esta mesma propriedade do problema da satisfatibilidade, ou seja, de poderem ser reduzidos a todos os problemas em $\mathcal{NP}$, é útil, pois, esforços com tentativas usuais de soluções de tempo polinomial para esses problemas podem ser poupados. A razão disto é que tal propriedade está fortemente associada à solução desses problemas e alguns desses problemas mesmo com grande importância estão em aberto há cerca de 80 anos, como o problema do Caixeiro Viajante, e continuam em aberto mesmo após 40 anos de mais importância lhes ter sido atribuída.

A ideia utilizada por Cook para sua demonstração, como ele mesmo chamou, foi de reduzir um problema a outro. Esta ideia se baseia em utilizar um algoritmo (o qual se supõe existir, a fim de estabelecer a redução) que decide uma das linguagens como subrotina para decidir a outra. Para isso é necessário associar a cada instância de um problema uma instância do outro, de forma que a pertinência das instâncias às suas respectivas linguagens estejam atreladas. Se for possível encontrar a associação e construir a instância associada em uma quantidade também polinomial de passos, então, está demostrado que a existência de algoritmo polinomial de um problema (o problema redutor) implica em algoritmo polinomial para o outro problema (o problema reduzido).

Cook aplicou essa redução de uma maneira genérica criando o primeiro problema ao quais todos os problema em $\mathcal{NP}$ estavam reduzidos. Sendo assim, resolver o Problema da Satisfatibilidade era pelo menos tão difícil quanto os problemas em $\mathcal{NP}$. A existência de um problema com essa propriedade permitiu a demostração de outros problemas com essa mesma propriedade, a partir de reduções deste problema original a esses outros. Em um dos trabalhos seguintes ao de Cook, Karp demostrou, através de reduções, 21 problemas com esta propriedade (KARP, 1972), após Karp muitos outros problemas também foram adicionados a esta lista. Assim, se estabeleceu a Classe *NP-difícil*.

A classe NP-difícil não está restrita a problemas de decisão e não está contida em $\mathcal{NP}$. Por outro lado, há problemas nesta classe que também pertencem à classe $\mathcal{NP}$, como o próprio Problema da Satisfatibilidade. Estes problemas, que pertencem à interseção dessas classes, constituem a classe de problemas *NP-completo*.

É possível ainda fazer distinções de interesse prático entre problemas NP-difíceis, considerando a representação dos problemas. Problemas computacionais muitas vezes possuem valores numéricos como parâmetros, sendo representados utilizando alguma base numérica, como, por exemplo, a notação binária (de base 2) ou a decimal (de base 10). Esses sistemas numéricos permitem a representação de valores exponencialmente maiores que o comprimento da notação em si utilizada; por exemplo, com 4 símbolos em binário é possível representar até 2^4 valores e 10^4 valores em decimal. Essa propriedade não ocorre quando o sistema unário (de base 1) é utilizado: neste sistema o valor representado e o comprimento da sua representação tem exatamente o mesmo tamanho. Como

o tempo de computação é referente ao tamanho da descrição das instâncias do problema, descrevê-las com o sistema unário pode tornar polinomial um algoritmo que é exponencial, quando uma base maior for utilizada. Um algoritmo com tal propriedade é dito ser *pseudo-polinomial*. Um problema em NP-completo é dito ser *fracamente* NP-completo se admitir algoritmo pseudo-polinomial, e dito ser *fortemente* NP-completo se a existência de tal algoritmo implica $\mathcal{P} = \mathcal{NP}$.

2.3 Teoria de Escalonamento

Um problema de escalonamento consiste em alocar um conjunto de n tarefas, usualmente referenciado como $J = \{J_0, J_1, ..., J_{n-1}\}$, a serem executadas por um determinado conjunto de m processadores (ou qualquer outra máquina ou recurso), referenciados por $P = \{P_0, P_1, ..., P_{m-1}\}$. Para cada tarefa, um escalonamento S designa um processador durante um certo período de tempo. Durante esse período, o processador não pode estar designado a nenhuma outra tarefa. Em alguns casos, os intervalos de tempo designados a cada uma das tarefas são restritos a intervalos discretos. Nesses casos, os intervalos de tempo de unidade mínima são intervalos unitários ou *slots*. O intervalo unitário, ou *slot*, que um escalonamento S designou para uma determinada tarefa J_i pode ser referenciado como $slot_S(J_i)$. De forma semelhante o processador desta tarefa pode ser referenciado como $P_S(J_i)$. Quando o escalonamento em questão está evidente pelo contexto, essas notações podem ser simplificadas para $slot(J_i)$ e $P(J_i)$, respectivamente. Outro termo importante é a carga de processamento total (*workload*) que se refere a soma do tempo processamento das tarefas, ou seja, o tempo de processamento ativo total.

Muitas possibilidades surgem em relação às características tanto das tarefas, quanto dos processadores, quanto das restrições. Em alguns problemas, algumas tarefas podem necessitar de mais tempo que outras para serem processadas ou necessitar ser processadas por um determinado subconjunto dos processadores. Os processadores, por sua vez, podem se diferenciar uns dos outros, por exemplo, no tempo necessário para processar cada tarefa. Restrições também podem ser impostas na ordem de processamento das tarefas, e, neste caso, obriga-se algumas tarefas a serem processadas anteriormente a outras. As restrições também podem ser impostas, por exemplo, a determinar que cada tarefa seja processada anteriormente a um determinado limite de tempo.

Essa vasta quantidade de parametrização dos problemas tornou suas definições extensas e, assim, as associações entre eles difíceis, sendo, as vezes, difícil até mesmo reconhecer quando dois trabalhos tratavam de problemas idênticos. Visando facilitar estas associações, Graham e outros (GRAHAM et al., 1979) propuseram uma notação sucinta, denominada de *notação de 3-campos*. Tal notação foi amplamente adotada e estendida à medida que novos artigos surgiam e necessitavam de algum parâmetro não previsto por ela inicialmente. Uma das extensões importantes foi a consideração de atrasos de comunicação, proposta em (VELTMAN; LAGEWEG; LENSTRA, 1990). Tais atrasos consistiam em estabelecer um intervalo de tempo mínimo entre o tempo de processamento de uma tarefa e sua tarefa predecessora quando estas fossem alocadas em processadores distintos.

Existe um vasto material disponível sobre a teoria envolvida na definição e resolução de problemas de escalonamento de tarefas (PINEDO, 2012) (RODRIGUES, 2009) (BRUCKER, 2007) (BLAZEWICZ et al., 2007) (LEUNG, 2004). Alguns trabalhos visam provar a complexidade computacional dos problemas de escalonamento ainda em abertos, ou mesmo estabelecer algoritmos ainda mais eficientes para problemas já provados estarem na classe P (RODRIGUES, 2009) (DOURADO; RODRIGUES; SZWARC-

FITER, 2009) (DOURADO; RODRIGUES; SZWARCFITER, 2011). Por outro lado, outros trabalhos visam estabelecer formulações matemáticas que possibilitem a elaboração de melhores estratégias algorítmicas exatas ou aproximadas para problemas de escalonamento NP-difíceis (PESSOA et al., 2010) (RODRIGUES et al., 2008) (AMORIM et al., 2013) (AMORIM et al., 2013).

2.3.1 Notação de Três Campos Estendida

A notação criada inicialmente para o modelo de escalonamento clássico, denominada de *notação de 3-campos*, foi posteriormente estendida para absorver outras possibilidades, como atrasos de comunicação. Neste seção é apresentado um subconjunto de tal notação, justamente contendo as opções utilizadas neste trabalho. Assim, a notação é definida com três campos $\alpha|\beta|\gamma$, cada um descrito a seguir.

O primeiro campo, o campo α, refere-se às definições sobre os processadores são elas:

Processadores Homogêneos (*identical parallel machines*)

P - A quantidade de processadores é informada na instância de cada problema.

P_m - A quantidade de m processadores está disponível para qualquer instância do problema.

P_∞ - Uma quantidade ilimitada de processadores está disponível, novamente para qualquer instância do problema.

Processadores Uniformes (*uniform parallel machines*)

Q - Quando existir processadores com capacidades de processamento distintas.

Q_m - A quantidade de m processadores está disponível para qualquer instância do problema e estes processadores possuem capacidades distintas, também determinada previamente.

Q_∞ - Uma quantidade ilimitada de processadores está disponível, novamente para qualquer instância do problema.

Processadores Não-Relacionados (*unrelated parallel machines*)

R - Quando o desempenho de cada processador depende da tarefa processada.

Restrição de Mapeamento (*multipurpose machines*)

MPM - Quando cada processador pode processar apenas um determinado subconjunto das tarefas.

O segundo campo, o campo β, pode se referir às características das tarefas como, por exemplo, o tempo de processamento. Pode se referir também às restrições em relação as tarefas, como ordem de processamento, ou pode se referir a possibilidades como interrupções nas tarefas. A seguir estão algumas possibilidades para este campo.

Ordem de Precedência (*precedence order*)

$prec$ - A relação de precedência é arbitrária (qualquer).

$chains$ - A relação de precedência estabelece um conjunto de cadeias.

out-tree - A relação de precedência estabelece uma árvore descendente.

in-tree - A relação de precedência estabelece uma árvore incidente.

interval-order - A relação de precedência estabelece uma ordem de intervalo.

Atrasos de Comunicação (*communication delays*)

$c = d$ - O atraso de comunicação (ou seja, o tempo mínimo entre uma tarefa e outra predecessora a esta quando processadas em processadores distintos) é constante e igual a d;

c - O atraso de comunicação é constante, sendo indicada a cada instância do problema.

c_i - Os atrasos de comunicação dependem apenas da tarefa predecessora (que transmite os dados).

$c_{i,j}$ - Os atrasos de comunicação dependem do par de tarefas envolvidos.

Atrasos de Precedência (*precedence delays*)

$\ell = d$ - Toda tarefa que possui outra como precedente (na ordem de precedência estabelecida) tem que aguardar d intervalos de tempos após a tarefa precedente, para poder ser processada independentemente do processador nas quais esse par de tarefas é processado.

ℓ - O tempo de espera entre tarefas precedentes é indicada a cada instância do problema.

ℓ_i - O tempo de espera entre tarefas precedentes depende apenas da tarefa predecessora.

$\ell_{*,j}$ - O tempo de espera entre tarefas precedentes depende apenas da tarefa sucessora.

$\ell_{i,j}$ - O tempo de espera entre tarefas precedentes depende do par de tarefas envolvidos.

Reprocessamento de Tarefas (*task duplication*)

dup - Indica que uma mesma tarefa pode ser processada em mais de um processador.

Tempo de Processamento (*processing time*)

$p_j = d$ - Todas as tarefas necessitam de d intervalos de tempo para ser completamente processadas.

Interrupção de Tarefas (*preemption*)

$pmtn$ - Tarefas podem ser interrompidas a qualquer instante de seu processamento e serem retomadas em qualquer outro processador a qualquer tempo.

Atrasos de Migração (*migration delays*)

$pmtn(delay = d)$ - Tarefas podem ser interrompidas a qualquer instante de seu processamento e serem retomadas a qualquer instante no mesmo processador ou d intervalos de tempo depois em qualquer outro processador.

$pmtn(delay)$ - Tarefas podem ser interrompidas a qualquer instante de seu processamento e serem retomadas a qualquer instante no mesmo processador ou intervalos de tempo depois, a ser definido a cada instância, em qualquer outro processador.

Data de Disponibilidade (*release date*)

r_j - Indica para cada tarefa um instante de tempo para o qual a tarefa não pode ser escalonada anteriormente.

Data Limite Ideal (*due date*)

d_j - Indica um instante limite para o qual a tarefa sendo processada posteriormente sofre algum tipo de penalidade a ser precisamente definida na função objetivo.

Data Limite Obrigatória (*deadline*)

D_j - Indica um instante limite para o qual a tarefa não pode ser processada posteriormente, ou seja, escaloná-la posteriormente a este limite torna o escalonamento inválido.

Problemas de escalonamento podem tratar apenas de encontrar um escalonamento válido sob as restrições de quantidade de processadores, tempo de processamento de cada tarefa e data limite obrigatória das tarefas. No entanto, usualmente, os problemas de escalonamento visam encontrar entre os escalonamentos válidos aqueles que otimizam algum critério. O último campo define qual o critério considerado nesta otimização. A seguir encontram-se as opções mais conhecidas para este critério, onde C_j indica o instante de tempo cujo processamento da tarefa J_j é finalizado.

Tempo Total

C_{max} **(**$makespan$**)** - Representa a minimização do maior C_j, ou seja, o tempo necessário para processar a última tarefa dada as restrições.

$\sum C_j$ *ou* $\sum w_j C_j$ **(**$flowtime$**)** - Representa a minimização da soma do tempo de completude das tarefas, ponderado ou não pelo peso das tarefas.

Atrasos de Tarefas

L_{max} $(lateness)$ - Representa a minimização de $\max(L_j)$, onde $L_j = (d_j - C_j)$. Ou seja, minimiza o tempo da tarefa que mais foi atrasada pelo escalonamento.

$\sum T_j$ *ou* $\sum w_j T_j$ $(tardiness)$ - Representa a minimização de $\sum T_j$, onde $T_j = \max\{L_j, 0\}$. Em outras palavras, representa a minimização da soma dos atrasos, ponderadas ou não pelo peso das tarefas.

$\sum U_j$ *ou* $\sum w_j U_j$ $(unit\ penalty)$ - Onde U_j é 1 se $L_j > 0$ e 0 caso contrário, representando a minimização da quantidade de tarefas atrasadas.

2.3.2 Problemas Clássicos e Principais Resultados

Os estudos no escalonamento de tarefas tiveram início no anos 50. Um dos primeiros e importantes resultados foi publicado por Hu (HU, 1961), que mostrou como encontrar o escalonamento ótimo em tempo linear para o problema com restrição de precedência de conjunto de árvores (*in-forest* ou *out-forest*), um número variável de processadores homogêneos e tarefas de tempo de processamento unitário ($P|tree; p_j = 1; |C_{max}$). A ideia do algoritmo se baseia em processar as tarefas priorizando aquelas de mais alto nível na árvore (estratégia gulosa conhecida como HLF - *Highest Level First*), estratégia que mais tarde se mostrou ótima em alguns problemas e é usada como heurística em outros. Outra estratégia gulosa também para tarefas de tempo unitário pode ser usada para encontrar escalonamento ótimo quando a precedência é uma ordem de intervalo ($P|interval - order; pj = 1; |C_{max}$) (PAPADIMITRIOU; YANNAKAKIS, 1979). Esta estratégia se baseia na ordenação decrescente pelo grau da tarefa-vértice. Se tarefas de tempo arbitrários forem permitidas, o problema é NP-completo (PAPADIMITRIOU; YANNAKAKIS, 1979). Nenhuma dessas estratégias, porém, funciona para uma precedência genérica ($P|prec; p_j = 1; |C_{max}$), que, de fato, havia sido mostrada ser $\mathcal{NP} - completo$ por Ullman (ULLMAN, 1975). Neste mesmo trabalho, Ullman também mostrou que mesmo restringindo a quantidade de processadores a apenas dois, basta-se admitir, além das tarefas de tempo de processamento igual a um, tarefas de tempo de processamento igual a dois para que o problema se mantenha NP-completo ($P_2|prec; p_j = \{1, 2\}; |C_{max}$). Quando tarefas de tempo de processamento igual a dois não são permitidas, a restrição de dois processadores garante a existência de algoritmo polinomial, este problema é referenciado como ($P_2|prec; p_j = 1|C_{max}$), resultado apresentado por Fujii *et al.* antes mesmo do artigo de Ullman (FUJII; KASAMI; NINOMIYA, 1969). A solução apresentada envolve emparelhamento no grafo complementar do fecho transitivo, tendo complexidade de pior caso $O(n^3)$ (FUJII; KASAMI; NINOMIYA, 1971). No entanto, outras soluções mais eficientes foram propostas. Coffman e Graham apresentaram solução com base em uma estratégia gulosa, a partir de uma enumeração das tarefas definida recursivamente a partir de seus sucessores (COFFMAN; GRAHAM, 1972). Mesmo essa solução pôde ser otimizada para $O(e + \alpha(n) * n)$ (GABOW, 1982) e ainda mais para $O(e + n)$ (STADTHERR, 1998).

Outro resultado que, como o de Hu, pertence à década de 60 foi o de Rothkopf (ROTHKOPF, 1966). Neste artigo, o autor apresenta um algoritmo de programação dinâmica para o problema de escalonamento onde não há precedência entre tarefas e cujas tarefas apresentam tempo de processamento qualquer ($P_m||C_{max}$). O algoritmo apresentado tem complexidade $O(n \times C^m)$, onde n é o número de tarefas, C é um limite para o tempo final de escalonamento e m é o número de processadores. Desta forma, o algoritmo só é polinomial sobre algumas restrições para m. A existência de solução polinomial para este problema implicaria $\mathcal{P} = \mathcal{NP}$, este fato é resultado da NP-completude do problema da partição (*Partition Problem*) apresentada no famoso artigo de Karp (KARP, 1972). Karp, neste mesmo trabalho, também mostrou a NP-completude de vários outros problemas, entre os quais está o de decidir se existe uma permutação de elementos sobre um conjunto de tarefas, para as quais está associada uma penalidade, um tempo de processamento e um tempo limite para execução, tal que a soma das penalidades das tarefas finalizadas após seu limite de para execução seja menor que um valor k. Para este problema todas as entradas consistem de valores inteiros.

O conhecimento da NP-completude do problema ($P_2||C_{max}$) impossibilitava considerações de precedência para este problema. No entanto, a existência de algoritmo

pseudo-polinomial induzia a ideia que talvez fosse possível inserir restrições de ordem de precedência ($P_2|prec;|C_{max}$) e continuar sendo possível computar a solução em tempo pseudo-polinomial. Esta ideia era reforçada pela existência de algoritmo polinomial para o problema $P_2|prec;p_j = 1;|C_{max}$, dada a similaridade entre os problemas. De fato, uma solução polinomial para o problema $P_2|prec;p_j = 1|C_{max}$ é pseudo-polinomial para ($P_2|prec;pmtn^*|C_{max}$), onde $pmtn^*$ indica a permissão às interrupções das tarefas nas transições dos intervalos discretos. No entanto, em 1991, Du mostrou que tal problema é fortemente NP-completo para qualquer número de processadores maior que um e mesmo quando a ordem de precedência é restrita a um conjunto de cadeias ($P_2|chains;|C_{max}$) (DU; LEUNG; YOUNG, 1991).

Finalmente, vale a menção do problema $P_3|prec;p_j = 1;|C_{max}$. Como consequência do artigo de Du, o problema de escalonamento $P_3|prec;|C_{max}$ é fortemente NP-completo, mas, quando as tarefas têm tempo de processamento unitário, não se conhece prova de NP-completude ou algoritmo polinomial. Este problema é um dos dois únicos problemas em aberto da lista definida por Garey e Jonhson (GAREY; JOHNSON, 1979).

Abaixo, a Tabela 2.1 apresenta os resultados de forma sintetizada.

Tabela 2.1: Resultados dos problemas de escalonamento clássico apresentado nesta seção.

Problema	Complexidade	Referência
$P\|tree;p_j = 1;\|C_{max}$	$O(n)$	Hu, 1961
$P\|interval-order;pj = 1;\|C_{max}$	$O(n)$	Papadimitriou, 1979
$P\|prec;p_j = 1;\|C_{max}$	NP-completo	Ullman, 1975
$P_2\|prec;p_j = \{1,2\};\|C_{max}$	NP-completo	Ullman, 1975
$P_2\|prec;p_j = 1\|C_{max}$	$O(e+\alpha(n)*n)$	Gabow, 1982
$P_m\|\|C_{max} < Cte$	$O(n \times C^m)$	Rothkopf, 1966
$P_2\|\|C_{max}$	NP-completo	Karp, 1972
$P_2\|chains;\|C_{max}$	NP-completo	Du, 1991
$P_3\|prec;p_j = 1;\|C_{max}$	Em aberto	

Foram apresentados nesta seção, resultados da literatura envolvendo problemas de escalonamento clássicos em máquinas paralelas, com um conjunto de tarefas com restrições de precedência, sempre considerando a minimização do *makespan*. No próximo capítulo, tais problemas voltam a ser considerados, mas envolvendo também outras restrições importantes no escalonamento de processos em sistemas paralelos e distríbuídos, principalmente atrasos de comunicação com valores extremos, extremamente grandes ou desprezíveis.

3 ESCALONAMENTO DE PROCESSOS COMPUTACIONAIS

Como já mencionado, existe um vasto material disponível na literatura sobre problemas de escalonamento, envolvendo tanto questões teóricas quanto aplicações específicas. Este capítulo se concentra em estudos de problemas de escalonamento de processos em sistemas paralelos e distribuídos. Na Seção 3.1, são apresentados os principais conceitos envolvidos neste cenário de aplicação. Em seguida, na Seção 3.2, são apresentados os principais modelos de escalonamento de processos vigentes em sistemas paralelos e distribuídos. Por fim, na Seção 3.3, são detalhados os problemas e resultados envolvendo atrasos de comunicação.

3.1 Definições Gerais

Um programa em um computador corresponde a execução de um conjunto de instruções. Tal conjunto, no entanto, nem sempre é conhecido a priori, pois ele pode ser modificado de acordo com a entrada do programa, já que o fluxo do programa pode ser alterado a partir de instruções condicionais. Assim, pode ocorrer de algumas instruções compiladas não serem executadas, enquanto outras serem executadas diversas vezes tanto em *loops*, como em múltiplas chamadas de funções etc. As instruções são unidades atômicas de processamento e, portanto, não podem ser divididas entre processadores. No entanto, para arquiteturas VLIW (*Very Long Instruction Word*) instruções podem ser executadas em conjunto por um mesmo processador fazendo uso do que é conhecido como paralelismo em nível de instrução (ILP - *Instruction Level Parallelism*). Desta forma, a maneira com que as instruções são agrupadas em um mesmo processador pode vir a modificar o tempo de execução final do programa. Alguns trabalhos se dedicam a otimizar o escalonamento neste nível de paralelismo (CHAUDHURI; WALKER; MITCHELL, 1994). Porém, este grau de paralelismo não é explorado nos modelos aqui tratados, a razão disso é que tal tratamento torna o modelo muito complexo para as considerações aqui pretendidas e, portanto, de difícil análise. Além disso, parte deste paralelismo pode ser obtido através de otimizações independentes feitas pelo compilador. Dinechin (DINECHIN, 2004) publicou um extenso relatório que esclarece as semelhanças e diferenças entre o tratamento em nível de instrução e em nível de tarefa .

Naturalmente, a ordem de execução das instruções deve obedecer algumas restrições. Uma das restrições é imposta pelo fato de que as informações necessárias para executar uma tarefa podem ser provenientes de outras tarefas, e, desta forma, obrigando que seja executada posteriormente a essas outras tarefas. Outra possível restrição pode ocorrer devido à reutilização de uma região de memória. Por exemplo, uma determinada tarefa

precisa de uma informação armazenada em certa região da memória e outra tarefa está designada para escrever outra informação nesta mesma região, assim, é necessário, portanto, que a segunda tarefa aguarde a primeira ler esta região antes de modificá-la. Tal fenômeno ocorre devido ao reuso de memória, que visa reduzir a memória utilizada durante a execução programa.

Essas relações de dependências podem ser representadas por um grafo direcionado, no qual cada tarefa é um vértice do grafo e a existência de uma aresta entre dois vértices indica a relação de precedência. A direção da aresta origina no vértice que representa a tarefa que deve ser processada primeiramente e incide no vértice que representa a tarefa que deve ser processada posteriormente. O escalonamento pode tratar cada instrução como um vértice a ser escalonado, no entanto, tal consideração pode ser inviável devido ao número de instruções, e, desta forma, pode ser preferível optar por unir as instruções em rotinas de códigos maiores e tratá-las como um único vértice.

Existem, porém, restrições que não definem uma ordem de precedência, mas uma impossibilidade de paralelismo. Por exemplo, duas tarefas são incompatíveis se fazem uso de um mesmo recurso que não permite acessos concorrentes (por exemplo, recurso de entrada ou saída - I/O), ou ainda, se ambas tarefas estão designadas a executar operações sobre uma estrutura de dados, modificando a estrutura (alterando ponteiros, por exemplo) durante sua execução. Ainda que ao final esta tarefa desfaça as modificações na estrutura, uma outra tarefa não poderá operar nessa estrutura em paralelo. Assim, essas tarefas podem executar anteriormente ou posteriormente uma a outra, mas não podem executar simultaneamente. Esse problema poderia ser sanado se a estrutura fosse replicada para cada tarefa que necessite utilizar-se dela. Isto porém pode não ser viável por razões de memória, ou mesmo, ineficiente a considerar o tempo para copiar tal estrutura. Outras aplicações podem ser encontradas na sintetização de circuitos digitais e no processo de compilação de arquiteturas VLIW (DINECHIN, 2007).

Esse tipo de restrição pode ser generalizado com restrições de *tarefas tipadas* (*Typed Task*). Tal restrição está intimamente relacionada a problemas abordados no final do Capítulo 4, onde tal problema é provado ser NP-completo sob restrições severas.

3.2 Modelos de Escalonamento em Sistemas Paralelos e Distribuídos

Nesta seção são apresentados o *Modelo Clássico* e o *Modelo Clássico Estendido*, ambos com atrasos de comunicação, que são os modelos considerados nos problemas de escalonamento tratados nesta dissertação. Também são apresentados nessa seção o Modelo LogP que representa de maneira mais fiel as arquiteturas atuais em termos de comunicação e o Modelo de Comunicação Hierárquica que possibilita distinções entre os tempo de comunicação as diferentes camadas de acoplamento das máquinas.

3.2.1 Modelos Clássico

O modelo de escalonamento clássico se baseia na máquina abstrata PRAM (sigla em inglês para Máquina Paralela de Acesso Randômico). Como o nome sugere, o modelo está fortemente atrelado ao modelo RAM, o qual assume que o acesso dos dados pelo processador é imediato. O modelo PRAM se diferencia do modelo RAM por permitir uma quantidade ilimitada de processadores, permitindo que qualquer processador possa acessar qualquer dado, à medida que são produzidos. No modelo de escalonamento clássico, porém, a quantidade de processadores pode ser limitada (DROZDOWSKI, 2009a).

A suposição de acesso imediato dos dados se mostrou distante para a maioria das

arquiteturas, em especial para memória distribuída (CHRETIENNE, 1989) (PAPADIMITRIOU; YANNAKAKIS, 1979) (VELTMAN; LAGEWEG; LENSTRA, 1990) (R. Giroudeau; KOENIG, 2007) (DROZDOWSKI, 2009b).

3.2.2 Modelo Clássico Estendido com Atrasos de Comunicação

Esse modelo estende o modelo clássico introduzindo o tempo de transferência de informação entre os processadores. Assume também que a transmissão dos dados é feito de forma paralela ao processamento, portanto, os processadores não deixam de processar tarefas enquanto transmitem ou recebem dados. Neste modelo, o atraso de comunicação pode depender tanto do par de processadores como também do par de tarefas em questão. Desta forma, pode ser definido como $c_{i,j,l,m}$, tal que $\{l, m\} \subseteq P$, e onde P é o conjunto de processadores, e $\{i, j\} \subseteq J$ sendo J o conjunto de tarefas. A dependência entre os processadores visa aderir à topologia das arquiteturas cujo número de comutares entre os pares de processadores são distintos, como arquiteturas circulares, em árvores, barris, etc. Enquanto isso, a dependência entre as tarefas assume as variações de tempo decorrente da quantidade de dados a serem trocados entre as tarefas (DROZDOWSKI, 2009b).

Na maioria dos estudos encontrados na literatura, o atraso independe dos processadores, sendo parametrizado apenas pelo par de tarefas em questão. Além disso, para muitos casos (a depender das aplicações e arquitetura em questão) as variações referentes à quantidade de informação trocada entre as tarefas são desprezíveis frente ao tempo gasto ao se iniciar uma comunicação. Desta forma, muitos estudos também desprezam essas variações, tornando, nesses casos, o atraso de comunicação o mesmo para qualquer par de tarefas e processadores.

3.2.3 Modelo LogP

O modelo $LogP$ modela arquiteturas que utilizem troca de mensagem nas comunicações (em contrapartida à memória compartilhada). Comumente, para tais arquiteturas, se utiliza ferramentas de comunicação baseada em mensagens, por exemplo, MPI (*Message Passing Interface*). Para caracterizar esse cenário o modelo se utiliza de quatro parâmetros.

Um dos parâmetros é a latência (representado pela letra 'L') que é o tempo necessário para enviar informação de um processador a qualquer outro independente do par de tarefas envolvido (pode-se considerar uma transmissão - *broadcast*).

Outro parâmetro neste modelo é a *sobrecarga* (*overhead* - representado pela letra 'o') que é o tempo gasto pelo processador, tanto para enviar como para receber uma mensagem. Durante este tempo, o processador deixa de estar apto para processar qualquer tarefa. A razão da existência deste parâmetro é baseada no fato de que, ainda que o processo de transferência de dados seja feito por um hardware a parte, é necessário ao processador preencher alguma região da memória (*buffer*) e há um desprendimento de tempo necessário para isso.

O *gap* (representado pela letra 'g') é o intervalo de tempo entre receber ou encaminhar mensagens consecutivas. Este parâmetro surge por supor que a região de memória do *buffer* seja limitada. Portanto, é necessário aguardar um tempo para que o hardware encarregado da transferência o esvazie (ou o preencha), à medida que a comunicação é feita.

Por último, o modelo é parametrizado pelo número de processadores representado pela letra 'P'.

Apesar de ser mais realista, há poucos trabalhos com resultados sobre este modelo.

Uma das razões, apontadas na literatura, é que a quantidade de detalhes dificultam resultados sobre este modelo (DROZDOWSKI, 2009b).

3.2.4 Modelo de Comunicação Hierárquica

Ainda mais recente, o modelo de comunicação hierárquica surgiu com o propósito de tratar as diferentes camadas de comunicação e, desta forma discriminar, por exemplo, as comunicações em um mesmo nó de *cluster* (mais rápidas) das comunicações entre nós distintos (mais lentas). Outra possível camada de comunicação é a conexão via rede Ethernet presentes em alguns sistemas distribuídos, que possuem um atraso ordens de grandezas maiores que as outras (DROZDOWSKI, 2009b).

Algumas das suposições feitas no modelo clássico também são válidas para este modelo. Como o fato dos processadores não participarem do processo de transferência de mensagem, e assim poderem processar ininterruptamente as tarefas.

Neste modelo é necessário distinguir os agrupamentos de processadores que pertencem à mesma camada hierárquica (e que, portanto, levam menos tempo para transferir informação). Portanto, é necessário outro parâmetro para descrever a quantidade de processadores em cada um dos agrupamentos. Alguns resultados polinomiais e de NP-completude podem ser encontrados na literatura (BAMPIS; GIROUDEAU; KöNIG, 2003) (GIROUDEAU; KöNIG, 2008).

3.3 Escalonamento com Atrasos de Comunicação

Como já mencionado, desprezar as penalidades impostas pelas arquiteturas pode gerar modelos que não se adequam as mesmas. O atraso de comunicação é, naturalmente, um fator limitante do paralelismo de uma aplicação. Observa-se que a razão entre tempo de processamento e tempo de comunicação diminui à medida que se paraleliza uma aplicação, sendo isto um dos mais significantes efeitos no encurvamento do *speed up* linear ótimo. Corrobora com isto o fato de que o tempo para apenas iniciar uma transferência entre processadores corresponde a cerca de 10.000 instruções de processadores (com poucas exceções) (DROZDOWSKI, 2009b).

Nesta seção estão apresentados resultados de escalonamento de atrasos de comunicação. O conteúdo está concentrado de maneira similar ao apresentado na Seção 2.3.2, mas em problemas que consideram esses atrasos, de forma que ele seja constante ou dependente do par de tarefas no qual a comunicação é realizada.

Problemas de escalonamentos com atrasos de comunicação são mais gerais do que os referidos na seção anterior. Dessa forma, pode-se pensar que os problemas com atrasos de comunicação deverão possuir complexidade de pior caso sempre maior que o problema correspondente sem comunicação. No entanto, isso pode não ser verdade, já que a entrada dos problemas com atrasos de comunicação podem vir a ter de descrever a comunicação entre cada par de tarefas ou par de processadores. Feita essa ressalva, as implicações de NP-Completude dos problemas citados anteriormente são válidos para os casos correspondentes com comunicação.

Um dos primeiros e importantes resultados de escalonamento com atrasos de comunicação foi apresentado por Chrétienne (CHRETIENNE, 1989), que mostrou que escalonar com atrasos de comunicação constantes, em que se admite qualquer precedência e tempos de execução de tarefa e de atrasos de comunicação arbitrários ($P|prec; c; |C_{max}$) é NP-completo, mesmo com uma quantidade ilimitada de processadores ($P_{\infty}|prec; c; |C_{max}$). Portanto, assim como era de se esperar, considerar os atrasos de comunicação tornam

NP-completos alguns problemas polinomiais, já que, quando a quantidade de processadores é ilimitada e atrasos de comunicação são desprezíveis, o problema admite solução trivial.

Na presença de atrasos de comunicação, pode ser mais vantajoso processar a mesma tarefa em mais de um processador, a fim de não aguardar o tempo de transmissão dos dados através da rede. Em 1989, Jung mostrou que, ao considerar essa possibilidade, o problema estudado por Chértiene admite solução pseudo-polinomial (JUNG; KIROUSIS; SPIRAKIS, 1989). Este problema pode ser referenciado como $(P_\infty|prec; c = d; dup; |C_{max})$. O algoritmo apresentado, que utiliza técnicas de programação dinâmica, é válido ainda que o custo de comunicação seja qualquer valor real. Mais precisamente, sua complexidade é n^{d+1}, em que n é o número de tarefas e d é tempo de comunicação entre as tarefas. De fato, não é possível encontrar algoritmo estritamente polinomial para este problema (exceto se $P = \mathcal{NP}$). Este resultado de NP-completude é ainda válido sobre restrições severas como, por exemplo, quando os tempos de execução das tarefas forem unitários (UET – *unit execution times*) $(P_\infty|prec; c; dup; pj = 1; |C_{max})$ (PAPADIMITRIOU; YANNAKAKIS, 1990). Uma restrição que torna o problema polinomial é restringir a precedência para árvore descendente $(P_\infty|out - trees; c_{i,j}; dup; |C_{max})$ (CHRETIENNE, 1994), ou limitar o tempo de processamento da menor tarefa de forma que ela seja sempre maior ou igual ao atraso de comunicação (SCT – *Small Communication Times*) $(P_\infty|prec; c_{i,j}; sct; dup; |C_{max})$. Para este problema, SCT é condição suficiente, mas não necessária. De fato, existe solução $O(n^2)$ com a seguinte restrição: $\forall j \in J, \min_{i \in ancest(j)}(p_i) \geq \max_{i \in ancest(j)}(c_{i,j})$ (COLIN; CHRETIENNE, 1991). Ainda, tal restrição pode ser relaxada de forma a manter o problema polinomial (DARBHA; AGRAWAL, 1998).

Naturalmente, quando os atrasos são determinados para cada par de tarefas, o problema se torna mais genérico e mais difícil. Alguns resultados de NP-completude só são válidos nestes casos. Como no caso de árvores descendentes, quando reprocessamento não é permitido $(P_\infty|out - trees; c_{i,j}; |C_{max})$ e em árvores incidentes admitindo ou não reprocessamento (CHRETIENNE, 1994).

O fato do tempo de comunicação ser limitado pelo tempo de execução da menor tarefa mostrou-se uma restrição importante, simplificando a solução do problema sobre diversas parametrizações. Essa restrição foi utilizada em outros estudos, ficando conhecida na literatura como pequenos atrasos de comunicação (SCT - *small communication times*) (PICOULEAU, 1995) (VARVARIGOU et al., 1996). Em contrapartida, os casos em que se admitem custos de comunicação maiores que o tempo de processamento da menor tarefa que ficaram conhecidos como longos atrasos de comunicação (LCT - *large communication times*)(GIROUDEAU et al., 2008) (AFRATI et al., 2005).

Em alguns casos, reprocessamento de tarefas pode não ser possível, por exemplo, devido a restrições de acesso a recursos. E, em outros casos, pode também não ser desejável, por exemplo, por conta do aumento do custo energético na execução do escalonamento encontrado. Não considerar esta possibilidade, no caso em que a quantidade de processadores é ilimitada, não modifica resultados de NP-completude para UET, pelo contrário, é conhecida a NP-completude para UET-UCT (RAYWARD-SMITH, 1987), consequentemente, não existe solução pseudo-polinomial, como apresentada em Jung (exceto se $P = \mathcal{NP}$). Além disso, a restrição de SCT mencionada é insuficiente para tornar o problema polinomial (PICOULEAU, 1995). No entanto, quando SCT é combinado com a restrição de árvores incidentes ou descendentes, o problema é polinomial (CHRETIENNE, 1989) (ANGER; HWANG; CHOW, 1990).

Em vista da NP-completude do problema, ainda que sobre restrições de tempo de processamento unitários (UET – *unit execution times*) e tempo de atrasos de comunicação unitários (UCT – *unit communication times*), foram propostas novas restrições a fim de delimitar a fronteira entre os casos em que a NP-completude era conhecida dos casos em que não era. Um dos trabalhos nesse sentido utilizou o comprimento do escalonamento ótimo como delimitador entre tais casos. Foi mostrado que se o comprimento for $\lambda \geq 6$, o problema é NP-completo, do contrário, é polinomial ($P_\infty|prec, c = 1, p_j = 1|C_{max} \leq 5$). Ainda, se a quantidade de processadores for indicada na instância do problema, então, decidir se existe escalonamento maior ou igual a 4 já é NP-completo e do contrário é polinomial($P|prec, c = 1, p_j = 1|C_{max} < 4$) ((HOOGEVEEN; LENSTRA; VELTMAN, 1994)). Se, ao invés de atrasos de comunicação unitário, forem considerados tempo de comunicação c, tal que $c > 1$, decidir sobre a existência de um escalonamento máximo maior ou igual que $c + 3$ é NP-completo (BAMPIS; GIANNAKOS; KöNIG, 1996) . De fato, este caso é fortemente NP-completo e se mantém assim, ainda que o número de processadores seja limitado em apenas dois e a ordem de precedência seja uma árvore binária $P_2|binary - tree; c \geq 2; p_j = 1; |C_{max}$ (AFRATI et al., 2005).

Apesar da dificuldade do problema ainda que a quantidade de processadores seja livre, resultados com quantidade de processadores indicados na instância dos problemas existem. Ali e El Rewini estenderam o resultado de Papadimitriou e Yannakakis apresentado para absorver tempos de comunicação unitários, apresentando algoritmo polinomial para encontrar o escalonamento ótimo quando a precedência é restrita a ordem de intervalo ($P|interval - order; c = 1; p_j = 1; |C_{max}$) (ALI; ELREWINI, 1995). Quando a precedência é uma árvore (descendente ou incidente), é possível utilizar algoritmo pseudo-polinomial na quantidade de processadores disponíveis. Mais precisamente, Varvarigou apresentou algoritmo para o problema ($P_m|tree; c = 1; p_j = 1; |C_{max}$) com complexidade $n^{2(m-2)}$, onde é o número de processadores disponíveis e o número de tarefas (VARVARIGOU et al., 1996). Engels e outros estenderam este resultado permitindo atrasos de comunicações distintos para os pares de tarefas, mas limitados por uma constante. A programação dinâmica apresentada possui dependência pseudo-polinomial no número de processadores disponíveis e exponencial no atrasos de comunicação ($P_m|tree; c_{i,j} \in \{1, 2, \ldots, D\}; p_j = 1; |C_{max}$) (ENGELS et al., 2001).

Os resultados apresentados nesta seção podem ser visualizados nn Tabela 3.1.

Tabela 3.1: A tabela apresenta os resultados desta seção em relação aos problemas com atrasos de comunicação.

Problema	Complexidade	Referência
$P_\infty\|prec; c; \|C_{max}$	NP-completo	Chretienne, 1988
$P_\infty\|prec; c = d; dup; \|C_{max}$	$O(n^{d+1})$	Jung, 1989
$P_\infty\|prec; c; dup; pj = 1; \|C_{max}$	NP-completo	Papadimitriou, 1990
$P_\infty\|out - trees; c_{i,j}; dup; \|C_{max}$	$O(n^2)$	Colin, 1991
$P_\infty\|out - trees; c_{i,j}; \|C_{max}$	NP-completo	Chretienne, 1994
$P_\infty\|prec, c = 1, p_j = 1\|C_{max} \geq 6$	NP-completo	Hoogeveen, 1994
$P_\infty\|prec, c = 1, p_j = 1\|C_{max} \leq 5$	O(1)	Hoogeveen1, 994
$P_2\|binary - tree; c \geq 2; p_j = 1; \|C_{max}$	NP-completo	Afrati, 2005
$P_m\|tree; c = 1; p_j = 1; \|C_{max}$	$O(n^{2(m-2)})$	Varvarigou, 1996
$P_m\|tree; c_{i,j} \in \{1, 2, \ldots, D\}; p_j = 1; \|C_{max}$	$O(m \times n^D)$	Engels, 2001

3.3.1 Resultados de Aproximações

Devido a NP-completude de grande parte dos problemas de escalonamento, muitos estudos da área se concentraram em criar heurísticas para minimizar o tempo total. Algumas dessas heurísticas utilizam escolhas gulosas como *HLF- Highest Level First* ou *ETF - Earliest Task First*. Tais heurísticas são às vezes referenciadas como heurísticas de estágio único, em contrapartida às heurísticas que optam por separar o processo de escalonamento em três etapas. As heurísticas de três etapas dividem o processo de escalonamento nas seguintes etapas: agrupamento (*clustering*), mapeamento (*mapping*) e sequenciamento (*sequencing*). O agrupamento trata de escolher as tarefas que devem ser processadas no mesmo processador, visando reduzir os custos de comunicação. A etapa seguinte, o mapeamento, ocorre apenas se houver algum tipo de heterogeneidade entre os processadores ou os barramentos de conexão que conecta pares dos processadores. Isto porque a heurística se encarrega de determinar qual o processador que cada um desses agrupamentos será processado, visando minimizar o custo tanto de comunicação quanto de processamento. Nesta etapa, finalmente, o sequenciamento determina a ordem em que as tarefas de um mesmo grupo devem ser processadas. Em algumas heurísticas essa etapa é realizada juntamente com o agrupamento (DROZDOWSKI, 2009b).

Alguns estudos apresentam algum tipo de garantia para as heurísticas apresentadas. Nos trabalhos já mencionados foram apresentados algoritmo de aproximação para os problemas tratados. Para o problema $P_\infty|prec; c; dup; pj = 1; |C_{max}$, foi apresentado um algoritmo que garante solução no máximo duas vezes maior que o escalonamento ótimo (PAPADIMITRIOU; YANNAKAKIS, 1990). Para $P|tree; c = 1; p_j = 1; |C_{max}$, a diferença entre o escalonamento ótimo e o escalonamento encontrado pelo algoritmo apresentado é no máximo $m - 2$, onde m é o número de processador disponível (VARVARIGOU et al., 1996). Essa diferença pôde ser reduzida pela metade (GUINAND; RAPINE; TRYSTRAM, 1997). Outro resultado importante foi apresentado para o problema $P|prec; c = 1; p_j = 1; |C_{max}$, a partir da conversão de um escalonamento aproximado para $P_\infty|prec; c = 1; p_j = 1; |C_{max}$. O fator de aproximação apresentado foi $\frac{7}{3} - \frac{4}{3m}$, onde m representa a quantidade de processadores utilizados (HANEN; MUNIER, 2001).

Para muitos problemas, foi demostrado que encontrar algoritmo de aproximação mais próximo que um certo limite é NP-completo. Entre os problemas NP-completos, as diferenças em relação a quão precisos os algoritmos podem ser são muito grande, alguns problemas NP-completos podem possuir limites nos fatores de aproximação que possuem crescimento assintótico exponencialmente maior que os limites de outros problemas (assumindo $\mathcal{P} \neq \mathcal{NP}$) (JOHNSON, 2006). Nesse sentido, muitos trabalhos se dedicaram a descobrir esses limites para os vários problemas de escalonamento. Novamente, alguns resultados de aproximações são resultados de trabalhos já apresentados nas seções anteriores. Como consequência do trabalho apresentado por Hoogeveen, não existe algoritmo polinomial com fator de aproximação menor que $\frac{5}{4}$ para o problema $P|prec, c = 1, p_j = 1|C_{max}$ nem menor que $\frac{7}{6}$ para $P_\infty|prec, c = 1, p_j = 1|C_{max}$ (HOOGEVEEN; LENSTRA; VELTMAN, 1994) e quando os atrasos de comunicação são maiores que uma unidade de tempo ($P_\infty|prec, c \geq 2, p_j = 1|C_{max}$), o menor fator de aproximação possível para um algoritmo polinomial tem de ser maior que $1 + \frac{1}{c+4}$ (supondo $\mathcal{P} \neq \mathcal{NP}$) (GIROUDEAU et al., 2008).

Há também inúmeros trabalhos na literatura referentes a algoritmos para os quais não existe nenhuma garantia, mas que apresentam resultados empíricos satisfatórios. Existe um grande conjunto de heurísticas gulosas *ISH - insertion scheduling heuristic*, *MCP - modified critical path*, *ETF - Earliest Time First*, *DLS - dynamic level scheduling* entre

outras. Cada uma dessas heurísticas, porém, se apresentam melhor ou pior a depender das condições estabelecidas. Devido a isto, a comparação entre as várias heurísticas não é tarefa simples. Em vista deste problema, Kwok e Ahmad propõem um conjunto de programas (*benchmarks*) que visa servir de base comparativa para problemas de escalonamento (AHMAD; KWOK, 1998). Neste vasto trabalho é apresentado gráficos comparativos entre as diversas heurísticas em relação aos programas propostos. Os resultados indicam que heurísticas baseadas no caminho crítico apresentam melhores desempenhos.

Os resultados apresentados podem ser visualizados na Tabela 3.2.

Tabela 3.2: A tabela apresenta os resultados desta seção. São apresentadas as garantias de aproximação e a complexidade do algoritmo conhecido ou a indicação de $\mathcal{NP}$-*completude*.

Problema	Aproximação	Complexidade	Referência
$P_\infty \vert prec; c; dup; pj = 1; \vert C_{max}$	$2 \times C_{max}$	$O(n)$	Papadimitriou, 1990
$P_m \vert tree; c = 1; p_j = 1; \vert C_{max}$	$C_{max} + (m-2)/2$	$O(n)$	Guinand, 1997
$P_m \vert prec; c = 1; p_j = 1; \vert C_{max}$	$(\frac{7}{3} - \frac{4}{3m}) \times C_{max}$,	$O(n)$	Hanen, 2001
$P \vert prec, c = 1, p_j = 1 \vert C_{max}$	$\leq \frac{5}{4} \times C_{max}$	NP-completo	Hoogeveen, 1994
$P_\infty \vert prec, c = 1, p_j = 1 \vert C_{max}$	$\leq \frac{7}{6} \times C_{max}$	NP-completo	Hoogeveen, 1994
$P_\infty \vert prec, c \geq 2, p_j = 1 \vert C_{max}$	$\leq 1 + \frac{1}{c+4} \times C_{max}$	NP-completo	Giroudeau, 2008

4 RESULTADOS

Neste capítulo, serão apresentados os resultados gerados nesta dissertação, envolvendo problemas de escalonamento de tarefas onde os atrasos de comunicação estão restritos a some dois valores extremos, $\{0, \infty\}$, e com tarefas de tempos de execução unitários que podem estar submetidas a alguma ordem de precedência. Desta forma, se o atraso de comunicação atrelado a um par de tarefas for ∞, tais tarefas devem ser processadas no mesmo processador. Na Seção 4.1, são apresentados os resultados de NP-completude para dois casos do problema geral em estudo. Na seção seguinte, Seção 4.2, são apresentados os resultados polinomiais estabelecidos: o primeiro, quando a ordem de precedência está restrita a árvores; e, o segundo, quando é permitido processar várias cópias de uma mesma tarefa em processadores distintos. No final do capítulo, na Seção 4.3, são apresentados algoritmos de programação dinâmica considerando escalonamentos de cadeias com restrição de alocação das tarefas nos processadores.

4.1 Resultados NP-Completos

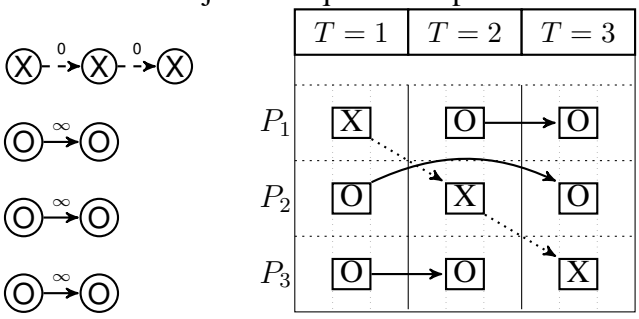
Esta seção apresenta dois estudos importantes para o problema de escalonamento geral em consideração, mostrando a NP-completude do problema proposto sob determinadas parametrizações. A seguir, é apresentado o trabalho de Sevastyanov e outros, cujo problema analisado é similar ao problema $P|chains; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$. Nesta seção, também é apresentado o resultado de NP-completude deste problema. Na seção seguinte, é visto que $P2|prec; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$ é NP-completo através de redução ao problema estudado por Jansen (JANSEN, 1994).

4.1.1 $P|chains; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$

O problema $P|chains; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$ pode ser interpretado da seguinte maneira. Cada subcadeia maximal, nas quais todas as tarefas estão ligadas através de arestas $c_{i,j} = \infty$, pode ser interpretada como uma única tarefa cujo tempo de processamento é a quantidade de tarefas (de tempo de processamento unitário) presentes na sub-cadeia. Além disso, é preciso permitir aos escalonamentos interromper essas tarefas nas transições dos intervalos unitários, ou seja, interromper de maneira que todas as partes da tarefa possua um intervalo inteiro.

Muitos problemas na literatura abordam a possibilidade de interrupções no processamento de tarefas, contudo as interrupções não são restritas da maneira mencionada. Além disso, são sempre acompanhadas da possibilidade de migrar (permitir que a tarefa seja interrompida em um processador e seja reiniciada em outro). É fácil verificar que, devido o atraso de comunicação infinitamente grande, tais migrações não devem ser permitidas

Figura 4.1: A Figura 4.2(a) mostra um conjunto de quatro cadeias com suas tarefas e suas respectivas comunicações. Em tracejado estão representadas as comunicações de atraso desprezíveis e, em linhas contínuas, estão representadas as comunicações de atraso infinito. A Figura 4.2(b) mostra que a cadeia completamente processada pelo processador P_2, ao ser interpretada como uma única tarefa, teria de ser interrompida em T_1 e retomada em T_3 para que o escalonamento seja ótimo quando disponibilidade de três processadores.



(a) Cadeias. (b) Escalonamento.

para que a correspondência seja válida. Por outro lado, a Figura 4.1 mostra um conjunto de cadeias no qual o tipo de interrupção mencionado é necessário ao escalonamento ótimo. Portanto, tais possibilidades de interrupção não podem ser descartadas. Na Figura 4.2(b), cada cadeia com duas tarefas é interpretada como uma tarefa de tempo de execução igual a dois, devido ao atraso de comunicação de suas arestas, onde o processador P_2 tem de interromper o processamento de uma tarefa em $T = 1$ e continua em $T = 3$, a fim de se escalonar em tempo ótimo.

O problema correspondente pode ser referenciado por $P|pmtn^*(delay = \infty); chains; |C_{max}$, onde $delay$ é o atraso do processo de migração e $pmtn^*$ representa a possibilidade de interromper as tarefas da maneira mencionada.

Em 2010, Sevastyanov e outros consideraram atrasos causados no processo de migração (SEVASTYANOV; SITTERS; FISHKIN, 2010). O problema analisado admitia qualquer atraso de comunicação, mas não assumia a existência de ordem de precedência nem restringia as interrupções a intervalos inteiros, podendo ser referenciado como $P|pmtn(delay = d); |C_{max}$.

Neste trabalho é visto que para o subproblema em que as instâncias são restritas, tais que $d \leq max\{\frac{1}{m}\sum_{j=1}^{n} p_j, p_{max}\} - p_{max}$, sendo p_{max} o tempo de processamento da maior tarefa, é possível encontrar escalonamento ótimo em tempo polinomial. Se tal restrição não for obedecida, o problema permitirá um conjunto de instâncias as quais é possível reduzir todas as instâncias de algum problema da família de problemas do α-PARTITION PROBLEM, sendo, portanto, NP-completo.

Apesar da similaridade entre os problemas, a prova apresentada por Sevastyanov e outros, faz uso de particularidades que não pode ser aproveitada diretamente para demostrar a NP-completude do problema $P|chains; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$. Em particular, utiliza-se da possibilidade de haver tarefas maiores que o atraso de comunicação e da possibilidade de migração. No entanto, é possível verificar que o problema α-PARTITION PROBLEM também pode ser reduzido ao problema tratado nessa seção. A prova de NP-

completude apresentada é feita através da redução do 3-PARTITION PROBLEM, que está definido a seguir.

3-PARTITION PROBLEM
Entrada: $\langle A \rangle$, onde A é um conjunto de inteiros positivos com a seguinte restrição $\forall a_i \in A \mid \frac{B}{4} < a_i < \frac{B}{2}$, onde $|A| = 3m$ e B = $\frac{\sum a_i}{m}$.
Saída: Decidir se é possível particionar o conjunto A em m sub-conjuntos de forma que a soma dos elementos seja a mesma para todos os subconjuntos.

Observa-se que se existir tal partição, todos os subconjuntos devem conter exatamente três elementos de A. Tal problema é fortemente NP-completo (GAREY; JOHNSON, 1979), o que significa que a existência de um algoritmo pseudo-polinomial implica em $P = NP$.

O problema tratado nesta seção pode ser precisamente definido como a seguir.

PROCESSADORES ARBITRÁRIOS
Entrada: $\langle Cadeias, C, P, t \rangle$, onde $Cadeias$ é um conjunto de cadeias de tarefas, C é uma função que associa todos os pares de tarefas consecutivas em uma mesma cadeia para um dos valores $\{0, \infty\}$ e P é um conjunto de processadores.
Saída: O problema consiste em decidir se existe escalonamento válido tal que o último $slot$ associado a alguma tarefas é menor ou igual a t.

Um escalonamento S é válido se associar para cada tarefa em alguma cadeia tanto um natural representando um $slot$ de tempo, como também um processador, tal que para todo par de tarefas $(J_{i,j}, J_{i,j+1})$, no qual $J_{i,j}$ é sucedida por $J_{i,j+1}$ em alguma cadeia, o slot de tempo $t_{i,j}$ associado por S para $J_{i,j}$ é anterior ao slot de tempo $t_{i,j+1}$ associado para $J_{i,j+1}$. Além disso, se $C(J_{i,j}, J_{i,j+1}) = \infty$, estas tarefas devem estar associadas ao mesmo processador.

Teorema 4.1.1. *PROCESSADORES ARBITRÁRIOS é NP-completo*

Prova. O problema pertence a $\mathcal{NP}$, pois o próprio escalonamento pode ser usado como certificador, já que tem tamanho polinomial na entrada do problema e a verificação a partir do escalonamento é polinomial. Para isso, basta verificar as restrições de ordem de precedência e as restrições de processamento em um mesmo processador impostas aos pares de tarefas tais que $C(J_{i,j}, J_{i,j+1}) = \infty$. Por último, verificar se de fato não existe tarefa processada após t.

A demostração que o problema é NP-difícil é feita a partir da redução do 3-PARTITION PROBLEM. Dada uma instância $\langle A \rangle$ qualquer do 3-PARTITION PROBLEM, monta-se uma instância $\langle Cadeias, C, P, t \rangle$ de TAREFAS ENCADEADAS da seguinte maneira. Para cada elemento de $a_i \in A$, cria-se uma cadeia em $Cadeias$ com a_i tarefas, tal que para cada par de tarefas $(J_{i,j}, J_{i,j+1})$ consecutivas nesta cadeia faz-se $C(J_{i,j}, J_{i,j+1}) = \infty$, por último, faz-se $P = \{1, 2, 3, ..., \frac{|A|}{3}\}$ e $t = \frac{\sum a_i}{\frac{|A|}{3}}$.

Primeiramente, demostra-se que se existir escalonamento válido para essa instância, então, existe a partição do 3-PARTITION PROBLEM. Um escalonamento S válido tem de processar todas as tarefas de uma mesma cadeia em um mesmo processador por conta das restrições impostas por C. Nota-se que a quantidade de $slots$ existentes até o $slot$ t é $P \times t$, dado que este valor é igual ao número de tarefas, não pode existir $slot$ vazio em S. Assim, S particiona as cadeias nos processadores de forma a executar em cada um

três cadeias cujo total de tarefas é t. Esta partição é uma partição válida para a instância original $\langle A \rangle$ do 3-PARTITION PROBLEM, relacionando as cadeias em $Cadeias$ com os elementos de A que lhe deram origem.

Não é difícil verificar que usando essa mesma correspondência é possível utilizar uma partição válida para o 3-PARTITION PROBLEM, para escolher os processadores de cada cadeia de forma a obter um escalonamento com as restrições desejadas. Portanto, se houver partição válida para o 3-PARTITION PROBLEM também existirá um escalonamento válido. □

4.1.2 $P_2|prec; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$

Jansen estudou o que ele denominou de escalonamento de tarefas tipadas ou diferenciadas (*typed task*) (JANSEN, 1993). Essa mesma restrição também possui outras denominações na literatura, tais como *multi-purpose machine scheduling, scheduling with eligibility constraints, scheduling with processing set restrictions* (LEUNG; LI, 2008). Esses problemas consistem em impor restrições sobre em quais processadores cada tarefa pode ser executada. O problema estudado por Jansen impunha a cada tarefa, ser processada em apenas um processador pré-determinado, e assim, o processador em que a tarefa era designada para processar estabelecia o tipo da tarefa. Jansen também mostrou que, ainda que a quantidade de tipos seja restrita a apenas dois, a quantidade de processadores seja apenas um processador por tipo e a precedência seja um conjunto de cadeias, o problema é NP-completo. Este problema pode ser precisamente definido da seguinte maneira.

TAREFAS TIPADAS
Entrada: $\langle Cadeias, Tipo, t \rangle$ que consiste de um conjunto de cadeias referenciado por $Cadeias$ (seja J o conjunto das tarefas de todas as cadeias), as quais para cada tarefa pertencente a J existe uma associação, feita por $Tipo$ a dois possíveis tipos de processador, sejam eles $Tipo_1$ e $Tipo_2$. Por último, é dado t um inteiro qualquer.
Saída: O problema consiste em decidir, se existe escalonamento válido, tal que o maior $slot$ de tempo associado a alguma tarefa seja menor que t.

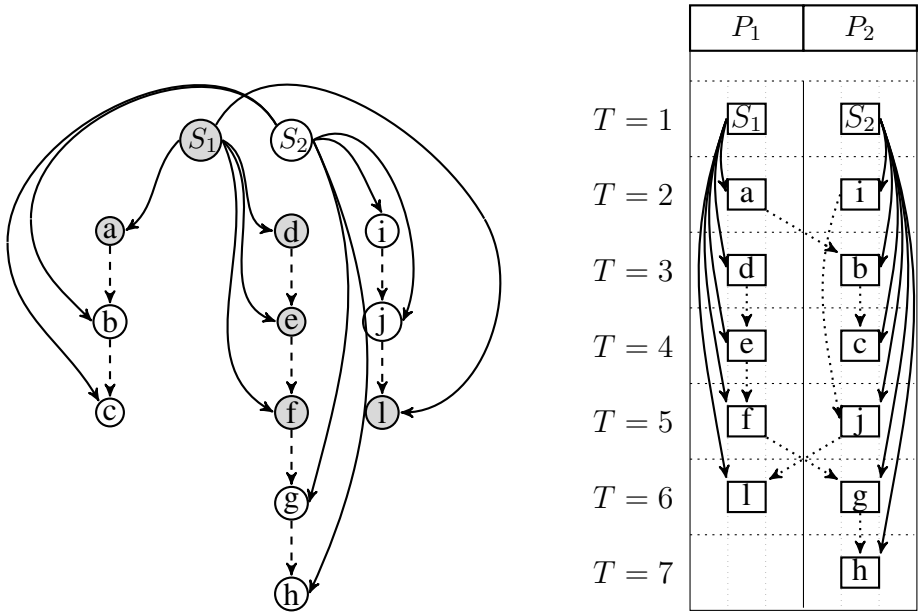
Um escalonamento, S, associa para cada tarefa um $slot$ de tempo para ser processada ($S : J \to \mathbb{N}$). S é válido se o $slot$ de tempo associados as tarefas obedecem ordem de processamento estabelecida pelas cadeias. Além disso, não pode haver duas tarefas processadas no mesmo $slot$ tal que estejam associadas por $Tipo$ para o mesmo tipo.

Este problema se reduz ao problema desta seção, cuja definição precisa é apresentada a seguir.

PRECEDÊNCIA ARBITRÁRIA
Entrada: $\langle G = (V, E), C, t \rangle$, onde G é um *grafo direcionado acíclico* e C é uma associação de E para $\{0, \infty\}$, referente ao custo de comunicação.
Saída: O problema novamente consiste em decidir se existe escalonamento válido, tal que o maior $slot$ de tempo associado a alguma tarefa seja menor que t.

Um escalonamento válido, S, é tal que estabelece para cada tarefa em V um entre dois processadores disponíveis, um $slot$ de tempo de forma que todo antecessor de $v \in V$ tenha $slot$ anterior ao $slot$ associado a v e o escalonamento obedeça as restrições de comunicações. Assim, se existe $e = (u, v) \in E | C(e) = \infty$, então u e v estão associados

Figura 4.2: A Figura 4.3(a) mostra um conjunto de três cadeias de tarefas tipadas, com a adição de duas tarefas-origem ou *source* (S_1 e S_2), cada uma se conectando através de arestas Inf com todas as tarefas de um mesmo tipo, onde os tipos estão diferenciados pelo preenchimento ou não do nó. A Figura 4.3(b) apresenta o escalonamento ótimo em relação a este conjunto. Note que apenas as arestas tracejadas podem cruzar o eixo dos processadores.



(a) Cadeias. (b) Escalonamento.

ao mesmo processador.

É possível transformar uma instância do problema *TAREFAS TIPADAS* para PRECEDENCIA ARBRITRÁRIA, como é mostrado na prova do teorema a seguir.

Teorema 4.1.2. *PRECEDÊNCIA ARBITRÁRIA é a NP-completo.*

Prova. O problema pertence a $\mathcal{NP}$, onde novamente verifica-se isso utilizando o próprio escalonamento, S, como certificador. Como S apenas estabelece dois elementos, um processador e um tempo, para cada tarefa em V, fica evidente que S é polinomial na entrada da instância. Além disso, a verificação a partir de S são duas simples conferências. A primeira é uma verificação de precedência dos $slots$ de tempos associados aos pares de tarefas, (u, v), para os quais exista $e = (u, v) \in E$. A segunda é uma verificação de igualdade nos processadores associados para as tarefas desses pares para os quais $C(e) = \infty$. É fácil ver que isto pode ser feito em tempo polinomial na instância do problema.

A prova de ser $\mathcal{NP}$-difícil se dá, como foi antecipado, pela redução do problema TAREFAS TIPADAS a este problema. Dada uma instância qualquer $\langle Cadeias, Tipo, t\rangle$ do TAREFAS TIPADAS constrói-se uma instância $\langle G = (V, E), C, t\rangle$ do PRECEDÊNCIA ARBITRÁRIA da seguinte maneira. Cria-se G de forma a preservar as cadeias em $Cadeias$, adicionando duas tarefas *sources*, J_{S_1} e J_{S_2}. Associa-se a cada uma dessas tarefas com cada um dos tipos presentes na imagem de $Tipo$ criando arestas de custo de comunicação ∞, que se originam nessas tarefas e incidem em cada tarefa do respectivo tipo (verifique a Figura 4.2). Desta forma tem-se, $V := J \cup \{J_{S_1}, J_{S_2}\}$, onde J é a

união de todas as tarefas em todas cadeias de $Cadeias$ e E é tal que para quaisquer duas tarefas,$J_{i,j}$, $J_{i,j+1}$, consecutivas em alguma cadeia $i \in Cadeias$, existe $e = (J_{i,j}, J_{i,j+1}) \in E$ com $C(e) = 0$. Além disso, para toda tarefa $J_{i,j} \in J|Tipo(J_{i,j}) = Tipo_k$ também existirá $e = (J_{S_k}, J_{i,j}) \in E$ com $C(e) = \infty$. Assim, todas as tarefas de um mesmo tipo terão de ser escalonadas no mesmo processador da tarefa *source* mãe. É possível verificar também que as duas tarefas *sources* serão processadas em processadores distintos, e que após o primeiro instante de tempo, no qual cada uma das tarefas *sources* são processadas, as restrições do problema se tornam exatamente as dos problema estudado por Jansen. Portanto, a correspondência entre a existência dos escalonamentos em cada uma dessas instâncias é direta. □

4.2 Resultados Polinomiais

Nesta seção, serão apresentados os resultados de polinomialidade obtidos para dois outros problemas de escalonamento com atrasos de comunicação de valores extremos, sendo o primeiro, na Seção 4.2.1, o problema $P_\infty|tree, c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$, e, o segundo, na Seção 4.2.2, o problema $P_\infty|prec; c_{i,j} \in \{0, \infty\}; dup, p_j = 1|C_{max}$.

4.2.1 $P_\infty|tree, c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$

O problema de escalonar árvores direcionadas enraizadas para qualquer número de processadores e qualquer custo de comunicação e tarefas de tempo de execução também quaisquer, $P\infty|tree, c_{i,j}, p_j|C_{max}$, foi mostrado NP-completo por Chértiene (CHRETIENNE, 1994). Neste mesmo artigo, Chértiene também apresentou algoritmo polinomial para este problema quando se permite re-processamento de tarefas. Quando o custo é restrito para SCT (*Small Communication Times*) o problema é solucionável em tempo polinomial para qualquer precedência (COLIN; CHRETIENNE, 1991). O problema abordado nesta seção, permite atrasos de comunicação maiores que o tempo de execução das tarefas, não sendo, portanto, classificado como SCT, e não permite re-processamento de tarefas. Será apresentada aqui estratégia de priorização que é ótima para o problema apresentado.

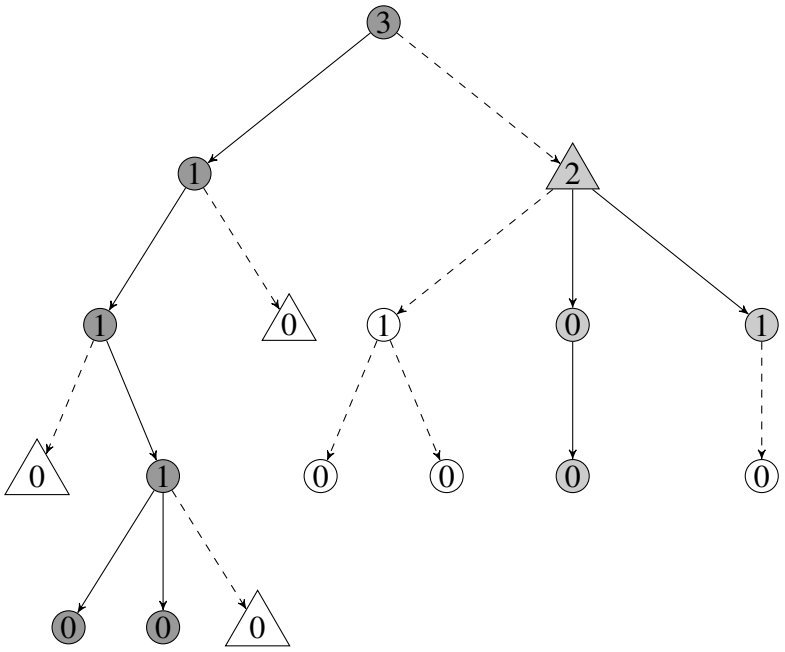
TAREFAS EM ÁRVORE
Entrada: $\langle T, C\rangle$, seja $T = (V, E)$ uma árvore direcionada enraizada que representa as tarefas, onde cada $e \in E$ está associada por $C : E \to \{0, \infty\}$ a um dos dois valores possíveis $\{0, \infty\}$, que representam o atraso de comunicação.
Saída: Um escalonamento válido o qual minimiza o último $slot$ associado a alguma tarefa.

Um escalonamento para o problema é uma associação dos elementos de V a dois números naturais, representando o processador e o $slot$ de tempo em que a tarefa foi processada. O escalonamento, S, é válido se duas condições forem satisfeitas. A primeira é S associar para todo vértice v um $slot$ de tempo posterior aos $slots$ associados ao seus ancestrais na árvore. A outra condição é S associar o mesmo processador para todo par de tarefas (u, v) para os quais existe $C(u, v) = \infty$.

Definição 4.2.1. *Define-se como* $arestasZero$*, o conjunto de arestas* $e \in E \mid C(e) = 0$.

Definição 4.2.2. *Define-se como* $arestasInf$*, o conjunto de arestas* $e \in E \mid C(e) = \infty$.

Figura 4.3: Esta figura representa uma árvore descendente. Estão destacadas duas sub-árvores *Árvores-Inf-Conectadas* maximais, uma em tom de cinza escuro outra em tom de cinza claro. Todos os outros vértices, formam sozinhos suas próprias *Árvores-Inf-Conectadas* maximais. Os nós triangulares representam raízes das *Árvores-Inf-Conectadas-Filhas*. No interior de cada vértice está indicada sua $Prioridade$ de acordo com o escalonamento ótimo.



Definição 4.2.3. *Define-se como Árvore-Inf-Conectada , o conjunto de vértices conexos por arestas de $arestasInf$, sendo ArvInfCon(v) a Árvore-Inf-Conectada maximal pertencentes a árvore enraizada em v(ver figura 4.3).*

Definição 4.2.4. *Define-se como Vértice-Conector, um vértice de uma Árvore-Inf-Conectada que possui descendentes conectados por $arestasZero$, cada árvore enraizada em um desses filhos é chamada de Árvore-Inf-Conectada-Filha(ver figura 4.3).*

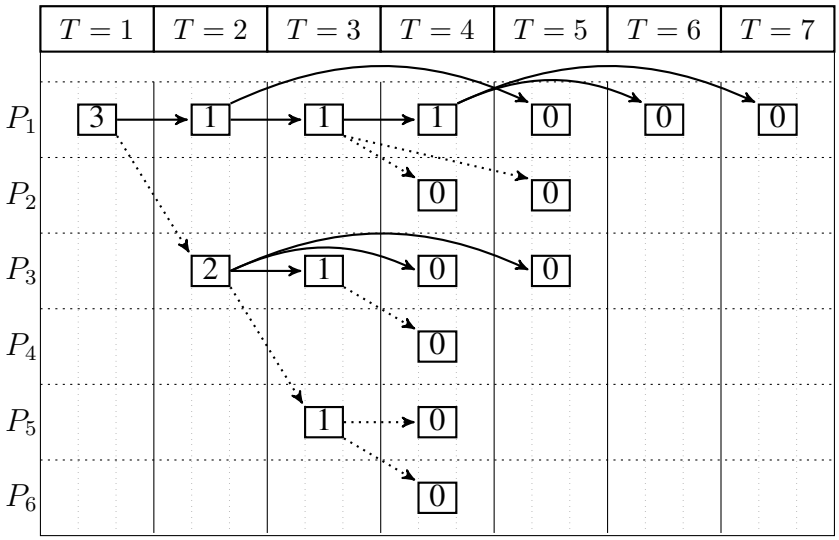
Definição 4.2.5. *Define-se como $mst(T_v)$ ($MinimalScheduleTime$), o menor tempo de processamento possível para escalonar a arvore T_v.*

Definição 4.2.6. *A Ordem de um vértice v, em relação a um escalonamento S, $Ordem_S(v)$, é a quantidade de vértices da ArvInfCon(v) que foram processados anteriormente a v em S, mais um.*

Definição 4.2.7. *A $Prioridade$ de um vértice v, em relação a um escalonamento S, $Pri_S(v)$, é o maior entre os $makespan$ das Árvore-Inf-Conectada-Filha cujas raízes são descendentes de v. Se v não possuir tais vértices como descendente, a prioridade de v é 0 (ver figura 4.3).*

Lema 4.2.1. *O tempo de escalonamento total t_S de um escalonamento S válido para uma árvore T_r enraizada em um vértice r é igual a $\max\{Ordem_S(v) + Pri_S(v)\}, \forall v \in$ ArvInfCon(r).*

Figura 4.4: Escalonamento da árvore exposta na Figura 4.3.



Prova. Nenhum vértice pertencente a *ArvInfCon(v)* é escalonado após $\max\{Ordem_S(v) + Pri_S(v)\}, \forall v \in$ *ArvInfCon(v)*, pois cada vértice v em *ArvInf-Con(r)* é escalonado exatamente no $slot\ Ordem_S(v)$.

Além disso, seja v um *Vértice-Conector*. Todo vértice pertencente a alguma *Árvore-Inf-Conectada-Filha* de v é escalonado até $Ordem_S(v) + Pri_S(v)$, já que v é escalonado no $slot\ Ordem_S(v)$ e, por definição, $Pri_S(v)$ é maior ou igual ao $makespan$ de toda árvore *Árvore-Inf-Conectada-Filha* descendente de v.

Por último, seja v um vértice pertencente a *ArvInfCon(v)* para o qual $Ordem_S(v) + Pri_S(v)$ é máximo. Então, v não possui nenhum descendente imediato u pertencente a *ArvInfCon(v)* tal que $Pri_S(v) = Pri_S(u)$, pois, certamente $Ordem_S(u) > Pri_S(v)$ e isso contrariaria o fato de $Ordem_S(v) + Pri_S(v)$ ser máximo. Se não existe tal descendente imediato, então, ou $Pri_S(v) = 0$ ou existe descendente imediato de v que é raiz de uma *Árvore-Inf-Conectada-Filha* cujo $makespan$ é igual a $Pri_S(v)$. Portanto, o escalonamento deve continuar por pelo menos $Ordem_S(v) + Pri_S(v)$. □

Todos os vértices de uma mesma *Árvore-Inf-Conectada* devem ser processados em um mesmo processador. Além disso, todos os outros vértices podem ser processadas em qualquer outro processador de forma independente. Devido a esta independência, pode-se usar recursão para calcular o menor tempo de processamento das sub-árvores maximais conectadas por $arestasZero$ a esta mesma *Árvore-Inf-Conectada* . A única restrição é que cada sub-árvore só poderá iniciar seu processamento após o processamento do pai da sua raiz, que é um *Vértice-Conector*. O Algoritmo 1 descrito nesta seção escalonará de forma gulosa os vértices de uma mesma *Árvore-Inf-Conectada* , a partir da prioridade definida.

Lema 4.2.2. *O escalonamento encontrado pelo Algoritmo 1 é válido.*

Prova. Seja S o escalonamento encontrado pelo algoritmo 1 e também seja $e = (v, u) \in$ E. Se $C(e) = 0$, então, o laço mais interno da função $PreencherSlots$ (linha 15) garante que u e todos seus descendentes estão em um o $slot$ de tempo de processamento posterior

Algorithm 1 Escalonador Árvores Descendentes

$T = (V, E)$
C ▷ $C : E \to \{0, \infty\}$
$Ordenacoes := \{\}$ ▷ cade elemento deste conjunto é uma $ArvInfCon$
$Slot_S := \{\}$ ▷ $Slot_S : V \to \mathcal{N}$
$P_S := \{\}$ ▷ $P_S : V \to \mathcal{N}$
$p := 0$ ▷ Número de processadores usados
function $PreencherSlots(v, t)$ ▷ $Slot_S : V \to \mathcal{N}$
    $ordenacao := Ordenacoes[v]$ ▷ elementos de $ArvInfCon(v)$ ordenados
    $p_v := p$ ▷ processador destinado aos elementos de $ArvInfCon(v)$
    $p := p + 1$
    for $i := 0 \to |ordenacao| - 1$ **do**
        $w := ordenacao[i]$
        $Slot_S\ add\ (w, t + i)$
        $P_S\ add\ (w, p_v)$
        for u *filho de* w **do**
            if $C[(w, u)] == 0$ **then**
                $Slot_S\ add\ PreencherSlots(u, t + i + 1)$
            end if
        end for
    end for
end function
function $Visitar(v, Prioridade)$
    $Prioridade\ add\ \{(v, 0)\}$ ▷ $Prioridade : V \to \mathcal{N}$
    for u *filho de* v **do**
        if $C[(v, u)] == inf$ **then**
            $Visitar(u, Prioridade)$
            $Prioridade[v] := max(Prioridade[v], Prioridade[u])$
        else
            $Prioridade[v] := max(Prioridade[v], MenorTempoDeEscalonamento(u))$
        end if
    end for
end function
function MENORTEMPODEESCALONAMENTO(v)
    $Prioridade = \{\}$
    $Visitar(v, Prioridade)$
    $ordenacao$ = Sequência de vértices em $Prioridade$ ordenados por Prioridade conservando as posições relativas a ordem de inserção entre vértices de mesma prioridade.
    $t := |ordenacao|$
    $Ordenacoes[v] := ordenacao$
    for $i := 0 \to |ordenacao| - 1$ **do**
        $t := max(t, Prioridade[ordenacao[i]] + i + 1)$
    end for
    return t
end function
function ESCALONARARVORE($T = (V, E), C$)
    MenorTempoDeEscalonamento($raiz(T)$)
    $PreencherSlots(raiz(T), 0)$
    return $(Slot_S, P_S)$
end function

a v. Se $C(e) = \infty$, v e u são pertencentes a mesma *Árvore-Inf-Conectada* , são processadas no mesmo processador pela função $PreencherSlots$ (linha 14) e obedecem a ordem de processamento imposto na instrução de ordenação presente em EscalonarArvoreOtimolinha (linha 36), a qual pelo critério de desempate respeita a ordem se inserção dos elementos no conjunto ordenado, portanto, é uma ordem de processamento válida. □

Teorema 4.2.3. *Um escalonamento que, dentre todas as tarefas cujos ancestrais já foram escalonados, escalona aquelas de maior* Prioridade*, é ótimo.*

Prova. Seja S um escalonamento para uma árvore T_r, enraizada em um vértice r, tal que S é obtido escalonando iterativamente os vértices de maior $prioridade$ dentro de uma mesma *Árvore-Inf-Conectada* . Seja também t_S o seu $makespan$.

Se *ArvInfCon(r)* não possui nenhuma *Árvores-Inf-Conectada-Filhas*, todos os vértices tem de ser processados em um mesmo processador, e o escalonamento descrito é, claramente, ótimo.

Para quando *ArvInfCon(r)* possui alguma *Árvore-Inf-Conectada-Filha*, a prova é feita por indução, assumindo que todas as *Árvore-Inf-Conectada-Filhas* são escalonadas otimamente. Com isso, S é tal que todos os vértices v pertencente a *ArvInfCon(r)* possuem $Pri_S(v)$ o menor entre todos os valores possíveis para a prioridade de v. Além disso, a $Ordem$ dos vértices em *ArvInfCon(r)* é o conjunto $\{1, 2, ..., |$*ArvInfCon(r)*$|\}$. Ainda, pelo Lema 4.2.1, o escalonamento ótimo deve minimizar $\max\{Ordem_S(v) + Pri(v)\}, \forall v \in$ *ArvInfCon(r)*. Portanto, como não é possível minimizar os valores das prioridades dos vértices nem da ordem deles, resta ao escalonamento ótimo somar o maior de um conjunto com o menor de outro. Claramente, o escalonamento S faz isso.

□

Cololário 4.2.4. *O Algoritmo 1 encontra o escalonamento ótimo.*

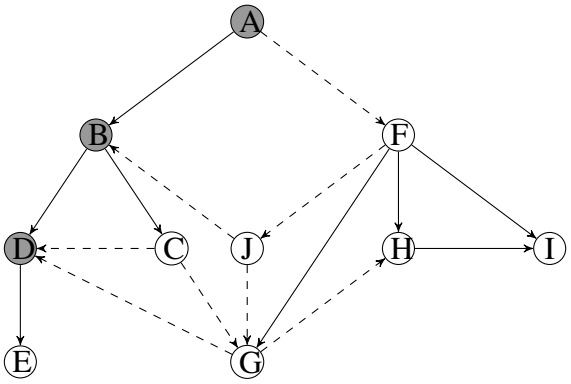
O Algoritmo 1 percorre a árvore agrupando os vértices de uma mesma *Árvore-Inf-Conectada* . Em cada raiz de uma *Árvore-Inf-Conectada* ocorre uma ordenação de seus elementos. Se a ordenação for efetuada por algoritmos usuais de ordenação, a complexidade total se dá por $\sum |A_i| \times \log |A_i|$, onde $|A_i|$ representa a quantidade de elementos presentes nesta árvore. Cada vértice pertece a somente uma destas árvores. Temos, portanto, que a quantidade de passos efetuados pelo algoritmo é $\sum |A_i| \times \log |A_i| \leq \sum |A_i| \times \log n \leq \log n \times \sum |A_i| = O(n \times \log n)$.

4.2.2 $P_\infty|prec; c_{i,j} \in \{0, \infty\}; dup, p_j = 1|C_{max}$

O problema de escalonar com qualquer precedência para um número qualquer de processadores é trivial. No entanto, quando atrasos de comunicações são considerados o problema se torna NP-completo, ainda que esses atrasos e o tempo de processamento das tarefas sejam unitários $P_\infty|prec, c = 1, p_j = 1|C_{max}$ (VELTMAN; LAGEWEG; LENSTRA, 1990). Quando atrasos de comunicação são considerados, pode ser mais vantajoso processar a mesma tarefa em diferentes processadores a fim de evitar a espera da chegada dos dados provenientes de outro processador. Considerando esta possibilidade e atrasos de comunicação constantes é conhecido algoritmo com complexidade exponencial no custo de comunicação ($O(n^{c+1})$) (JUNG; KIROUSIS; SPIRAKIS, 1989), quando este atraso não é limitado o problema, $P_\infty|prec, c, dup, p_j = 1|C_{max}$, é NP-completo (PAPADIMITRIOU; YANNAKAKIS, 1990).

Esta seção mostra que quando o atraso definido para cada par de tarefas é restrito a atrasos desprezíveis e atrasos muito longos ($c_{i,j} \in \{0, \infty\}$), o problema admite solução polinomial.

Figura 4.5: Esta figura representa um digrafo. As arestas de comunicação desprezíveis estão pontilhadas e as arestas de atrasos infinitamente grandes estão em linhas contínuas. Os vértices em cinza são os ancestrais inf conectados do vértice representado pela letra E.



Para este problema um escalonamento S associa para cada tarefa pelo menos um momento e um processador.

TAREFAS COM REPROCESSAMENTO
Entrada: $\langle G, C\rangle$, seja $G = (V, E)$ um grafo direcionado acíclico e $C : E \to \{0, \infty\}$ uma associação das arestas aos atrasos de comunicação para transmitir informações entre os vértices relacionados.
Saída: Encontrar um escalonamento válido que minimiza o processamento total.

Um escalonamento S é válido para o problema TAREFAS COM REPROCESSAMENTO, se S associa para cada $v \in V$um conjunto não vazio de pares ordenados de números naturais $(\mathbb{N}, \mathbb{N})$ e para toda tarefa escalonada existe pelo menos uma cópia de cada ancestral previamente escalonado. Além disso, se existe $e = (u, v) \in E|C(e) = \infty$, então, existe pelo menos uma cópia de u processada previamente em todos os processadores em que alguma cópia de v foi processada.

O Algoritmo 2 executa pelo menos uma cópia da tarefa-vértice presente em V em seu mínimo tempo possível. Com isso, o escalonamento visa não tardar o processamento das tarefa predecessoras. Além disso, precisa garantir que, para cada tarefa, as tarefas antecessoras conectadas por uma aresta $e \in arestasInf$ terão uma cópia processadas no mesmo processador. Novamente, o algoritmo apresentado faz uso de recursão usufruindo da natureza independente das cópias alocadas em processadores distintos.

Definição 4.2.8. *Seja S um escalonamento válido para o problema, P_S é o conjunto de processadores utilizados por S, sendo $P_S(v)$ um processador com uma cópia de v no $slot = mst_S(v)$.*

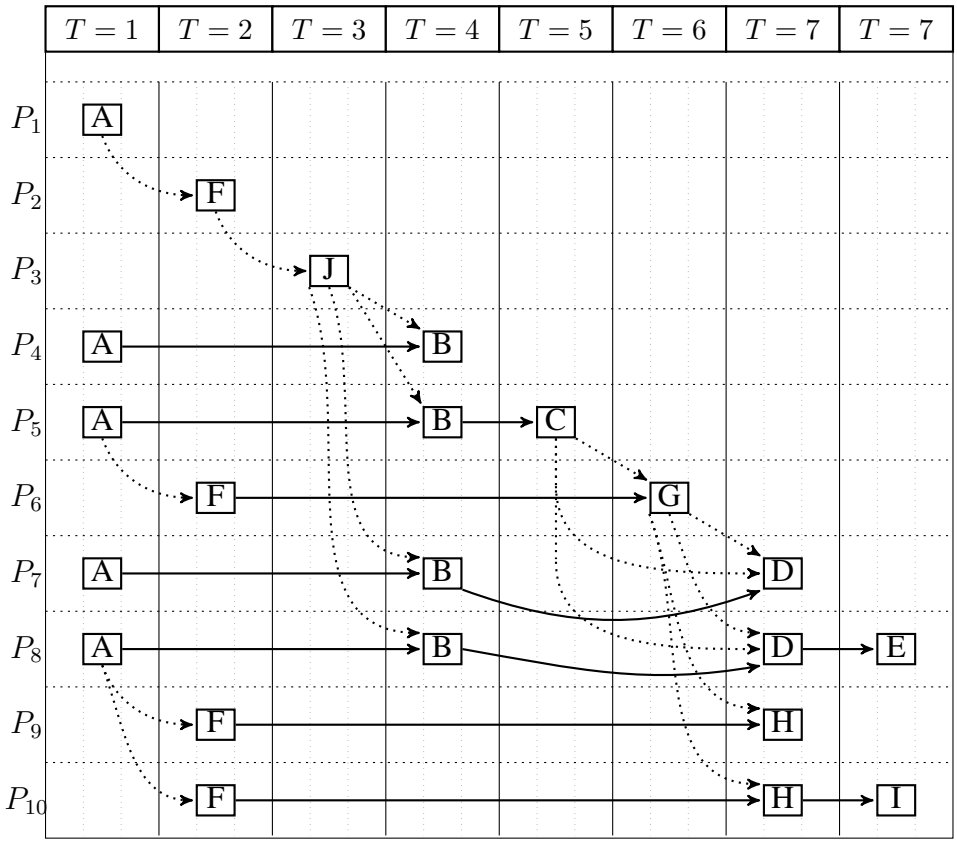
Definição 4.2.9. *O $mst_S(v)$ de uma tarefa v referente a um escalonamento S é o menor tempo de escalonamento para o qual existe uma cópia desta tarefa escalonada por S em algum processador.*

Definição 4.2.10. *Os $AncestInfConec(v)$ é um conjunto maximal de tarefas pertencentes aos ancestrais de v para os quais existe caminho à v somente por arestas de custo infinito.*

Algorithm 2 Escalonador Com Reprocessamento

$p := 0$ ▷ Índice do Processador Atual
$S := \{\}$ ▷ $S(v)$ é o conjunto de pares de $slot$ e processadores onde v é processada
function $Visitar(v)$
  $CompInfConect := \{(v, MenorTempo(v)\}$
  for $e := (u, v) \in E$ **do**
    if $C(e) = inf$ **then**
      $CompInfConect = CompInfConect \cup Visitar(u)$
    end if
  end for
end function
function $MenorTempo(v)$
  $menorTempo := 0$ ▷ $slot$ em a partir do qual todos os ancestral-zero de v possuem alguma cópia processada
  $AncestInfConect := \{\}$
  for $e := (u, v) \in E$ **do**
    if $C(e) = inf$ **then**
      $AncestInfConect := AncestInfConect \cup Visitar(u)$
    else
      $menorTempo := \max(menorTempo, MenorTempo(u))$
    end if
  end for
  $Ordenar(AncestInfConect)$ ▷ Ordena crescentemente pelo MenorTempo
  $SlotAtual := 0$ ▷ Indica o $Slot$ de tempo em p atualmente desocupado
  for $i := 0 \rightarrow |AncestInfConect| - 1$ **do**
    $SlotAtual := max(SlotAtual, AncestInfConect[i].MenorTempo)$
    $S(AncestInfConect[i].Vertice) := S(AncestInfConect[i].Vertice) \cup (SlotAtual, p)$
    $SlotAtual := SlotAtual + 1$
  end for
  $S(v) := S(v) \cup (max(SlotAtual, menorTempo), p)$
  $p := p + 1$
  return $max(SlotAtual, menorTempo)$
end function
function $Escalonar(G, C)$
  $makespan := 0$
  for $v \in V$ **do**
    $makespan := max(makespan, MenorTempo(v) + 1)$
  end for
  return S
end function

Figura 4.6: Escalonamento do digrafo da Figura 4.5. As comunicações de atrasos desprezíveis estão em pontilhado, enquanto as comunicações de atraso infinito estão em arestas contínuas. Nota-se que as arestas contínuas não podem ligar elementos entre processadores distintos.



Definição 4.2.11. *Os $AncestZeroConec(v)$ é um conjunto maximal de tarefas pertencentes aos ancestrais de v que estejam conectados a v por arestas de custo zero.*

Definição 4.2.12. *O $Vazios_S(p)$ é o conjunto de $slots$ de tempos referente a um escalonamento S para o qual não existe tarefa escalonada neste $slot$ para p.*

Teorema 4.2.5. *O escalonamento encontrado pelo Algoritmo 2 é válido.*

Prova. O laço da linha 23 garante que para cada $AncestInfConec(v)$ de v existirá uma cópia no mesmo processador p e processada num tempo anterior ao processamento de v em p atribuído na linha 28. Como todas as tarefas de $AncestInfConec(v)$ são alocadas em $slots$ de tempo igual ou posterior aos seus respectivos mst_S, então, existe pelo menos uma cópia para os respectivos $AncestZeroConec$ de cada $AncestInfConec(v)$ alocadas em *slots* anteriores em algum outro processador. Além disso, como o conjunto de todos os ancestrais de v contém o conjunto dos ancestrais de cada um de seus ancestrais, todos os $AncestInfConec(v)$ de cada um dos ancestrais de v também estão processados em p. Qualquer ancestral de um vértice possui mst_S menor que este vértice, assim a ordenação pelo mst_S garante que cada ancestral de v tem seus $AncestInfConec(v)$ alocados anteriormente no mesmo processador. Além disso, a linha 28 também garante que o tempo de processamento será posterior aos $AncestZeroConec$ e a função $Escalonar$ chama $MenorTempo$ para todos os vértices em V, portanto, todas as tarefas são processadas. □

Lema 4.2.6. *Seja S um escalonamento do algoritmo 2 para uma determinada instância. Seja P_v um processador utilizado por S para processar uma cópia de um vértice $v \in V$. Então para todo S' válido, todos os $slots$ de tempo deixados vazios em P_v por S, que sejam anteriores a esta cópia de v, também serão anteriores a qualquer cópia de v escalonado por S' em qualquer processador. Equivalentemente, $\forall slot_{vazio} \in Vazios_S(p), v \in V, (t,p) \in S(v)\ slot_{vazio} < t \Rightarrow slot_{vazio} < t_{S'}$.*

Prova. Seja v uma tarefa escalonada em P_v posteriormente a $slot_{vazio} \in Vazios_S(P_v)$. Como todas as tarefas escalonadas em um mesmo processador estão ordenadas pelo seu mst_S, se v não possuir antecessor também escalonado em P_v após este $slot_{vazio}$, v deveria ter sido escalonada neste mesmo $slot_{vazio}$ (laço da linha 23), exceto se $mst_S(u) \geq slot_{vazio}$ para algum $u \in ancest(v)$. Se existirem antecessores também escalonados após $slot_{vazio}$, pode-se argumentar o mesmo para todos esses antecessores. Isto também se verifica nas atribuições feita no laço da linha 23. Desta forma, se existir S' contra-exemplo do lema, seja v uma das tarefas de menor profundidade que contradiz o lema, então S', escalonou tal que $mst_{S'}(v) < mst_S(v)$, mas para isso teria de escalonar $u \in ancest(v)$ de forma que $mst_{S'}(u) < mst_S(u)$, mas v é a tarefa de menor profundidade dentre tais tarefas, portanto seu ancestral não pode ser uma dessas tarefas. □

Teorema 4.2.7. *O escalonamento encontrado pelo Algoritmo 2 encontra o escalonamento ótimo.*

Prova. Seja S' um escalonamento cujo tempo final é menor que o escalonamento S dado pelo algoritmo 2 para um instância $\langle G, C \rangle$. Seja v uma tarefa de maior profundidade em G, tal que, $mst(v)_{S'} < mst(v)_S$. A maneira em que a cópia de v é escalonada no processador $P_S(v)$ na sua respectiva execução em $MenorTempo(v)$ pelo 2 garante que ela será a última tarefa a ser processada nesse processador. Se no processador $P_S(v)$ o

$slot$ referente ao tempo $mst(v)_{S'}$ for vazio ou anterior a algum $slot$ vazio em $P_S(v)$, S' não é válido pelo lema 4.2.6.

Se tal $slot$ não existe, deve haver uma tarefa $AncestInfConec(v)$ processada em P_S no $slot$ de tempo igual a $mst(v)_{S'}$. Esta tarefa tem de ser processada em P_S, e anteriormente a v, mas pelo Lema 4.2.6 deve ser processada após o último $slot$ vazio em P_S, o que não é possível sem deixar de escalonar outro ancestral de v.

□

Assim como o Algoritmo 1, o Algortimo 2, apresentado nesta seção, ordena os elementos dos $AncestInfConectados$ de cada vértice. Contudo, os $AncestInfConectados$ de vértices distintos podem possuir uma interseção não vazia. Desta forma, temos que a complexidade é da ordem de $\sum |A_i| \times \log |A_i|$, onde A_i são os $AncestInfConectados$ do ***i-ésimo*** vértice. Assim, $\sum |A_i| \times \log n \leq \log n \times \sum |A_i| = O(n^2 \times \log n)$. De fato, esta quantidade de passos é atingida quando a interseção dos $AncestInfConectados$ é maximizada, isto ocorre quando o DAG é uma cadeia.

4.3 Resultados envolvendo Cadeias de Tarefas Tipadas

Como foi mencionado na primeira seção deste capítulo, o problema $Pm|prec; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$ está fortemente relacionado ao problema de escalonar sob restrições de processamento. Em particular, este problema contém o problema TAREFAS TIPADAS estudado por Jansen (JANSEN, 1993).

O problema considerado aqui está fortemente atrelado às instâncias do TAREFAS LIVRE COM COMUNICAÇÃO, geradas a partir da transformação utilizada no Teorema 4.1.2 de instâncias do problema TAREFAS TIPADAS. Tal transformação está exemplificada na Figura 4.3(a). O problema é precisamente definido a seguir.

CADEIAS TIPADAS
Entrada: $\langle Cadeias, Tipo, p, t\rangle$, onde $Cadeias$ é um conjunto de cadeias de tarefas, $Tipo$ é uma associação de cada tarefa presente em um das cadeias com algum "tipo", e p e t são inteiros.
Saída: Decidir se existe escalonamento válido, tal que, o maior $slot$ associado a alguma tarefa é menor ou igual a t.

Um escalonamento S para o problema mencionado associa um $natural$ representando um $slot$ para cada tarefa e um elemento de {1, 2, 3, ..., p}, que representa o processador no qual a tarefa foi processado. S é válido quando as seguintes condições são obedecidas. Primeiramente, o $slot$ associado à tarefa J_i deve ser menor que o $slot$ associado a qualquer J_j antecessor de J_i. Além disso, todas as tarefas tais que $Tipo(J_i) = Tipo(J_j)$, devem estar associadas ao mesmo processador. Por último, não pode haver duas tarefas associadas ao mesmo tempo e processador.

Desta forma, o problema está relacionado ao TAREFAS TIPADAS, mas permite uma a quantidade qualquer de processadores e "tipos". Além disso, não existe um processador predeterminado para cada tipo, mas as tarefas de mesmo tipo devem ser processadas no mesmo processador.

Os parâmetros que descrevem o problema analisado aqui estão limitados em diferentes formas nas seções que se seguem. Na primeira é visto que este problema é NP-completo, ainda que o conjunto $Cadeias$ seja restrito a apenas dois elementos. Na segunda e terceira seção, são apresentados algoritmos de programação dinâmica polinomiais quando a

quantidade de tipos são limitadas por constantes, sendo que na segunda seção a quantidade de elemento em $Cadeias$ também é limitada por constante e na terceira é o tamanho das cadeias que é limitado.

4.3.1 NP-Completude de Tarefas Tipadas com Duas Cadeias

Esta seção mostra que decidir se existe um escalonamento para o problema apresentado com tempo de processamento final menor que um determinado valor é NP-completo, mesmo quando apenas duas cadeias são permitidas. Isto pode ser verificado através da redução do problema da coloração de vértices a este problema. O problema da coloração é precisamente definido a seguir.

COLORACAO DE VERTICES
Entrada: $\langle G, k \rangle$, onde $G = (V, E)$ é um grafo e k um inteiro.
Saída: O problema consiste em decidir se é possível criar associação de cada vértice para algum elemento em {1, 2, 3, ..., k}, tal que, não exista $e = (u, v) \in E$ para qual u e v estejam associados aos mesmos elementos.

O problema considerado é definido como segue:

DUAS CADEIAS TIPADAS
Entrada: $\langle Cadeia_0, Cadeia_1, Tipo, p, t \rangle$, onde $Cadeia_0$ e $Cadeia_1$ são cadeias de tarefas, $Tipo$ é uma associação de cada tarefa presente em um das cadeias com algum "tipo", e p e t são inteiros.
Saída: Decidir se existe escalonamento válido, tal que o maior $slot$ associado a alguma tarefa é menor ou igual a t.

Teorema 4.3.1. *O problema DUAS CADEIAS TIPADAS é NP-completo.*

Prova. Dada uma instância $\langle G = (V, E), k \rangle$ qualquer do problema COLORACAO DE VERTICES, cria-se a seguinte instância para o problema DUAS CADEIAS TIPADAS: Cada vértice em V corresponderá a um $tipo$ possível na imagem de $Tipo$. Para cada $e = (u, v) \in E$ cria-se duas tarefas, uma tarefa com o $tipo$ u em uma das cadeias, e outra com $tipo$ v na outra cadeia de forma que ambas as tarefas tenham o mesmo índice (mesma quantidade de ancestrais) nas suas respectivas cadeias. Assim, são definidas as duas cadeias, $Cadeia_0$ e $Cadeia_1$ e a função $Tipo$ para cada tarefa. Por último, faz-se $p = k$ e $t = |E|$.

Primeiramente será mostrado que se existir escalonamento com tempo final t, então G é p-colorível, onde p é o número de processadores disponíveis. Para isso, cada processador será usado como rótulo de cor para colorir os vértices.

Todas as tarefas que possuem o $tipo$ v (portanto, referente a um vértice v) foram processadas no mesmo processador, seja tal processador P_v. Isto permite colorir o vértice v com a cor do processador P_v. Como o escalonamento possui tempo final t, que é o número de tarefas em cada cadeia, então, a i-ésima tarefa de cada cadeia foi processada no i-ésimo $slot$. Desta forma, dois vértices adjacentes em G não possuem a mesma cor, porque para cada aresta em E existem duas tarefas, uma em cada cadeia, que foram processados no mesmo instante. Isto garante que os processadores que processaram tais tarefas sejam diferentes, o que é o mesmo que dizer que a cor desses vértices são diferentes. Verifica-se também que, se existe uma k-coloração para os vértices de G, existe escalonamento com

tempo igual a t, pois, como $k = p$ é possível associar para cada processador uma cor e processar no i-ésimo slot de tempo o i-ésimo elemento de cada uma das cadeias. Cada uma dessas tarefas está associada a um processador diferente pois existe uma aresta em G ligando os vértices referentes a essas tarefas. □

4.3.2 Quantidade de Cadeias e de Tipos Limitadas Por Constantes

Esta seção mostrará que é possível escalonar tais cadeias se o número de cadeias e de tipos for limitados por uma constante. Para isso, será apresentado algoritmo polinomial usando técnicas de programação dinâmica.

Primeiramente, o algoritmo precisa definir em quais processadores irá processar as tarefas de um determinado tipo. Cada tipo possui p opções de processadores onde suas tarefas podem ser processadas, sendo p também menor que uma constante, já que é limitado pela quantidade de *tipo*, pois, em nada modifica adicionar mais processadores que tipos. Verifica-se que o número de possibilidades totais é limitado por p^{tipos}, sendo, portanto, uma constante. Para cada uma dessas possibilidades, executa-se uma programação dinâmica, cujo estado é o conjunto de tarefas as quais todos os ancestrais já foram escalonados, ou equivalentemente, um índice para a tarefa em cada cadeia que ainda não foi processada, mas todos seus antecessores já foram processados. A quantidade de estados é, portanto, o produto dos tamanho das cadeias, $\prod |c_i|$, que não pode ser maior que $n^{|Cadeias|}$. Desta forma, em um determinado estado, escolhe-se um subconjunto de cadeias para as quais as tarefas devem ser processados. Por último, verifica-se a possibilidade de se executar essas tarefas verificando se nenhuma dessas tarefas estão designadas a serem processadas no mesmo processador. A complexidade total é $O(p^{tipos} \times n^{|Cadeias|} \times p \times 2^{|Cadeias|})$, para as restrições dadas é $O(n^{|Cadeias|})$.

4.3.3 Tamanho de Cadeias e de Tipos Limitadas Por Constante

Quando, além dos tipos possíveis para cada tarefa, o tamanho da cadeia é limitado por uma constante, também é possível utilizar técnicas de programação dinâmica para se obter um algoritmo em tempo polinomial. O algoritmo apresentado aqui se utiliza de ideia semelhante a apresentada anteriormente, como também do fato de que, se houverem muitas cadeias, algumas delas devem coincidir em altura e tipos das tarefas na sequência apresentada. Em outras palavras, as variações possíveis para as cadeias é limitada.

Uma cadeia possui no máximo h_{max} tarefas, onde cada tarefa possui exatamente um dentre k tipos possíveis. Assim, as cadeias podem se distinguir entre si pela altura, que é menor ou igual a h_{max}, e pelos tipos escolhidos para cada tarefa da cadeia. Desta forma, são $\sum_{i=1}^{h_{max}} k^i$ variações distintas de cadeias, sendo isto $O(k^{h_{max}+1})$, e cujo valor é constante pelas restrições dadas, valor aqui denominado de k'.

A programação dinâmica apresentada aqui utiliza um vetor de k' dimensões, onde cada dimensão se refere uma das k' variações de cadeias e o valor do vetor nesta dimensão indica a quantidade de cadeias existentes de cada tipo de cadeia. A quantidade de estados é limitado polinomialmente em n, em particular é $O(n^{k'})$, pois não pode haver mais que n cadeias de um mesmo tipo.

Nota-se que as cadeias de uma determinada variação não podem ser processadas em um mesmo instante, pois as tarefas livres dessas cadeias possuem o mesmo tipo e devem ser processadas em um mesmo processador.

Novamente, assim como na ideia apresentada na seção anterior, é preciso definir em quais processadores cada tipo deve ser processado. Como ambos os valores são limitados

por constantes, pode-se tentar todas as possibilidades.

Assim, com a informação de quantas cadeias existem para cada variação de cadeia, é possível escolher para todo $i \leq p$ um subconjunto dos tipos de cadeias para serem processadas e verificar se nenhum par dentre essas tarefas estão designadas a processar no mesmo processador. Tal operação custa $\sum_{i=1}^{min(p,k')} i \times \binom{k'}{i} \leq min(p,k') \times \sum_{i=1}^{min(p,k')} \binom{k'}{i} \leq min(p,k') \times 2^{k'}$. Finalmente, se recorre a programação dinâmica com os ajustes necessários quanto à quantidade de cada tipo de cadeia, reduzindo-se a quantidade de tarefas total a cada estado. A complexidade total da programação dinâmica é $O(n^{k'} \times 2^{k'})$.

5 CONCLUSÃO

Nesta dissertação de mestrado, foram obtidos e apresentados resultados relativos ao problema de escalonar tarefas de tempos de processamento unitários em máquinas paralelas, objetivando minimizar o tempo em que a última tarefa a ser processada termina seu processamento (minimização do *makespan*), isto quando atrasos de comunicação são determinados para cada par de tarefas precedentes e restritos a valores extremos, 0 ou ∞.

Dois problemas foram demonstrados serem NP-completos. No primeiro deles, a quantidade de processadores disponível é indicada na instância do problema. Este resultado é válido ainda que as relações de precedência sejam restrita a um conjunto de cadeias. No segundo problema, a ordem de precedência é arbitrária. Este resultado foi demostrado ser válido ainda que uma quantidade fixa de apenas dois processadores esteja disponível.

Dois outros problemas foram demostrados serem solúveis em tempo polinomial. Ambos consideram uma quantidade ilimitada de processadores disponível. No primeiro deles a relação de precedência é restrita para uma árvore descendente. No segundo, é permitido o reprocessamento de tarefas.

Foram também apresentados, resultados para o problema de escalonar tarefas particionadas em conjuntos, cujas tarefas necessitam ser processadas no mesmo processador. Este problema foi demonstrado ser NP-completo se a quantidade de processadores disponíveis é determinada a cada instância do problema. Mais especificamente, este resultado foi demonstrado ser válido ainda que a precedência seja restrita a duas cadeias. Ainda, quando a quantidade de partições está limitada por uma constante, bem como a quantidade de cadeias ou a quantidade de tarefas por cadeia, também é limitado, o problema admite solução em tempo polinomial.

Como trabalhos futuros, os estudos realizados deixam em aberto se é possível encontrar escalonamento polinomial sob tais restrições, envolvendo uma quantidade fixa de processadores e uma relação de precedência restrita, por exemplo, por uma cadeia $P_m|chain; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$. Tais restrições são equivalentes as restrições do problema $P_m|chain; pmtn^*(delay = \infty)|C_{max}$, onde $pmtn^*$ indica a possibilidade de interromper o processamento de tarefas na transição dos intervalos unitários de tempo e $delay = \infty$ indica que o atraso de migração da tarefa é infinitamente grande. A similaridade com o problema $P_2|chain;|C_{max}$ é notória, já que neste problema migrações também não são permitidas. Assim, a distinção entre os problema se dá pela possibilidade ou não de interrupção das tarefas. O fato de $P_2|chain;|C_{max}$ ser fortemente NP-difícil (DU; LEUNG; YOUNG, 1991) pode indicar que $P_m|chain; c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$ também seja.

REFERÊNCIAS

AFRATI, F. et al. Scheduling trees with large communication delays on two identical processors. **Journal of Scheduling**, [S.l.: s.n.], v.8, n.2, p.179–190, Apr. 2005.

AHMAD, I.; KWOK, Y.-k. On exploiting task duplication in parallel program scheduling. **IEEE Transactions on Parallel and Distributed Systems**, [S.l.: s.n.], v.9, n.9, p.872–892, 1998.

ALI, H.; ELREWINI, H. An optimal algorithm for scheduling interval ordered tasks with communication on N processors. **Journal of Computer and System Sciences**, [S.l.: s.n.], v.51, n.2, p.301–306, Oct. 1995.

AMORIM, R. X. d. et al. A hybrid genetic algorithm with local search approach for E/T scheduling problems on identical parallel machines. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, GECCO 2013, New York, New York, USA. **Proceedings...** ACM Press, 2013. p.1–2.

AMORIM, R. X. d. et al. Single and multi-start methods based on local search and path-relinking technique for earliness-tardiness parallel machine scheduling problems. **Expert Systems with Applications**, [S.l.: s.n.], v.1, 2013.

ANGER, F. D.; HWANG, J.-J.; CHOW, Y.-C. Scheduling with sufficient loosely coupled processors. **Journal of Parallel and Distributed Computing**, [S.l.: s.n.], v.9, n.1, p.87–92, May 1990.

BAMPIS, E.; GIANNAKOS, A.; KöNIG, J.-C. On the complexity of scheduling with large communication delays. **European Journal of Operational Research**, [S.l.: s.n.], v.94, n.2, p.252–260, Oct. 1996.

BAMPIS, E.; GIROUDEAU, R.; KöNIG, J.-C. An approximation algorithm for the precedence constrained scheduling problem with hierarchical communications. **Theoretical Computer Science**, [S.l.: s.n.], v.290, n.3, p.1883–1895, Jan. 2003.

BARKER-PLUMMER, D. Turing Machines. In: ZALTA, E. N. (Ed.). **The Stanford Encyclopedia of Philosophy**. Summer 201.ed. [S.l.: s.n.], 2013.

BLAZEWICZ, J. et al. **Handbook of Scheduling - From Theory to Applications**. [S.l.]: Chapman and Hall/CRC, 2007.

BONDY, A.; MURTY, U. S. R. **Graph theory**. New York, London: Springer, 2008. (Graduate texts in mathematics, v.244).

BORKAR, S.; CHIEN, A. A. The future of microprocessors. **ACM Communications**, [S.l.: s.n.], v.54, n.5, p.67, May 2011.

BRUCKER, P. **Scheduling Algorithms**. [S.l.]: Springer, 2007. 1–52p. v.93, n.1.

CAO, J. et al. A taxonomy of application scheduling tools for high performance cluster computing. **Cluster Computing**, [S.l.: s.n.], v.9, n.3, p.355–371, July 2006.

CHAUDHURI, S.; WALKER, R.; MITCHELL, J. Analyzing and exploiting the structure of the constraints in the ILP approach to the scheduling problem. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, [S.l.: s.n.], v.2, n.4, p.456–471, Dec. 1994.

CHRETIENNE, P. A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints. **European Journal of Operational Research**, [S.l.: s.n.], v.43, n.2, p.225–230, Nov. 1989.

CHRETIENNE, P. Tree scheduling with communication delays. **Discrete Applied Mathematics**, [S.l.: s.n.], v.49, n.1-3, p.129–141, Mar. 1994.

COFFMAN, E.; GRAHAM, R. Optimal scheduling for two-processor systems. **Acta Informatica**, [S.l.: s.n.], 1972.

COLIN, J. Y.; CHRETIENNE, P. Scheduling with small communication delays and task duplication. **Operations Research**, [S.l.: s.n.], v.39, n.4, p.680–684, July 1991.

COOK, S. A. The complexity of theorem-proving procedures. In: ACM SYMPOSIUM ON THEORY OF COMPUTING - STOC '71, New York, New York, USA. **Proceedings...** ACM Press, 1971. p.151–158.

COPELAND, B. J. The Church-Turing Thesis. In: ZALTA, E. N. (Ed.). **The Stanford Encyclopedia of Philosophy**. Fall 2008.ed. [S.l.: s.n.], 2008.

DARBHA, S.; AGRAWAL, D. Optimal scheduling algorithm for distributed-memory machines. **IEEE Transactions on Parallel and Distributed Systems**, [S.l.: s.n.], v.9, n.1, p.87–95, 1998.

DIAS, M. F.; WEBER, L. Introdução. In: **Teoria da Recursão**. [S.l.]: UNESP, 2010. p.15–23.

DIESTEL, R. **Graph Theory**. 2o ed..ed. Berlin: Springer, 2000.

DINECHIN, B. D. de. From machine scheduling to VLIW instruction scheduling. **ST Journal of Research**, [S.l.: s.n.], v.1, n.1.2, p.1–35, 2004.

DINECHIN, B. D. de. Scheduling monotone interval orders on typed task systems. **PlanSIG 2007**, [S.l.: s.n.], p.25, 2007.

DOURADO, M. C.; RODRIGUES, R. F.; SZWARCFITER, J. L. Scheduling unit time jobs with integer release dates to minimize the weighted number of tardy jobs. **Annals of Operations Research**, [S.l.: s.n.], v.169, p.81–91, 2009.

DOURADO, M. C.; RODRIGUES, R. F.; SZWARCFITER, J. L. Scheduling problems with multi-purpose parallel machines. **Discrete Applied Mathematics (to appear)**, [S.l.: s.n.], 2011.

DROZDOWSKI, M. Classic scheduling theory. In: **Scheduling for Parallel Processing**. London: Springer London, 2009. p.55–86. (Computer Communications and Networks).

DROZDOWSKI, M. Scheduling with communication delays. In: **Scheduling for Parallel Processing**. London: Springer London, 2009. p.209–299. (Computer Communications and Networks).

DU, J.; LEUNG, J. Y.-T.; YOUNG, G. H. Scheduling chain-structured tasks to minimize makespan and mean flow time. **Information and Computation**, [S.l.: s.n.], v.92, n.2, p.219–236, June 1991.

ENGELS, D. W. et al. Parallel processor scheduling with delay constraints. In: ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS. **Proceedings...** Society for Industrial and Applied Mathematics, 2001. p.577–585.

FUJII, M.; KASAMI, T.; NINOMIYA, K. Optimal sequencing of two equivalent processors. **SIAM Journal on Applied Mathematics**, [S.l.: s.n.], v.17, n.4, p.784–789, July 1969.

FUJII, M.; KASAMI, T.; NINOMIYA, K. Erratum "Optimal sequencing of two equivalent processors". **SIAM J. Appl. Math**, [S.l.: s.n.], v.20, n.1, p.141, 1971.

GABOW, H. N. An almost-Linear algorithm for two-processor scheduling. **Journal of the ACM**, [S.l.: s.n.], v.29, n.3, p.766–780, July 1982.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: a guide to the theory of np-completeness**. [S.l.]: Freeman New York, 1979.

GIROUDEAU, R. et al. Complexity and approximation for precedence constrained scheduling problems with large communication delays. **Theoretical Computer Science**, [S.l.: s.n.], v.401, n.1-3, p.107–119, July 2008.

GIROUDEAU, R.; KöNIG, J. General scheduling non-approximability results in presence of hierarchical communications. **European Journal of Operational Research**, [S.l.: s.n.], v.184, n.2, p.441–457, Jan. 2008.

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

GUINAND, F.; RAPINE, C.; TRYSTRAM, D. Worst case analysis of Lawler's algorithm for scheduling trees with communication delays. **IEEE Transactions on Parallel and Distributed Systems**, [S.l.: s.n.], v.8, n.10, p.1085–1086, 1997.

HANEN, C.; MUNIER, a. An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. **Discrete Applied Mathematics**, [S.l.: s.n.], v.108, n.3, p.239–257, Mar. 2001.

HOOGEVEEN, J.; LENSTRA, J.; VELTMAN, B. Three, four, five, six, or the complexity of scheduling with communication delays. **Operations Research Letters**, [S.l.: s.n.], v.16, n.3, p.129–137, Oct. 1994.

HU, T. C. Parallel sequencing and assembly line problems. **Operations Research**, [S.l.: s.n.], v.9, n.6, p.841–848, 1961.

IMMERMAN, N. Computability and Complexity. In: ZALTA, E. N. (Ed.). **The Stanford Encyclopedia of Philosophy**. Fall 2011.ed. [S.l.: s.n.], 2011.

JANSEN, K. Scheduling with constrained processor allocation for interval orders. **Computers & Operations Research**, [S.l.: s.n.], v.20, n.6, p.587–595, 1993.

JANSEN, K. Analysis of scheduling problems with typed task systems. **Discrete Applied Mathematics**, [S.l.: s.n.], v.52, n.3, p.223–232, Aug. 1994.

JOHNSON, D. S. The NP-completeness column. **ACM Transactions on Algorithms**, [S.l.: s.n.], v.2, n.3, p.473–489, July 2006.

JUNG, H.; KIROUSIS, L.; SPIRAKIS, P. Lower bounds and efficient algorithms for multiprocessor scheduling of dags with communication delays. In: FIRST ANNUAL ACM SYMPOSIUM ON PARALLEL ALGORITHMS AND ARCHITECTURES - SPAA '89, New York, New York, USA. **Proceedings...** ACM Press, 1989. p.254–264.

KARP, R. Reducibility Among Combinatorial Problems. In: YEARS OF INTEGER PROGRAMMING 1958-2008, 50. **Anais...** [S.l.: s.n.], 1972. p.85–103.

LEUNG, J.; LI, C. Scheduling with processing set restrictions: a survey. **International Journal of Production Economics**, [S.l.: s.n.], v.116, p.251–262, 2008.

LEUNG, J. Y.-T. **Handbook of Scheduling - Algorithms, Models and Performance Analysis**. [S.l.]: Chapman and Hall/CRC, 2004.

PADOIN, E.; NAVAUX, P. HPC vs HPC High Performance Computing vs High Power Consumption. In: WORKSHOP DE PROCESSAMENTO PARALELO E DISTRIBUÍDO (WSPPD 2011). **Anais...** [S.l.: s.n.], 2011. n.November.

PAPADIMITRIOU, C. H.; YANNAKAKIS, M. Scheduling interval-ordered tasks. **SIAM Journal on Computing**, [S.l.: s.n.], v.8, n.3, p.405–409, Aug. 1979.

PAPADIMITRIOU, C. H.; YANNAKAKIS, M. Towards an architecture-independent analysis of parallel algorithms. **SIAM Journal on Computing**, [S.l.: s.n.], v.19, n.2, p.322–328, Apr. 1990.

PESSOA, A. et al. Exact Algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. **Mathematical Programming Computation**, [S.l.: s.n.], v.2, p.259–290, 2010.

PICOULEAU, C. New complexity results on scheduling with small communication delays. **Discrete Applied Mathematics**, [S.l.: s.n.], v.60, n.94, p.331–342, 1995.

PINEDO, M. L. **Scheduling - Theory, Algorithms, and Systems**. [S.l.: s.n.], 2012. 1–104p. v.4a ed.

PIRES, R. F. et al. On parallel machine scheduling with special communication delays. In: **Workshops on Job Scheduling Strategies for Parallel Processing (submitted)**. [S.l.: s.n.], 2014. (Lecture Notes in Computer Science).

R. Giroudeau; KOENIG, J. Scheduling with communication delay. In: LEVNER, E. (Ed.). **Multiprocessor Scheduling, Theory and Applications**. [S.l.: s.n.], 2007. n.December, p.63–84.

RAYWARD-SMITH, V. UET scheduling with unit interprocessor communication delays. **Discrete Applied Mathematics**, [S.l.: s.n.], v.18, n.1, p.55–71, Sept. 1987.

RODRIGUES, R. et al. Heuristic algorithm for the parallel machine total weighted tardiness scheduling problem. **Relatórios de pesquisa em engenharia de produção**, [S.l.: s.n.], v.8, p.1–12, 2008.

RODRIGUES, R. F. **Caracterizações e Algoritmos para Problemas Clássicos de Escalonamento**. 2009. Tese (Doutorado em Ciência da Computação) — UFRJ, Rio de Janeiro.

ROTHKOPF, M. H. Scheduling independent tasks on parallel processors. **Management Science**, [S.l.: s.n.], v.12, n.5, p.437–447, Jan. 1966.

SEVASTYANOV, S. V.; SITTERS, R. a.; FISHKIN, a. V. Preemptive scheduling of independent jobs on identical parallel machines subject to migration delays. **Automation and Remote Control**, [S.l.: s.n.], v.71, n.10, p.2093–2101, Oct. 2010.

STADTHERR, H. The two processor scheduling problem. In: **Work Efficient Parallel Scheduling Algorithms**. [S.l.: s.n.], 1998. p.88.

ULLMAN, J. NP-complete scheduling problems. **Journal of Computer and System Sciences**, [S.l.: s.n.], v.10, n.3, p.384–393, June 1975.

VARVARIGOU, T. et al. Scheduling in and out forests in the presence of communication delays. **IEEE Transactions on Parallel and Distributed Systems**, [S.l.: s.n.], v.7, n.10, p.1065–1074, 1996.

VELTMAN, B.; LAGEWEG, B.; LENSTRA, J. Multiprocessor scheduling with communication delays. **Parallel Computing**, [S.l.: s.n.], v.16, n.2-3, p.173–182, Dec. 1990.