UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UM ESTUDO SOBRE FORMULAÇÕES MATEMÁTICAS E ESTRATÉGIAS ALGORÍTMICAS PARA PROBLEMAS DE ESCALONAMENTO EM MÁQUINAS PARALELAS COM PENALIDADES DE ANTECIPAÇÃO E ATRASO

RAINER XAVIER DE AMORIM

Março de 2013

Manaus - AM

RAINER XAVIER DE AMORIM

UM ESTUDO SOBRE FORMULAÇÕES MATEMÁTICAS E ESTRATÉGIAS ALGORÍTMICAS PARA PROBLEMAS DE ESCALONAMENTO EM MÁQUINAS PARALELAS COM PENALIDADES DE ANTECIPAÇÃO E ATRASO

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Informáticado Instituto de Computação da Universidade Federal do Amazonas (PPGI/IComp, UFAM) como parte dos requisitos necessários à obtenção do título de Mestre em Informática.

Orientadora: Rosiane de Freitas Rodrigues, D.Sc.

Março de 2013

Manaus - AM

PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UFAM

FOLHA DE APROVAÇÃO

Um Estudo sobre Formulações Matemáticas e Estratégias Algorítmicas para Problemas de Escalonamento em Máquinas Paralelas com Penalidades de Antecipação e Atraso

RAINER XAVIER DE AMORIM

Dissertação defendida e aprovada pela banca examinadora constituída pelos Professores:

Rosiane de Freitas Rodrigues
PROFA. ROSIANE DE FREITAS RODRIGUES – PRESIDENTE

PROF. EDUARDO UCHOA BARBOZA – MEMBRO

Fabíola Guerra Nakamura
PROFA. FABIOLA GUERRA NAKAMURA – MEMBRO

Arilo Claudio Dias Neto
PROF. ARILO CLÁUDIO DIAS NETO – MEMBRO

PROF. JOSÉ REGINALDO HUGHES CARVALHO – MEMBRO

Manaus, 27 de março de 2013.

A Deus,
aos professores,
aos colegas de curso e
aos meus familiares.

Agradecimentos

Agradeço primeiramente a Deus, hoje e sempre, pois tenho certeza que Ele me permitiu mais essa benção e esteve sempre ao meu lado em todos os momentos.

A minha orientadora, Rosiane de Freitas Rodrigues, pela oportunidade de enriquecimento profissional e pessoal, dedicação, lições e pela competência com que sempre orientou esta pesquisa.

Aos meus amigos, em especial ao amigo Bruno Dias, que me ajudaram diretamente e indiretamente neste trabalho e aos professores e pesquisadores com quem tive a oportunidade de estudar e que deram valiosas sugestões que contribuíram para o enriquecimento deste trabalho.

Aos meus pais, Mazionete Amorim e Célio Amorim, a minha irmã Kássia Amorim e a minha namorada e amiga Lídia Mayara Almeida pelo apoio e constante estímulo.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo suporte financeiro.

Resumo da Dissertação apresentada ao PPGI/IComp/UFAM como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática.

UM ESTUDO SOBRE FORMULAÇÕES MATEMÁTICAS E ESTRATÉGIAS ALGORÍTMICAS PARA PROBLEMAS DE ESCALONAMENTO EM MÁQUINAS PARALELAS COM PENALIDADES DE ANTECIPAÇÃO E ATRASO

Rainer Xavier de Amorim

Março/2013

Orientadora: Profa. Rosiane de Freitas Rodrigues, D.Sc.

Esta dissertação apresenta um estudo sobre problemas de escalonamento com penalidades de antecipação e atraso em máquinas paralelas, considerando tarefas independentes, ponderadas e de tempos de execução arbitrários. Uma análise sobre as principais formulações matemáticas em programação inteira é dada, bem como apresentados os principais resultados da literatura. Uma formulação matemática de programação inteira baseada no modelo de fluxo em redes também foi proposta para o problema, que pode ser aplicada em ambientes mono e multiprocessado sem tempo ocioso. Métodos de enumeração implícita foram estudados e aplicados aos problemas em questão através do resolvedor de programação linear inteira CPLEX e da biblioteca UFFLP, principalmente, estratégias algorítmicas aproximadas de otimização global baseadas em heurísticas de busca local e técnica de reconexão de caminhos foram desenvolvidas. Os experimentos computacionais mostram que as estratégias propostas são competitivas em relação aos resultados existentes na literatura para ambientes de escalonamento monoprocessados, envolvendo instâncias baseadas no *benchmark* da *OR-Library* para 40, 50, 100, 150, 200 e 300 tarefas, onde todos os ótimos foram encontrados, e, principalmente, sendo a melhor estratégia apresentada para ambientes multiprocessados, envolvendo 2, 4 e 10 máquinas paralelas idênticas.

Palavras-chave: algoritmos, otimização combinatória, penalidades de antecipação e atraso, programação linear inteira, teoria de escalonamento.

Abstract of Dissertation presented to PPGI/IComp/UFAM as a partial fulfillment of the requirements for the degree of Master in Informatics.

A STUDY OF MATHEMATICAL FORMULATIONS AND ALGORITHMIC STRATEGIES FOR SCHEDULING PROBLEMS ON PARALLEL MACHINES WITH EARLINESS AND TARDINESS PENALTIES

Rainer Xavier de Amorim

March/2013

Advisor: Profa. Rosiane de Freitas Rodrigues, D.Sc.

This dissertation presents a study on scheduling problems with earliness and tardiness penalties on identical parallel machines, considering independent and weighted jobs with arbitrary processing times. An analysis of the major mathematical formulations in integer programming is given, and presented the main results from the literature. An integer mathematical formulation based on network flow model was also proposed for the problem, which can be applied on single and parallel machines without idle time. Exact methods of implicit enumeration were studied and applied for the problem through the integer linear programming solver CPLEX and the UFFLP library and, mainly, algorithmic strategies of global optimization based on local search heuristic and path-relinking technique were developed. The computational experiments shows that the proposed algorithmic strategies are competitive in relation to existing results from the literature for single-machine scheduling, involving instances based on OR-Library benchmark for 40, 50, 100, 150, 200 and 300 jobs, where all the optimal values were found, and, mainly, being the best algorithmic strategy for multiprocessor environments, involving 2, 4 and 10 identical parallel machines.

Keywords: algorithms, combinatorial optimization, earliness and tardiness penalties, integer linear programming, scheduling theory.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Os problemas de escalonamento em geral vêm sendo estudados desde a década de 1950 [dFR09], [Bru06], quando o ferramental matemático pôde ser usado para modelar problemas reais complexos com o auxílio do computador e questões econômicas motivaram cada vez mais o seu estudo em diversas áreas do conhecimento. Por isso, existe na literatura um vasto material sobre o assunto com inúmeras classes de complexidade com diferentes características, condições de execução, restrições e ambientes de processamento. Apesar do grande interesse de diversas áreas na investigação destes problemas, ainda existem muitos problemas em aberto, com pequenas diferenças em relação aos problemas já bem resolvidos ou já provados serem NP-Difíceis. Além disso, constantemente surgem novos problemas, conceitos, formas de modelagem, técnicas de resolução e aplicação, o que enriquece ainda mais a literatura e desperta cada vez mais o interesse em buscar novas e melhores soluções algorítmicas e novos modelos matemáticos cada vez mais eficientes para os problemas de escalonamento do mundo real [dFR09].

Dentre os problemas de escalonamento existentes, os que consideram datas de término na execução de tarefas vêm sendo estudados desde a década de 1970 e constituem uma das principais vertentes de pesquisa sobre a classe geral de problemas de escalonamento [EC77], [GS99], [SKS08], [Józ07], [Pin12], [Tan12b], tanto pela grande representatividade de situações reais, quanto pela maior complexidade teórica envolvida. Especialmente, o estudo de problemas de escalonamento que consideram simultaneamente penalidades de antecipação e atraso na execução de tarefas foi motivado pela adoção nas indústrias de processos produtivos com o conceito de produção sem folga (nem antes nem depois da data de término estipulada).

Esta ideia de produção sem folga ficou conhecida como *Just-in-Time* (JIT), tal termo surgiu no Japão na década de 70, na indústria automotiva, onde se buscava um sistema em que fosse possível coordenar uma determinada produção com a sua demanda específica e com o menor tempo de atraso possível. Este sistema tem como objetivo a melhoria do processo produtivo e fluxo de manufatura, eliminação de estoques e desperdícios. A aplicação deste sistema é muito ampla, envolvendo a produção de produtos perecíveis por exemplo, mas abrangendo qualquer sistema em que as tarefas devem ser finalizadas o mais próximo possível da data de término estipulada [LA04].

Muitas vezes é bastante difícil atingir esses requisitos de produção, logo, a redução de custos de estoque e atraso na produção acabam se tornando grandes desafios. Se a demanda dos consumidores é conhecida no início do planejamento, é natural definir as datas de término dos produtos obedecendo a demanda de cada consumidor. Assim, a partir do cronograma principal de produção, onde as datas de término sugeridas dos consumidores estão definidas, as informações das demandas são propagadas para os níveis restantes de produção. O objetivo é estipular as datas de término de cada etapa de produção e controle de matérias-primas, procurando assim atender ao cronograma principal de produção [Józ07].

Naturalmente, completar uma produção após a data de entrega não é algo desejável. No caso de produtos finalizados, o atraso não lida apenas com a insatisfação do cliente mas também pode resultar na perda de clientes. Em geral, os custos incorridos pela conclusão tardia de um pedido incluem penalizações devido a quebra de cláusulas contratuais, custos de expedição de tarefas que estão com atraso e custos de atualização do cronograma de produção [Józ07].

Por outro lado, quando uma produção é concluída antes da data de entrega, essa produção deve ser armazenada, o que pode gerar custos indesejáveis de estoque. Os custos de inventário dependem da quantidade de inventário, o valor do item e o tempo em que o inventário é realizado. Os custos de estoque incluem o custo de capital comprometido com o inventário, o custo dos seguros contratados em estoques, custo de obsolência ou limitações de vida útil do produto e os custos operacionais envolvidos no armazenamento do inventário. O custo de capital é geralmente expresso como uma taxa de juros com base no custo de obtenção de empréstimos bancários para financiar o investimento em estoque. Os custos operacionais incluem o custo do aluguel de espaço de armazenamento público,

ou o custo de possuir e operar instalações de armazenamento (tais como aquecimento, luz e trabalho) [Józ07].

O objetivo deste trabalho consiste em elaborar formulações matemáticas e desenvolver estratégias algorítmicas para problemas de escalonamento de tarefas com penalidades de antecipação e atraso. Os objetivos específicos são:

- Adquirir um conhecimento aprofundado na área de otimização combinatória em geral, com ênfase em teoria de escalonamento, programação matemática e métodos aproximados.
- Analisar formulações matemáticas para problemas clássicos de escalonamento com penalidades de antecipação e atraso.
- Desenvolver algoritmos exatos e aproximados para tal problema.
- Realizar experimentos computacionais envolvendo instâncias de testes disponíveis na literatura e comparar os resultados com trabalhos relacionados.
- Registrar e divulgar os resultados da pesquisa em artigos e conferências de relevância.

Este trabalho justifica-se pela oportunidade de lidar com problemas de escalonamento de tarefas envolvendo a minimização das penalidades de antecipação e atraso através de um enfoque teórico-computacional. Sendo que, muitos problemas reais podem ser modelados como problemas de escalonamento e a literatura oferece várias ferramentas e técnicas de resolução para tais problemas.

Nas organizações em geral, é importante que estratégias de alocação de recursos sejam bem definidas, como postos e máquinas da linha de produção de uma fábrica, organização do quadro de funcionários de uma empresa, fluxo de matérias-primas, de modo a minimizar os encargos de atraso ou antecipação nas atividades a fim de minimizar custos em geral e maximizar o lucro e a satisfação dos clientes.

A classe de problemas de escalonamento contém problemas em diversas classes de complexidade. Dentre elas, estão os problemas de escalonamento que estão na classe P, NP-Difícil e os problemas que estão em aberto. Tem-se então uma rica área de pesquisa em Otimização Combinatória e Complexidade Computacional, que além de possuir aplicações práticas diversas, possui uma enorme importância teórica, o que desperta o

interesse em buscar algoritmos e formulações matemáticos mais eficientes para estes problemas.

Com relação a organização, esta dissertação encontra-se dividida em 05 (cinco) capítulos, que são:

- Neste capítulo é apresentada a introdução desta dissertação, juntamente com o objetivo geral e específicos, justificativa e organização da dissertação.
- No Capítulo 2 é apresentado o referencial teórico necessário para o entendimento da pesquisa realizada, onde é introduzida a área de otimização combinatória juntamente com algumas abordagens de resolução, bem como as definições e notações amplamente utilizadas nos problemas clássicos de escalonamento.
- No Capítulo 3, os problemas de escalonamento de tarefas com penalidades de antecipação e atraso são apresentados, bem como os trabalhos correlatos e as suas estratégias de resolução.
- No Capítulo 4, os métodos propostos neste trabalho são detalhados, envolvendo abordagens exatas e aproximadas.
- No Capítulo 5, os resultados computacionais e uma análise empírica realizada podem ser consultados.
- No Capítulo 6, são expostas as conclusões do trabalho e diretivas para trabalhos futuros.

Capítulo 2

Fundamentos teóricos

Este capítulo apresenta a fundamentação teórica necessária para estudo na área de otimização combinatória, com ênfase nos métodos de resolução de problemas, bem como o estudo da teoria de escalonamento.

Dessa forma, são apresentados os conceitos e os principais problemas de otimização combinatória, incluindo seus principais métodos de resolução (exatos e aproximados). A teoria de escalonamento, que se propõe a conceituar, classificar e apresentar modelos teóricos, notações e políticas de escalonamento, é também resumida neste capítulo. As seções seguintes abordam os tópicos citados acima.

2.1 Otimização combinatória

Otimização Combinatória é uma área de pesquisa que trata um tipo especial de problema de otimização matemática cujo conjunto de soluções factíveis pode ser representado por um espaço de busca discreto. Onde se deseja encontrar, dentre todas as possíveis soluções, aquela solução (ou soluções) cujo custo maximize ou minimize uma determinada função objetivo (ou múltiplos objetivos), respeitando um determinado conjunto de restrições [MR11] [dFR96]. Dependendo da complexidade do problema, uma forma de solucionar seria simplesmente enumerar todas as soluções possíveis e retornar a melhor delas. Entretanto, para problemas com um espaço de busca exponencial, este método torna-se impraticável, o que desperta cada vez mais a atenção de pesquisadores da área em propor modelos teóricos mais robustos e métodos mais eficientes, a fim de obter melhores resultados em tempo de execução adequado.

Os problemas de Otimização Combinatória surgem nos mais diversos contextos e, certamente, em todas as áreas tecnológicas, de logística e de gestão industrial [Law01]. Um grande número de problemas de otimização combinatória ocorre em projetos de sistemas de distribuição e alocação de recursos, projetos de redes de computadores, roteamento de veículos, escalonamento de tarefas em máquinas, empacotamento de caixas em contêineres, cortes em placas ou madeira, sequenciamento de genes e DNA, modelagem molecular, localização e alocação de canais em redes de sensores sem fio, dentre outros. Estes problemas exigem, cada vez mais, novas abordagens e métodos, envolvendo novas ideias teóricas, matemático-computacionais, para serem solucionados de maneira eficiente [MM81].

Estes problemas podem ser classificados quanto a sua complexidade computacional, o que permite a separação dos problemas fáceis dos difíceis. Todos os problemas de otimização combinatória possuem uma versão de decisão, onde suas soluções consistem em uma resposta positiva (sim) ou negativa (não) para um determinado questionamento quanto a existência de alguma solução. A classificação de tais problemas é feita através da **Teoria da NP-Completude**, onde o problema é tratado como um conjunto de parâmetros (definição das instâncias) e um conjunto do propriedades (restrições do problema). Para as instâncias de um mesmo problema, as únicas variações estão nos conjuntos de parâmetros. A teoria da complexidade considera o tamanho das instâncias como sendo a quantidade de *bits* necessária para representá-las em computadores digitais [GJ79].

De um modo geral, os problemas de interesse neste trabalho envolvem problemas de Otimização Combinatória conhecidos como NP-Difíceis, que são aqueles para os quais não se conhece nenhum algoritmo que retorne a solução ótima em tempo polinomial e sim, apenas algoritmos de tempo de execução exponencial, o que é considerado intratável do ponto de vista computacional, dado que para grandes instâncias pode se tornar completamente inviável obter todas as soluções possíveis ou gerar a melhor solução para o problema.

Nas próximas subseções, serão detalhados os métodos de resolução de problemas de Otimização Combinatória, sendo divididos em exatos (Subseção 2.1.1) e aproximados (Subseção 2.1.2). Métodos exatos são todos aqueles que devolvem a melhor solução possível - solução ótima - para o problema que está sendo resolvido [dFR96]. Métodos aproximados são aqueles que, de modo geral, devolvem uma solução válida para o pro-

blema sem a garantia de que seja a melhor solução possível e, neste caso, podem ser determinísticos (maioria das heurísticas simples) ou estocásticos (meta-heurísticas e outros métodos que fazem uso de aleatoriedade e processos estocásticos) [dFR96]. Dentre os métodos aproximados, existem também os chamados algoritmos aproximativos, que são aqueles que devolvem uma solução aproximada cuja razão de aproximação da solução ótima é conhecida. Esta seção baseia-se nas seguintes referências: Rodrigues [dFR96], Rodrigues [dFR09], Talbi [Tal09] e Luke [Luk09].

2.1.1 Métodos exatos

Os métodos exatos possuem a característica de fornecerem a solução ótima para um problema de Otimização Combinatória, dada a especificação do problema expressa através de um modelo teórico. Geram sempre a melhor solução possível, mas, se o problema for NP-difícil, não garantem a resolução em um tempo computacional adequado, sendo de tempo exponencial no geral. O método mais intuitivo seria um procedimento de **enumeração explícita** (de **busca exaustiva** ou, popularmente chamado de **força bruta**), que consiste em enumerar explicitamente todas as possibilidades de solução do problema e, então, escolher a melhor solução encontrada [dFR96].

No entanto, a busca exaustiva é inviável computacionalmente para os problemas NP-difíceis de grande porte. Desta forma, na tentativa de tornar o processo algorítmico menos custoso, métodos de enumeração implícita são utilizados, consistindo em usar modelos teóricos mais robustos e estratégias algorítmicas mais sofisticadas para uma resolução mais rápida do problema, gerando-se a solução ótima, mas sem ter o custo de se gerar todas as soluções explicitamente. Dentre tais métodos de enumeração implícita, os mais clássicos e aplicados são: programação dinâmica, *branch-and-bound*, métodos de corte e de geração de colunas, *branch-and-cut*, *branch-and-price* e combinações entre os mesmos [dFR96].

O método exato desenvolvido no trabalho foi um *branch-and-cut*, através da ferramenta CPLEX. Assim, nas seções seguintes tal método é brevemente descrito e, para um melhor entendimento do mesmo, os métodos de *branch-and-bound* e de cortes no plano em geral, são brevemente descritos.

2.1.1.1 *Branch-and-bound*

O método *branch-and-bound* também adota uma estratégia de **enumeração implícita** para a resolução de problemas de Otimização Combinatória. A ideia básica consiste em um particionamento inteligente do espaço de busca (que pode ser representado por uma árvore), descartando partes que não poderão fornecer a solução ótima para o problema [PS82]. A divisão do problema corresponde ao processo de enumerar as possibilidades válidas ou simplesmente ramificação (do inglês, *branching*), combinada com o processo de verificação do valor da função objetivo que se baseia em limites inferiores e superiores para o problema (do inglês, *bounding*), e que deste modo, pode desconsiderar a ramificação por subárvores que matematicamente não produzirão valores melhores do que os já obtidos.

Dessa forma, seja um problema de minimização P. Os subproblemas de P são os subconjuntos S' do conjunto S de soluções viáveis de P. Sendo assim, a partir do problema original P, pode-se estabelecer um limite inferior para o problema desconsiderando-se a restrição de integralidade e, então, o problema relaxado (relaxação linear) é resolvido com um método de solução de programação linear contínua, como o Simplex. Se a solução obtida for inteira (todas as variáveis de decisão forem inteiras na solução do Simplex), então, considera-se o problema resolvido. Caso contrário, consideram-se os seguintes passos [Bru06]:

- **Ramificação (*branching:*)** S é substituído por problemas menores $S_i (i = 1, ..., n)$. Este processo é chamado de **ramificação** (*branching*), que trata-se de um processo recursivo, ou seja, S_i é a base de outra ramificação. Todo o processo é representado por uma **árvore de ramificações** (*branching tree*), onde S é a raiz da árvore de ramificação e $S_i (i = 1, ..., n)$ são os filhos de S. Os ramos da árvore recebem restrições incorporadas pelo processo de ramificação, sobre uma variável de decisão x_i (ramificações com restrições $x_i \leq n$ e $x_i \geq n + 1$, por exemplo), esses ramos são chamados de **subproblemas** (Figura 2.1);

- **Definição dos limites (*boundings*):** no processo de ramificação pode-se chegar nas seguintes situações nos subproblemas:

    1. **A restrição adicionada torna o problema inviável:** assim não há mais ramificação a partir deste nó (o nó é **podado**);

2. **A solução do nó é inteira:** não haverá mais ramificação a partir deste nó e a solução deste nó será a melhor solução inteira atual, onde, a partir daí, é analisado se esta for a primeira melhor solução inteira, ela passará a ser um **limite superior** do problema original de minimização (ou inferior se for de maximização), ou seja, se as próximas soluções encontradas forem maiores que esta, não haverá ramificação (também serão podados). Se já houver um limite superior e se a nova solução obtida for menor que o limite superior armazenado, a nova solução passará a ser o novo limite superior do problema de minimização *P*.

- Quando não for mais possível ramificar, o último limite superior armazenado é a solução ótima do problema de programação inteira.

O método *branch-and-bound* termina quando todos os subproblemas tiverem sido resolvidos. A escolha de bons algoritmos para o cálculo dos limites inferiores e superiores é crucial, justamente para que se possa eliminar uma grande parte do conjunto de soluções factíveis sem precisar enumerá-las [dFR96].

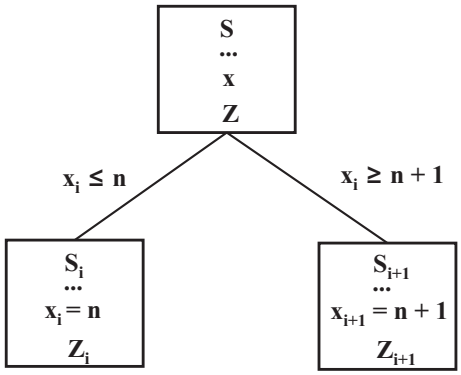


Figura 2.1: Ramificação da árvore de *branch-and-bound* para um conjunto de soluções *S*.

2.1.1.2 Planos de corte

Um plano de corte define uma restrição a mais para o problema, eliminando assim pontos extremos contínuos, sem remover nenhum ponto inteiro factível. Algoritmos de cortes no plano (*cutting plane algorithms*) iniciam seu processamento através de uma relaxação linear do problema e a partir daí, novas desigualdades (cortes no plano) vão sendo adicionadas ao problema relaxado. O objetivo é colocar a solução ótima do problema de

programação inteira como um vértice do politopo, de forma que um método de solução, como o Simplex, possa encontrar e retornar a solução.

A partir de uma solução contínua da relaxação do problema, pode-se utilizar esses dados para criar uma nova restrição que corta pontos contínuos do politopo. Este processo de adicionar cortes que não excluam pontos inteiros viáveis do problema é ilustrado na Figura 2.2 [PS82]. A Figura 2.2 (a) apresenta o problema de programação inteira e a solução ótima contínua x^*. Uma restrição linear, corte no plano (ou simplesmente corte), é adicionado na Figura 2.2 (b), modificando assim o conjunto de soluções viáveis do problema (sem a perda de pontos inteiros) e o novo ótimo contínuo é movido de lugar. A Figura 2.2 (c) apresenta o resultado após a adição de outro plano de corte, o efeito agora é fazer o ótimo contínuo virar uma solução inteira para o problema.

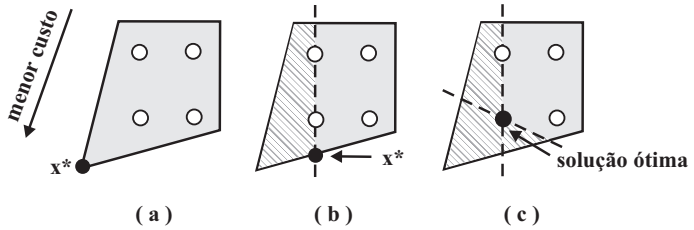


Figura 2.2: Ilustração do algoritmo de cortes no plano: (a) solução ótima contínua x^*. (b) nova solução ótima contínua x^* após um corte. (c) solução do problema de programação inteira após dois cortes [PS82].

A inequação adicionada no problema relaxado é chamada de **Cortes**. A aplicação repetida destes cortes no problema relaxado garante chegar no ponto ótimo do problema, conforme mostrado na Figura 2.2.

2.1.1.3 *Branch-and-cut*

O método *branch-and-cut* (B&C) é um método exato de programação linear inteira, proposto por Padberg e Rinaldi [PR87] e aplicado inicialmente para solucionar o problema do caixeiro viajante. Este algoritmo combina os métodos *branch-and-bound* e planos de corte. Apesar de ambos serem capazes de solucionar um problema de programação linear inteira, eles levam um certo tempo para encontrar uma solução ótima. Dessa forma, através da combinação dos dois métodos é possível diminuir esse tempo. A cada nó da árvore, busca-se o máximo de cortes que podem ser inseridos em um tempo razoável, reduzindo assim o limite superior do nó e, consequentemente, reduzindo também o tamanho da árvore de *branch-and-bound*, onde os cortes podem ser controlados por algum critério, tais

como:

- Realizar *n* cortes de Gomory a cada nó;
- Adicionar inequações específicas do problema original;
- Efetuar cortes a cada nível da árvore que seja múltiplo de algum valor.

No *branch-and-cut* também consideram-se os cortes nos estágios de enumeração da árvore de pesquisa. Assim, através desta abordagem, é possível que se guarde uma restrição necessária para um nó de uma determinada sub-árvore, de modo elas sejam válidas para os nós de outras sub-árvores da árvore de pesquisa. O procedimento é iniciado com a relaxação do problema de programação inteira, então o problema relaxado é resolvido e se a solução violar as restrições de integralidade do problema original, o corte é aplicado (adiciona-se mais uma restrição). O objetivo é achar a solução ótima do problema sem que todas as restrições do problema original sejam adicionadas no problema relaxado.

Também existem técnicas de pré-processamento, onde tenta-se melhorar a relaxação linear através de considerações algébricas, onde variáveis e restrições são checadas para evitar redundância, alguns coeficientes de restrições são ajustados e algumas variáveis são fixadas a certos valores, sem que isto afete o valor da solução ótima do problema. Assim, é possível reduzir o número de variáveis significativamente, tornando o problema mais fácil de se resolver. Métodos heurísticos podem ser aplicados a fim de que se obtenha boas soluções, em um curto espaço de tempo, e assim reduzir o tamanho da árvore de busca do *branch-and-cut* [dFR96].

2.1.2 Métodos aproximados

Os métodos aproximados para resolução de problemas de otimização combinatória, ao contrário dos métodos exatos, não garantem a obtenção de uma solução ótima para o problema, mas tentam gerar uma boa solução para o problema (ou até mesmo ser a ótima) podendo ter um tempo computacional relativamente menor do que os métodos exatos, em alguns casos. Esta característica, de gerar boas soluções viáveis rapidamente, e sem precisar manipular modelos teóricos complicados, faz com que tais métodos sejam muito utilizados na resolução de problemas de Otimização Combinatória, principalmente em grandes aplicações reais.

Nas próximas subseções são abordados os principais métodos heurísticos e meta-heurísticos amplamente utilizados na literatura para resolução problemas de Otimização Combinatória.

2.1.2.1 Heurísticas

Os métodos heurísticos são utilizados para se encontrar boas soluções para problemas difíceis em um razoável espaço de tempo, mas existem outras razões para se utilizar os métodos heurísticos, dentre eles destacam-se [MR11]:

- Quando nenhum método de resolução do problema em sua otimalidade é conhecido;
- Quando, embora exista algum método exato para solução do problema, tal método não pode ser utilizado no ambiente computacional disponível para processamento;
- Quando a flexibilidade do método heurístico for fundamental, permitindo, por exemplo, a incorporação de condições que são difíceis de modelar;
- Quando o método heurístico for usado como parte de um procedimento global que garanta achar a solução ótima de um problema.

Uma boa heurística deve possuir as seguintes propriedades:

- Gerar uma solução com um esforço computacional razoável;
- Gerar uma solução próxima da solução ótima do problema (com alta probabilidade);
- Gerar uma solução muito ruim (longe do ótimo) com baixa probabilidade.

Os métodos heurísticos podem ser divididos em sete grandes grupos, que são [dFR96]:

- **Heurísticas de construção:** onde se constroem uma solução completa (válida para o problema), a partir de uma solução parcial (incompleta ou nula) através da inserção sucessiva de seus componentes individuais. Geralmente são métodos determinísticos que tendem a se basear na melhor escolha a cada iteração;

- **Heurísticas de melhoria:** essa heurística inicia seu processamento com uma solução válida e vai melhorando essa solução através de sucessivas trocas de elementos. Estas heurísticas também são conhecidas como heurísticas de **busca local**, onde é realizada uma busca exaustiva na vizinhança da solução inicial até que seja encontrado a melhor solução dessa vizinhança;

- **Heurísticas de decomposição:** o problema original é quebrado em subproblemas simples de se resolver, levando-se em consideração que, de um modo geral, esses subproblemas pertencem a classe de um mesmo problema;

- **Heurísticas de Particionamento:** o problema é dividido em vários problemas menores e, então, as soluções são unidas posteriormente;

- **Heurísticas de relaxação:** o espaço do problema é relaxado com a finalidade de se ter um algoritmo mais rápido e, a solução obtida é então ajustada de maneira que ela se torne factível para o problema original. Relaxar significa retirar algumas restrições do problema;

- **Heurísticas baseadas em formulações matemáticas:** alteram um algoritmo baseado na formulação matemática do problema, objetivando um melhor desempenho no tempo de processamento.

2.1.2.2 Meta-heurísticas

Meta-heurísticas são métodos aproximados de resolução de problemas de Otimização Combinatória que fazem uso de mais de um processo heurístico na busca por soluções. São aplicadas em problemas complexos, mal-definidos ou com grande dificuldade computacional, em geral problemas NP-difíceis. Diferentemente das heurísticas, tais métodos conseguem explorar um grande espaço de busca, através de saltos aleatórios no mesmo. As meta-heurísticas servem para três propósitos principais: solucionar problemas de modo rápido, resolver problemas de grande porte, e conter outras heurísticas em seus procedimentos internos. Além disso, eles são simples de criar e implementar, e são muito flexíveis [Tal09]. As meta-heurísticas também fazem uso da aleatoriedade ou pesquisa estocástica, ao contrário das heurísticas determinísticas. Esta seção baseia-se nas seguintes referências: Rodrigues [dFR96], Talbi [Tal09] e Luke [Luk09].

2.1.2.2.1 Busca local iterada

A busca local começa com uma solução inicial. A cada iteração, a heurística substitui a solução corrente por uma solução vizinha cuja função objetivo (FO) é melhor que a solução corrente. A busca termina quando todas soluções vizinhas são piores que a solução corrente, as soluções vizinhas da solução corrente fazem parte de um subconjunto do espaço de soluções. Na Figura 2.3 é apresentado um gráfico com uma curva representando as soluções e uma estratégia de seleção (seleção da solução vizinha).

A busca local pode ser vista como uma busca de melhores soluções em um grafo que representa o espaço de busca. Esse grafo pode ser definido como $G = (S, M)$, onde S representa um conjunto de todas as soluções factíveis do espaço de busca e M representa a relação de vizinhança. No grafo G, uma aresta (i, j) será conectada a qualquer solução vizinha s_i e s_j. Para uma dada solução s, o número de arestas associadas será $|N(s)|$ (número de soluções vizinhas).

Figura 2.3: Funcionamento do algoritmo de busca local em um determinado espaço de busca.

Seleção da melhor solução vizinha

Muitas estratégias podem ser aplicadas na seleção da melhor solução vizinha (Figura 2.4):

- **Melhor encontrado (*best improvement*):** Nesta estratégia, a melhor solução vizinha (solução que possui o melhor valor de função objetivo) é selecionada. A vizinhança de soluções é avaliada de uma forma totalmente determinística. Por isso, a exploração das soluções vizinhas é exaustiva, todos os possíveis movimentos são utilizados para que seja encontrada a melhor solução. Este tipo de exploração pode consumir bastante tempo para um espaço muito grande de soluções vizinhas;

- **Primeira melhora (*first improvement*):** Esta estratégia consiste em escolher a primeira solução vizinha que é melhor que a solução corrente. Então, a solução corrente é imediatamente substituída por essa solução melhorada. Esta estratégia envolve uma avaliação parcial das soluções vizinhas. Em uma exploração cíclica, as soluções vizinhas são exploradas de modo determinístico seguindo uma dada ordem de geração de soluções. No pior caso (quando nenhuma melhoria é encontrada), uma avaliação completa das soluções vizinhas é realizada;

- **Seleção aleatória (*random selection*):** Nesta estratégia, uma seleção aleatória é aplicada nas soluções vizinhas a fim de melhorar a solução corrente.

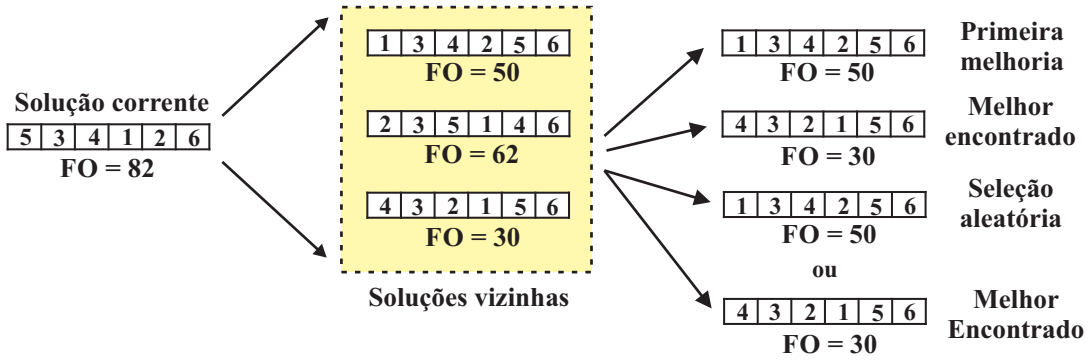


Figura 2.4: Funcionamento do algoritmo de busca local em um determinado espaço de busca.

A qualidade do ótimo local obtido pelo algoritmo de busca local depende de uma solução inicial. Como se pode gerar um ótimo local com tanta variabilidade, a busca local iterada pode ser utilizada para melhorar a qualidade dos sucessivos ótimos locais.

A busca local iterada é baseada no seguinte princípio: Primeiramente, uma busca local é aplicada em uma solução inicial. Assim, a cada iteração, é realizada uma perturbação do ótimo local obtido e, finalmente, uma busca local é aplicada na solução perturbada. A solução gerada é aceita como a nova solução corrente sob determinadas condições. Este processo é repetido durante um determinado número de iterações. O Algoritmo 1 descreve o algoritmo de busca local.

Três elementos básicos compõem uma busca local iterada (Figura 2.5):

- **Busca local:** qualquer meta-heurística (determinística ou estocástica) pode ser utilizada em conjunto com a busca local iterada;

Algoritmo 1: Busca local iterada (*Iterated local search*).

$s_* \longleftarrow BuscaLocal(s_0)$ ▷ Aplica um dado algoritmo de busca local;
repita
  $s' \longleftarrow Perturba(s_*)$ ▷ Perturba a solução corrente;
  $s'_* \longleftarrow BuscaLocal(s')$ ▷ Aplica a busca local na solução perturbada;
  $s_* \longleftarrow AceitaSolução(s_* , s'_*)$ ▷ Critério de aceitação;
até *Critério de parada satisfeito*;
retorna *Melhor solução encontrada.*

- **Método de perturbação:** a perturbação pode ser vista como um movimento aleatório da solução corrente. A perturbação deve manter alguma parte da solução original e fortemente perturbar a outra parte da solução para que seja possível explorar outras regiões do espaço de busca;

- **Critério de aceitação:** define as condições que o novo ótimo local deve possuir para que ele possa substituir a solução corrente.

Figura 2.5: Funcionamento do algoritmo de busca local iterada em um determinado espaço de busca.

2.1.2.2.2 Algoritmo genético

Os algoritmos genéticos (GAs) foram originalmente propostos por Holland [Hol75]. Esses algoritmos utilizam o processo de evolução baseado na teoria da evolução de Darwin para resolver problemas, nessa teoria, os indivíduos mais adaptados sobrevivem. Nesse contexto, o termo **evolução** representa uma solução para o problema (formado por uma cadeia de genes); O termo **gene** representa uma certa característica da solução representada no cromossomo; O termo **alelo** representa o valor de gene (usualmente 0 ou 1); A

função de avaliação (*fitness*) representa o valor associado a um cromossomo para representar quão boa é a solução; A **população** é o conjunto de possíveis soluções (cromossomos); A **geração** é a operação para se gerar uma nova população.

Existem muitas implementações de algoritmos genéticos para uma variedade de problemas, os componentes principais de algoritmos genéticos são os seguintes:

- **Solução codificada:** uma representação de cromossomo para as soluções;
- **População inicial:** criação de uma população inicial de cromossomos;
- **Função de avaliação:** medida de avaliação baseada na função objetivo;
- **Seleção:** seleção natural de alguns cromossomos (pais) na população para geração de novos membros (filhos) na população;
- **Operações genéticas:** operadores genéticos aplicados aos cromossomos cuja regra é criar novos membros (filhos) na população pelo cruzamento de genes de dois cromossomos (operações de cruzamento) ou pela modificação de genes de um cromossomo (operações de mutação);
- **Substituição:** seleção natural dos membros da população que irão sobreviver;
- **Parâmetro de seleção:** convergência natural de toda população que é globalmente melhorada a cada passo do algoritmo.

O desempenho do GA depende muito da concepção dos componentes supracitados e da escolha dos parâmetros como o tamanho da população, probabilidades de operações genéticas (taxa de cruzamento e taxa de mutação), e número de gerações.

Métodos de seleção

O mecanismo de seleção é um dos componentes principais dos algoritmos genéticos. O princípio do método de seleção é "quão melhor é um indivíduo, maior será a sua chance de ser um pai". Através de métodos de seleção, os cromossomos (pais) são selecionados da população para serem combinados e formarem novos cromossomos (filhos), a fim de que sejam aplicadas novas operações genéticas. Dentre os métodos de seleção existentes na literatura, os principais são:

- **Seleção por roleta (*Roulette wheel selection*):** é o método mais comum de seleção, onde é designado a cada indivíduo uma seleção probabilística que é proporcional ao seu valor de avaliação. Seja f_i o valor de avaliação baseado na função objetivo de um indivíduo p_i na população P. A sua probabilidade de ser selecionado é $p_i = f_i/(\sum_{j=1}^{n} f_i)$. E seja um gráfico de pizza onde cada indivíduo é posicionado no gráfico de acordo com o seu valor de avaliação. A seleção é realizada através da agulha de uma roleta que é posicionada em volta desse gráfico de pizza. A seleção de μ indivíduos é realizada através de μ giros independentes na seleção por roleta, como pode ser observado na Figura 2.6 (a);

- **Amostragem universal estocástica (*Stochastic universal sampling*):** esse tipo de seleção é semelhante a seleção por roleta, mas para selecionar μ indivíduos utiliza-se μ agulhas igualmente posicionadas. Nesta estratégia, em um único giro da roleta são selecionados μ indivíduos para reprodução, como pode ser observado na Figura 2.6 (b);

- **Seleção por torneio (*Tournament selection*):** é um método que consiste em sucessivas disputas para realizar a seleção. Primeiramente seleciona-se randomicamente k indivíduos, onde o parâmetro k é o tamanho do grupo do torneio. Um torneio é então aplicado nestes k membros do grupo a fim de selecionar o melhor dentre eles. Para selecionar μ indivíduos, o procedimento de torneio é realizado μ vezes.

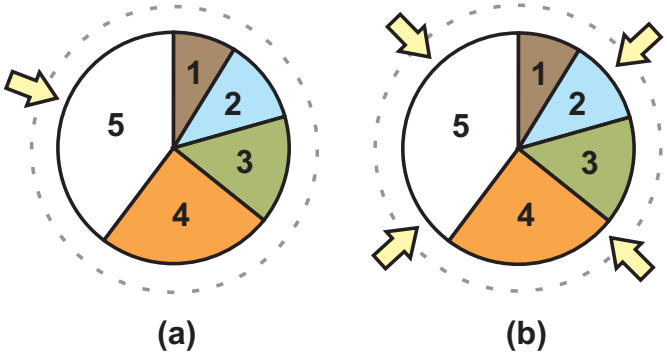


Figura 2.6: (a) seleção por roleta e (b) seleção por amostragem universal estocástica. Adaptado de Talbi [Tal09].

Reprodução

Uma vez que a seleção dos indivíduos é executada, a fase da reprodução conta com a aplicação dos operadores de reprodução, que são:

- **Mutação:** é aplicada em um único indivíduo, a mutação representa pequenas mudanças em indivíduos previamente selecionados na população. A probabilidade p_m define a probabilidade de mutação de cada elemento (gene) da solução. Geralmente pequenos valores são recomendados para essa probabilidade ($p_m \in [0.001, 0.01]$);
- **Recombinação e cruzamento:** recombina dois ou mais cromossomos para gerar novas soluções.

Elitismo

O elitismo consiste em guardar as melhores soluções encontradas durante a busca de soluções. Onde uma população secundária pode ser utilizada para armazenar estas soluções de alta qualidade. O elitismo tem sido utilizado para prevenir a perda de soluções ótimas encontradas durante um processo de busca. Dessa forma, a população secundária não exerce nenhuma influência nas operações de busca da população corrente.

Estratégias de substituição

A fase de substituição diz respeito à seleção dos sobreviventes tanto dos pais como das soluções filhas. Uma vez que o tamanho da população é constante, os indivíduos devem ser retirados de acordo com uma determinada estratégia de seleção, que são:

- **Substituição de gerações (*Generational replacement*):** abrange toda a população de tamanho μ. A população descendente irá substituir sistematicamente a população original;
- **Substituição de regime permanente (*Steady-state replacement*):** A cada geração do algoritmo, somente um descendente é gerado. E, a partir daí, o pior indivíduo da população original é substituído pelo descendente gerado.

2.1.2.2.3 Reconexão de caminhos (*path relinking*)

A técnica de reconexão de caminhos foi originalmente proposta por Glover et al. [GLM00] para a diversificação do algoritmo de busca dispersa. Esta técnica permite explorar caminhos que ligam duas soluções elite encontradas pela busca dispersa, entretanto, esta estratégia pode ser generalizada e aplicada a outras meta-heurísticas para gerar boas soluções. A ideia principal desta técnica é de gerar e explorar uma trajetória no espaço de busca conectando uma solução inicial s e uma solução guiada ou de destino s'. O caminho entre duas soluções no espaço de busca (vizinhança) geralmente resultam em soluções que compartilham atributos comuns das soluções iniciais. O Algoritmo 2 apresenta o esquema de funcionamento do algoritmo de reconexão de caminhos, onde, a cada iteração, o melhor movimento em termos de função objetivo é escolhido e há uma diminuição da distancia d entre duas soluções. Este procedimento é repetido até que a distancia entre as duas soluções seja 0. A melhor solução encontrada na trajetória é retornada pelo algoritmo.

Algoritmo 2: Reconexão de caminhos (*path relinking*).

Entrada: solução inicial s e solução guiada s'
$x \longleftarrow s$;
enquanto $dist(x,s') \neq 0$ **faça**
  Encontre o melhor movimento m que diminua a $dist(x \oplus m, s')$;
  $x \longleftarrow x \oplus m$ ▷ Aplica o movimento m na solução x;
fim enquanto
retorna *Melhor solução encontrada na trajetória entre s e* s'

Os principais questionamentos do algoritmo de reconexão de caminhos são [Tal09]:

- **Seleção do caminho:** o que deve ser considerado na geração do caminho. Uma heurística pode ser utilizada para gerar o caminho que deve ser utilizado para minimizar a distância até a solução guiada. Para isso, uma distância d deve ser definida no espaço de busca associado ao problema. A complexidade computacional para esse procedimento deve ser levada em consideração;

- **Operações intermediárias:** que operações devem ser aplicadas a cada passo da geração do caminho. Algumas soluções selecionadas no caminho podem ser consideradas. Por exemplo, para cada solução intermediária no caminho, um procedimento de busca local pode ser aplicado a fim de melhorar a qualidade da solução.

Além disso, para cada par de soluções de s e s', existem várias alternativas diferentes para seleção da solução inicial e solução guiada (Figura 2.7):

- **Caminho avante (*forward*):** a pior solução entre s e s' é utilizada como solução inicial.

- **Caminho reverso (*backward*):** a melhor solução entre s e s' é utilizada como solução inicial. Como a vizinhança da solução inicial é mais explorada do que a vizinhança da solução guiada, a estratégia de caminho reverso é em geral melhor do que a estratégia do caminho adiantado.

- **Reconexão avante-reverso (*back and forward relinking*):** dois caminhos são construídos em paralelo, utilizando alternativamente a solução s como solução inicial e guiada. Apesar desta estratégia apresentar boas soluções, este método adiciona uma sobrecarga no tempo de computação.

- **Reconexão mista (*mixed relinking*):** assim como na estratégia de reconexão avante-reverso, dois caminhos são construídos em paralelo partindo de s até s', mas agora, a solução guiada é uma solução intermediária m, que está localizada a uma mesma distância de s e s'. Esta estratégia paralela pode reduzir o tempo de execução.

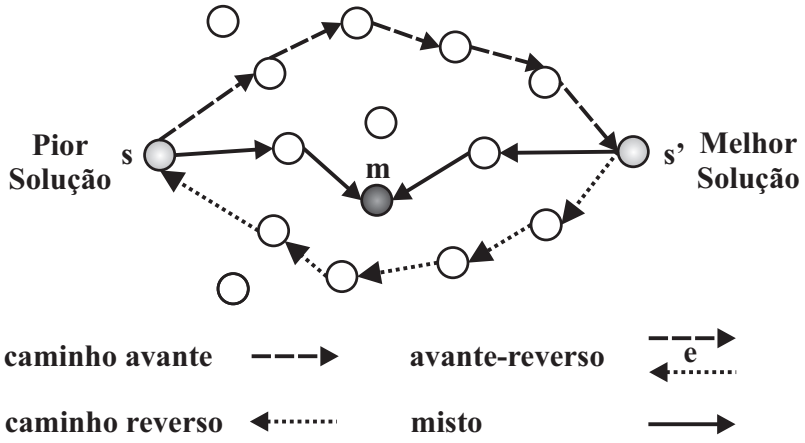


Figura 2.7: Diferentes estratégias de reconexão de caminhos em termos das soluções inicial e guiada. Adaptado de Talbi [Tal09].

2.2 Teoria de escalonamento

De acordo com Rodrigues [dFR09], problemas de escalonamento podem ser definidos como sendo a designação ou alocação de determinados recursos a determinadas atividades em função do tempo. Tal alocação sobre o tempo envolve um processo de tomada de decisão que visa otimizar um ou mais critérios de medida de desempenho. Devido ao fato de um dos primeiros problemas modelados como sendo de escalonamento envolver a otimização da produção de uma fábrica, virou consenso definir de maneira geral um problema de escalonamento como sendo a alocação de tarefas a máquinas ou processadores de tal forma a otimizar algum critério de produção. Tais critérios são modelados como uma função matemática de maximização ou minimização chamada de função objetivo.

Existem muitos problemas de escalonamento, variando de acordo com o tempo de processamento, tipos e quantidade de restrições dos elementos a serem escalonados, condições de execução e critério de otimização, isto é, o número de máquinas pode variar e eles podem ser iguais ou diferentes, os elementos podem ou não ter tempo de início e término bem-definidos, pode haver ou não interrupção nas tarefas (preempção), entre outros.

Nos problemas de escalonamento considerados neste trabalho, o número de tarefas e de máquinas são finitos. O número de tarefas é denotado por n e o número de máquinas é denotado por m. Geralmente, j refere-se a uma tarefa enquanto que i refere-se a uma determinada máquina. Se uma tarefa requer uma determinado tipo de processamento ou operação, então o par (i, j) refere-se a ao processamento ou operação de uma tarefa j na máquina i [Bru06], [Pin12]. Sejam m as máquinas $M_j(j = 1, ..., m)$ que devem processar n tarefas $J_i(i = 1, ..., n)$, um escalonamento é para cada tarefa uma alocação de um ou mais intervalos a uma ou mais máquinas. Esta seção baseia-se nas seguintes referências: Rodrigues [dFR09], Brucker [Bru06] e Pinedo [Pin12].

2.2.1 Representação e notação

Graham et al. [GLLK79] introduziram um esquema de classificação para problemas de escalonamento em termos da notação de três campos $\alpha|\beta|\gamma$ onde α especifica o **ambiente de processamento** que possui somente uma entrada, β especifica os detalhes das **características das tarefas** e restrições de escalonamento, pode ter muitas ou nenhuma entrada e

γ especifica a **função objetivo (FO)** a ser otimizada (*critério de otimização*), geralmente tendo somente uma entrada.

Um escalonamento pode ser representado pelo **gráfico de Gantt** (Figura 2.8), que mostra um exemplo de saída **orientado-a-máquina** (Figura 2.8 (a)) onde as tarefas são representadas por caixas retangulares em uma representação do primeiro quadrante do plano cartesiano - o eixo das abcissas representa o tempo de processamento e o eixo das ordenadas representa as máquinas. Outra representação do gráfico de Gantt é a representação **orientado-a-tarefa** (Figura 2.8 (b)), onde a diferença é a troca entre as tarefas e máquinas.

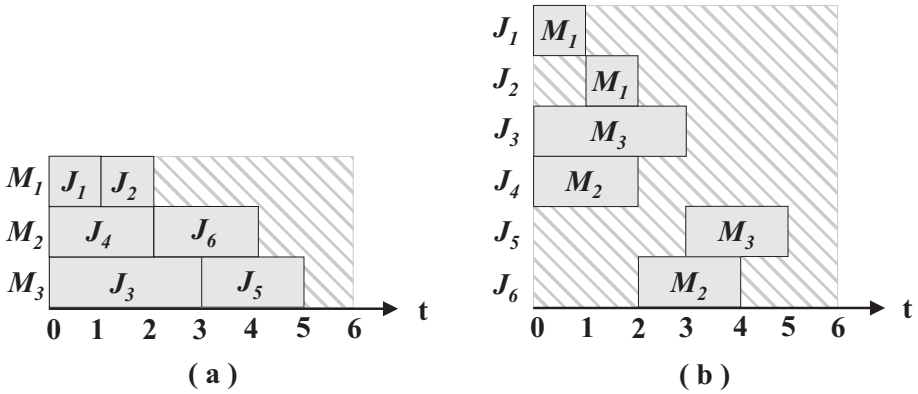


Figura 2.8: Exemplo de representação no gráfico de Gantt: (a) orientado-a-máquina e (b) orientado-a-tarefa [dFR09].

2.2.1.1 Ambiente de processamento

Os principais ambientes de processamento em um escalonamento e suas notações são definidas a seguir:

- **Ambiente monoprocessado:** quando há um único processador no sistema. A notação utilizada é 1;
- **Processadores ou máquina paralelas idênticas:** quando existem m máquinas idênticas em paralelo. Utiliza-se a notação P_m se o número de processadores é fixo (m é uma constante), e utiliza-se a notação P se o número de processadores é parte da entrada (m é uma variável);
- **Processadores ou máquinas paralelas uniformes:** quando existem m máquinas paralelas com velocidades diferentes. A notação utilizada neste caso é Q e Q_m;

- **Processadores ou máquinas paralelas não-relacionadas:** quando existem m máquinas em paralelo com desempenhos dependentes da tarefa a ser executada. A notação utilizada neste caso é R e R_m;

- **Processadores ou máquinas de propósito geral:** quando as m máquinas são divididas em subconjuntos μ, a notação desses subconjuntos é $PMPM$ se as máquinas forem idênticas. Se as máquinas forem paralelas uniformes, a notação é $QMPM$. E, se as máquinas paralelas forem não-relacionadas, a notação é $RMPM$. A notação utilizada para os processadores de propósito geral é MPM;

- ***Flow shop*:** quando existe a mesma sequência de máquinas para cada tarefa. Dessa forma, considera-se m máquinas em série, cada tarefa deve ser processada e cada uma dessas m máquinas. Todas as tarefas devem seguir a mesma configuração de processamento, ou seja, elas devem ser processadas primeiramente na máquina 1, depois na máquina 2 e assim sucessivamente. Depois de finalizar a sua execução em uma máquina, a tarefa entra na fila para ser executada na próxima máquina. Geralmente, todas as filas são organizadas por ordem de chegada (*First In First Out - FIFO*). A notação utilizada é F e F_m;

- ***Job shop*:** quando existem máquinas diferentes e em ordem diferente em cada tarefa. A notação utilizada é J e J_m;

- ***Open shop*:** quando a tarefa é executada em cada máquina exatamente uma vez em cada sequência, esta sequência é diferente para cada tarefa. A notação utilizada é O e O_m.

2.2.1.2 Características das tarefas

Cada **tarefa (do inglês, *job* ou *task*)** pode ter muitas restrições associadas, como as seguintes:

- **Data de chegada ou de disponibilidade (*release date*):** define o momento em que a tarefa pode começar a ser executada. A notação utilizada é r_j;

- **Data de término sugerida (*due date*):** indica o tempo que a tarefa dever ser finalizada, notação d_j, sob pena de sofrer alguma penalidade caso ultrapasse tal tempo;

- **Data de término obrigatória (*deadline*):** indica o tempo que cada tarefa deve ser finalizada, notação D_j, não sendo permitido ultrapassar tal tempo;

- **Tempos de processamento (*processing time*):** restringe o tempo de processamento de cada tarefa, notação p_j, por exemplo, quando $p_j = p$, significa que todas as tarefas possuem o mesmo tempo de processamento, igual a p, quando $p_j = 1$, significa que todas as tarefas têm o mesmo tempo de processamento igual a 1, mas quando a notação p_j é omitida da descrição do problema, as tarefas podem possuir tempos de processamento quaisquer;

- **Peso (*weight*):** que pode representar prioridades iguais ou diferentes. A notação utilizada é w_j;

- **Preempção (*preemption*):** indica se uma preempção (interrompe e retoma sua execução) é permitida ou não. A notação utilizada é *pmtn*;

- **Sem-espera (*no-wait*):** indica que a tarefa não pode ficar em estado de espera depois de iniciada sua execução. Somente para *flow shops*. A notação utilizada é *nwt*;

- **Restrição no número de tarefas:** restringe o número de tarefas, ou seja, $nbr_j = 5$ indica que existem no máximo 5 tarefas a serem processadas. A notação utilizada é *nbr*;

- **Restrição de precedência (*precedence constraints*):** essas restrições podem aparecer nos ambientes mono e multi-processado, sendo que uma ou mais tarefas devem ser completadas antes que outra tarefa possa começar o seu processamento. Existem casos especiais de restrição de precedência: se cada tarefa tem pelo menos um predecessor e pelo menos um sucessor, as restrições são definidas como *chains*. Se cada tarefa tem pelo menos um sucessor, as restrições são definidas como *intree*. Se cada tarefa tem pelo menos um predecessor as restrições são definidas como *outtree*. A notação utilizada é *prec*;

- **Sequência dependente de tempos de preparação (*sequence dependent setup times*):** onde s_{jk} representa uma sequência dependente de tempos de preparação das tarefas j e k, S_{0k} denota o tempo de preparação para a tarefa k se a tarefa k é a

primeira na sequência e s_{j0} se a tarefa j é a última da sequência. Se o tempo de preparação entre as tarefas j e k depende da máquina, então o índice i é incluído, ou seja, s_{ijk}. Se s_{jk} não aparecer no problema, ou seja, se s_{jk} não aparecer no campo β, todos os tempos de preparação possuem o valor 0. A notação utilizada é s_{jk};

- **Processamento em lotes (*batch processing*):** onde uma máquina deve ser capaz de processar um número de tarefas, b, simultaneamente, ou seja, pode processar um lote de b tarefas ao mesmo tempo. Os tempos de processamento de todas as tarefas no lote não devem ser os mesmos e o lote todo é finalizado somente quando a última tarefa do lote é completada, implicando que o tempo de processamento de todo o lote é determinado pela tarefa com o maior tempo de processamento. Se $b = 1$, então o problema é reduzido ao problema convencional de escalonamento. Outro caso especial é o $b = \infty$, ou seja, não há limite no número de tarefas que a máquina pode processar ao mesmo tempo.

  Utilizam-se as notações $p-batch$, quando o comprimento do lote for igual ao maior dos tempos de processamento dentre todas as tarefas, e a notação $s-batch$, quando o comprimento do lote for igual a soma dos tempos de processamento de todas as tarefas.

É chamada **janela de tempo** (*time window*), um intervalo de tempo determinado por uma data de início e uma data de término. A janela de tempo de uma determinada tarefa, para que seja executada em tempo, é definida por sua data de chegada e por sua data de término sugerida. Uma tarefa pode ser executada em qualquer máquina com mesmo desempenho ou diferentemente em cada máquina, o que pode tomar mais ou menos tempo para ser executada. Neste caso, p_{ij} representa o tempo de processamento que a tarefa J_j demora pra ser processada na máquina M_i. O critério de otimização ou função objetivo $f_j(t)$ representa o custo induzido pela tarefa J_j quando é executada em um processador até o tempo t.

2.2.1.3 Critério de otimização

De acordo com Brucker [Bru06], existem dois tipos de função de custo, que são: $f_{max}(C) = max\{f_i(C_i)|i = 1,...,n\}$ e $\sum f_i(C) = \sum_{i=1}^{n} f_i(C_i)$ chamados de funções objetivo de gargalo (*bottleneck objectives*) ou do tipo MinMax (MaxMin), e funções objetivo

de soma total (*sum objectives*), respectivamente. O objetivo é achar um escalonamento factível que minimize uma função de custo obedecendo a certos critérios de otimização. Existem determinadas **funções objetivos clássicas**, a maioria delas sendo uma **função regular**, que possui um comportamento não-decrescente em relação aos tempos de completude das tarefas. O tempo de completude da tarefa j na máquina i é denotado por C_{ij}. Se a tarefa não for dependente da máquina, o tempo que a tarefa j sai do sistema, ou seja, o tempo que a tarefa finaliza o seu processamento na máquina onde foi processada, é denotado por C_j.

A seguir serão apresentados exemplo de funções regulares que consideram escalas de tempo, conhecidas como **critérios proporcionais**, que são [dFR09]:

- **Maior tempo de completude (*makespan*)** (Notação: C_{max}) refere-se ao tempo de completude da última tarefa a terminar sua execução, ou de outra forma, maior tempo de completude dentre todas as tarefas executadas. $C_{max} = max\{C_1, ..., C_m\}$. Deseja-se minimizar C_{max}, pois um *makespan* mínimo geralmente significa uma maior utilização da máquina;

- **Máxima latência (*maximum lateness*)** (Notação: L_{max}) é definida como $L_j = C_j - d_j$, que será um valor positivo se a tarefa J_j for executada com atraso, será um valor negativo se a tarefa for antecipada ou, senão, será um valor nulo. A função de máxima latência (L_{max}), então, retorna o valor do maior atraso possível ou menor antecipação se não houver atrasos, considerando todas as tarefas executadas. $L_{max} = max\{L_1, ..., L_n\}$, onde deseja-se minimizar o L_{max};

- **Tempo total de completude (*total completion time*)** (Notação: $\sum C_j$ ou $\sum w_j C_j$) soma de todos os tempos de completude das tarefas (ponderadas ou não), onde deseja-se minimizar $\sum C_j$ ou $\sum w_j C_j$;

- **Tempo total de atraso (*tardiness*) (Notação: $\sum T_j$ ou $\sum w_j T_j$)** trata-se da soma de todos os tempos de atraso das tarefas (ponderadas ou não). Dessa forma, o atraso assume um valor positivo se as tarefas forem atrasadas, caso contrário, o atraso assume o valor 0. Sendo $T_j = max\{L_j, 0\}$, onde deseja-se minimizar $\sum T_j$ ou $\sum w_j T_j$.

Também existem funções objetivo que não dependem diretamente do tempo, sendo conhecidos como **critérios permanentes**, que são:

- **Número de tarefas tardias** (Notação: $\sum U_j$ ou $\sum w_j U_j$) é o número total de tarefas executadas com atraso (ponderadas ou não), onde U_j é uma penalidade unitária caso a tarefa esteja atrasada (e nula caso contrário), como no exemplo a seguir:

$$U_j = \begin{cases} 1 \text{ se } C_j > d_j \\ 0 \text{ caso contrário} \end{cases} \tag{2.1}$$

A função objetivo que envolve a minimização de atraso pertence a classe de funções objetivo que possui medidas **regulares** de desempenho. Uma função objetivo possui uma medida de desempenho regular quando é não-decrescente em $C_1, ..., C_n$. Pesquisas recentes consideram cada vez mais o estudo de funções objetivo que são **não-regulares**, como por exemplo, a que representa penalidades de antecipação (ponderadas ou não). A antecipação assume um valor positivo se as tarefas forem antecipadas, caso contrário, o valor de antecipação assume o valor 0. Desta forma, a penalidade de antecipação é não-crescente em C_j e pode ser formalmente descrita como segue:

- **Tempo total de antecipação (*earliness*) (Notação: $\sum E_j$ ou $\sum w_j E_j$)** trata-se da soma de todos os tempos de antecipação das tarefas (ponderadas ou não). Desta forma, a antecipação assume um valor positivo se as tarefas forem antecipadas, caso contrário, o valor de antecipação é valor 0. Sendo $E_j = max d_j - C_j, 0$, onde deseja-se minimizar $\sum E_j$ ou $\sum w_j E_j$.

Para atender ao conceito surgido nas indústrias de produção sem folga, ou seja, de produto produzido o mais próximo possível da data de entrega (do inglês *Just-in-Time* - JIT), aplica-se uma combinação dos critérios de antecipação e atraso anteriormente descritos, e, desta forma, a função objetivo consiste na soma das tarefas escalonadas com antecipação e atraso, ponderadas ou não, tal como segue:

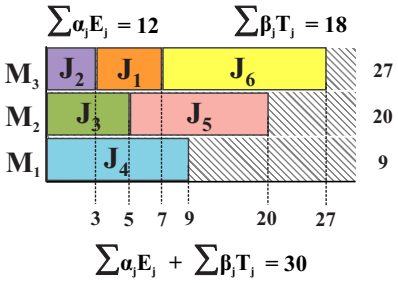
$$\sum_{j=1}^{n} \alpha_j E_j + \sum_{j=1}^{n} \beta_j T_j. \tag{2.2}$$

Observa-se que na formulação acima, as tarefas são ponderadas e o peso associado à antecipação da tarefa j (α_j) difere do peso associado ao atraso da tarefa j (β_j). Se os pesos forem iguais para ambas penalidades considerando a mesma tarefa, então, basta isolar a constante do somatório.

Um exemplo para o escalonamento com penalidades de antecipação e atraso em ambiente de máquinas paralelas idênticas é apresentado na Figura 2.9 onde, na Figura 2.9 (a) é apresentado um exemplo de instância com 6 tarefas para o problema com seus respectivos valores para tempos de processamento (p_j), data de término sugerida (d_j), penalidades de antecipação (α_j), penalidades de atraso (β_j), tempo de completude (C_j) das tarefas, valores de antecipação (E_j) e atraso (T_j) das tarefas, e os valores da antecipação ponderada ($\alpha_j E_j$) e do atraso ponderado das tarefas ($\beta_j T_j$) e na Figura 2.9 (b) pode-se observar como fica a representação do escalonamento em três máquinas para essa instância. O valor de antecipação é obtido pela diferença entre data de término sugerida e o tempo de completude, e o valor de atraso é obtido através de diferença entre o tempo de completude da tarefa e a data de término sugerida, caso o resultado de uma dessas diferenças seja negativo, é atribuído o valor 0 no lugar do resultado negativo. Ao final, o peso ou penalidade de cada tarefa é multiplicado com o seu respectivo valor de antecipação (E_j) ou atraso (T_j) e, depois esses resultados são somados obtendo assim o valor total de antecipação ponderada e de atraso ponderado das tarefas.

J_j	p_j	d_j	α_j	β_j	C_j	E_j	T_j	$\alpha_j E_j$	$\beta_j T_j$
J_1	4	7	3	5	7	0	0	0	0
J_2	3	4	4	5	3	1	0	4	0
J_3	5	5	8	8	5	0	0	0	0
J_4	9	10	8	10	9	1	0	8	0
J_5	15	17	6	4	20	0	3	0	12
J_6	20	25	7	3	27	0	2	0	6
						$\sum\alpha_j E_j$ e $\sum\beta_j T_j$ =		12	18

(a)



(b)

Figura 2.9: (a) Exemplo de uma instância de 6 tarefas para o problema de escalonamento com antecipação e atraso (b) representação do escalonamento através do gráfico de Gantt orientado-a-máquina.

2.2.2 Hierarquia de complexidade

Os problemas de escalonamento de tarefas são problemas de **otimização combinatória**. Existem inúmeros problemas de escalonamento, classificados em diferentes classes de **complexidade computacional**. Tal complexidade determina a natureza do algoritmo a ser desenvolvido e, muitas vezes, um algoritmo para um determinado problema de escalonamento pode ser aplicado para a solução de outro problema, pois muitos desses problemas tratam-se de casos particulares ou generalizações de outros. Por exemplo, o pro-

blema $1||\sum C_j$ é um caso especial do problema $1||\sum w_j C_j$, dessa forma, um algoritmo para $1||\sum w_j C_j$ pode também ser utilizado para solucionar o problema $1||\sum C_j$. Na terminologia de complexidade é dito que o problema $1||\sum C_j$ é redutível ao problema $1||\sum w_j C_j$. Que é denotado por suas **hierarquias de complexidade** [Pin12],[dFR09],[Bru06]:

$$1||\sum C_j \propto 1||\sum w_j C_j. \tag{2.3}$$

Entretanto, também podem existir muitos problemas que não são comparáveis uns com os outros. Por exemplo, $P_m||\sum w_j T_j$ não é comparável com o problema $J_m||C_{max}$.

Na Figura 2.10 (a) são apresentadas as reduções elementares para os principais ambientes de processamento; na Figura 2.10 (b) são apresentados os diagramas de inclusão para algumas restrições de processamento; e na Figura 2.10 (c) são apresentadas as reduções elementares para as funções objetivo.

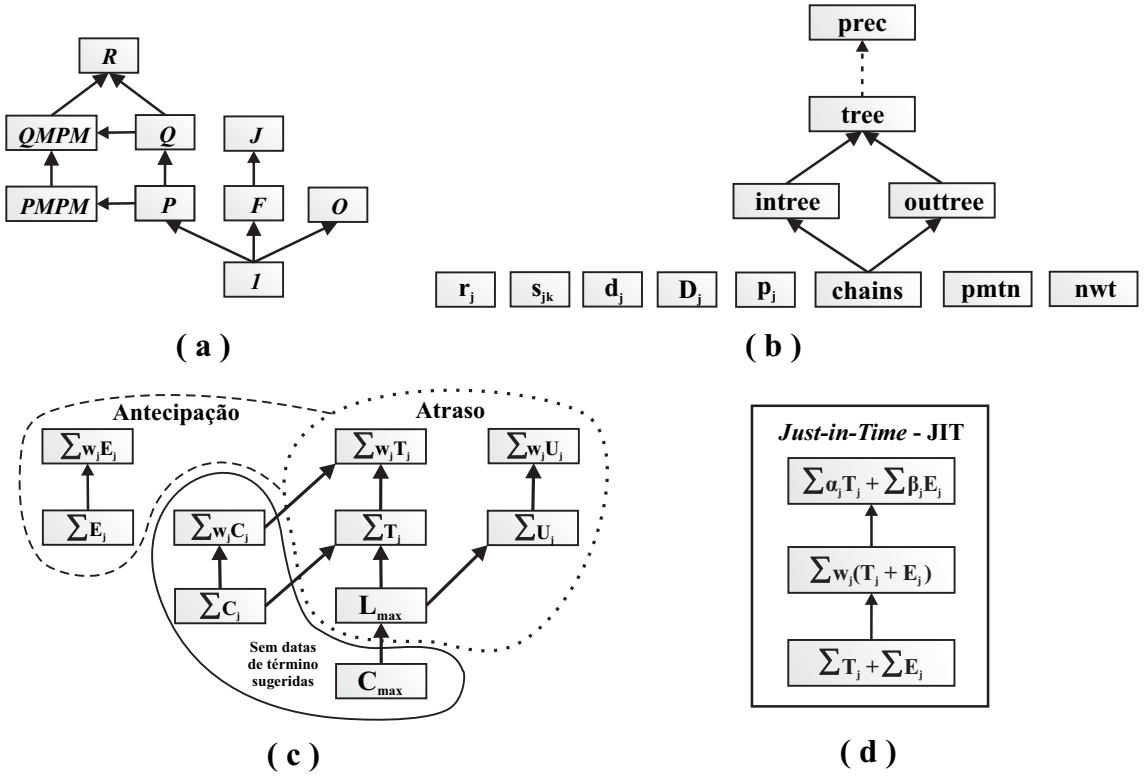


Figura 2.10: Hierarquias de complexidade dos problemas de escalonamento determinísticos: (a) relação de complexidade entre os ambientes de processamento (b) relação entre as principais características das tarefas em um escalonamento (c) relação de complexidade entre as funções objetivo clássicas (d) relação de complexidade entre as funções objetivo com antecipação e atraso.

2.3 Considerações finais

Este Capítulo apresentou a fundamentação teórica necessária para a resolução de Problemas de Otimização Combinatória, incluindo métodos exatos e aproximados. Na Teoria de Escalonamento foram resumidos os principais conceitos, classificações, notações e políticas de escalonamento. O estudo destes conceitos foram de extrema importância para o desenvolvimento deste trabalho.

Capítulo 3

Escalonamento de tarefas com penalidades de antecipação e atraso

Neste capítulo é abordada a classe de problemas de principal interesse deste trabalho, envolvendo problemas de escalonamento de tarefas com penalidades de antecipação e atraso. Para isto, cada tipo de penalidade - antecipação ou atraso - é considerada em separado e depois em conjunto. São apresentadas as definições e conceitos principais, notação clássica e exemplos de aplicação, bem como uma revisão da literatura tanto para ambiente monoprocessado quanto para máquinas paralelas. Questões estruturais de escalonamentos válidos são analisadas, destacando-se uma análise sobre soluções diferentes com empate no valor da função objetivo e soluções com simetria. Por fim, uma análise das formulações matemáticas disponíveis na literatura para o problema também é apresentada.

Na literatura existe uma ampla e crescente quantidade de trabalhos que consideram problemas de escalonamento com antecipação e atraso, dentre eles destacam-se as revisões da literatura apresentadas por Baker e Scudder [BS90], Gordon et al. [GPC02] e Shabtay e Steiner [SS12]. Este capítulo foi estruturado com base na análise realizada sobre o problema, e os trabalhos e resultados encontrados.

3.1 Escalonamento com atraso de tarefas

Minimizar o atraso total de uma tarefa é uma das considerações mais importantes na produção industrial, onde se trabalha com datas de entrega contratuais. Na literatura, existem

diversos trabalhos propondo modelos teóricos diferenciados e novos métodos de resolução. As principais abordagens exatas para este problema são, em sua maioria, baseadas em algoritmos de Programação Dinâmica e *branch-and-bound*, mas, os melhores resultados para o problema clássico em máquinas paralelas envolvem um método de *branch-cut-and-price* [PUPdF10]. Abordagens aproximadas são baseadas em regras de despacho ou políticas de escalonamento de tarefas, embutidas em heurísticas e meta-heurísticas.

A Figura 3.1 mostra a definição do atraso de uma tarefa na linha do tempo, onde a mudança da curva ocorre no ponto definido pela data de término sugerida (*due date*, d_j,). Pode-se observar que antes da data de término sugerida d_j, o valor de atraso é nulo e, após essa data, o valor de atraso é monotonicamente crescente. A Tabela 3.1 apresenta uma classificação dos problemas de escalonamento de tarefas com atraso, investigados nesta seção.

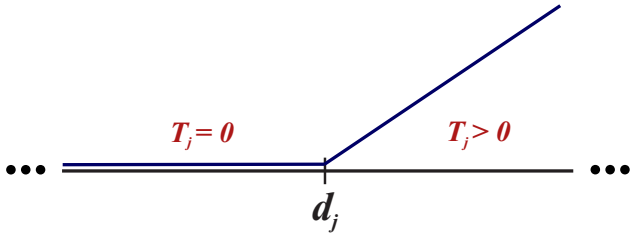


Figura 3.1: Definição do valor de atraso na linha do tempo.

3.1.1 Ambiente monoprocessado com atraso de tarefas

O problema $1||\sum T_j$ recebeu bastante atenção na literatura pois durante muitos anos sua complexidade permaneceu em aberto, até que o mesmo foi provado como NP-difícil em 1990 por Du e Leung [DL90]. Abordagens exatas para este problema são, em sua maioria, baseadas em algoritmos de programação dinâmica e *branch-and-bound*. Abordagens aproximadas são baseadas em regra de despacho ou políticas de escalonamento de tarefas, embutida em heurísticas e meta-heurísticas.

Tan e Narasimhan [TN97] propuseram um algoritmo de recozimento simulado (*simulated annealing*). Abordagens utilizando algoritmos genéticos também podem ser encontradas em Liu et al. [LAR05] e Demirel et al. [DOcDT11], bem como quando adicionado ao problema clássico acima, tempos de preparação (*setups*) (Tan et al. [TNRR00] e França et al. [FMM01]). Alguns trabalhos apresentam estratégias híbridas como em Feili [FHG12], onde são propostos três estratégias algorítmicas para o problema de esca-

lonamento com atraso ponderado de tarefas baseado em recozimento simulado, algoritmo genético e busca local.

Estratégias eficientes de busca em vizinhança baseada em movimentos generalizados de troca de pares são apresentados em Rodrigues et al. [dFPUP08] e Grosso et. al. [GDCT04]. Gupta e Smith [GS06] propuseram um algoritmo GRASP com reconexão de caminhos e um algoritmo de Busca Local para a minimização do atraso ponderado de tarefas com tempos de preparação. Outro método de busca foi apresentado por Mendes et al. [MFM02], envolvendo busca múltipla (*multi-start*), que trata-se de um algoritmo que parte de várias soluções iniciais aleatórias seguidas de busca local a fim de encontrar uma solução ótima na vizinhança do espaço de soluções.

Trabalhos que consideram estratégias exatas baseadas em Programação Dinâmica foram considerados em Baptiste [Bap00] e Tanaka e Fujikuma [TF11]. Métodos de *branch-and-bound* foram considerados em Wodecki [Wod08], Babu [BPP04] e Lu e Chu [LC06], incluindo resultados para 40 e 50 tarefas.

Tian et al. [TNC06] forneceram um algoritmo de tempo $O(n^2)$ para solucionar o problema $1|r_j, p_j = p, pmtn|\sum T_j$,envolvendo janelas de tempo, preempção e tempos iguais de processamento, usando o conceito de blocos (técnica de decomposição), onde o problema de escalonamento é decomposto em dois subproblemas e cada subproblema envolvendo um escalonamento ótimo, que é então distribuído para cada bloco. Finalmente, é investigado o escalonamento de um bloco com tarefas de tempos iguais e algumas propriedades são derivadas para a elaboração de um algoritmo para solucionar o problema.

3.1.2 Máquinas paralelas com atraso de tarefas

Uma abordagem sumarizada para os problemas de escalonamento em máquinas paralelas e uma exposição sobre suas complexidades pode ser consultada em Kravchenko e Werner [KW11]. Nesta subseção, para se discorrer sobre os problemas de escalonamento em máquinas paralelas, são considerados os três tipos clássicos de ambientes, onde em todos eles, cada tarefa pode ser processada em qualquer uma das m máquinas. Tais ambientes podem envolver: *Máquinas paralelas idênticas* (P), onde os tempos de processamento independem da máquina, ou seja, $p_{ij} = p_j$; *Máquinas paralelas uniformes* (Q) onde cada máquina possui um desempenho/velocidade diferente, onde $p_{ij} = p_j/\tau_i$ e τ_i denota a velocidade uniforme da máquina M_i e; *Máquinas paralelas não-relacionadas* (R) onde cada

máquina tem um comportamento/velocidade diferente para cada tarefa.

3.1.2.1 Máquinas paralelas idênticas

Considere um problema de escalonamento em máquinas paralelas que inclui um conjunto de tarefas independentes, $J = \{J_1, J_2, ..., J_n\}$, em um conjunto de máquinas paralelas idênticas, $P = \{1, 2, ..., m\}$, minimizando uma função objetivo como o atraso total ponderado.

Armentano e Yamashita [AY00] consideraram o algoritmo de **busca tabu** para escalonar as tarefas em máquinas paralelas idênticas com o objetivo de minimizar o atraso. A abordagem dos autores inicia com uma solução obtida pela heurística KPM, que por sua vez pode ser descrita em dois passos: no primeiro passo, todas as tarefas não escalonadas são listadas usando a política de escalonamento do menor para o maior tempo de processamento (*shortest processing time first* - SPT) e geram-se m cópias desta lista - uma para cada máquina. No segundo passo, determina-se a próxima tarefa a ser escalonada usando a heurística PSK (desenvolvida por Panwalkar et al. [PSK93]) para minimizar o atraso total em uma máquina única.

Após a solução gerada pela heurística KPM, Armentano e Yamashita [AY00] aplicaram a busca tabu nas soluções vizinhas para obter novas e melhores soluções, sendo a vizinhança definida por dois movimentos. O primeiro envolve inserções que consistem em transferir cada tarefa de uma máquina para outra e a segunda envolve trocas, que são obtidas pelas trocas de pares de tarefas de duas máquinas.

Existem na literatura abordagens para o ambiente de máquinas paralelas idênticas, mas que também consideram o ambiente monoprocessado, como pode ser observado em Rodrigues et al. [dFPUP08], onde foi proposto um algoritmo heurístico para o problema $P||\sum w_j T_j$ em que o escalonamento tanto uma máquina quanto em máquinas paralelas é representado por uma lista sequencial de tarefas (Figura 4.1 (a)). A Figura 4.1 (b) mostra representação de um escalonamento em máquinas paralelas idênticas através do **gráfico de Gantt**.

Para este mesmo problema de escalonamento (considerando somente máquinas paralelas idênticas), Croce et al. [CGG12] apresentaram uma heurística de melhoria cujos resultados computacionais apresentados são melhores que os resultados existentes na literatura, incluindo os apresentados por Rodrigues et al. [dFPUP08]. Um algoritmo exato pode ser consultado em Pessoa et al. [PUPdF10], onde foi proposto um método

branch-cut-and-price para o problema $P||\sum w_j T_j$, considerando o ambiente monoprocessado, além de máquinas paralelas idênticas.

Kravchenko e Werner [KW09] mostraram que os problemas $P|pmtn|\sum T_j$, com preempções permitidas, e $P|r_j, p_j = p, pmtn|\sum T_j$, com datas de chegada definidas, tarefas de tempos de processamento iguais e preempções permitidas, são NP-difíceis.

3.1.2.2 Máquinas paralelas uniformes

Considere um problema de escalonamento em máquinas paralelas que inclui um conjunto de tarefas independentes, $J = \{J_1, J_2, ..., J_n\}$, em um conjunto de máquinas paralelas uniformes ou com velocidades diferentes, $Q = \{1, 2, ..., m\}$, minimizando uma função objetivo como o atraso total ponderado.

Dessouky et al. [DLLvdV90] consideraram o caso de escalonar n tarefas idênticas em m máquinas paralelas uniformes, problema $Q|p_j = 1|\sum T_j$, onde foram propostos algoritmos polinomiais de tempo $O(n \log n)$ para minimizar o atraso total ponderado e a latência máxima. O problema foi solucionado através de um algoritmo de trocas ou alocação simples, que consiste em organizar as tarefas em ordem não-decrescente de datas de término e associá-las de acordo com os tempos de completude $C_1, ..., C_n$.

Kravchenko e Werner [KW09] consideraram o caso onde preempções são permitidas, ou seja, o tempo de processamento de qualquer tarefa pode ser interrompido a qualquer instante e recomeçar depois, possivelmente em uma máquina diferente. Foram desenvolvidos algoritmos de tempo polinomial para os problemas $Q|p_j = p, pmtn|\sum T_j$ e $Q|p_j = p, pmtn|\sum T_j$.

Dourado et al. [DdFS10] apresentaram algoritmos para os problemas de escalonamento $Q|p_j = 1|\sum w_j T_j$ e $Q|p_j = p|\sum w_j T_j$, onde é dado um conjunto $J = \{J_1, ..., J_n\}$ de n tarefas a serem escalonadas em m máquinas paralelas uniformes $Q = \{Q_1, ..., Q_m\}$, onde cada máquina Q_i tem uma velocidade específica q_i e toda tarefa J_j tem um tempo de processamento $p_j = p$. A execução da tarefa J_j na máquina Q_i requer tempo p_j/q_j. Para o problema $Q|p_j = 1|\sum w_j T_j$, a estratégia utilizada para resolver o problema envolve primeiramente solucionar uma versão não ponderada do problema ($Q|p_j = 1|\sum T_j$), resultando em um escalonamento com bipartição de tarefas (tarefas finalizadas em suas datas de término sugeridas e tarefas com atraso). Uma estratégia ótima para o caso ponderado é obtida através da troca das tarefas tardias com as tarefas finalizadas em suas datas de

término sugeridas. Desta forma, essa mesma estratégia pode ser utilizada para o problema $Q|p_j = p|\sum w_j T_j$, por ser um problema equivalente ao problema $Q|p_j = 1|\sum w_j T_j$.

3.1.2.3 Máquinas paralelas não-relacionadas

Considere um problema de escalonamento em máquinas paralelas que inclui um conjunto de tarefas independentes, $J = \{J_1, J_2, ..., J_n\}$, em um conjunto de máquinas paralelas não-relacionadas, $R = \{1, 2, ..., m\}$, onde cada tarefa possui um desempenho associado a cada máquina, com o objetivo de minimizar uma função objetivo como o atraso total ponderado.

Bilyk e Mönch [BM10] estudaram um problema de planejamento e escalonamento em máquinas paralelas não relacionadas. Foram consideradas n tarefas que devem ser atribuídas e escalonadas em m máquinas paralelas não relacionadas, sendo que cada tarefa possui um peso que representa a prioridade do pedido do cliente correspondente, uma data de término sugerida e uma data de chegada. Foi modelado um problema de escalonamento em máquinas paralelas não relacionadas em um ambiente de fabricação de placas de fiação impressa (PWB), que consiste em múltiplos estágios de produção, no qual cada placa deve passar por uma sequência pré-determinada de passos de fabricação. Para resolver este problema foi desenvolvida uma metodologia heurística baseada em decomposição e busca em vizinhanças variáveis (*variable neighborhood search* - VNS).

3.2 Escalonamento com antecipação de tarefas

Em ambientes reais, os fabricantes não costumam estipular uma data de término sugerida para uma tarefa específica; ao invés disso, eles possuem datas pré-definidas ou previstas em que suas tarefas devem ser finalizadas, um exemplo seria as tarefas que devem ser entregues no final de cada dia de trabalho cuja antecipação é definida como a diferença entre o tempo de entrega e o tempo de completude (C_j) dessa tarefa [Yan00]. A Figura 3.2 mostra onde é possível obter o valor de antecipação de uma tarefa na linha do tempo. Observa-se que após a data de término sugerida d_j, o valor de antecipação é nulo mas, antes desta data de término sugerida, o valor de antecipação é monotonicamente decrescente. A Tabela 3.2 apresenta uma classificação dos problemas de escalonamento de tarefas com antecipação investigados nesta seção.

Tabela 3.1: Classificação dos problemas de escalonamento com atraso de tarefas.

Ambiente	Problemas	Estratégias de resolução	Referências
1	$1\|ST_{sd}\|\sum T_j$, onde ST_{sd} refere-se ao tempo de preparação dependente da sequência	Recozimento simulado	Tan e Narasimhan [TN97]
	$1\|ST_{sd}\|\sum T_j$	*Branch-and-bound*, *simulated annealing*, algoritmo genético e técnica de troca de pares (*pairwise interchange*)	Tan et al. [TNRR00]
	$1\|s_{ij}\|\sum T_j$	*Branch-and-bound*	Lu e Chu [LC06]
	$1\|\|\sum w_jT_j$	*Branch-and-bound*	Babu [BPP04], Wodecki [Wod08]
	$1\|ST_{sd}\|\sum T_j$	Algoritmo genético	França et al. [FMM01]
	$1\|s_{ij}\|\sum w_jT_j$	GRASP com reconexão de caminhos e busca local	Gupta e Smith [GS06]
	$1\|\|\sum w_jT_j$	Algoritmo genético	Liu et al. [LAR05]
	$1\|\|\sum T_j$	Algoritmo genético	Demirel et al. [DOcDT11]
	$1\|s_{ij}\|\sum w_jT_j$	Recozimento simulado, algoritmo genético, busca local e algoritmo genético com busca local	Feili [FHG12]
	$1\|ST_{sd}\|\sum T_j$	Busca local *multi-start*	Mendes et al. [MFM02]
	$1\|r_j,p_j=p,pmtn\|\sum T_j$	Técnica de decomposição	Tian et al. [TNC06]
	$1\|\|\sum w_jT_j$	Método de busca *dynasearch*	Grosso et. al. [GDCT04]
	$1\|\|\sum w_jT_j$	Lista sequencial de busca local baseada em movimentos GPI, com critério de desempate	Rodrigues et al. [dFPUP08]
	$1\|r_j,p_j=p\|\sum_j T_j$	Programação Dinâmica	Baptiste [Bap00]
	$1\|r_j\|\sum w_jT_j$	Programação dinâmica	Tanaka e Fujikuma [TF11]
P	$P\|\|\frac{1}{n}\sum_{k=1}^{m}T(S_k)$	Busca tabu	Armentano e Yamashita [AY00]
	$P\|\|\sum w_jT_j$	ILS com técnicas de busca em vizinhança de grande porte (VLNS)	Croce et al. [CGG12]
	$P\|\|\sum w_jT_j$	Lista sequencial de busca local baseada em movimentos GPI, com critério de desempate	Rodrigues et al. [dFPUP08]
	$P\|\|\sum w_jT_j$	Algoritmo *branch-cut-and-price*	Pessoa et al. [PUPdF10]
Q	$Q\|p_j=1\|\sum T_j$	Algoritmo de troca de argumentos	Dessouky et al. [DLLvdV90]
	$Q\|p_j=p,pmtn\|\sum T_j$	Algoritmo de tempo polinomial	Kravchenko e Werner [KW09]
	$Q\|p_j=1\|\sum w_jT_j$	Algoritmo de tempo polinomial	Dourado et al. [DdFS10]
R	$R_m\|r_j\|\sum_{j=1}^{n}w_jT_j = \sum_{j=1}^{n}w_j(C_j-d_j)^+$, onde MP é múltiplas máquinas	Heurística baseada em decomposição e busca em vizinhança variável (VNS)	Bilyk e Mönch [BM10]

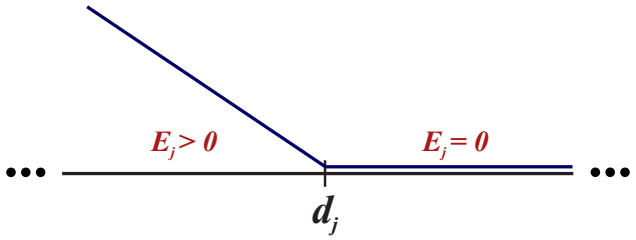


Figura 3.2: Definição do valor de antecipação na linha do tempo.

Cheng et al. [CGK96] estudaram o problema de escalonamento em lotes (*batches*), onde as tarefas em um lote são entregues ao cliente juntamente com o tempo de completude da última tarefa no lote, a antecipação dessa tarefa é definida como a diferença entre o tempo de entrega do lote, em que essa tarefa está inserida, e o seu tempo de completude. A abordagem utilizada foi a de programação dinâmica para o escalonamento de máquinas paralelas e uma abordagem semelhante foi considerada para o problema de escalonamento em lotes para o ambiente monoprocessado.

Gordon e Strusevich [GS99] consideraram o ambiente monoprocessado com datas de término sugeridas e problemas de escalonamento com n tarefas em que as datas de término sugeridas são obtidas a partir de tempos de processamento com folgas q (*slack due dates*). Eles fazem uma consideração para as seguintes funções objetivo (importantes para o escalonamento e controle de aplicações): antecipação ponderada $\sum_{j=1}^{n} w_j E_j$, onde w_j é um dado peso da tarefa j que indica sua relativa importância e; antecipação com ponderações exponenciais $\sum_{j=1}^{n} w_j exp(\gamma E_j)$, onde $\gamma \neq 0$.

Koksalan et al. [KAK98] consideraram o problema de escalonamento bicritério, considerando a minimização do tempo de fluxo e máxima antecipação de tarefas em ambiente monoprocessado, eles desenvolveram uma heurística para gerar sequências eficientes para o caso em que as máquinas podem possuir tempo ocioso e também fizeram considerações para casos sem tempo ocioso (um resumo sobre problemas de escalonamento que permitem ou não tempo ocioso é feito na seção 3.3.1).

Um caso de problemas de escalonamento com entrega em lotes e penalidades de antecipação foi estudado por Yang [Yan00], onde foram considerados os problemas de escalonamento que minimizam dois tipos de penalidades de antecipação. No primeiro problema, o objetivo é achar uma sequência ótima tal que o tempo obtido no somatório das antecipações ponderadas seja mínimo. No segundo problema, o objetivo é achar uma sequência ótima tal que o valor obtido no somatório das antecipações ponderadas seja mínimo.

Pathumnakul e Egbelu [PE05] propuseram uma heurística para minimizar a antecipação ponderada de tarefas sujeitas a uma data máxima de término (*deadlines*) em ambiente monoprocessado, baseada em condições locais de otimalidade. A heurística começa escalonando cada tarefa de modo que o seu tempo de completude coincida com o seu *deadline*. A solução obtida é conhecida como solução ideal. Se não houver conflito de execução entre as tarefas em uma máquina, isto é, se não houver sobreposição entre os tempos nos quais as tarefas deveriam estar sendo executadas, a solução obtida é ótima, senão a solução obtida é infactível. A heurística trabalha nestes conflitos até que não seja mais possível melhorar a solução. Os autores apresentam resultados para 10, 15, 20, 25 e 30 tarefas, sendo que só foi obtido o ótimo para 10 e 15.

Um algoritmo *branch-and-bound* foi proposto por Wu e Lee [WL06] para a minimização do tempo de preparação e da antecipação das tarefas em ambiente monoprocessado, onde o problema abordado consiste em várias tarefas que são agrupadas vários grupos (esses grupos representam um determinado pedido de um cliente). O tempo de preparação é utilizado somente quando uma tarefa muda de um grupo de tarefas para outro, o algoritmo *branch-and-bound* foi capaz de prover uma solução ótima para instâncias de somente 12, 15 e 18 tarefas. Outro algoritmo *branch-and-bound* também foi proposto por Yin et al. [YWW$^+$12] juntamente com o algoritmo de Recozimento Simulado, que é utilizado para gerar uma solução inicial para o algoritmo, o algoritmo *branch-and-bound* proposto é capaz de resolver instâncias do problema de até 28 tarefas.

Zhao e Tang [ZT07] consideraram o problema de escalonamento em ambiente monoprocessado com efeitos tardios, eles propuseram soluções polinomiais para a minimização do problema de *makespan* (momento em que termina a execução da última tarefa), o problema de minimização da soma dos tempos de completude e a soma de tarefas antecipadas ponderadas.

3.3 Escalonamento com antecipação e atraso

Problemas de escalonamento com antecipação e atraso (Escalonamento E/T) representam situações importantes do mundo real, com o surgimento motivado pela adoção nas indústrias de processos produtivos com o conceito de produção sem folga (nem antes nem depois da data de término sugerida), ou seja, de produtos com produção concluída

Tabela 3.2: Classificação dos problemas de escalonamento com antecipação de tarefas.

Ambiente	Problemas	Estratégias de resolução	Referências
1	$1\|prec, C_j \leq d_j = p_j + q\|\sum w_j E_j$, onde q é a folga adicionada a data de término sugerida (*slack due dates*)	Algoritmo de tempo polinomial com determinação de datas de término sugeridas	Gordon e Strusevich [GS99]
	$1\|C_j \leq d_j = p_j + q\|\sum w_j E_j$	Algoritmo de tempo polinomial com determinação de datas de término sugeridas	Gordon e Strusevich [GS99]
	$1\|D_j\|\sum w_j E_j$	Algoritmo heurístico baseado em condições locais de otimalidade	Pathumnakul e Egbelu [PE05]
	$1\|s_j\|\sum E_j$	Algoritmo *branch-and-bound*	Wu e Lee [WL06]
	$1\|\|\sum w_j E_j$	Algoritmo *branch-and-bound*	Yin et al. [YWW+12]

no momento em que devem ser entregues (do inglês, *Just-in-Time* (JIT)). Tal termo surgiu no Japão na década de 70, na indústria automotiva, onde se buscava um sistema em que fosse possível coordenar uma determinada produção com a sua demanda específica e com o menor tempo de atraso possível. Esse sistema tem como objetivo a melhoria do processo produtivo e fluxo de manufatura, eliminação de estoques e desperdícios. A aplicação desse sistema é muito ampla, envolvendo a produção de produtos perecíveis por exemplo, mas abrangendo qualquer sistema em que as tarefas devam ser finalizadas o mais próximo possível do data de término estipulada [LA04].

De acordo com Xiao e Li [XL02], decisões na determinação de datas de término sugeridas de tarefas são importantes no planejamento e controle de muitas operações de uma organização. Uma cotação rápida de entrega de um determinado trabalho pode ser muito atrativa para o cliente, mas pode aumentar a chance de entregas atrasadas. Por outro lado, a demora na cotação de um trabalho pode diminuir a chance de um trabalho ser entregue com atraso, mas será penalizado com o risco de se perder um determinado negócio em potencial.

A Figura 3.3 mostra onde é possível obter o valor de antecipação e atraso de uma tarefa na linha do tempo, onde o que se deseja é minimizar o valor de antecipação e atraso de modo que se tenha uma tarefa o mais próxima possível da data de término sugerida (d_j). Desta forma, quanto mais próxima a tarefa estiver da data de término estipulada, menor será o seu valor de antecipação e atraso. A Tabela 3.3 apresenta uma classificação dos problemas de escalonamento de tarefas com antecipação e atraso investigados nesta seção:

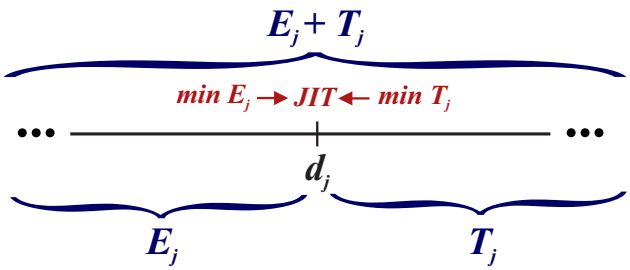


Figura 3.3: Definição do valor de antecipação e atraso na linha do tempo.

O conceito JIT também pode ser utilizado para gerir as atividades externas de uma empresa, como o processo de compra de insumos de outras fábricas, fabricação e montagem de novos produtos a partir desses insumos e distribuição ou exportação dos itens produzidos. JIT auxilia em todo esse processo, pois parte da ideia de 'conduzir' a produção a partir de uma determinada demanda, ou seja, a fábrica produz somente o necessário e nas quantidades necessárias quando requerido, evitando assim a quebra da cadeia de suprimentos da empresa e o acúmulo de itens produzidos no estoque, pois esses produtos são logos enviados ao mercado consumidor. A Figura 3.4 apresenta uma aplicação do conceito JIT em uma empresa automotiva do polo industrial de Manaus.

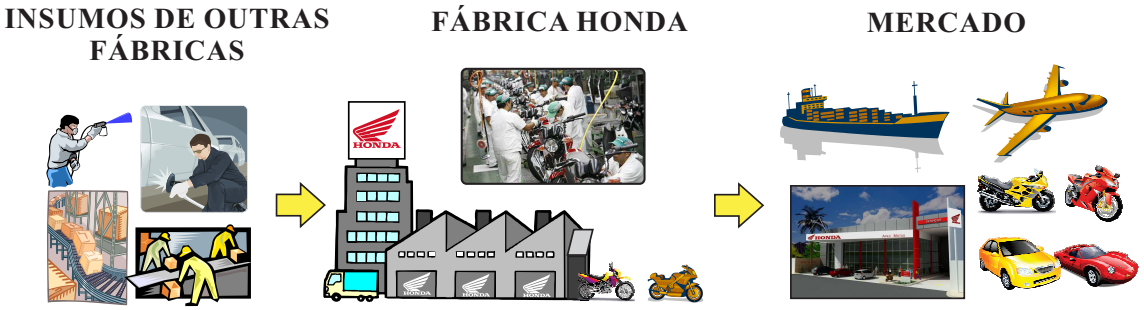


Figura 3.4: Aplicação do conceito JIT em uma empresa automotiva do Pólo Industrial de Manaus.

Na literatura, os problemas de escalonamento que envolvem penalidades de antecipação e atraso são classificados em duas categorias principais, segundo os tipos de datas de término sugeridas, de acordo como abordado em Hassin e Shani [HS05]:

- **Problemas de escalonamento com datas de término iguais (*common due dates*)**, $d_j = d$ - denotados como CDD, esse tipo de problemas são sub-divididos por aquele com **datas de término restritas** e com **datas de término irrestritas**. O problema de escalonamento é restrito, quando nenhuma tarefa pode inicializar seu processamento antes do instante zero. O problema de escalonamento é irrestrito, quando

a data de término sugerida d é suficientemente grande, por exemplo $d \geq \sum_{j=1}^{n} p_j$. Uma solução ótima para problemas de escalonamento com datas de término iguais devem satisfazer as seguintes propriedades (Baker e Scudder [BS90] e Hassin e Shani [HS05]):

1. não há tempo ocioso inserido (se a tarefa j segue imediatamente a tarefa i no escalonamento, $C_j = C_i + p_j$);
2. a sequência deve possuir uma função de crescimento em **forma de V** (*V-shaped*), ou seja, tarefas antecipadas ($C_j \leq d$) são sequenciadas em ordem não-crescente e tarefas atrasadas ($C_j > d$) são sequenciadas em uma ordem não-decrescente, não permitindo tempo ocioso entre as tarefas;
3. a $b-$ésima tarefa da sequência é concluída precisamente em sua data de término sugerida, onde b é o menor valor inteiro que satisfaz a inequação $\sum_{j=1}^{b} \alpha_j \geq \sum_{i=b+1}^{n} \beta_j$ ($C_j = d$ para alguma tarefa j).

Revisões da literatura e uma classificação dos problemas de escalonamento com penalidades de antecipação e atraso com datas de término iguais podem ser consultados em Baker e Scudder [BS90], Hall et. al. [HKS91] e Gordon et al. [GPC02].

- **Problemas de escalonamento com datas de término distintas (*distinct due dates*)**, d_j - denotados como DDD e cujas três propriedades acima descritas, que definem uma solução ótima para problemas de escalonamento com datas de término iguais, não necessariamente podem ser utilizados para problemas de escalonamento com datas de término distintas. Baker e Scudder [BS90] afirmam que nenhuma das propriedades descritas para datas de término iguais podem ser aplicadas a problemas com datas de término distintas, pois **um escalonamento ótimo para datas de término distintas pode requerer tempo ocioso entre as tarefas**. Logo, a primeira propriedade é violada e a segunda também, uma vez que tais propriedades não permitem tempo ocioso ou a utilização de datas de término distintas. A terceira propriedade não se aplica, uma vez que múltiplas datas de término não podem ser acomodados de modo a terminarem todos na mesma data de término sugerida.

  Para resolver isso, James e Buchanan [JB97] redefiniram tais propriedades para os problemas de escalonamento com datas de término distintas. Eles definiram um bloco como sendo um conjunto máximo de tarefas que são escalonados sem tempo

ocioso, onde qualquer solução ótima para o problema satisfaz as três propriedades descritas acima com modificações simples, no que diz respeito a blocos de tarefas. Uma abordagem heurística híbrida para o problema, sem tempo ocioso, envolvendo algoritmo genético, busca local e recozimento simulado pode ser consultado em Rym e M'Hallah [M'H07]. Kanet e Sridharan [KS00] apresentaram uma revisão de problemas com tempos ociosos inseridos (IIT), onde é definida uma taxonomia dos problemas com IIT.

Existem casos especiais de problemas irrestritos com penalidades de antecipação e atraso unitários, ou seja, $\alpha = \beta = 1$, para $1 \leq j \leq n$, onde deseja-se minimizar $\sum_{j=1}^{n}(E_j + T_j)$, dessa forma, o problema de escalonamento que envolve antecipação e atraso ponderados de tarefas é uma generalização desse problema, onde $\alpha_j = \alpha$ e $\beta_j = \beta$ para $1 \leq j \leq n$ e deseja-se minimizar $\sum_{j=1}^{n}(\alpha_j E_j + \beta_j T_j)$. Verma e Dessouky [VD98] consideraram o problema em ambiente monoprocessado considerando tempos de processamento unitários e casos de pesos de antecipação e atraso unitários, onde $\alpha_i = \beta_i = 1$ $\forall i \in n$, pesos de antecipação e atraso idênticos, onde a taxa α_i/β_i ou a diferença $\alpha_i - \beta_i$ é o mesmo para todas as tarefas, penalidades idênticas de antecipação, $\alpha_i = \alpha$ $\forall i \in n$ e penalidades idênticas de atraso, $\beta_i = \beta$ $\forall i \in n$. Shabtay e Steiner [SS12] apresentam uma revisão da literatura sobre problemas de escalonamento JIT, onde é apresentado o problema de maximização do número ponderado de tarefas que são finalizadas exatamente em suas datas de término sugeridas e também são apresentados vários ambientes de escalonamento com tempos de processamento fixos e variáveis.

Nas próximas sub-seções serão apresentadas as principais abordagens da literatura sobre problemas de escalonamento de tarefas com penalidades de antecipação e atraso, com tempo ocioso e não-ocioso, juntamente com trabalhos relacionados que envolvem ambientes monoprocessados e máquinas paralelas idênticas, uniformes e não-relacionadas.

3.3.1 Tempo ocioso × não-ocioso

Como visto na seção anterior, um escalonamento com datas de término iguais não possuem tempo ocioso em um escalonamento ótimo, antes que a última tarefa do escalonamento seja completada. Entretanto, no caso das tarefas com datas de término distintas, pode ser vantajoso segurar a produção por algum tempo a fim de reduzir os custos decor-

rentes a antecipação de tarefas, assim, é possível que se tenha tempo ocioso no escalonamento. Esta seção baseia-se na seguinte referência: Józefowska [Józ07].

A Figura 3.5 apresenta dois exemplos de escalonamento, sem tempo ocioso (Figura 3.5 (a)) e com tempo ocioso (Figura 3.5 (b)), considerando duas tarefas com tempo de processamento igual a 6, datas de término sugeridas $d_1 = 8$ e $d_2 = 17$, em ambiente monoprocessado. Se o tempo ocioso não é permitido (Figura 3.5 (a)), a primeira tarefa inicia o seu processamento no instante 0, e possui um tempo de completude $C_1 = 6$. Dessa forma, a segunda tarefa inicia no instante 6 e finaliza seu processamento no instante 12, $C_2 = 12$. No escalonamento da Figura 3.5 (a) as tarefas são completadas antes da data de término sugerida, o valor de antecipação associado a cada tarefa é obtido desde que $d_1 - C_1 = 2 > 0$ e $d_2 - C_2 = 5 > 0$. No escalonamento da Figura 3.5 (b) dois períodos de tempo ocioso ocorrem, como resultado, as tarefas são completadas exatamente em suas datas de término sugeridas. Assim, os custos de antecipação e atraso são nulos.

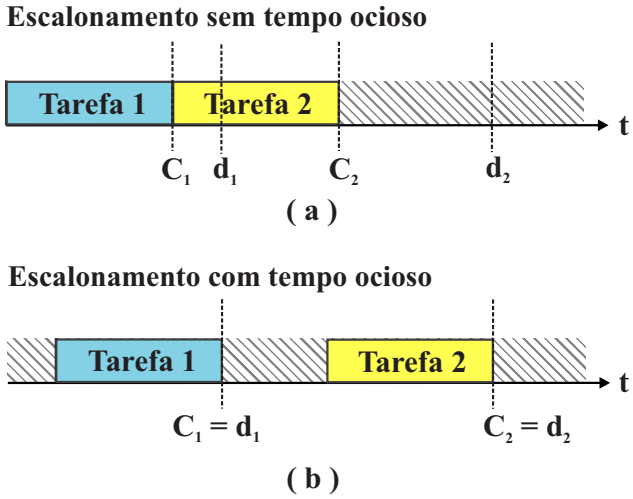


Figura 3.5: Escalonamento com tempo ocioso × não-ocioso [Józ07].

A inserção de tempo ocioso no escalonamento resulta em uma subutilização da máquina. A maximização na utilização da máquina geralmente entra em conflito com o valor de função objetivo JIT, o que pode impactar diretamente nos custos de estoque de uma empresa. Apesar do custo decorrente da existência de tempo ocioso na máquina não ser desprezível, ele é compatível com o conceito JIT quando se quer obter o menor valor de antecipação e atraso possível. Mas se o custo do tempo ocioso na máquina for elevado, pode ser necessária a aplicação de uma restrição adicional a fim de evitar tempo ocioso no escalonamento. Assim, tem-se dois grupos principais de problemas de escalonamento: problemas com tempo ocioso permitido e problemas sem tempo ocioso. O primeiro caso

é mais consistente com a ideia do escalonamento JIT, enquanto que a versão sem tempo ocioso pode ter seu uso justificado em outras aplicações.

Dependendo do problema de antecipação e atraso considerado, tem problemas de escalonamento que não permitem tempo ocioso entre as tarefas devido as restrições do problema, como as linhas de produção de uma fábrica, por exemplo, que não podem parar e devem seguir um fluxo contínuo de produção. Outro exemplo é quando se tem máquinas muito caras, que precisam ser utilizadas em uma determinada aplicação, logo, a sua utilização deve ser maximizada. Dessa forma, a restrição que evita tempo ocioso, apesar de não ser muito compatível com o conceito JIT, é também considerada na literatura.

Por outro lado, há problemas que permitem tempo ocioso no escalonamento, também devido as restrições do problema, ou seja, existem problemas que requerem que as tarefas terminem o mais próximas da data de término possível, a fim de atender a um determinado cronograma de produção ou atender as necessidades de um cliente, por exemplo. Dessa forma, com um número razoável de tarefas terminando na data de término sugerida, os custos das penalidades de antecipação e atraso tendem a ser reduzidos. Assim, pode ser mais lucrativo considerar tempo ocioso em alguns problemas.

Os problemas de escalonamento com e sem tempo ocioso podem ser encontrados na literatura com as seguintes variações de datas de término sugeridas:

- Escalonamento com tempo ocioso:

  a) Pesos arbitrários (*arbitrary weights*): $\alpha \neq \beta$.

  b) Pesos proporcionais (*proportional weights*): $\alpha_i = \alpha p_i$ e $\beta_i = \beta p_i$, $i = 1, ..., n$, $\alpha, \beta \geq 0$.

- Escalonamento sem tempo ocioso:

  a) Pesos arbitrários (*arbitrary weights*): $\alpha \neq \beta$.

  b) Pesos independentes de tarefa (*job independent weights*): $\alpha_i = \alpha$ e $\beta_i = \beta$, $i = 1, ..., n$.

3.3.2 Ambiente monoprocessado com antecipação & atraso

O problema de Escalonamento E/T em máquina única consiste em encontrar uma sequência ótima de um conjunto de n tarefas a serem escalonadas em uma única máquina. Cada

tarefa J_j possui um tempo de processamento inteiro p_j e uma data de término sugerida inteira d_j. Caso as tarefas estejam prontas para serem executadas a partir do instante zero e não houver preempção das tarefas, não é permitido tempo ocioso porque a capacidade da máquina é limitada, comparada a demanda, e não é tecnologicamente viável deixar a máquina parada por um longo tempo [M'H07].

Herrmann e Lee [HL93] resolveram o problema de escalonamento para minimizar a soma das penalidades de antecipação e atraso e os custos de entrega, onde as tarefas atrasadas são concluídas em lotes, com um custo fixo por lote, como no problema: min $\sum(\alpha_j E_j + \beta_j T_j + K y_j)$, onde α_j é a taxa de penalidade de antecipação, β_j é a taxa de penalidade de atraso, K é o custo da entrega do lote, d é uma dada data de término igual, D_j é a data de entrega da tarefa (onde $D_j \geq C_j$ e $D_j = d$ se $C_j \leq d$) e y_j = 1 se $D_j = C_j > d$ e 0 caso contrário. A estratégia algorítmica utilizada para o problema é um algoritmo pseudopolinomial de programação dinâmica.

Sourd e Kedad-Sidhoum [SKS03] lidaram com problemas envolvendo datas de término sugeridas, custos de antecipação e custos de atraso distintos e, com o objetivo de determinar o custo mínimo do problema, eles propuseram novos limites inferiores através da decomposição de cada tarefa em operadores unários, que por sua vez são atribuídos a cada instante de tempo, o que fornece um escalonamento preemptivo. Assim, dependendo de onde esses operadores unários são atribuídos na janela de tempo, gera uma custo associado no escalonamento, a atribuição que gerar o menor custo para o escalonamento é assumido como o limite inferior válido.

Um algoritmo *branch-and-bound* é proposto baseado neste limite inferior, obtendo resultados rápidos para instâncias envolvendo 8, 10, 15, 20, 25 e 30 tarefas. Em outro trabalho, os autores Sourd e Kedad-Sidhoum [SKS08] apresentam um novo algoritmo *branch-and-bound* capraz de resolver instâncias com até 50 tarefas, o algoritmo é baseado na combinação da relaxação lagrangeana de restrição de recursos e novas regras de dominância.

Uma meta-heurística híbrida foi proposta por M'Hallah [M'H07] para o problema $1|d_j|\sum E_j + \sum T_j$. Essa heurística é baseada em população, envolvendo Algoritmo Genético e a exploração do espaço de busca envolve Busca Local e Recozimento Simulado. A heurística baseada em população objetiva uma otimização global enquanto que a heurística de busca local se esforça para a otimização de um ótimo local, a otimização global

equipara-se a evolução, enquanto que a otimização local equipara-se à aprendizagem. O autor afirma que a sincronização da evolução com aprendizagem gera uma heurística híbrida eficiente. O problema $1|d_j|\sum E_j + \sum T_j$ é NP-difícil, sendo uma generalização de $1|d_j|\sum T_j$ que também é NP-difícil. O algoritmo proposto possui dois níveis de hibridização, que são: baixo nível (onde é verificada a habilidade do Algoritmo Genético, de atingir boas soluções, substituindo a operação genética de mutação pela busca local) e o alto nível (onde são utilizadas três heurísticas gulosas para gerar uma população inicial, e a população é refinada a cada geração pelo algoritmo de Recozimento Simulado). A abordagem além de gerar resultados promissores em um tempo razoável, atingindo o ótimo na maioria dos casos testados, também pode ser estendido para outros problemas que envolvem antecipação e atraso.

Shabtay e Steiner [SS08] apresentam um estudo envolvendo os dois tipos de abordagem para datas de término sugeridas citadas anteriormente: datas de término iguais (*common due dates*) e datas de término irrestritas (*unrestricted due dates*). Com base nestes dois, trabalharam também com a ideia de **datas de término com folga** (*slack due dates*), geralmente referenciado como SLK, onde as datas de término de todas as tarefas recebem folgas (*slk*), o que reflete a um mesmo tempo de espera para todas as tarefas, ou seja, $d_j = p_j + slk$ para $j = 1, ..., n$, onde $slk \geq 0$ é uma variável de decisão.

Kedad-Sidhoum et al. [KSSS08] propuseram duas novas famílias de limites inferiores para o problema de escalonamento em que as tarefas possuem datas de término distintas com custos de antecipação e atraso. Primeiro foi feita uma generalização de duas atribuições de limites inferiores para o problema com máquinas monoprocessadas e máquinas paralelas, e segundo, foi investigado uma formulação indexada por tempo para o problema a fim de se obter eficientes limites inferiores através da geração de colunas e relaxação lagrangeana. Para a geração de limite superior, foi apresentado um algoritmo de busca local.

Um algoritmo *branch-and-bound* para o problema de escalonamento com penalidades de antecipação e atraso pode ser consultado em Tanaka et al. [TSA03], onde o procedimento começa com uma solução inicial é obtida através do algoritmo de troca de pares adjacentes (*adjacent pairwise interchange* - API), Figura 3.6, esse algoritmo sequencia a solução inicial ordenando as tarefas a partir do menor para a maior data de término sugerida seguido de movimentos API até que a solução não possa mais ser melhorada.

Tanaka et al. [TFA09] também consideraram um algoritmo de programação dinâmica para solucionar os problema de escalonamento com antecipação e atraso ponderados de tarefas, apresentando resultados para até 300 tarefas. Uma outra versão desse algoritmo é apresentada em Tanaka et al. [Tan12b], onde o algoritmo começa o seu processamento através da relaxação lagrangeana do problema original e, então, restrições são adicionadas até que a diferença entre o limite inferior e superior se torne zero. As relaxações são resolvidas pela programação dinâmica, e os estados desnecessários do algoritmo de programação dinâmica são eliminados no decorrer da execução do algoritmo, a fim de evitar o crescimento excessivo de estados causados pela adição de restrições.

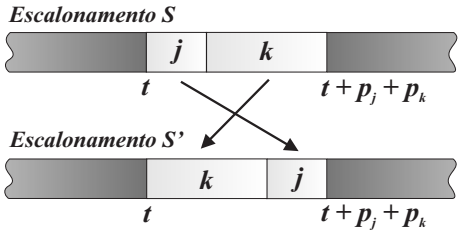


Figura 3.6: Exemplo do método de troca de pares adjacentes para as tarefas j e k. Adaptado de Pinedo [Pin12].

Diferentes estratégias aproximadas envolvendo o ambiente monoprocessado pode ser consultado em Sourd [Sou06], que considerou o método de busca em vizinhança chamado de *dynasearch* ou busca em vizinhança muito grande, onde a vizinhança é obtida através da composição de várias operações de troca de posições do escalonamento e, como a vizinhança é exponencial, um algoritmo de programação dinâmica foi proposto a fim de se obter a solução ótima da vizinhança. Júnior e Carvalho [JC07], considerando janelas de entrega e tempo de preparação da máquina dependente da sequência de produção, propuseram um método heurístico baseado em GRASP, busca local iterada e descida em vizinhança variável (*variable neighborhood descent*), onde a heurística possui dois passos: o primeiro é a determinação da sequência de tarefas e o segundo passo é a determinação da data ótima de início processamento de cada tarefa na sequência.

Penna et al. [PSdCAGO12] consideraram o escalonamento de tarefas com penalidades de antecipação e atraso com janelas de entrega distintas (onde há um período para a conclusão de cada tarefa), onde utilizou-se um algoritmo heurístico baseado em três fases, a primeira delas diz respeito ao algoritmo GRASP para a geração de uma solução inicial viável, a segunda fase é baseada em busca tabu a fim de se seja refinada a solução encontrada e a terceira fase conta com o algoritmo de reconexão de caminhos como estra-

tégia de pós-otimização. O algoritmo foi testado com até 75 tarefas e foi comparado com algoritmos da literatura, superando todos eles. Um algoritmo genético em ambiente monoprocessado foi proposto por Yousefi et al. [YY12], onde foi considerado um estudo de caso de uma aplicação do algoritmo na indústria, onde são consideradas tarefas que possuem datas de término iguais, e o algoritmo é aplicado para o escalonamento de produtos em uma linha de produção.

3.3.3 Máquinas paralelas com antecipação & atraso

Os problemas de escalonamento que envolvem restrições de antecipação e atraso consistem em achar a sequência ótima para um dado conjunto de n tarefas $N = \{J_1, J_2, ..., J_n\}$ com tempos de processamento $p_j (j = 1, 2, ..., n)$ a serem processadas em: **máquinas paralelas idênticas** P (cada máquina possui a mesma velocidade, e tempo de processamento para a tarefa j é p_j); **máquinas paralelas uniformes** Q (cada máquina i possui a sua própria velocidade s_i, e o tempo de processamento da tarefa j nessa máquina será $p_{ij} = p_j/s_i$) ou **máquinas paralelas não-relacionadas** R (a velocidade da máquina é dependente da tarefa a ser executada, e o tempo de processamento da tarefa j na máquina i é $p_{ij} = p_j/s_{ij}$). Os casos de *job shop* (máquinas diferentes e em ordem diferente para cada tarefa), *flow shop* (onde se tem uma sequência de máquinas para cada tarefa) e *open shop* (onde cada máquina opera exatamente uma vez em cada sequência, diferentes para cada tarefa) também são detalhados na literatura [LW04]. Kravchenko e Werner [KW11] apresentam uma revisão da literatura envolvendo problemas de escalonamento em máquinas paralelas.

3.3.3.1 Máquinas paralelas idênticas

Nesta subseção, são apresentados os principais trabalhos encontrados na literatura envolvendo ambientes de máquinas paralelas idênticas (P). Ressalta-se que os trabalhos encontrados que envolvem máquinas paralelas uniformes e não-relacionadas também se adequam, a menos de algumas adaptações necessários em alguns casos, ao caso de ambiente em máquinas paralelas idênticas. Ainda assim, não foram encontrados muitos trabalhos para este caso específico somente, e os dois principais encontrados são apresentados a seguir.

Kedad-Sidhoum et al. [KSSS08], resolvem o problema $P|r_j|\sum_j \alpha_j E_j + \beta_j T_j$ utilizando

programação inteira, através de uma estratégia algorítmica combinando geração de colunas, relaxação lagrangeana e heurística de busca local.

Xiao e Li [XL02] propuseram duas heurísticas para minimizar o problema $nP_Dd + \sum_{j=1}^{n}(P_E E_j + P_T T_j)$, onde as penalidades que são aplicadas às tarefas são: P_E que representa as penalidades de antecipação, P_T que representa as penalidades de atraso e nP_Dd que é aplicado quando há datas de término iguais. A primeira heurística desenvolvida minimiza a somatória ponderada das datas de término sugeridas, tempo total de antecipação e tempo total de atraso. A segunda heurística desenvolvida possui a melhor performance no pior caso para o caso de penalidades de antecipação antecipadas, ou seja, corresponde a situação onde ocorre o custo de armazenamento por causa das tarefas finalizadas antes da data de término sugerida, que é desprezível se for comparado com as datas de término sugeridas e penalidades sobre tarefas atrasadas.

3.3.3.2 Máquinas paralelas uniformes

Nesta subseção, são apresentados os principais trabalhos encontrados na literatura envolvendo ambientes de máquinas paralelas uniformes (Q).

Tuong e Soukhal [TS08] consideraram problemas de escalonamento com n tarefas com tempos iguais de processamento que devem ser escalonados em m máquinas uniformes, onde l é uma dada data de término sugerida $(l < n)$, foi mostrado que o problema de escalonamento em m máquinas uniformes é solucionável em tempo polinomial.

Soukhal e Toung [ST12] estudaram problemas com datas de término iguais de antecipação e atraso em ambiente monoprocessado e máquinas paralelas em que o tempo de processamento das tarefas são idênticos, onde dois casos de datas de término sugeridas foram considerados. No primeiro caso, é o caso em que a data de término sugerida é suficientemente adiantada (versão restritiva) ou não (versão não-restritiva) para restrição de solução ótima, o segundo caso lida com datas de término desconhecidas que são as variáveis de decisão. É feita uma demonstração que o problema $Q_m|p_j = p, d_j = d$, não restritivo $|\sum(\alpha E_i + \beta T_i)$ pode ser executado em tempo polinomial, $O(m^2)$, com penalidades unitárias para antecipação e atraso.

3.3.3.3 Máquinas paralelas não-relacionadas

Nesta subseção, são apresentados os principais trabalhos encontrados na literatura envolvendo ambientes de máquinas paralelas não-relacionadas (R).

Sung e Vlach [SV05] consideraram os problemas de escalonamento não-preemptivos com o objetivo de maximizar o número ponderado de tarefas que são completadas exatamente em duas datas de término sugeridas. Foi apresentado um algoritmo de programação dinâmica de tempo polinomial quando o número de máquinas é fixo e posteriormente é mostrado que se o número de máquinas faz parte da entrada, o problema se torna NP-Difícil.

Beyranvand et al. [BPG11] lidaram com este problema com datas de término restritivas, na literatura, este problema é denotado por $R|d_i = d^r|\sum_i \alpha_i E_i + \beta_i T_i$, onde α_i e β_i são as penalidades de antecipação e o atraso, respectivamente, E_i e T_i são os valores de antecipação e atraso, respectivamente, d^r é a data de término restritiva, isso significa que todas as tarefas devem idealmente serem concluídos, ou seja, existe pelo menos uma máquina m com $\sum_j p_{ij} > d^r$. Para solucionar esse problema de escalonamento, foi proposto o modelo de programação quadrática com restrições lineares com o objetivos de se obter soluções ótimas para o problema.

Vallada e Ruiz [VR12] consideraram o problema de escalonamento em máquinas não-relacionadas, eles estudaram formulações matemáticas existentes, como o modelo de programação inteira mista (PIM) e um algoritmo genético foi proposto para o problema $R|S_{ijk}|\sum_{j=1}^{n}(w_j^{'}E_j + w_j T_j)$. No problema estudado, $w_j^{'}$ é o peso da antecipação ou prioridade e w_j é o peso do atraso ou prioridade, não necessariamente igual a $w_j^{'}$. O tempo de preparação (*setup*) S_{ijk} é fixo e não-negativo, sendo o tempo necessário para a máquina i, $\forall i \in M$, processar a tarefa j quando a tarefa k for a próxima da sequência, $\forall j$, k, $j \neq k$, $\in N$.

Estratégias aproximadas podem ser consultadas em Nogueira et al. [NAG12], onde foi proposta uma heurística híbrida baseada na meta-heurística GRASP e reconexão de caminhos, de maneira que, para cada solução gerada pela heurística é utilizado um algoritmo de tempo polinomial para determinar a data ótima de início das tarefas, são apresentados resultados para 8, 9, 10, 20 e 30 tarefas.

Tabela 3.3: Classificação dos problemas de escalonamento com antecipação e atraso de tarefas.

Ambiente	Problemas	Estratégias de resolução	Referências
1	$1\|s-batch,d_j=d\|\sum(\alpha_jE_j+\beta_jT_j+Ky_j)$, Ky_j é o custo da entrega do lote	Programação dinâmica	Herrmann e Lee [HL93]
	$1\|\|\sum\alpha_iE_i+\sum\beta_iT_i$	Programação dinâmica	Tanaka et al. [TFA09], Tanaka et al. [Tan12b]
	$1\|ST_{sd}\|\sum w_jE_j+\sum w_jT_j$	Busca tabu	Kolahan e Liang [KL98]
	$1\|ST_{sd}\|\sum w_jE_j+\sum w_jT_j$	Método de busca *Dynasearch*	Sourd [Sou06]
	$1\|\|\sum\alpha_jE_j+\sum\beta_jT_j$	Algoritmo *branch-and-bound*	Sourd e Kedad-Sidhoum [SKS03], [Sou09]
	$1\|r_j\|\sum\alpha_jE_j+\sum\beta_jT_j$	Algoritmo *branch-and-bound*	Sourd e Kedad-Sidhoum [SKS08]
	$1\|ST_{sd}\|\sum E_j+\sum T_j$	Algoritmo *branch-and-bound*	Rabadi [RMA04]
	$1\|d_j\|\sum E_j+\sum T_j$	Algoritmo genético híbrido envolvendo busca local e recozimento simulado	M'Hallah [M'H07]
	$1\|ST_{sd},s_{ij}\|\sum_{j=1}^{n}(\alpha_iT_j+\beta_jE_j)$	GRASP, busca local iterada e descida em vizinhança variável	Carvalho [JC07]
	$1\|d_i\|\sum w_iE_i+\sum h_iT_i$	Algoritmo *branch-and-bound*	Li [Li97], Rebai e Kacem [RK08]
	$1\|\|\sum\alpha_iE_i+\sum\beta_iT_i$	Algoritmo *branch-and-bound*	Tanaka et al. [TSA03]
	$1\|ST_{sd},E_i\leq d_i\leq T_i\|\sum_{i=1}^{n}(\alpha_iE_i+\beta_iT_i)$, onde $E_i\leq d_i\leq T_i$ indica a janelas de entrega distintas	GRASP, busca tabu e reconexão de caminhos	Penna et al. [PSdCAGO12]
	$1\|d_j=d\|\sum_{i=1}^{n}(\alpha_iE_i+\beta_iT_i)$	Algoritmo genético	Yousefi et al. [YY12]
P	$P\|\|nP_Dd+\sum_{j=1}^{n}(P_EE_j+P_TT_j)$, nP_Dd é uma penalidade aplicada quando há datas de término iguais	Abordagem heurística baseada em formulação matemática	Xiao e Li [XL02]
	$P\|r_j\|\sum_j\alpha_jE_j+\beta_jT_j$	Programação inteira, geração de colunas, relaxação lagrangeana e busca local	Kedad-Sidhoum et al. [KSSS08]
Q	$Q_m\|p_i=p,d_i\in D,\|D\|\leq l,l$ fixo $\|\sum(\alpha_iE_i+\beta_iT_i)$	Prova matemática que o problema é solucionável em tempo polinomial	Tuong e Soukhal [TS08]
	$Q_m\|p_j=p$, $d_j=d$, não-restritivo $\|\sum(\alpha E_i+\beta T_i)$	Prova matemática que o problema é solucionável em tempo polinomial	Soukhal e Toung [ST12]
R	$R_m\|\|w_j(\sum E_j+\sum T_j)$	Programação dinâmica	Sung e Vlach [SV05]
	$R\|s_{ijk}\|\sum_{j=1}^{n}(w_j^{'}E_j+w_jT_j)$	Algoritmo genético	Vallada e Ruiz [VR12]
	$R\|s_{ijk}\|\sum_{j=1}^{n}(\alpha_iT_j+\beta_jE_j)$	GRASP com reconexão de caminhos	Nogueira et al. [NAG12]

3.4 Critério de desempate

Durante o processo de busca por melhores soluções, há grande possibilidade que se encontre muitas soluções diferentes mas de mesmo custo em relação à função objetivo que penaliza tarefas ponderadas com antecipação ou atraso. Assim, faz-se necessário a escolha de um critério sobre o qual uma solução deve ser escolhida ou priorizada uma solução em relação a outra de igual valor de função objetivo. Rodrigues et al. [dFPUP08] propuseram uma estratégia de **Critério de Desempate** (CD) para o problema de escalonamento com função objetivo que penaliza o atraso de tarefas ponderadas e que permite escolher, entre duas soluções com o mesmo valor de função objetivo, qual solução deve ser considerada a melhor solução. O critério de desempate proposto se baseia em datas de término sugeridas (*due dates*) das tarefas escalonadas e na posição das mesmas na sequência do escalonamento (solução), priorizando aqueles escalonamentos que possuem tarefas com datas de término menores escalonadas primeiro no escalonamento. Tal critério de desempate foi motivado pela política de escalonamento bem conhecida que consiste em escalonar primeiro as tarefas de datas de término menores (*earliest due date first rule*), o que é uma regra ótima - gera a solução ótima - para o problema $1||\sum w_j T_j$ quando não existirem tarefas tardias (quando o valor da função objetivo for zero).

Formalmente, dado um escalonamento representado por uma sequência de tarefas $\pi = (\pi_1, ..., \pi_n)$, define-se $b(\pi)$ como uma pontuação da permutação π, que será dada por:

$$b(\pi) = \sum_{j=1}^{n} d_{\pi_j} \cdot (n - j + 1). \tag{3.1}$$

Assim, dada duas sequências π e ρ de tarefas com o mesmo custo de função objetivo, se $b(\rho) < b(\pi)$, então ρ é considerada melhor solução que π.

Neste trabalho, o problema de escalonamento principal abordado envolve não somente a penalização de atraso, mas simultaneamente, a penalização de antecipação e de atraso de tarefas ponderadas. Ainda assim, como a região de soluções viáveis para o problema permanece a mesma e a nova função objetivo é uma composição envolvendo a função anterior, pode ser observado através de uma análise simples e posteriormente comprovados por análise empírica, que empates entre soluções continuam ocorrendo em grande número considerando esta nova função objetivo.

Sendo assim, tal critério de desempate foi aplicado no processo de busca local pro-

posto neste trabalho, onde frequentemente soluções vizinhas no espaço de busca possuem custos iguais apesar de serem soluções diferentes e, então, o processo de busca local pode ser interrompido prematuramente, sem realmente devolver um ótimo local que expressa uma solução de melhor qualidade. Assim, como será apresentado posteriormente, tal critério de desempate foi adotado na heurística de busca local implementada, bem como alguns resultados da análise de ocorrência de empates segundo este critério são apresentados.

Uma questão interessante não explorada neste trabalho, seria adequar uma política de escalonamento mais adequada para tal função composta, de tal forma a favorecer casos em que ocorra um índice de antecipação muito grande. Isto não está sendo tratado neste trabalho e apenas o atraso está sendo priorizado como critério de desempate. Outra estratégia interessante a ser explorada seria a determinação de soluções simétricas no espaço de busca, o que pode representar um considerável ganho de desempenho na execução de estratégias exatas e aproximadas, a seção seguinte apresenta um estudo inicial sobre simetria em escalonamento de tarefas.

3.5 Identificando simetria

Além de existirem muitas soluções diferentes com mesmo valor de função objetivo, é possível a existência de simetria no escalonamento, ou seja, soluções que a princípio são diferentes, envolvendo tarefas alocadas em diferentes máquinas e posições no tempo, mas que na verdade são similares em relação a estrutura geral do problema, de tal forma que bastaria considerar uma delas em um processo algorítmico de geração de soluções. Em suma, a identificação e remoção de tais soluções simétricas pode reduzir consideravelmente o esforço computacional de uma determinada estratégia algorítmica, evitando o esforço repetitivo do método em recalcular soluções equivalentes desnecessárias no processo de busca.

Neste trabalho, o estudo da ocorrência de simetria nos problemas de escalonamento em análise, propiciou a detecção de **blocos de tarefas simétricos** no escalonamento considerando máquinas paralelas idênticas. Dado um escalonamento, considera-se aqui que um **bloco de tarefas** consiste em uma janela de tempo no escalonamento cujas tarefas contidas são completamente executadas neste intervalo (começam e terminam suas exe-

cuções nesta janela de tempo). Sendo assim, um bloco de tarefas em um escalonamento é simétrico a outro bloco, de outro escalonamento, se contém as mesmas tarefas escalonadas, mesmo estando em máquinas e ordem diferentes. Conceitualmente, em tais blocos simétricos pode existir tempo ocioso entre as tarefas escalonadas ou não.

Detectar blocos simétricos é uma outra forma de refinar o processo de obtenção de soluções, podendo melhorar o desempenho dos algoritmos relacionados. Uma solução (um mesmo escalonamento de tarefas) pode possuir um ou vários blocos simétricos. A Figura 3.7 apresenta um exemplo de blocos simétricos no escalonamento, onde pode-se observar duas soluções diferentes para o problema, mas que possuem o mesmo custo e valores diferentes dados pelo critério de desempate. A Figura 3.7 (a) apresenta um exemplo de instância de 8 tarefas para o problema de escalonamento com penalidades de antecipação e atraso. Baseado nesta instância, são apresentadas duas soluções ótimas para o problema: a primeira delas contém dois blocos simétricos no escalonamento, ilustrando a possibilidade da existência de vários blocos simétricos no escalonamento, Figura 3.7 (b); a segunda delas apresenta uma solução ótima para o problema sem blocos simétricos no escalonamento, mas todo o escalonamento pode ser tratado como um único bloco simétrico, Figura 3.7 (c).

A Figura 3.8 (b) apresenta seis exemplos de soluções simétricas, que equivalem a solução ótima para a instância da Figura 3.8 (a), onde pode-se observar que apesar das soluções serem equivalentes, o valor de critério de desempate associado a cada solução é diferente. Dessa forma, a simetria lida apenas com a identificação de soluções equivalentes de modo que elas não precisem mais ser recalculadas no escalonamento, e o critério de desempate lida somente com soluções de mesmo custo no escalonamento e define qual das duas soluções com empate deve ser escolhida, escolhendo assim a solução que possuir o menor valor de critério de desempate.

O estudo de ocorrência de simetria em problemas é foco de estudo há bastante tempo e recentemente vem ganhando uma importância ainda maior, principalmente no que tange ao desenvolvimento de métodos eficientes para a resolução de problemas computacionalmente difíceis, com o intuito justamente de minimizar o esforço necessário destes métodos na obtenção de boas soluções. Particularmente sobre problemas de escalonamento, Ostrowski et al. [OAV10] apresentaram um estudo sobre remoção de simetria em problemas de escalonamento utilizando programação linear inteira e mostraram que es-

J_j	p_j	d_j	α_j	β_j	C_j	E_j	T_j	$\alpha_j E_j$	$\beta_j T_j$
J_1	3	3	3	5	3	0	0	0	0
J_2	2	6	4	5	6	0	0	0	0
J_3	3	6	8	8	6	0	0	0	0
J_4	4	4	8	10	4	0	0	0	0
J_5	6	6	6	4	6	0	0	0	0
J_6	7	10	7	3	13	0	3	0	9
J_7	3	11	4	2	9	1	0	4	0
J_8	3	8	5	8	9	0	1	0	8
						$\sum\alpha_j E_j$ e $\sum\beta_j T_j$ =		4	17

(a)

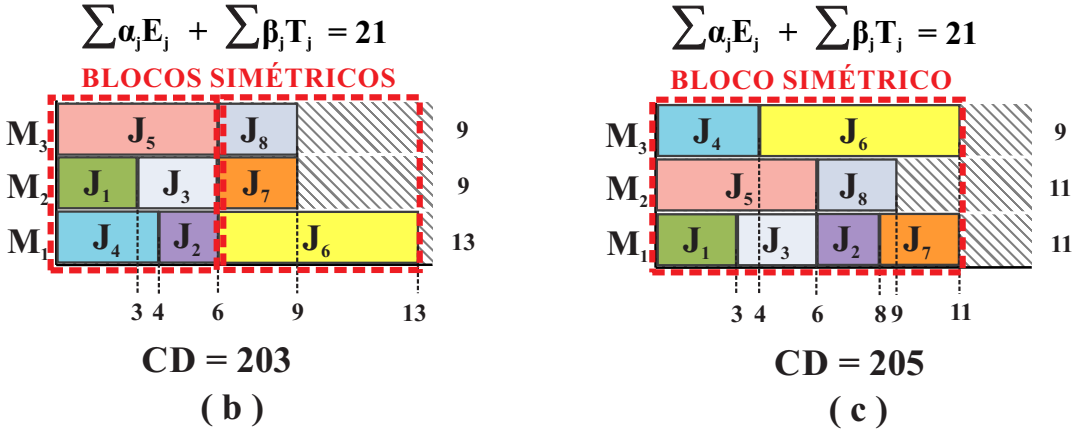
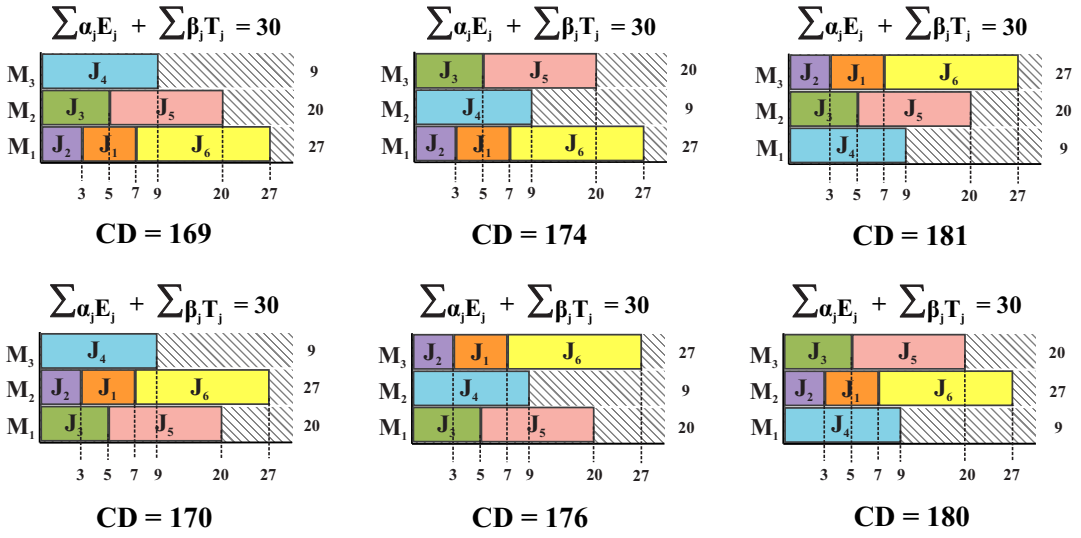


Figura 3.7: (a) Exemplo de uma instância de 8 tarefas para o problema de escalonamento com penalidade de antecipação e atraso com uma solução ótima que contém um bloco simétrico no escalonamento (b) e outra solução ótima de mesmo custo sem blocos simétricos (c) com seus respectivos valores de antecipação e atraso e valores de critério de desempate.

tratégias de remoção de simetria podem representar um enorme ganho de desempenho nos métodos exatos. Os autores afirmam que a presença de simetria pode representar um efeito negativo no desempenho dos algoritmos exatos, citando pontualmente o método de enumeração implícita de *branch-and-bound*, pois existem muitos subproblemas na árvore de *branch-and-bound* que são equivalentes. Assim, é importante que esses subproblemas equivalentes sejam detectados, pois caso contrário, milhares de subproblemas desnecessários podem ser calculados, tornando problemas relativamente fáceis impossíveis de resolver através da técnica de *branch-and-bound*. Para evitar que isso aconteça, as soluções equivalentes e subproblemas devem ser identificados e removidos da busca.

J_j	p_j	d_j	α_j	β_j	C_j	E_j	T_j	$\alpha_j E_j$	$\beta_j T_j$
J_1	4	7	3	5	7	0	0	0	0
J_2	3	4	4	5	3	1	0	4	0
J_3	5	5	8	8	5	0	0	0	0
J_4	9	10	8	10	9	1	0	8	0
J_5	15	17	6	4	20	0	3	0	12
J_6	20	25	7	3	27	0	2	0	6
							$\sum\alpha_j E_j$ e $\sum\beta_j T_j$ =	12	18

(a)



(b)

Figura 3.8: Possíveis soluções ótimas simétricas para a instância apresentada em (a) com seus respectivos valores de função objetivo ($\sum \alpha_j E_j + \sum \beta_j T_j$) e valores de critérios de desempate (CD) apresentados em (b).

3.6 Formulações matemáticas

Nesta seção serão detalhadas as principais formulações matemáticas existentes na literatura para o problema de escalonamento clássico com penalidades de antecipação e atraso (tarefas independentes, com tempos arbitrários de processamento, sem preempção permitida, com datas de chegada iguais e datas de término distintas). Sendo assim, duas questões ajudam na classificação das formulações matemáticas existentes:

1. permitem tempo ocioso no escalonamento ou não;

2. consideram escalonamento em máquinas paralelas ou somente em uma única máquina.

Kanet e Sridharan [KS00] apresentaram uma revisão da literatura sobre problemas de escalonamento com tempos ociosos inseridos, onde são considerados problemas de escalonamento que envolvem penalidades de antecipação e atraso, onde resumem os resultados gerais sobre quando considerar ou não tempo ocioso em problemas de escalonamento no geral. Assim, afirmam que não é necessário considerar ou tratar tempo ocioso nos dois seguintes casos:

1. Problemas de escalonamento em ambiente monoprocessado.

2. Problemas de escalonamento em máquinas paralelas, quando todas as tarefas do problema estão simultaneamente disponíveis (quando se tem datas de chegadas iguais) e quando o problema de escalonamento apresentar medidas regulares de performance.

3.6.1 Formulação de programação linear inteira mista para o escalonamento em máquinas paralelas com tempo ocioso (PIM-TO)

A primeira formulação considerada é o modelo de programação linear inteira mista (PIM) proposto por Arenales et al. [AMAY07] para o problema de escalonamento em máquinas paralelas. A formulação proposta assume que: todas as tarefas devem estar disponíveis no instante zero (datas de chegadas iguais, $r_j = 0$, sem perda de generalidade); cada tarefa possui a sua própria data de término sugerida (d_j distintos); qualquer máquina pode processar qualquer tarefa; cada tarefa pode ser executada somente uma vez; a preempção de uma tarefa não é permitida; cada máquina pode processar exatamente uma tarefa em um dado instante de tempo; o número de tarefas e máquinas são fixos e os tempos de processamento também são fixos. E apresenta tempos de preparação relacionadas a máquina e tarefas consecutivamente escalonadas.

A variável de decisão inteira é binária e tri-indexada, sendo: x_{ijk} que terá o valor 1 se a tarefa i precede imediatamente a tarefa j na máquina k, 0 caso contrário. As outras variáveis, que podem receber valores reais, são: C_{ik} que é o instante de término do processamento da tarefa i na máquina k; e, T_i e E_i representam o atraso e a antecipação da tarefa i, respectivamente.

A formulação matemática também apresenta os seguintes parâmetros não-negativos:

- p_{ik} é o tempo de processamento da tarefa i na máquina k (para máquinas paralelas idênticas a notação muda para p_j);
- s_{ijk} é o tempo de preparação da máquina k para processar a tarefa j imediatamente após a tarefa i (caso o problema não tenha tempos de preparação, assume-se o valor zero para o tempo de preparação da máquina);
- d_j é a data de entrega da tarefa j; e, M é uma constante suficientemente grande (conhecida pelo termo em inglês, *big M*).

A seguir é apresentada a formulação matemática em programação inteira mista, com restrições e função objetivo lineares.

$$\text{Min} \sum_{j=1}^{n} (E_j + T_j) \tag{3.2}$$

$$\text{S. a.} \sum_{k=1}^{m} \sum_{i=0}^{n} x_{ijk} = 1 \qquad j = 1,2,...,n. \tag{3.3}$$

$$\sum_{j=1}^{n} x_{0jk} \leq 1 \qquad k = 1,2,...,m. \tag{3.4}$$

$$\sum_{i=0,\ i\neq h}^{n} x_{ihk} - \sum_{j=0,\ j\neq h}^{n} x_{hjk} = 0 \qquad h = 1,...,n \text{ e } k = 1,...,m. \tag{3.5}$$

$$C_{0k} = 0 \qquad k = 1,...,m. \tag{3.6}$$

$$C_{jk} \geq C_{ik} - M + (p_{jk} + s_{ijk} + M)x_{ijk} \quad i = 0,...,n; j = 1,...,n \text{ e } k = 1,...,m. \tag{3.7}$$

$$E_i \geq d_i - C_{ik} \qquad i = 1,...,n \text{ e } k = 1,...,m. \tag{3.8}$$

$$T_i \geq C_{ik} - d_i \qquad i = 1,...,n \text{ e } k = 1,...,m. \tag{3.9}$$

$$T_i \geq 0, E_i \geq 0 \qquad i = 1,...,n. \tag{3.10}$$

$$x_{ijk} \in \{0,1\} \qquad i,j = 0,...,n \text{ e } k = 1,...,m. \tag{3.11}$$

Na formulação acima, a função objetivo 3.2 minimiza a antecipação e o atraso das tarefas. As restrições 3.2, 3.3, 3.4 asseguram uma sequência exclusiva de tarefas para cada uma das máquinas. Sendo que o conjunto de restrições 3.3 indicam que cada tarefa j na máquina k tenha apenas uma tarefa precedente. O conjunto de restrições 3.4 garantem que cada máquina k, se usada, tenha uma sequência exclusiva de tarefas. O conjunto de restrições 3.5 definem que cada tarefa j tenha uma única tarefa imediata sucessora, com

exceção da tarefa 0 que estabelece o início e o fim de uma sequência de tarefas na máquina k. Para a tarefa 0, o conjunto de restrições 3.6 estabelecem que o instante de termino das tarefas nas máquinas devem ser iguais a zero. No conjunto de restrições 3.7 são verificados os instantes de conclusão das tarefas nas máquinas onde são executadas. Nas restrições 3.8 e 3.9, são definidos, respectivamente, a antecipação e o atraso associado a cada uma das tarefas. As restrições 3.10 e 3.11 indicam o domínio das variáveis utilizadas na formulação.

As restrições acima descritas não excluem a ocorrência de tempo ocioso entre as tarefas, como pode-se observar na Figura 3.9 (a), que apresenta uma solução viável para a instância da Figura 3.8 (a) e mostra que a aplicação da restrição 3.7 (Figura 3.9 (b)) permite tempo ocioso.

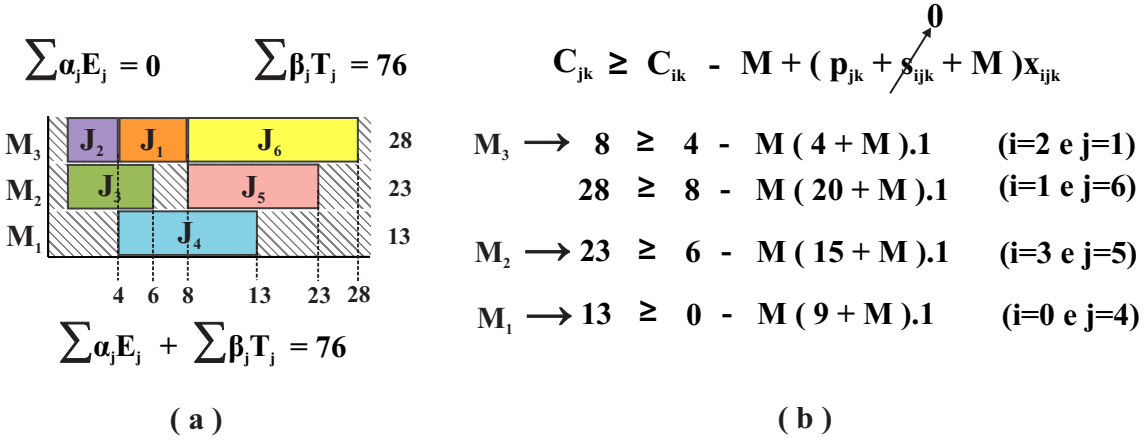


Figura 3.9: Solução do escalonamento de tarefas com tempo ocioso (a) e aplicação da restrição 3.7 para a solução (b).

3.6.2 Formulação de programação linear inteira mista para o escalonamento em máquinas paralelas sem tempo ocioso (PIM-STO)

A segunda formulação considerada neste trabalho é o modelo de PIM proposto por Tsai e Wang [TW12] para máquinas paralelas não-relacionadas ($R||\sum_{i=1}^{n} T_j + \sum_{i=1}^{n} E_j$) baseado na formulação apresentada por Rabadi et al. [RMAS06] para a minimização do *makespan* em máquinas paralelas não-relacionadas. A formulação proposta por Tsai e Wang [TW12] também assume que: a formulação proposta assume que: todas as tarefas devem estar disponíveis no instante zero (datas de chegadas iguais, $r_j = 0$, sem perda de generalidade); cada tarefa possui a sua própria data de término sugerida (d_j distintos);

qualquer máquina pode processar qualquer tarefa; cada tarefa pode ser executada somente uma vez; a preempção de uma tarefa não é permitida; cada máquina pode processar exatamente uma tarefa em um dado instante de tempo; o número de tarefas e máquinas são fixos e os tempos de processamento também são fixos e apresenta tempos de preparação relacionadas a máquina e tarefas consecutivamente escalonadas.

Apresenta a mesma variável de decisão inteira binária e tri-indexada, sendo: x_{ijk} uma variável binária que terá valor 1 se a tarefa j é processada diretamente depois da tarefa i na máquina k, 0 caso contrário (com os casos especiais para: x_{0jk},variável binária que terá valor 1 se a tarefa j é a primeira a ser processada na máquina k, 0 caso contrário, e x_{i0k}, variável binária que terá valor 1 se a tarefa j é a última a ser processada na máquina k, 0 caso contrário. As variáveis contínuas são: C_j, tempo de completude da tarefa j; T_j, atraso da tarefa j, e E_j, antecipação da tarefa j.

A formulação matemática também apresenta os seguintes parâmetros não-negativos: n**:** número de tarefas; m**:** número de máquinas; p_{jk}**:** tempo de processamento da tarefa j na máquina k; d_j**:** data de término sugerida da tarefa j; e, M**:** uma constante suficientemente grande.

A seguir é apresentada a formulação matemática em programação inteira mista, com restrições e função objetivo lineares.

$$\text{Min} \sum_{j=1}^{n} E_j + \sum_{j=1}^{n} T_j \tag{3.12}$$

$$\text{S. a.} \sum_{i=0,\ i\neq j}^{n} \sum_{k=1}^{m} x_{ijk} = 1 \qquad j = 1,2,...,n. \tag{3.13}$$

$$\sum_{i=0,\ i\neq h}^{n} x_{ihk} - \sum_{j=1,\ j\neq h}^{n} x_{hjk} = 0 \qquad h = 1,2,...,n;\ \ k = 1,2,...,m. \tag{3.14}$$

$$C_i + \sum_{k=1}^{m} x_{ijk}(p_{jk}) + M(\sum_{k=1}^{m} x_{ijk} - 1) \leq C_j \qquad i = 0,1,...,n;\ \ j = 1,2,...,n. \tag{3.15}$$

$$\sum_{j=0}^{n} x_{0jk} = 1 \qquad k = 1,2,...,m. \tag{3.16}$$

$$T_j = C_j - d_j \qquad j = 1,2,...,n. \tag{3.17}$$

$$E_j = d_j - C_j \qquad j = 1,2,...,n. \tag{3.18}$$

$$C_0 = 0 \tag{3.19}$$

$$C_j, E_j, T_j \geq 0 \qquad j = 1,2,...,n. \tag{3.20}$$

$$x_{ijk} \in \{0,1\} \quad i,j = 1,2,...,n; \;\; k = 1,2,...,m. \tag{3.21}$$

$$C_0 + \sum_{k=1}^{m} x_{0jk}(p_{jk}) \geq C_j + M(\sum_{k=1}^{m} x_{0jk} - 1) \qquad j = 1,2,...,n. \tag{3.22}$$

$$C_i + \sum_{k=1}^{m} x_{ijk}(p_{jk}) \geq C_j + M(\sum_{k=1}^{m} x_{ijk} - 1) \qquad i = 0,1,...,n; \;\; j = 1,2,...,n. \tag{3.23}$$

Para adaptar o modelo para o problema envolvendo máquinas paralelas idênticas (e não o caso geral, de não-relacionadas), $P||\sum_{i=1}^{n} \alpha_j E_j + \sum_{i=1}^{n} \beta_j T_j$, a notação referente ao tempo de processamento das tarefas deve ser mudada de p_{jk} (que indica que o tempo de processamento da tarefa j será processada na máquina k) para somente p_j (já que as máquinas são idênticas e não possuem desempenhos dependentes da tarefa a ser executada) e as penalidades de antecipação e atraso devem ser adicionadas na função objetivo, que fica como a seguir:

$$\text{Min} \qquad \sum_{j=1}^{n} \alpha_j E_j + \sum_{j=1}^{n} \beta_j T_j \tag{3.24}$$

Na formulação acima, a função objetivo 3.12 minimiza a antecipação e o atraso das tarefas. A restrição 3.13 assegura que uma tarefa seja escalonada somente uma vez e processada por somente uma máquina. A restrição 3.14 assegura que cada tarefa i não seja nem precedida e nem sucedida por mais de uma tarefa j. A restrição 3.15 é utilizada para calcular o tempo de completude das tarefas e assegurar que nenhuma tarefa i deve preceder e suceder ao mesmo tempo uma determinada tarefa j no escalonamento.

A restrição 3.16 assegura que não mais que uma tarefa seja escalonada primeiro em uma máquina. O valores para atraso e antecipação das tarefas são calculados nas restrições 3.17 e 3.18, respectivamente. A restrição 3.19 define o tempo de completude da tarefa fictícia 0 seja zero. A restrição 3.20 assegura que os valores para tempos de completude, antecipação e atraso sejam não-negativos. A restrição 3.21 assegura que a variável de decisão x_{ijk} seja binária.

As restrições acima descritas não excluem a ocorrência de tempo ocioso entre as ta-

refas, como pode-se observar na Figura 3.10 (a), que apresenta uma solução viável para a instância da Figura 3.8 (a) e mostra que a restrição 3.15 (Figura 3.10 (b)) permite tempo ocioso.

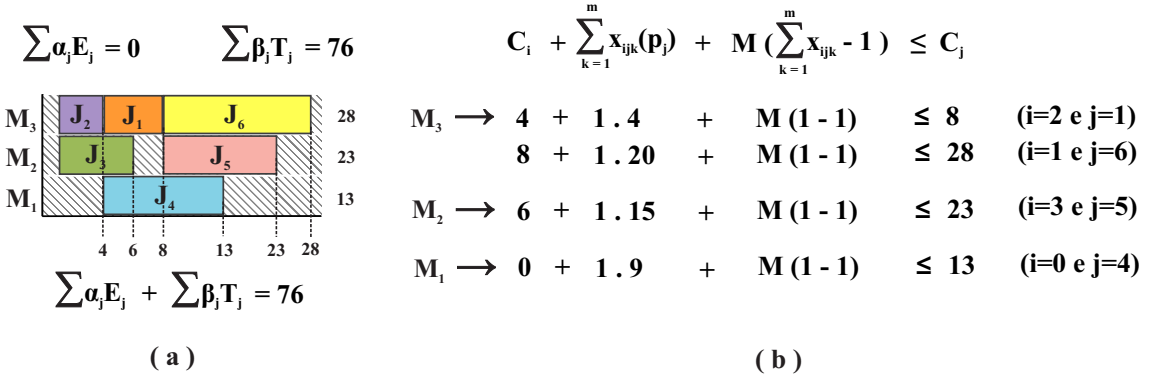


Figura 3.10: Escalonamento de tarefas: solução com tempo ocioso (a) e aplicação da restrição 3.15 para a solução (b).

Dessa forma, Tsai e Wang [TW12] propuseram mais duas novas restrições para solucionar o problema. A primeira proposta é a restrição 3.22, que assegura que o tempo de completude da primeira tarefa no escalonamento seja igual ao seu tempo de processamento. A Figura 3.11 (a) apresenta uma solução viável para a instância da Figura 3.8 (a) e mostra que a restrição 3.22 (Figura 3.11 (b)) não permite tempo ocioso no início de uma sequência em cada máquina.

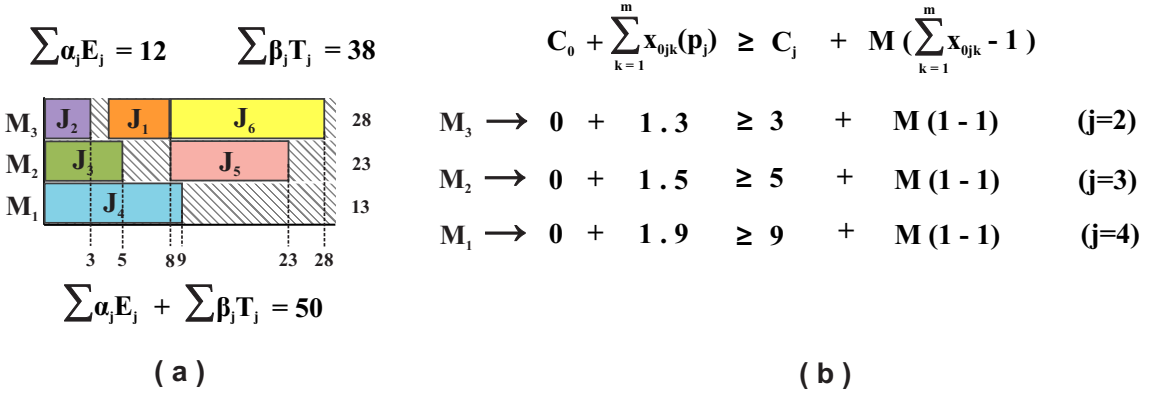


Figura 3.11: Escalonamento de tarefas: solução com tempo ocioso e com todas as tarefas iniciando no instante 0 (a) e aplicação da restrição 3.22 para a solução (b).

A segunda restrição 3.23, assegura que o tempo de completude de uma tarefa *j* seja igual a soma do tempo de completude da tarefa que foi processada anteriormente a ele (tarefa *i*) com o seu tempo de completude. A Figura 3.12 (a) apresenta uma solução viável para a instância da Figura 3.8 (a) e mostra que a restrição 3.23 (Figura 3.12 (b)) não permite tempo ocioso no escalonamento.

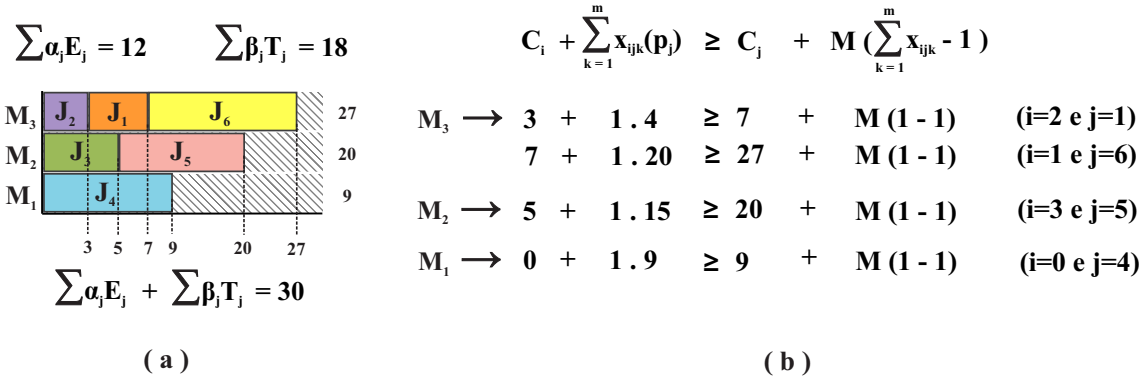


Figura 3.12: Escalonamento de tarefas: solução sem tempo ocioso e com todas as tarefas iniciando no instante 0 (a) e aplicação da restrição 3.23 para a solução (b).

3.6.3 Formulação indexada pelo tempo para o escalonamento em ambiente monoprocessado (PI-STO-IT)

A terceira formulação considerada neste trabalho é um modelo de programação linear inteira (PI) proposto por Tanaka et al. [TFA09] para a minimização das penalidades de antecipação e atraso em ambiente monoprocessado (baseado nas formulações propostas por Pritsker et al. [PWW68], Dyer e Wolsey [DW90], Sousa and Wolsey [SW92] e Van den Akker et al. [vdAvHS99]).

Apresenta uma variável de decisão inteira binária e bi-indexada x_{it} $(i \in N, 1 \leq t \leq T)$, tal que x_{it} é igual a 1 se $t \geq p_i$ e se a tarefa i é completada no instante t $(t = C_i)$, e é igual a 0, caso contrário.

Apresenta os seguintes parâmetros não-negativos: p_j, indicando o tempo de processamento da tarefa j; e, T uma constante indicando o limite superior para o intervalo de tempo em que as tarefas devem ser escalonadas (intervalo de tempo entre 1 e T.

Desta forma, o problema pode ser formulado como segue:

$$\text{Min} \sum_{i=1}^{n} \sum_{t=1}^{T} f_i(t) x_{it} \tag{3.25}$$

$$\text{S. a.} \sum_{i=1}^{n} \sum_{s=t}^{min(T,\ t+p_i-1)} x_{is} = 1 \quad 1 \leq t \leq T \tag{3.26}$$

$$\sum_{t=1}^{T} x_{it} = 1 \quad i \in N \tag{3.27}$$

$$x_{it} \in \{0,1\}, \quad i \in N, \quad 1 \leq t \leq T \tag{3.28}$$

A restrição (3.26) define a restrição de capacidade da máquina, onde cada máquina deve processar exatamente uma tarefa em um determinado instante de tempo. A restrição (3.27) indica o número de ocorrências das tarefas no escalonamento. Para a função objetivo (3.25), minimização da antecipação e atraso, tem-se: $f_j(t) = \alpha_i \cdot max\{d_i - t, 0\} + \beta_i \cdot max\{t - d_i, 0\}$.

3.6.4 Formulação clássica geral indexada pelo tempo para o escalonamento em ambiente monoprocessado (PI-TO-IT)

A quarta formulação considerada se baseia na formulação geral proposta para problemas de escalonamento com funções objetivo regulares (não-decrescentes em relação aos tempos de completude das tarefas) proposta por Dyer e Wolsey [DW90].

As variáveis de decisão são binárias bi-indexadas, y^t_j, indicando que uma tarefa j inicia o seu processamento no instante t em alguma máquina.

Apresenta os seguintes parâmetros não-negativos: p_j, indicando o tempo de processamento da tarefa j; e, T uma constante indicando o limite superior para o intervalo de tempo em que as tarefas devem ser escalonadas (neste caso, intervalo de tempo entre 0 e T.

Desta forma, o problema pode ser formulado como segue:

$$\text{Min} \sum_{j \in J} \sum_{t=0}^{T-p_j} f_j(t + p_j) y^t{}_j \tag{3.29}$$

$$\text{S. a.} \sum_{t=0}^{T-p_j} y^t{}_j = 1 \quad (\forall j \in J) \tag{3.30}$$

$$\sum_{\substack{j \in J, \\ t+p_j \leq T}} \sum_{s=max\{0, t-p_j+1\}}^{t} y^s{}_j \leq 1 \quad (t = 0, ..., T-1) \tag{3.31}$$

$$y^t{}_j \in \{0, 1\} \quad (\forall j \in J; t = 0, ..., T-1) \tag{3.32}$$

Na função objetivo (3.29), a função $f_j(x)$ é definida para cada tarefa, onde x é o instante onde cada tarefa finaliza o seu processamento na linha do tempo. Se a função objetivo é para a minimização do atraso ponderado, por exemplo, $f_j(x)$ é igual a

$\beta_j \times max\{0, x - d_j\}$. A restrição (3.30) determina que a tarefa deve ser processada exatamente uma vez. A restrição (3.31) determina que a máquina pode processar no máximo uma tarefa em um dado instante de tempo, sendo que a soma de tal instante com o seu respectivo tempo de processamento não pode ser maior que o máximo tempo de execução. Somente 1 tarefas pode ser executada ao mesmo tempo (na única máquina disponível. Esta formulação pode ser adaptada para ambiente multiprocessado, trocando-se para m o lado direito desta restrição, indicando que no pior caso, somente m tarefas podem ser executadas ao mesmo tempo, onde m é o número de máquinas disponíveis. A restrição (3.32) define os tipos de variáveis utilizadas.

Uma vantagem importante da formulação indexada pelo tempo é que essa formulação pode ser utilizada para modelar vários critérios de desempenho, do tipo regulares (funções objetivo regulares) para problemas de escalonamento. A mudança na formulação se dá através da mudança dos custos da função objetivo. A desvantagem na utilização deste modelo é no tamanho: onde existem $n + T$ restrições e aproximadamente nT variáveis, onde T é a soma de todos os tempos de processamento $\sum_{j=1}^{n} p_j$ (para ambiente monoprocessado).

A Figura 3.13 apresenta uma solução viável em ambiente monoprocessado para a instância da Figura 3.8 (a). Para máquinas paralelas, Pessoa et al. [PUPdF10] definiu o valor de T como $\lfloor (\sum_{j=1}^{n} p_j - p_{max})/m \rfloor + p_{max}$, onde p_{max} é o maior tempo de processamento da tarefa, este valor é válido pois se uma tarefa i é completada após $\lfloor (\sum_{j=1}^{n} p_j - p_i)/m \rfloor + p_i$ em uma certa máquina, pode-se concluir que pelo menos uma outra máquina estará disponível no instante $\lfloor (\sum_{j=1}^{n} p_j - p_i)/m \rfloor$, dessa forma, a tarefa pode ser movida para essa outra máquina, reduzindo assim o tempo de completude do escalonamento. Desta forma, instâncias com muitas tarefas, ou tarefas com um grande tempo de processamento, demandam muito tempo para solução.

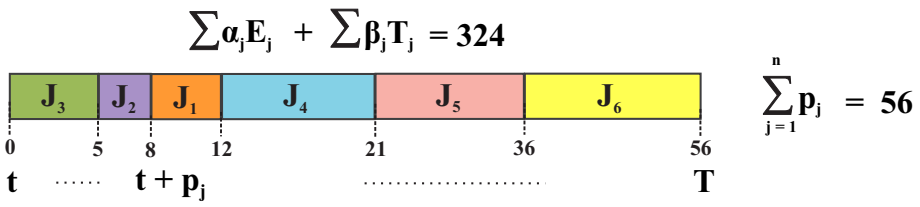


Figura 3.13: Solução do escalonamento para ambiente monoprocessado.

A formulação indexada pelo tempo contou com uma adaptação para o problema de escalonamento com penalidades de antecipação e atraso na função objetivo. Apesar da for-

mulação não permitir tempo ocioso na minimização do atraso ponderado de tarefas (funções regulares), é necessário que algumas restrições sejam adicionadas para evitar tempo ocioso na minimização da antecipação e atraso ponderados (funções não-regulares). As restrições a serem adicionadas seriam as que não permitissem tempo ocioso no início e no meio da sequência.

Assim, para evitar o tempo ocioso no início de uma sequência em uma determinada máquina, a restrição 3.29, propusemos a seguinte restrição (Figura 3.14):

$$\sum_{j \in J} y_j^0 = m. \tag{3.33}$$

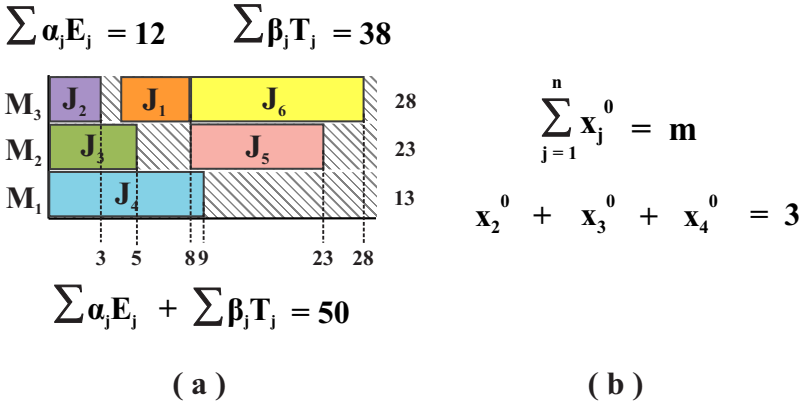


Figura 3.14: Solução do escalonamento com todas as tarefas iniciando no instante 0 (a) e aplicação da restrição 3.33 para a solução (b).

Desta forma, todas as tarefas iniciam no instante zero. Entretanto, não foi definida ainda uma restrição que evite tempo ocioso entre as tarefas em uma determinada sequência de uma máquina nesta formulação.

Para evitar tempo ocioso entre as tarefas, foi proposta neste trabalho uma formulação de programação inteira que se baseia na adaptação da formulação clássica geral de escalonamento. A formulação proposta permite a utilização de funções objetivos não-regulares (não-crescentes em relação aos tempos de completude das tarefas) sem tempo ocioso entre as tarefas em ambiente mono e multiprocessado.

Tempo ocioso nas máquinas não é permitido e as máquinas não podem processar mais que uma tarefa em um dado instante. Todas as tarefas podem ser processadas a partir do instante 0 (zero). Portanto, as tarefas devem ser processadas no intervalo de tempo $[0,T]$, onde $T = \left\lfloor \frac{\sum_{j=1}^{n} p_j - p_{max}}{m} \right\rfloor + p_{max}$, p_{max} é o maior tempo de processamento de todas as tarefas e m é o número de máquinas. As variáveis de decisão binárias y_j^t indicam que

a tarefa j inicia o seu processamento no instante t em alguma máquina. Desta forma, o problema pode ser formulado como segue (Formulação indexada pelo tempo baseada no modelo de fluxo em redes para o escalonamento em máquinas paralelas - PI-STO-JIT):

$$\text{Min} \sum_{t=0}^{T} \sum_{j=1}^{n} f_j(C_j) y^t{}_j \tag{3.34}$$

$$\text{S. a.} \sum_{t=0}^{T} y^t{}_j = 1 \ \ (j = 1,2,...,n) \tag{3.35}$$

$$\sum_{j=1}^{n} y_j^{t-p_j} - \sum_{j=0}^{n} y_j^t = 0 \ \ (t = 1,...,T) \tag{3.36}$$

$$\sum_{j=1}^{n} y_j^0 = m \tag{3.37}$$

$$y^t{}_j \in \{0,1\} \ \ (j = 1,2,...,n; t = 0,...,T) \tag{3.38}$$

A função objetivo $f_j(C_j)$ (3.34) desta formulação é baseada no problema $P||\sum \alpha_j E_j + \sum \beta_j T_j$. Assim, o valor de função objetivo pode ser calculado como segue: $f_j(C_j) = \sum \alpha_j \cdot max\{0, d_j - C_j\} + \sum \beta_j \cdot max\{0, C_j - d_j\}$. A restrição (3.35) determina que a tarefa deve ser processada exatamente uma vez. A restrição (3.36) determina a representação do escalonamento através de fluxo em redes, o que garante que não se tenha tempo ocioso entre as tarefas no escalonamento, neste caso, o tempo ocioso é permitido somente quando todas as tarefas em cada máquina são processadas, o tempo ocioso é representado pela tarefa fictícia 0 (zero), que é utilizada no final da sequência de tarefas em cada máquina, como esta tarefa fictícia é a última da sequência de tarefas, ela é direcionada ao instante final do escalonamento, essa restrição é ilustrada na Figura 3.15. A restrição (3.37) define o instante de início do escalonamento, no início do escalonamento é assegurado que a primeira tarefa da sequência em cada máquina comece no instante 0 (zero).

A Figura 3.16 (b) apresenta o escalonamento utilizando a representação de fluxo em redes para a solução apresentada na Figura 3.16 (a), onde cada caminho é representado por diferentes setas na rede da Figura 3.16 (b), que indica uma sequência de tarefas em cada máquina.

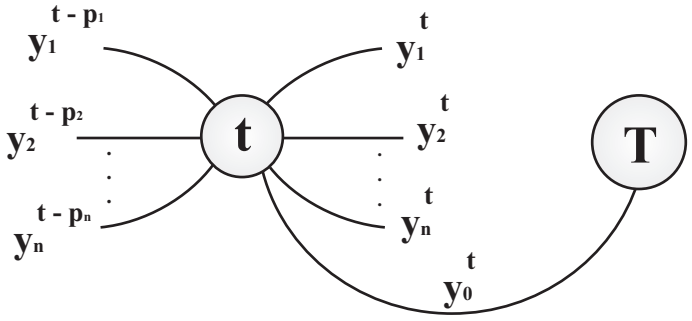


Figura 3.15: Esquema da restrição (3.36) para a determinação da sequência de tarefas através de fluxo em redes.

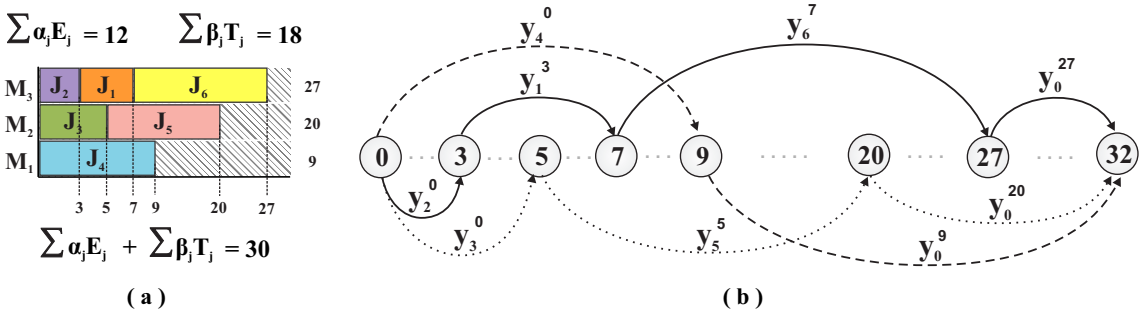


Figura 3.16: (a) Solução utilizando o gráfico de Gannt. (b) representação do escalonamento no modelo de fluxo em redes para a formulação clássica do problema.

3.6.5 Formulação baseada no modelo de fluxo em redes e roteamento de veículos para o o escalonamento em máquinas paralelas - *Arc-time indexed formulation* (PI-STO-ArcTime)

A quinta formulação, proposta por Pessoa et al. [PUPdF10] para a minimização do atraso ponderado de tarefas em ambiente mono e multiprocessado, baseia-se no modelo de fluxo em redes e roteamento de veículos (*Arc-time indexed formulation*) considera o escalonamento no intervalo de tempo de 0 a T, onde as máquinas estão ociosas no instante 0 e devem estar novamente ociosas no instante T. As variáveis binárias x_{ij}^t, $i \neq j$, indicam que a tarefa i termina sua execução e a tarefa j inicia a sua execução no instante t, em alguma máquina. As variáveis x_{0j}^t indicam que a tarefa j inicia seu processamento no instante t em alguma máquina que estava ociosa no intervalo $t-1$ a t, em particular, as variáveis x_{0j}^0 indicam que a tarefa j inicia em alguma máquina no instante 0. As variáveis x_{i0}^t indicam que a tarefa i finaliza o seu processamento no instante t em uma máquina que ficará ociosa nos instantes t a $t+1$, as variáveis x_{i0}^T indicam que a tarefa i será finalizada no instante final do intervalo de tempo. As variáveis x_{00}^t indicam que o número de máquinas que estavam livres no intervalo $t-1$ a t e irá permanecer ociosa no intervalo t a

$t+1$.

$$\text{Minimizar} \sum_{j\in J_+} \sum_{j\in J\setminus\{i\}} \sum_{t=p_i}^{T-p_j} f_j(t+p_j)x_{ij}^t \tag{3.39}$$

$$\text{Sujeito a} \sum_{i\in J_+\setminus\{j\}} \sum_{t=p_i}^{T-p_j} x_{ij}^t = 1 \ \ (\forall j \in J) \tag{3.40}$$

$$\sum_{j\in J_+\setminus\{i\},\ t-p_j\geq 0} x_{ji}^t - \sum_{j\in J_+\setminus\{i\},\ t+p_i+p_j\leq T} x_{ij}^{t+p_i} = 0 \ \ (\forall j \in J; \tag{3.41}$$
$$t = 0,...,T-p_i)$$

$$\sum_{j\in J_+,\ t-p_j\geq 0} x_{j0}^t - \sum_{j\in J_+,\ t+p_j+1\leq T} x_{0j}^{t+1} = 0 \ \ (t = 0,...,T-1) \tag{3.42}$$

$$\sum_{i\in J_+} x_{0j}^0 = m \tag{3.43}$$

$$x_{ij}^t \in Z_+ \ \ (\forall i \in J_+; \forall j \in J_+\setminus\{j\}; t = p_i,...,T-p_j) \tag{3.44}$$

$$x_{00}^t \in Z_+ \ \ (t = 0,...,T-1) \tag{3.45}$$

As restrições (3.41), (3.42) e (3.43) definem o fluxo em redes de m unidades (ou m máquinas) em um grafo acíclico $G = (V,A)$. Neste fluxo existem apenas uma origem e um destino, qualquer solução pode ser decomposta em um conjunto de m caminhos correspondentes a cada escalonamento (sequência de tarefas com seus tempos ociosos) de cada máquina. A restrição (3.40) define que cada tarefa deve estar em exatamente um caminho, e por isso, processada em somente uma máquina.

A Figura 3.17 apresenta o escalonamento utilizando a representação de fluxo em redes e roteamento de veículos, para a instância apresentada na Figura 3.8 (a), onde cada caminho na rede representa o escalonamento em uma máquina.

Esta formulação foi adaptada para a minimização conjunta das penalidades de antecipação e atraso, sem tempo ocioso entre as tarefas, o tempo ocioso neste caso seria considerado somente após o fim da execução das tarefas nas máquinas.

A seguir são resumidas as principais características das formulações apresentadas. A Tabela 3.4 apresenta sete colunas: formulação matemática, número de restrições, número de variáveis, tipos de variáveis, ambiente de processamento, tipo de formulação e referência.

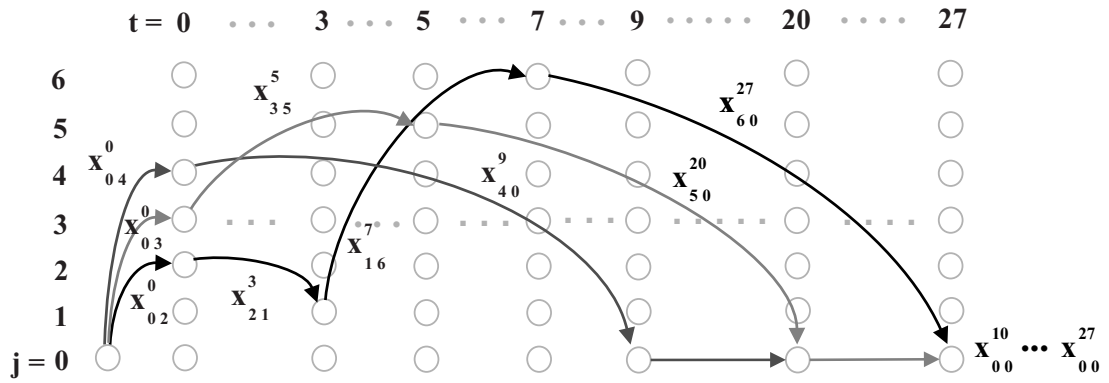


Figura 3.17: Representação do escalonamento no modelo de fluxo em redes.

Tabela 3.4: Dimensão das formulações estudadas

Formulação Matemática	Número de restrições	Número de variáveis	Tipos de Variáveis	Ambiente de Processamento	Tipo de formulação	Referência
PIM-TO	$O(n^2m)$	$O(n^2m)$	mono, bi e tri-indexadas	mono e multi-processado	PIM	Arenales, Morabito, Armentano e Yanasse (2007) [AMAY07]
PIM-STO	$O(n^2)$	$O(n^2m)$	mono, bi e tri-indexadas	mono e multi-processado	PIM	Tsai e Wang (2012) [TW12]
PI-STO-IT	$O(n+T)$	$O(nT)$	bi-indexadas	monoprocessado	PI	Tanaka, Fujikuma e Araki (2009) [TFA09]
PI-TO-IT	$O(n+T)$	$O(nT)$	bi-indexadas	monoprocessado	PI	Dyer e Wolsey [DW90] e Pessoa, Uchoa, Aragão e Rodrigues (2010) [PUPdF10]
PI-STO-ArcTime	$O(nT)$	$O(n^2T)$	tri-indexadas	mono e multi-processado	PI	Pessoa, Uchoa, Aragão e Rodrigues (2010) [PUPdF10]

3.7 Considerações finais

Este Capítulo apresentou uma revisão da literatura envolvendo problemas de escalonamento de tarefas com penalidades de antecipação e atraso, considerando os ambientes mono e multiprocessado. Foram apresentadas as definições e conceitos principais, notação clássica e exemplos de aplicação juntamente com propriedades estruturais que definem um escalonamento ótimo. Uma análise das formulações matemáticas para os problemas de escalonamento E/T foi apresentada, onde foi proposta uma formulação para o problema de escalonamento com antecipação e atraso em ambientes mono e multiprocessado sem tempo ocioso entre as tarefas. Esta formulação foi baseada na formulação clássica geral indexada pelo tempo de Dyer e Wolsey [DW90] e em um modelo de fluxo em redes.

Capítulo 4

Métodos propostos

Neste capítulo são detalhadas cinco estratégias algorítmicas aproximadas para o problema de escalonamento de tarefas que envolve penalidades de antecipação e atraso. Estas estratégias são baseadas em otimização global com busca simples (*single-start*) ou busca múltipla (*multi-start*) e baseadas em busca local com reconexão de caminhos, cujo objetivo foi analisar o processo de convergência, tempo de execução e a qualidade das soluções obtidas, afim de se atingir as melhores soluções possíveis. Os métodos propostos neste trabalho são resumidos a seguir:

- **Otimização global com busca simples:**
  1. Busca local iterada - ILS;
  2. Busca local iterada com reconexão de caminhos - ILS+PR.
- **Otimização global com busca múltipla baseada em população:**
  1. Algoritmo genético com reconexão de caminhos - GA+PR;
  2. Algoritmo genético com busca local e reconexão de caminhos - GA+LS+PR.
- **Otimização global com busca múltipla baseada em um conjunto de soluções:**
  1. Algoritmo ILS busca múltipla - ILS-M.

O primeiro método envolve uma adaptação do algoritmo de *busca local iterada*, proposto inicialmente por Rodrigues et al. [dFPUP08] para a minimização do atraso ponderado de tarefas. Este método iterativo é baseado em otimização global com busca simples, onde a cada iteração, a solução corrente é substituída por uma solução vizinha de melhor

qualidade, usando a representação de uma lista sequencial para o escalonamento em múltiplas máquinas e vizinhança baseada no 2-Opt com movimentos generalizados de troca de pares e critério de desempate. A segunda estratégia algorítmica proposta envolve *busca local iterada com a técnica de reconexão de caminhos* [Amorim et al. [ADdF12a]], cujo objetivo é encontrar melhores soluções no espaço de busca formado entre dois ótimos locais.

As outras três estratégias propostas foram baseadas em otimização global com busca múltipla utilizando o *Algoritmo Genético* como base. Assim, a terceira estratégia algorítmica envolve o *Algoritmo Genético com reconexão de caminhos*, cujo objetivo é verificar se a técnica de reconexão de caminhos consegue explorar e atingir boas soluções entre soluções arbitrárias da população gerada. A quarta estratégia proposta combina o *Algoritmo Genético usando Busca Local com reconexão de caminhos*, cujo objetivo é melhor explorar o espaço de busca entre soluções ótimas locais, como no método da *busca local iterada com a técnica de reconexão de caminhos*, mas agora com otimização global com busca múltipla, afim de convergir rapidamente para melhores soluções, mesmo em instâncias maiores. Nas duas abordagens é gerada uma nova população com soluções cada vez melhores e diversificadas a cada iteração. A quinta e última estratégia algorítmica proposta envolve ILS com busca múltipla, onde o objetivo é verificar se o ILS consegue atingir melhores soluções, em instâncias de tamanho maior, considerando um conjunto de soluções diversificadas a cada iteração.

Além dos métodos aproximados propostos, o método exato considerado neste trabalho envolveu a utilização do algoritmo *branch-and-cut* da ferramenta IBM/ILOG CPLEX 12.4, onde foi considerada a implementação da formulação de programação inteira proposta neste trabalho (PI-STO-JIT), utilizando-se a biblioteca UFFLP, que ajudou no processo de conversão das instâncias de teste e resolução do modelo matemático utilizando o ambiente CPLEX como resolvedor. Nas seções seguintes, os métodos propostos são detalhados.

4.1 Busca local iterada - ILS

A primeira abordagem algorítmica trata-se do algoritmo de Busca Local Iterada (ILS) para o problema de escalonamento com antecipação e atraso ponderados, $P||\sum \alpha_j E_j + \beta_j T_j$, baseado no método ILS proposto inicialmente por Rodrigues et al. [dFPUP08] para o problema de escalonamento com atraso ponderado em máquinas paralelas ($P||\sum w_j T_j$).

A ideia principal da heurística de Rodrigues et al. [dFPUP08] é representar um escalonamento mínimo para o problema com atraso total ponderado de tarefas como uma lista sequencial de tarefas. A Figura 4.1 ilustra a representação da solução utilizada pela heurística para minimização das penalidades de antecipação e atraso de tarefas, onde a Figura 4.1 (b) mostra a representação do escalonamento através de uma lista sequencial de tarefas utilizada pelo algoritmo, que trata-se somente de uma permutação do conjunto de tarefas para o ambiente monoprocessado. O algoritmo consegue atingir soluções ótimas para instâncias com 40 e 50 tarefas. A seguir serão considerados os detalhes dessa heurística de busca local para o problema de escalonamento com antecipação e atraso.

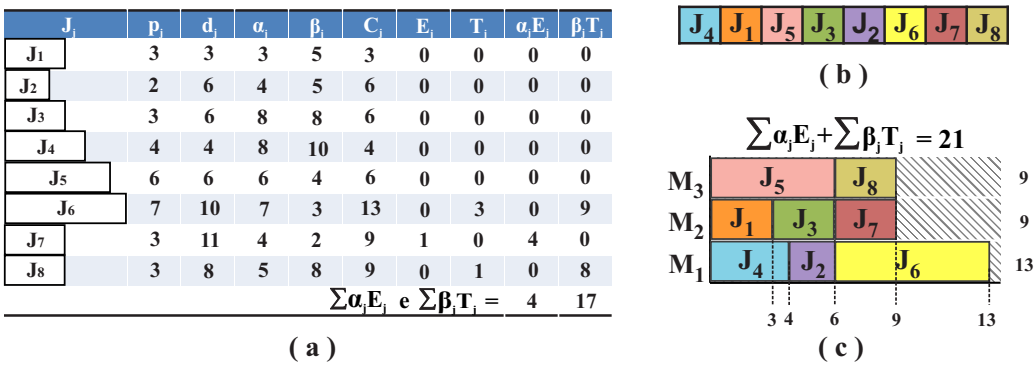


Figura 4.1: Representação da solução utilizada pelo algoritmo de busca local iterada de Rodrigues et al. [dFPUP08] para o escalonamento E/T, onde tem-se: (a) uma instância de 8 tarefas. (b) a representação do escalonamento através de uma lista sequencial e a sua respectiva representação em máquinas paralelas idênticas em (c).

O Algoritmo 3 apresenta uma adaptação da heurística de Rodrigues et al. [dFPUP08] para a minimização das penalidades de antecipação e atraso. Onde N é o número de iterações, r é o número de vezes que uma permutação aleatória é gerada e k é o número de movimentos generalizados de troca de pares (*generalized pairwise interchange* - GPI) aplicados a solução π (solução apresentada na Figura 4.1 (b)).

Os movimentos GPI são definidos por dois tipos de movimentos, o primeiro deles é o movimento de troca (*swap*), onde a posição das duas tarefas i e j (não necessariamente

adjacentes) na sequência é trocada em π, Figura 4.2 (a); e o segundo deles é o movimento de remoção (*move*), onde a tarefa i é removida de sua posição original e inserida na posição da tarefa j em π, a subsequência de tarefas entre as tarefas i e j são reposicionadas para dar espaço a tarefa i na sua nova posição, Figura 4.2 (b). Cada busca local é realizada até que não seja mais possível melhorar a solução corrente. Um busca completa na vizinhança custa $O(n^2)$ movimentos GPI e o custo de transformar uma lista sequencial no escalonamento para máquinas paralelas é de $O(nlogm)$.

Seja uma solução inicial s para o problema $1||\sum \alpha_j E_j + \sum \beta_j T_j$, obtida por uma permutação inicial π de tarefas, $\pi = (\pi_1, \pi_2, ..., \pi_n)$, onde todas as máquinas estão ociosas no instante 0. Assim, a partir desta solução inicial π, uma solução factível é gerada através do sequenciamento de tarefas da menor para a maior data de término sugerida (*earliest due date* - EDD) e é armazenada em π^*. Na primeira iteração, é gerada uma permutação aleatória em π.

Algoritmo 3: Adaptação do algoritmo de busca local iterada de Rodrigues et al. [dFPUP08] para a minimização das penalidades de antecipação e atraso.

$i \leftarrow 1$;
$\pi^* \leftarrow$ uma permutação gerada pelo sequenciamento de tarefas da menor para a maior data de término sugerida;
enquanto $i < N$ **faça**
  se i ***é um múltiplo de*** r **então**
    $\pi \leftarrow$ **uma permutação aleatória**;
  fim se
  Aplique movimentos GPI em π, até que não seja possível melhorar o valor de $\sum \alpha_j E_j + \sum \beta_j T_j$;
  se $w(\pi) < w(\pi^*)$ **então**
    $\pi^* \leftarrow \pi$;
  fim se
  Aplique k movimentos de troca escolhidos aleatoriamente em π;
  $i \leftarrow i + 1$.
fim enquanto

A busca na vizinhança é baseada em movimentos GPI e a busca na vizinhança da solução π termina quando o primeiro movimento de melhora é achado - neste caso, π é atualizado e uma nova busca é iniciada. Isto é feito até que uma busca seja completada sem melhorias. Se a solução achada na busca local for melhor que a melhor solução global atual, então a mesma será armazenada como π^*. Finalmente, k movimentos GPI escolhidos aleatoriamente são aplicados em uma tentativa de sair de regiões de ótimo local ruins. A cada r iterações, uma nova permutação totalmente aleatória substitui a solução

gerada na iteração anterior. Uma busca completa na vizinhança requer um tempo de $O(n^2)$ movimentos, e a avaliação de cada movimento requer a construção do escalonamento usando a regra de despacho, o que requer tempo $O(n \log m)$ em uma implementação com filas de prioridade.

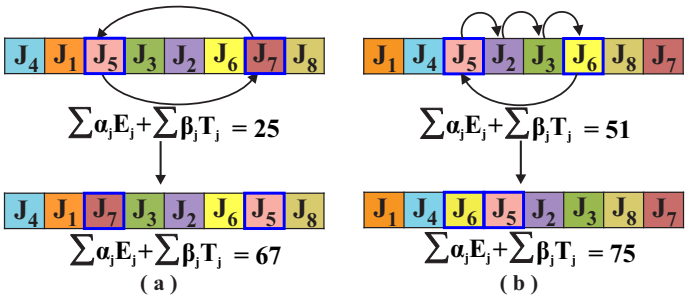


Figura 4.2: Exemplo de movimentos generalizados de troca de pares: (a) movimento de troca e (b) movimento de remoção.

A busca local conta com a estratégia da melhor solução encontrada (*best improvement*), ou seja, a mudança é aceita somente quando a nova solução é melhor que a solução corrente. Entretanto, soluções ótimas locais geralmente possuem muitas soluções vizinhas com o mesmo custo. Dessa forma, para impedir uma busca prematura, ou seja, para que o algoritmo continue efetuando a busca por soluções melhores, o algoritmo possui um Critério de Desempate (seção 3.4) de soluções que define, dentre outras soluções de mesmo valor de função objetivo, qual deve ser considerada a melhor solução. Quando há algum empate entre máquinas, ou seja, quando se tem duas máquinas ociosas, é escolhida a máquina com o menor índice.

O processo de busca local é repetido durante x iterações, onde $x = k \cdot n \cdot m$, onde k é uma dada constante, n é o número de tarefas e m é o número de máquinas. Quando o número da iteração corrente é um múltiplo de uma dada constante r, uma nova sequência randômica é gerada onde a busca local será aplicada novamente. Finalmente, k movimentos GPI escolhidos aleatoriamente são aplicados em uma tentativa de explorar outras sub-regiões do espaço de busca e fugir de ótimos locais.

4.2 Busca local iterada com reconexão de caminhos entre soluções ótimas locais - ILS+PR

A segunda abordagem algorítmica aplicada problema de escalonamento E/T consiste na combinação do algoritmo ILS com a técnica de reconexão de caminhos (*path relinking*), técnica que foi originalmente proposta para diversificação do algoritmo *scatter search* [GLM00]. Com essa estratégia combinada, é possível convergir rapidamente para bons resultados do que o algoritmo ILS sozinho, e assim obter melhores soluções para instancias maiores.

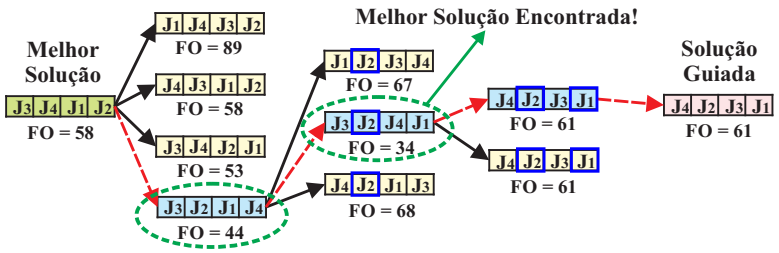
O método ILS+PR começa o seu processamento com duas soluções viáveis, essa soluções são geradas através de uma permutação randômica das sequências únicas seguindo a regra de sequência de tarefas da menor para a maior data de término sugerida (EDD). Em seguida, o algoritmo realiza uma busca local sobre as duas soluções viáveis. Após a busca local, tem-se duas soluções ótimas locais, e a técnica de reconexão de caminhos é aplicada com o objetivo de se explorar o espaço de soluções existente entre os dois ótimos locais. Com essa estratégia foi possível provar que existem melhores soluções no espaço de busca entre dois ótimos locais.

Na implementação proposta, a técnica de reconexão de caminhos começa a sua execução a partir da melhor solução ótima local, da iteração, para a pior solução ótima local, essa estratégia é conhecida como **reconexão de caminhos reversos** (*backward path relinking*). A melhor solução encontrada no caminho é armazenada e, finalmente, as três soluções (duas soluções obtidas por buscas locais e uma solução obtida pela reconexão de caminhos) são comparadas entre si e a melhor solução global (solução corrente do algoritmo). A melhor solução entre as três soluções candidatas é então considerada a melhor nova solução global, como pode ser observado no esquema da Figura 4.3 (b), que considera a instância apresentada na Figura 4.3 (a) e, como ambiente de processamento, foram utilizadas duas máquinas paralelas idênticas.

Na técnica de reconexão de caminhos, a melhor solução das duas soluções viáveis iniciais encontradas (ótimos locais) é guardada. Esta solução, que será chamada de **caminho de sequência**, será alterada para se tornar igual à solução guiada. O algoritmo então verifica, para cada posição do caminho de sequência, uma solução que, quando alterada, se torne igual a solução guiada, retornando assim a melhor solução.

J_j	p_j	d_j	α_j	β_j
J_1	4	7	3	5
J_2	3	4	4	5
J_3	5	5	8	8
J_4	9	10	8	10

(a)



(b)

Figura 4.3: (a) Exemplo de instância para o algoritmo de reconexão de caminhos. (b) esquema para o algoritmo de reconexão de caminhos reversos (*backward path relinking*), onde as soluções com as setas rachuradas representam o caminho percorrido de melhores soluções no espaço de soluções existente entre as duas soluções iniciais, melhor solução e solução guiada.

Dessa forma, para fazer a posição j no caminho de sequência θ se tornar igual à posição j da solução guiada π, deve-se encontrar a posição k do elemento π_i no caminho de sequência. Encontrada a posição k, os elementos θ_i e θ_j são trocados no caminho de sequência. Assim, quando o melhor movimento é encontrado no caminho de sequência, e quando a posição do caminho de sequência se torna igual a posição da solução guiada, esta posição é marcada como fixa e não será mais modificada. Este procedimento é repetido até que todas as posições se tornem fixas (n iterações).

Para ajudar o algoritmo de reconexão de caminhos explorar cada vez mais o espaço de soluções, a cada cinco iterações do algoritmo é realizada uma perturbação na solução global de forma a sempre ser possível buscar uma nova e extensa área do espaço de soluções, que pode ser explorada por esses procedimentos (duas buscas locais e reconexão de caminhos). O Pseudocódigo para este algoritmo melhorado integrando busca local iterada e reconexão de caminhos é apresentado no Algoritmo 4 e o pseudocódigo que detalha a técnica de reconexão de caminhos é apresentado no Algoritmo 5.

Algoritmo 4: ILS para o problema $1||\sum \alpha_j E_j + \sum \beta_j T_j$ com reconexão de caminhos reversos.

$i \leftarrow 1$;
$\pi \leftarrow$ uma permutação seguindo a regra da menor para a maior data de término sugerida;
$\theta \leftarrow$ uma permutação randômica de π;
$\pi^* \leftarrow$ melhor solução corrente;
enquanto $i < N$ **faça**
  se $i > 0$ **então**
    π e $\theta \leftarrow$ permutações randômicas de π e θ;
  fim se
  Aplique movimentos generalizados de troca de pares em π e em θ, até que não seja possível melhorar o valor de $\sum \alpha_j E_j + \sum \beta_j T_j$;
  se $w(\pi) < w(\theta)$ **então**
    $\gamma \leftarrow$ Reconexão-de-caminhos(π, θ);
  fim se
  senão
    $\gamma \leftarrow$ Reconexão-de-caminhos(θ, π);
  fim se
  se $w(\pi) < w(\theta)$ ***e*** $w(\pi) < w(\gamma)$ **então**
    se $w(\pi) < w(\pi^*)$ **então**
      $\pi^* \leftarrow \pi$;
    fim se
    senão
      se $w(\theta) < w(\pi)$ ***e*** $w(\theta) < w(\gamma)$ **então**
        se $w(\theta) < w(\pi*)$ **então**
          $\pi^* \leftarrow \theta$;
        fim se
        senão
          se $w(\gamma) < w(\pi*)$ **então**
            $\pi^* \leftarrow \gamma$;
          fim se
        fim se
      fim se
    fim se
  fim se
  $i \leftarrow i + 1$;
fim enquanto

Algoritmo 5: Reconexão-de-caminhos(Início,SoluçãoGuiada).

Entrada: Início,SoluçãoGuiada
Saída: MelhorEncontrado

```
Caminho ← Início;
Contador ← 1;
MelhorSoluçãoEncontrada ← ∞;
enquanto Contador ≤ n faça
    MelhorSoluçãoAlterada ← ∞;
    enquanto j ≤ n faça
        se j não é fixo então
            k ← posição da SoluçãoGuiada_j no Caminho;
            Troque Caminho_j e Caminho_k;
            se w(caminho) < MelhorSoluçãoAlterada então
                MelhorJ ← j;
                MelhorK ← k;
                MelhorSoluçãoAlterada ← w(caminho);
            fim se
            Troque Caminho_k e Caminho_j;
        fim se
    fim enquanto
    Troque Caminho_MelhorJ e Caminho_MelhorK;
    se w(caminho) < MelhorSoluçãoEncontrada então
        MelhorEncontrado ← Caminho;
        MelhorSoluçãoEncontrada ← w(caminho);
    fim se
    Marque j como fixo;
    Contador ← Contador +1;
fim enquanto
```

4.3 Algoritmo genético com reconexão de caminhos entre soluções arbitrárias - GA+PR

O terceiro método proposto neste trabalho envolve o algoritmo genético (GA) com Reconexão de Caminhos entre soluções arbitrárias (e o último método proposto é entre ótimos locais).

Neste algoritmo também é utilizada a representação em lista sequencial para o escalonamento de tarefas de Rodrigues et al. [dFPUP08] (Figura 4.1 (b)). Esta sequência é tratada como como um cromossomo que é utilizado nas operações do algoritmo (**cruzamento baseado em posição** e **mutação**). Cada gene representa uma tarefa e cada iteração do algoritmo uma nova população é gerada, conforme ilustrado na Figura 4.4. Quando é necessário calcular o valor de função objetivo (*fitness*) de um cromossomo (solução), as tarefas são distribuídas nas máquinas e o valor de antecipação e atraso ponderados das tarefas é calculado (Figura 4.1 (c)). O algoritmo foi baseado nas operações genéticas propostas por Liu et al. [LAR05], proposto inicialmente para a minimização o atraso ponderado de tarefas em ambiente monoprocessado ($1||\sum w_j T_j$). A seguir serão detalhadas as operações genéticas e estratégias desenvolvidas.

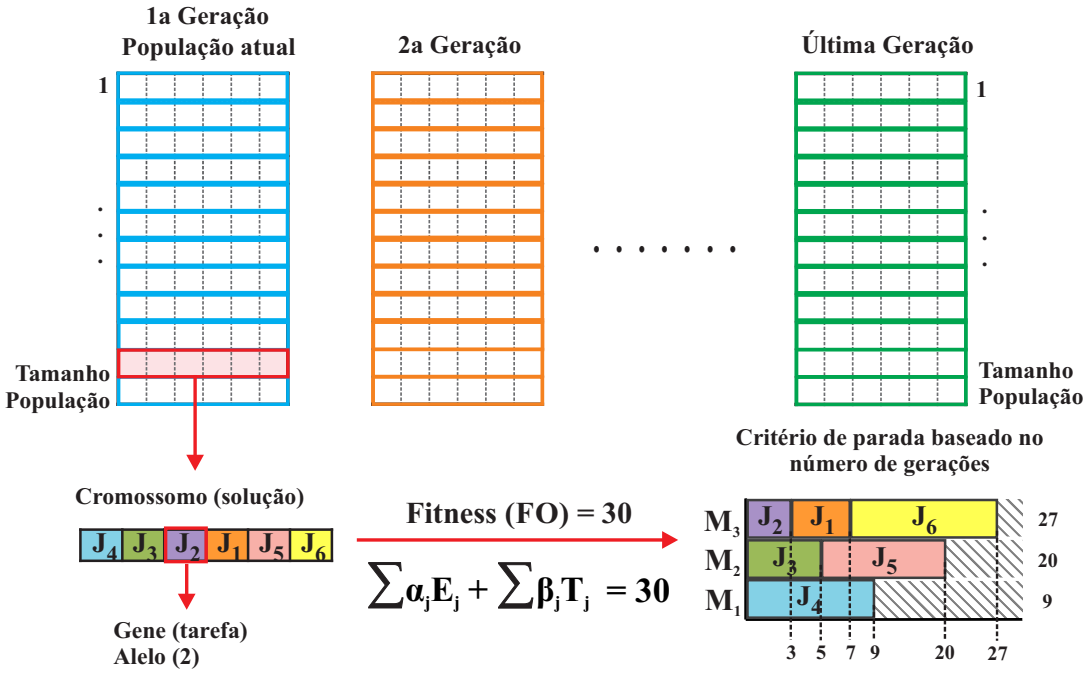


Figura 4.4: Estratégia do Algoritmo Genético proposto, onde a cada iteração uma nova população é gerada.

4.3.1 Função de avaliação (*fitness*)

Utilizando a abordagem de lista sequencial, apresentada na Figura 4.1 (b), o cromossomo é codificado através de movimentos GPI com cada gene sendo uma tarefa que deve ser escalonada na máquina disponível mais cedo, como apresentado na Figura 4.1 (c). Quando uma população é gerada, cada cromossomo é avaliado de acordo com o seu valor de função de avaliação, que trata-se do valor de função objetivo com as penalidades de antecipação e atraso para cada cromossomo.

4.3.2 Operações genéticas

a) **Cruzamento (*crossover*):** a regra de operação de cruzamento é a de combinar elementos de dois cromossomos pais e gerar um ou mais cromossomos filhos. Foi utilizado o **cruzamento baseado em posição**, onde as posições são randomicamente escolhidas (são escolhidas $0.4 * n$ posições, onde n é o número de tarefas), e as características (genes) nesta posição são guardadas e a descendência é mantida intacta. A Figura 4.5 mostra um exemplo de cruzamento para duas soluções da instância apresentada na Figura 4.1 (a), onde os dois pais (**A** e **B**) foram escolhidos pela estratégia de seleção por torneio, e mostra os dois filhos gerados (**C** e **D**) pelo cruzamento;

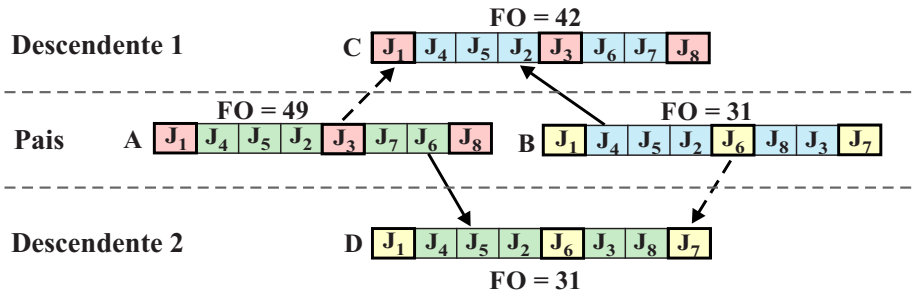


Figura 4.5: Operações Genéticas de cruzamento realizadas pelo algoritmo.

b) **Mutação:** A regra da operação de mutação é prover e manter a diversidade em uma população tal que os operadores podem continuar trabalhando. A mutação utilizada no algoritmo é a mutação baseada em ordem, onde duas posições são selecionadas randomicamente, e assim, as características (genes) destas posições são trocadas. A seleção por torneio também é utilizada pela operação de mutação, mas apenas um

indivíduo é selecionado para receber a mutação, a Figura 4.6 mostra um exemplo de mutação para uma solução da instância da Figura 4.1 (a).

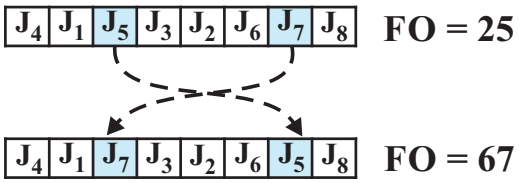


Figura 4.6: Operações Genéticas de Mutação realizadas pelo algoritmo.

Assim, após a operação genética de cruzamento, onde geram-se dois novos descendentes (ou soluções arbitrárias), outra técnica pode ser aplicada com objetivo de intensificar o processo de busca por melhores soluções. Dessa forma, foi desenvolvido um passo extra onde as novas soluções descendentes geradas são utilizadas pelo algoritmo de reconexão de caminhos, onde, a melhor solução das duas soluções candidatas é utilizada como solução inicial e a pior delas é utilizada como solução guiada, a melhor solução encontrada no algoritmo de reconexão de caminhos é guardada na nova população, de acordo com o esquema da Figura 4.7 (sem o passo da busca local - que será detalhado na seção seguinte). A estratégia utilizada neste algoritmo é a estratégia de substituição de gerações, onde a cada iteração, uma nova população é gerada, e se nesse processo for encontrada uma solução melhor que a solução global corrente, a melhor solução encontrada é assumida como nova solução global. O Algoritmo 6 apresenta a ideia proposta.

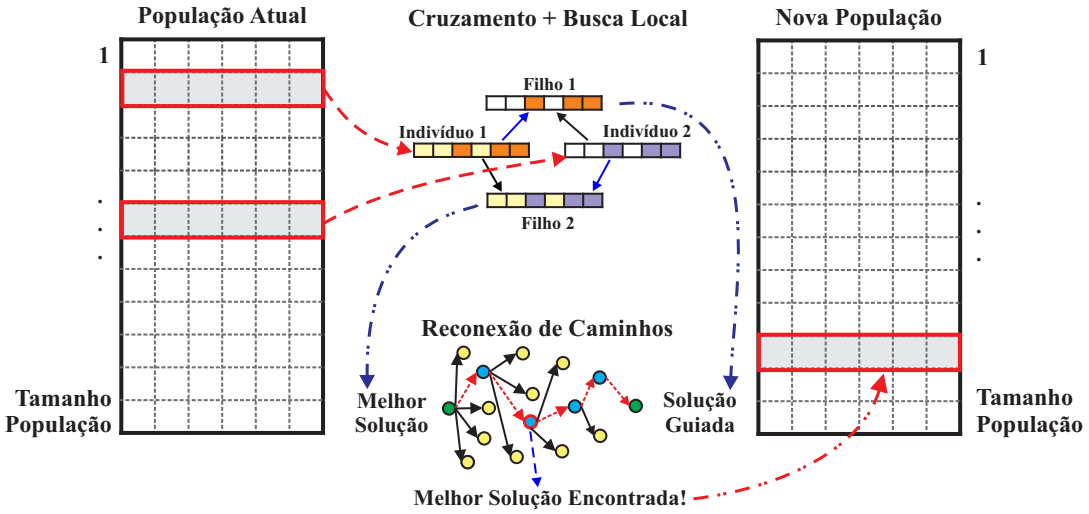


Figura 4.7: Esquema da operação de cruzamento seguida reconexão de caminhos entre soluções arbitrárias.

Algoritmo 6: Algoritmo Genético com Reconexão de Caminhos entre soluções arbitrárias para o escalonamento de máquinas paralelas.

$\prod \longleftarrow$ um conjunto de permutações aleatórias geradas de tarefas;
Calcule os valores de *fitness* de cada permutação;
enquanto $i < N$ **faça**
  $\prod^* \longleftarrow$ melhores soluções de $\prod$ ▷ Elitismo;
  $\prod^* \longleftarrow$ *Mutação* na solução π;
  $\prod^* \longleftarrow$ *Cruzamento* usando as soluções π e θ;
  $\prod^* \longleftarrow$ *Cruzamento com Busca Local* usando as soluções π e θ;
  π^* e $\theta^* \longleftarrow$ *Cruzamento baseado em posição* em π e θ;
  se $w(\pi^*) < w(\theta^*)$ **então**
    $\prod^* \longleftarrow$ **Reconexão-de-Caminhos**(π^*, θ^*);
  fim se
  senão
    $\prod^* \longleftarrow$ **Reconexão-de-Caminhos**(θ^*, π^*);
  fim se
  $i \longleftarrow i + 1$;
fim enquanto
Retorne a melhor solução encontrada.

4.4 Algoritmo genético com reconexão de caminhos entre ótimos locais - GA+LS+PR

O quarto método proposto envolve uma estratégia algorítmica com algoritmo genético com busca local e a técnica de reconexão de caminhos (GA+LS+PR). Dessa forma, dada a performance do algoritmo da heurística de busca local, agora o algoritmo genético proposto possui dois passos extras: para cada descendente gerado no cruzamento, é realizada uma busca local com o objetivo de melhorar cada vez mais a geração de novos indivíduos na população, esta abordagem pode reduzir o tempo necessário para se atingir uma solução ótima ou próximo do ótimo, e para o segundo passo, após a busca local nos novos descendentes gerados, tem-se dois ótimos locais, esses dois ótimos locais são utilizados na técnica de reconexão de caminhos e a melhor solução encontrada no caminho é armazenada na nova população (população que será utilizada na próxima iteração do algoritmo). Juntamente com o Elitismo, operações genéticas (mutação e cruzamento) e agora, com busca local e o passo de reconexão de caminhos, a cada iteração, é gerada uma nova população onde é possível obter boas soluções que jamais são perdidas no processo evolucionário. A Figura 4.8 apresenta o esquema do Algoritmo Genético com Busca Local e Reconexão de Caminhos proposto.

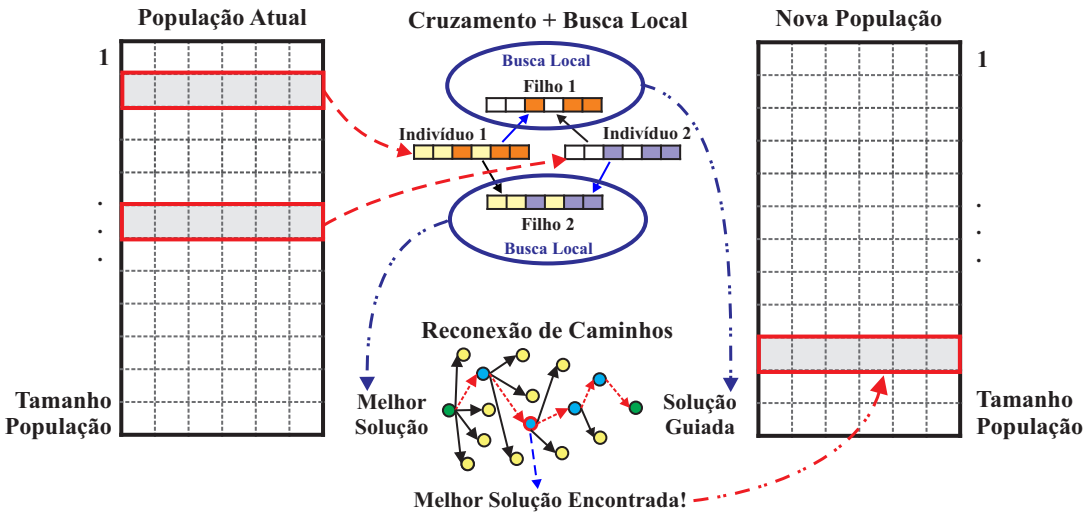


Figura 4.8: Esquema da operação de cruzamento seguida de busca local e reconexão de caminhos entre ótimos locais.

O Algoritmo 7 apresenta os passos gerais da ideia proposta, onde a cada iteração a população é ordenada de acordo com o valor de função objetivo (da melhor para a pior solução), e somente as melhores soluções são copiadas para a nova população do algoritmo, garantindo assim o ***Elitismo*** da população. A seguir, é realizada uma seleção por torneio para selecionar os pais para operações genéticas (um indivíduo para *mutação* e dois para *cruzamento*). O cruzamento é realizado novamente com o objetivo de selecionar dois descendentes que serão utilizados pelo algoritmo de reconexão de caminhos afim de explorar o espaço de soluções entre estes dois descendentes. Enquanto as novas soluções são geradas, estas novas soluções são guardadas em uma nova população ou população auxiliar de modo que essa nova população seja utilizada na próxima iteração do algoritmo. Se alguma melhor solução for encontrada durante esse processo, a melhor solução global do algoritmo é substituída com a nova melhor solução encontrada. Através desta abordagem é possível obter novas soluções e melhores soluções em um razoável período de tempo, melhor que os outros métodos propostos.

O algoritmo genético com busca local é também conhecido como algoritmo memético, mas neste trabalho é chamado de "algoritmo genético híbrido"cuja estratégia de seleção de indivíduos utilizada foi a seleção por torneio, para as operações genéticas de mutação e cruzamento. O valor de função objetivo para o problema de escalonamento com antecipação e atraso ponderados ($\sum \alpha_j E_j + \sum \beta_j T_j$) foi utilizado como valor de *fitness*.

Algoritmo 7: Algoritmo genético com reconexão de caminhos entre ótimos locais.

$\prod \longleftarrow$ um conjunto de permutações aleatórias geradas de tarefas;
Calcule os valores de *fitness* de cada permutação;
enquanto $i < N$ **faça**
  $\prod^* \longleftarrow$ melhores soluções de $\prod$ ▷ Elitismo;
  $\prod^* \longleftarrow$ *Mutação* na solução π;
  $\prod^* \longleftarrow$ *Cruzamento* usando as soluções π e θ;
  $\prod^* \longleftarrow$ *Cruzamento com Busca Local* usando as soluções π e θ;
  π^* e $\theta^* \longleftarrow$ *Cruzamento baseado em posição* em π e θ;
  para *cada filho* π^* *e* θ^* **faça**
    Aplique movimentos GPI em π^* e em θ^*, até que não seja possível melhorar o valor de $\sum \alpha_j E_j + \sum \beta_j T_j$;
  fim para
  se $w(\pi^*) < w(\theta^*)$ **então**
    $\prod^* \longleftarrow$ **Reconexão-de-Caminhos**(π^*, θ^*);
  fim se
  senão
    $\prod^* \longleftarrow$ **Reconexão-de-Caminhos**(θ^*, π^*);
  fim se
  $i \longleftarrow i + 1$;
fim enquanto
Retorne a melhor solução encontrada.

4.5 Algoritmo ILS busca múltipla - ILS-M

O quinto e último método proposto envolve busca múltipla (multi-start) incluindo Busca Local como estratégia de melhoria. A diferença entre este algoritmo e o ILS (apresentado na seção 4.1) é que este método considera um conjunto de soluções iniciais do espaço de busca, que são obtidas através de uma permutação de tarefas. Dessa forma, em cada iteração do algoritmo, tem-se um conjunto diferente de soluções onde dois passos são considerados. O primeiro passo dedica-se em construir novas soluções através da aplicação de perturbações em algumas soluções randomicamente selecionadas do conjunto corrente de soluções do algoritmo, com o objetivo de explorar a maior parte possível do espaço de soluções. O segundo passo dedica-se em melhorar as soluções obtidas através da aplicação de Busca Local em algumas soluções randomicamente escolhidas do conjunto de soluções da iteração corrente. Através desta abordagem é possível atingir melhores soluções que os outros métodos propostos. O Algoritmo 8 apresenta a estratégia proposta, onde x e y são constantes, N é o número de iterações e $\prod$ representa o conjunto de soluções utilizado em cada iteração. A Figura 4.9 ilustra a ideia proposta.

Algoritmo 8: Algoritmo ILS busca múltipla.

$i \leftarrow 1$;
$\prod \longleftarrow$ um conjunto de permutações aleatórias de tarefas;
Calcule o valor de função objetivo de cada sequência no conjunto;
enquanto $i < N$ **faça**
  $\prod \longleftarrow$ permutações aleatórias em x soluções randomicamente selecionadas;
  $\prod \longleftarrow$ movimentos GPI em y soluções randomicamente selecionadas até que nenhuma melhoria seja possível;
  $i \leftarrow i + 1$;
fim enquanto
Retorne a melhor solução encontrada.

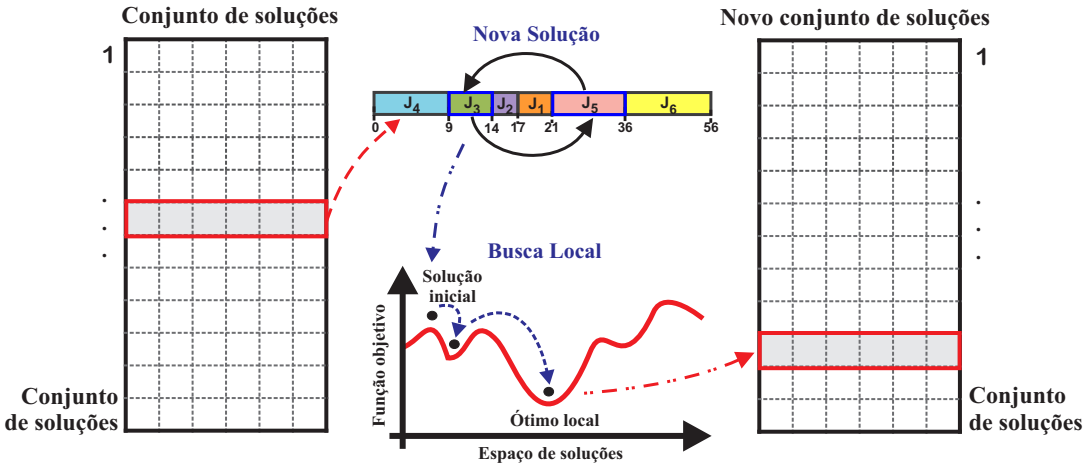


Figura 4.9: Estratégia utilizada no algoritmo ILS busca múltipla.

4.6 Branch-and-cut via CPLEX

Para o método exato, foi considerado o algoritmo *branch-and-cut* do CPLEX, que será utilizado para fins de comparação com os métodos aproximados desenvolvidos. Para isso, foi implementado um programa em C++ que efetua a conversão de uma determinada instância para um arquivo em formato conhecido pelo CPLEX, como o formato LP, ou seja, o algoritmo desenvolvido gera o modelo a ser resolvido pelo CPLEX com o auxílio da ferramenta de programação matemática UFFLP [PBas], o Apêndice A apresenta maiores detalhes sobre esta biblioteca assim como a sua utilização. No Apêndice B é apresentada a ferramenta CPLEX juntamente com a sua estratégia algorítmica e um exemplo de saída do algoritmo para uma dada instância de exemplo.

4.7 Considerações finais

Neste Capítulo foram detalhadas cinco estratégias algorítmicas aproximadas para o problema de escalonamento de tarefas que envolve penalidades de antecipação e atraso, para os ambientes mono e multiprocessado. Estas estratégias são baseadas em otimização global com busca simples (*single-start*) ou busca múltipla (*multi-start*) e baseadas em busca local com reconexão de caminhos.

Os métodos apresentados neste capítulo foram: *busca local iterada*, *busca local iterada com reconexão de caminhos entre soluções ótimas locais*, *algoritmo genético com reconexão de caminhos entre soluções arbitrárias*, *algoritmo genético com reconexão de caminhos entre ótimos locais* e *algoritmo de busca local iterada com busca múltipla*. Onde o objetivo dos métodos propostos foi analisar o processo de convergência, tempo de execução e a qualidade das soluções obtidas, afim de se atingir as melhores soluções possíveis. O método exato considerado foi o algoritmo *branch-and-cut* via CPLEX. Os resultados obtidos através da execução dos métodos foram reportados no Capítulo 5.

Capítulo 5

Análise empírica dos resultados computacionais

Neste capítulo são apresentados os resultados dos experimentos computacionais realizados com os métodos propostos no Capítulo 4, onde foram considerados os ambientes mono e multi-processado e as instâncias propostas por Tanaka [Tan12b], que são baseadas nas instâncias da *OR-Library* para ambiente monoprocessado, sendo aqui adaptadas para ambiente multiprocessados também. As estratégias propostas apresentam soluções ótimas, dentre eles, destacam-se que os ótimos da literatura foram todos obtidos, com tempos razoáveis de processamento. Os métodos propostos nesta dissertação são adequados tanto para mono quanto para ambientes multiprocessados, atingindo boas soluções em um tempo de execução razoável. Infelizmente, não existem *benchmarks* disponíveis na literatura para o problema de escalonamento com antecipação e atraso em máquinas paralelas, mas, espera-se que as instâncias adaptadas (para máquinas paralelas) e os resultados computacionais apresentados possam servir de parâmetro para outros trabalhos de pesquisa.

5.1 Ambiente computacional

Os experimentos computacionais dos métodos propostos neste trabalho foram realizados em um servidor Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 12GB, Sistema Operacional Linux, utilizando-se linguagem de programação C/C++, ferramenta CPLEX e biblioteca UFFLP, para a resolução de problemas de programação inteira.

5.2 Instâncias de teste

As instâncias utilizadas para teste são as instâncias geradas por Tanaka [Tan12a] para o problema de escalonamento com antecipação e atraso ponderados em ambiente monoprocessado sem tempo ocioso, onde, as instâncias de 40, 50 e 100 tarefas foram geradas a partir das instâncias da *OR-Library* [CP98] que contou apenas com a adição das penalidades de antecipação. As instâncias de 150, 200, 250 e 300 tarefas foram geradas de um modo similar a geração das instâncias da *OR-Library*. Para os experimentos em ambiente de máquinas paralelas, as instâncias do Tanaka [Tan12a] receberam uma modificação em suas datas de término sugeridas, ou seja, as datas de término das instâncias foram divididas pela quantidade de máquinas, pois caso contrário, com uma data de término sugerida relativamente grande (compatível com ambiente monoprocessado) o valor de antecipação e atraso ponderados poderiam ser nulos. Os ótimos disponibilizados por Tanaka [Tan12a] foram obtidos através da execução de um algoritmo exato baseado em programação dinâmica.

As instâncias da *OR-Library* [CP98] foram geradas randomicamente de acordo com a configuração a seguir: para cada tarefa $j(j = 1,...,n)$, foi gerado um tempo de processamento p_j a partir de uma distribuição uniforme no intervalo $[1,100]$ e um valor inteiro associado ao peso ou penalidade w_j foi gerado a partir e uma distribuição uniforme no intervalo $[1,10]$. Para se ter instâncias com complexidades distintas, as datas de término sugeridas d_j foram geradas através de diferentes distribuições uniformes. Desta forma, para um determinado intervalo de datas de término sugeridas $RDD(RDD = 0.2,0.4,0.6,0.8,1.0)$ e para uma determinada média correspondente ao fator de atraso $TF(TF = 0.2,0.4,0.6,0.8,1.0)$, uma data de término sugerida d_j para cada tarefa j é randomicamente gerada através da distribuição uniforme no intervalo $[P(1-TF-RDD/2),P(1-TF+RDD/2)]$, onde $P = \sum_{i=1}^{n} p_j$. Cinco instâncias foram geradas para cada 25 pares de valores de RDD e TF, obtendo assim 125 instâncias no total para cada valor de n [CPW98]. Um exemplo de instância é apresentado na Tabela 5.1.

5.3 Resultados - métodos aproximados

Os experimentos com o algoritmo ILS contou com $30nm$ iterações (sendo n o número de tarefas e m o número de máquinas), onde a cada 5 iterações uma nova solução é gerada e

Tabela 5.1: Exemplo de instância utilizada.

	1^a coluna	2^a coluna	3^a coluna	4^a coluna
1^a linha	8 (número de tarefas)			
2^a linha em diante	tempo de processamento	datas de término sugeridas	penalidades de antecipação	penalidade de atraso
	3	3	3	5
	2	6	4	5
	3	6	8	8
	4	4	8	10
	6	6	6	4
	7	10	7	3
	3	11	4	2
	3	8	5	8

a cada 3 iterações a solução corrente é perturbada com o objetivo de se explorar outras regiões do espaço de busca de soluções, evitando assim ótimos locais. O algoritmo ILS+PR contou com duas buscas locais a fim de que se tenham dois ótimos locais como solução inicial para o algoritmo de reconexão de caminhos. O algoritmo também foi executado com $25nm$ iterações, onde a cada iteração são geradas duas soluções, e é realizada uma busca local em cada solução gerada para que se tenham dois ótimos locais. A cada 5 iterações, estas duas soluções geradas recebem uma perturbação, para que, posteriormente, seja aplicada uma busca local em cada solução perturbada. Assim, juntamente com a exploração do espaço de soluções formado por dois ótimos locais, espera-se explorar a maior parte possível do espaço de busca de soluções.

Os métodos baseados em população foram desenvolvidos através do algoritmo genético com busca local e reconexão de caminhos (GA+LS+PR) e (GA+PR), que contou com a geração de uma nova população a cada iteração e testado com a seguinte configuração:

- **primeira parte ($1/3$ das iterações):** a nova população gerada, a cada iteração, é baseada em 5% de elitismo, 50% de mutação, 30% de cruzamento e 15% de cruzamento com busca local. O objetivo das primeiras $1/3$ iterações é aumentar a diversidade de indivíduos e assim obter a maior amostra possível do espaço de soluções;

- **para a se segunda parte ($2/3$ das iterações):** a nova população gerada, a cada iteração, é baseada em 10% de elitismo, 30% de mutação, 10% de cruzamento, 20% de cruzamento com busca local e 30% de cruzamento usando busca local e

reconexão de caminhos. Esta segunda configuração permite achar soluções ótimas ou próximas do ótimo na população diversificada obtida nas primeiras iterações $(1/3)$, as próximas $2/3$ iterações do algoritmo são dedicadas em achar as melhores soluções possíveis, através da geração de uma nova população, a cada iteração, com soluções cada vez melhores, mas nada impede que soluções ótimas podem ser encontradas nas primeiras iterações $(1/3)$.

O algoritmo ILS-M, método de otimização global com busca múltipla (*multi-start*), foi testado com um conjunto de $12n$ soluções a cada iteração e 60 iterações, onde n é o número de tarefas e m é o número de máquinas.

5.3.1 Resultados ambiente monoprocessado

Os resultados obtidos para o ambiente monoprocessado foram reportados nas Tabelas 5.2 e 5.3, que apresentam os resultados para 40, 50, 100, 150, 200 e 300 tarefas, onde os resultados são comparados com o *benchmark* da literatura, Tanaka [Tan12b], atingindo a maioria de seus valores ótimos.

No ambiente monoprocessado é possível perceber que a maioria das soluções ótimas ou próximo do ótimo obtidas, foram alcançadas na primeira geração dos algoritmos GA+LS+PR e ILS-M para 40, 50 e 100 tarefas. Isto se deve a quantidade de buscas locais realizadas a cada geração o algoritmo, ou seja, 15% da população total no primeiro $1/3$ de geração, e por este motivo o tempo de execução do algoritmo tende a ser maior, é a partir de 100 tarefas que o número de gerações necessárias para atingir a melhor solução começa a crescer. Por outro lado, os algoritmos ILS e ILS+PR levam um número maior de iterações para chegar na melhor solução, pois o algoritmo ILS realiza uma busca local a cada iteração e o ILS+PR realiza duas buscas locais por iteração.

A primeira coluna das tabelas apresenta a quantidade de tarefas utilizadas, a segunda coluna apresenta o número referente a instância de *benchmark* da literatura cuja solução ótima é apresentada na terceira coluna, Tanaka [Tan12b]. As outras colunas das tabelas apresentam a melhor solução obtida pelos algoritmos ILS, ILS+PR, GA+PR, GA+LS+PR e ILS-M, a média de soluções obtidas a partir de três execuções de cada algoritmo para uma determinada instância, quantidade de iterações dos algoritmos para chegar na melhor solução, tempo gasto que os algoritmos levaram para chegar na melhor solução e o tempo total de execução do algoritmos.

Tabela 5.2: Resultados para 40, 50, 100, 150, 200 e 300 tarefas em ambiente monoprocessado ($1||\sum \alpha_j E_j + \sum \beta_j T_j$).

J_i	Inst.	Tanaka	Melhor Solução				Média das soluções				Iterações/Gerações				Melhor Tempo				Tempo Total			
			ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M
40	1	54640	54640	54640	54640	54640	54640.0	54640.0	54640.0	54640.0	7	1	1	1	0.0	0.0	0.0	0.0	3.0	5.7	18.3	10.8
	11	29924	29924	29924	29924	29924	29924.0	29924.0	29924.0	29924.0	1	3	1	1	0.0	0.0	0.0	0.0	2.7	5.4	7.9	12.4
	21	77819	77819	77819	77819	77819	77819.0	77819.0	77819.0	77819.0	3	1	1	1	0.0	0.0	0.0	0.0	2.3	4.2	11.2	12.0
	31	23291	23291	23291	23291	23291	23291.0	23291.0	23291.0	23291.0	1	1	1	1	0.0	0.0	0.0	0.0	3.3	5.9	12.2	11.2
	41	59093	59093	59093	59093	59093	59093.0	59093.0	59093.0	59093.0	3	1	1	1	0.0	0.0	0.0	0.0	3.1	6.1	13.2	10.6
	51	47667	47667	47667	47667	47667	47667.0	47667.0	47667.0	47667.0	2	6	1	1	0.0	0.1	0.0	0.1	3.1	6.2	8.2	11.4
	61	21737	21737	21737	21737	21737	21737.0	21737.0	21737.0	21737.0	11	1	1	1	0.0	0.0	0.0	0.0	3.6	6.9	14.0	12.3
	71	90521	90521	90521	90521	90521	90521.0	90521.0	90521.0	90521.0	1	1	1	1	0.0	0.0	0.0	0.0	3.0	5.6	18.5	10.3
	81	12552	12552	12552	12552	12552	12552.0	12552.0	12575.0	12552.0	92	32	1	1	0.3	0.2	0.0	0.2	3.4	6.6	12.6	12.7
	91	48311	48311	48311	48311	48311	48311.0	48311.0	48311.0	48311.0	1	1	1	1	0.0	0.0	0.0	0.0	3.4	6.6	6.6	11.5
	101	85961	85961	85961	85961	85961	85961.0	85961.0	85961.0	85961.0	4	5	1	1	0.1	0.1	0.0	0.1	3.8	7.2	6.6	11.3
	111	32374	32374	32374	32374	32374	32374.0	32374.0	32377.3	32374.0	5	2	1	1	0.0	0.0	0.0	0.0	3.5	6.9	6.7	11.9
	121	122538	122538	122538	122538	122538	122538.0	122538.0	122540.3	122538.0	40	9	1	1	0.1	0.1	0.0	0.1	3.3	6.5	6.7	11.2
50	1	130548	130548	130548	130548	130548	130548.0	130548.0	130570.3	130548.0	11	15	1	1	0.1	0.2	0.1	0.0	6.5	12.7	17.6	23.6
	11	54167	54167	54167	54167	54167	54167.0	54167.0	54167.0	54167.0	3	19	1	1	0.0	0.2	0.0	0.2	0.0	12.2	17.7	23.9
	21	214555	214555	214555	214555	214555	214555.0	214555.0	214555.0	214555.0	5	2	1	1	0.0	0.0	0.0	0.0	5.9	11.2	16.6	21.4
	31	37565	37565	37565	37565	37565	37565.0	37565.0	37573.7	37565.0	5	21	1	1	0.0	0.3	0.1	0.0	7.5	14.4	18.7	25.6
	41	71949	71949	71949	71949	71949	71949.0	71949.0	71949.0	71949.0	4	1	1	1	0.0	0.0	0.0	0.0	6.8	12.9	18.2	24.1
	51	56208	56208	56208	56208	56208	56208.0	56208.0	56208.0	56208.0	1	1	1	1	0.0	0.0	0.0	0.0	8.6	15.8	20.2	31.7
	61	34640	34640	34640	34640	34660	34640.0	34640.0	34676.7	34640.0	121	88	1	1	0.7	1.2	0.1	0.6	8.4	16.3	19.8	28.0
	71	150978	150978	150978	150978	150978	150978.0	150978.0	150978.0	150978.0	36	16	1	1	0.2	0.2	0.0	0.0	7.0	14.1	17.6	23.9
	81	17433	17433	17433	17433	17433	17433.0	17433.0	17433.0	17433.0	13	13	1	1	0.1	0.2	0.0	0.0	8.8	16.7	22.1	28.8
	91	89715	89715	89715	89715	89715	89715.0	89715.0	89715.0	89715.0	31	8	1	1	0.2	0.1	0.0	0.0	8.3	16.1	18.8	26.6
	101	90880	90880	90880	90880	90880	90880.0	90880.0	90883.7	90880.0	38	4	1	1	0.2	0.1	0.1	0.1	8.5	15.8	18.8	28.5
	111	30170	30170	30170	30170	30170	30170.0	30170.0	30170.0	30170.0	8	8	1	1	0.0	0.1	0.1	0.0	8.1	15.8	19.7	27.9
	121	79007	79007	79007	79007	79007	79007.0	79007.0	79007.0	79007.0	2	1	1	1	0.0	0.0	0.0	0.0	7.1	13.8	18.1	25.1
100	1	405122	405122	405122	405122	405122	405122.0	405122.0	405122.0	405122.0	707	41	1	1	24.7	3.7	1.7	11.0	107.6	201.5	535.1	372.7
	11	252623	252623	252623	252623	252623	252633.3	252623.0	252637.7	252623.0	1858	321	1	1	66.1	26.2	0.7	2.5	109.1	204.9	528.9	360.5
	21	898927	898927	898927	898927	898927	898927.0	898927.0	898927.0	898927.0	1	1	1	1	0.1	0.1	0.1	0.1	85.6	151.6	481.4	307.6
	31	193480	193480	193480	193480	193480	193480.0	193480.0	193525.3	193480.0	69	50	2	1	3.2	4.8	5.0	3.6	124.5	227.7	568.3	370.9
	41	463554	463554	463554	463554	463554	463554.0	463554.0	463556.3	463554.0	593	56	1	1	21.5	5.8	0.3	1.7	107.8	226.7	534.2	355.2
	51	444991	444991	444991	444991	444991	444991.0	444991.0	444991.0	444991.0	2	15	1	1	0.1	1.5	0.5	0.4	123.5	233.1	549.0	388.9
	61	102185	102185	102185	102185	102185	102185.0	102185.0	102187.0	102185.0	568	61	2	1	23.8	6.4	4.7	11.7	128.7	245.9	602.9	404.4
	71	640932	640932	640932	640932	640932	640932.0	640932.0	640932.0	640932.0	342	72	1	1	11.8	5.7	0.1	1.1	101.9	197.4	523.7	392.4
	81	47629	47629	47629	47629	47629	47629.0	47629.0	47680.0	47629.0	252	24	2	1	11.1	2.3	5.3	3.7	125.8	255.9	630.2	467.8
	91	249743	249743	249743	249743	249743	249743.0	249743.0	249743.0	249743.0	672	374	2	1	24.8	32.4	4.1	3.2	112.4	219.3	554.6	391.0
	101	356397	356397	356397	356397	356397	356397.0	356397.0	356397.0	356397.0	15	4	1	1	0.7	0.5	0.2	0.3	129.4	239.9	550.2	373.3
	111	161642	161653	161680	161691	**161642**	161669.7	161682.0	161691.0	161674.7	109	883	4	8	5.2	97.2	10.6	91.7	147.3	274.8	639.0	460.3
	121	471761	471768	**471761**	**471761**	**471761**	471768.0	471761.0	471766.0	471761.0	2863	825	3	1	151.4	74.9	8.0	4.6	158.8	230.4	528.5	428.8
150	1	855097	855097	855097	855097	855097	855097.0	855097.0	855097.0	855097.0	56	27	2	1	12.9	7.8	31.6	31.5	822.4	1012.9	4245.7	2475.5
	31	378921	378921	378921	378921	378921	378924.3	378924.3	378921.0	378921.0	2065	1012	2	2	406.7	293.1	28.2	151.8	867.1	1100.6	4376.9	2684.3
	61	235255	235358	235275	235752	**235255**	235451.0	235291.7	235752.0	235255.0	1230	1397	4	6	273.3	460.9	60.1	535.1	997.3	1237.0	4572.4	3129.7
	91	416842	416842	416842	416842	416842	416842.0	416842.0	416842.0	416842.0	77	65	3	1	17.9	21.2	45.9	36.4	973.1	1224.3	4503.5	2949.6
	121	629821	629821	629821	629821	629821	629821.0	629821.0	629821.0	629821.0	44	39	1	1	8.9	10.8	3.8	12.0	892.7	1075.5	4448.2	2742.7
200	1	1508466	1508473	1508473	1508674	**1508466**	1508473.0	1508475.6	1508674.0	1508479.0	1863	2959	4	3	1463.6	5184.7	153.5	646.6	4709.0	8764.2	17461.8	7468.9
300	1	3184308	3184669	3184707	3185010	3184598	3184883.8	3184781.2	3185010.0	3184606.0	1386	20	6	8	4045	127.6	2811.9	10995.2	26655.7	45057.7	135907.7	43990.6

Tabela 5.3: Resultados para 40, 50, 100, 150, 200 e 300 tarefas em ambiente monoprocessado ($1||\sum \alpha_j E_j + \sum \beta_j T_j$) - GA+PR e GA+LS+PR.

J_i	Inst.	Tanaka	Melhor Solução		Média das soluções		Iterações		Melhor tempo		Tempo total	
			GA+PR	GA+LS+PR	GA+PR	GA+LS+PR	GA+PR	GA+LS+PR	GA+PR	GA+LS+PR	GA+PR	GA+LS+PR
40	1	54640	61662	54640	66490.0	54640.0	14	1	0.0	0.0	0.6	18.3
	11	29924	35028	29924	40712.7	29924.0	436	1	0.2	0.0	0.6	7.9
	21	77819	82771	77819	83325.0	77819.0	16	1	0.0	0.0	0.6	11.2
	31	23291	34155	23291	37775.0	23291.0	14	1	0.0	0.0	0.7	12.2
	41	59093	71509	59093	73803.3	59093.0	16	1	0.0	0.0	0.6	13.2
	51	47667	57644	47667	62202.7	47667.0	15	1	0.0	0.0	0.6	8.2
	61	21737	32166	21737	33611.7	21737.0	15	1	0.0	0.0	0.7	14.0
	71	90521	98639	90521	100008.3	90521.0	15	1	0.0	0.0	0.6	18.5
	81	12552	25239	12552	27550.7	12575.0	16	1	0.0	0.0	0.6	12.6
	91	48311	57762	48311	60402.0	48311.0	16	1	0.0	0.0	0.7	6.6
	101	85961	91461	85961	96966.3	85961.0	487	1	0.2	0.0	0.7	6.6
	111	32374	46167	32374	48827.3	32377.3	13	1	0.0	0.0	0.6	6.7
	121	122538	128310	122538	130731.7	122540.3	334	1	0.1	0.0	0.6	6.7
50	1	130548	136812	130548	138395.7	130570.3	1348	1	0.8	0.1	1.2	17.6
	11	54167	67582	54167	70256.3	54167.0	19	1	0.0	0.0	1.2	17.7
	21	214555	220962	214555	227411.0	214555.0	22	1	0.0	0.0	1.1	16.6
	31	37565	53983	37565	59410.3	37573.7	16	1	0.0	0.1	1.2	18.7
	41	71949	84157	71949	86294.0	71949.0	21	1	0.0	0.0	1.1	18.2
	51	56208	75182	56208	79038.7	56208.0	17	1	0.0	0.0	1.1	20.2
	61	34640	51643	34660	52953.3	34676.7	19	1	0.0	0.1	1.2	19.8
	71	150978	160379	150978	163808.3	150978.0	21	1	0.0	0.0	1.2	17.6
	81	17433	38594	17433	41752.3	17433.0	17	1	0.0	0.0	1.1	22.1
	91	89715	99850	89715	107286.7	89715.0	18	1	0.0	0.0	1.2	18.8
	101	90880	107291	90880	115951.3	90883.7	16	1	0.0	0.1	1.2	18.8
	111	30170	56800	30170	60965.7	30170.0	18	1	0.0	0.1	1.2	19.7
	121	79007	84838	79007	95219.7	79007.0	21	1	0.0	0.0	1.2	18.1
100	1	405122	453932	405122	458321.7	405122.0	34	1	0.1	1.7	9.4	535.1
	11	252623	273936	252623	283357.0	252637.7	37	1	0.1	0.7	9.1	528.9
	21	898927	914618	898927	927073.3	898927.0	39	1	0.1	0.1	8.8	481.4
	31	193480	247159	193480	252896.3	193525.3	38	2	0.1	5.0	9.0	568.3
	41	463554	511288	463554	519833.7	463556.3	44	1	0.1	0.3	9.3	534.2
	51	444991	490791	444991	506307.3	444991.0	37	1	0.1	0.5	11.4	549.0
	61	102185	153880	102185	171502.7	102187.0	39	2	0.1	4.7	9.2	602.9
	71	640932	669460	640932	678100.3	640932.0	40	1	0.1	0.1	9.2	523.7
	81	47629	139317	47629	146335.7	47680.0	36	2	0.1	5.3	9.2	630.2
	91	249743	289048	249743	314394.3	249743.0	41	2	0.1	4.1	11.3	554.6
	101	356397	400453	356397	437361.7	356397.0	39	1	0.1	0.2	8.9	550.2
	111	161642	212962	161691	240935.3	161691.0	39	4	0.1	10.6	9.4	639.0
	121	471761	508129	471761	525144.3	471766.0	40	3	0.1	8.0	9.3	528.5
150	1	855097	924683	855097	924683.0	855097.0	65	2	0.4	31.6	43.4	4245,7
	31	378921	500312	378921	500312.0	378921.0	52	2	0.2	28.2	43.1	4376.9
	61	235255	349451	235752	349451.0	235752.0	52	4	0.3	60.1	48.8	4572.4
	91	416842	501233	416842	501233.0	416842.0	54	3	0.2	45.9	46.9	4503.5
	121	629821	773678	629821	773678.0	629821.0	60	1	0.3	3.8	44.2	4448.2
200	1	1508466	1640019	1508674	1664243.3	1508674.0	80	4	0.7	153.5	116.5	17461.8
300	1	3184308	3465295	3185010	3498471.3	3185010.0	119	6	36.6	2811.9	486.9	135907.7

5.3.2 Resultados ambiente multiprocessado

Os resultados obtidos para o ambiente multiprocessado foram reportados nas Tabelas 5.4 e 5.5, que apresentam os resultados para 40, 50, 100, 150 e 200 tarefas em 2 máquinas. Na Tabela 5.4 pode-se observar que os algoritmos GA+LS+PR e ILS-M atingiram soluções ótimas ou próximas do ótimo para a maioria das instâncias testadas.

O resultados do método GA+LS+PR e ILS-M também se destacam nos experimentos

para 4 e 10 máquinas, como pode ser observado nas Tabelas 5.6 e 5.7, que apresentam os resultados computacionais das instâncias adaptadas para máquinas paralelas de 40, 50 e 100 tarefas em 4 máquinas, e nas Tabelas 5.8 e 5.9, que apresentam os resultados para 40, 50 e 100 tarefas em 10 máquinas. Apesar do método GA+LS+PR atingir boas soluções, pode-se observar na Tabela 5.6 que o algoritmo GA+LS+PR não retorna boas soluções para algumas instâncias de 50 e 100 tarefas, quando comparado com as soluções obtidas pelo método ILS-M.

O GA+PR (método baseado em população envolvendo algoritmo genético com reconexão de caminhos entre duas soluções arbitrárias) não gerou bons resultados, comparado com os outros três métodos aproximados propostos (todos baseados em busca local), o que mostra como é importante a estratégia da busca local nos métodos. Os resultados do método GA+PR foram reportados nas Tabelas 5.5, 5.7 e 5.9, onde pode-se concluir que sem a busca local, o algoritmo genético não consegue atingir boas soluções quando comparado com os outros métodos propostos.

A primeira coluna das tabelas apresenta a quantidade de tarefas utilizadas, a segunda coluna apresenta o número referente a instancia de *benchmark* da literatura, que foi adaptada para a utilização em máquinas paralelas, onde as datas de término sugeridas das instâncias foram divididas pelo número de máquinas. As outras colunas das tabelas apresentam a melhor solução obtida pelos algoritmos ILS, ILS+PR, GA+LS+PR e ILS-M, a média de soluções obtidas a partir de três execuções de cada algoritmo para uma determinada instância, quantidade de iterações dos algoritmos para chegar na melhor solução, tempo gasto que os algoritmos levaram para chegar na melhor solução e o tempo total de execução do algoritmos.

Os resultados dos métodos propostos neste trabalho para máquinas paralelas foram comparados com as soluções ótimas obtidas com o algoritmo *branch-and-cut* da ferramenta IBM/ILOG CPLEX 12.4. Onde foi considerada a implementação da formulação indexada pelo tempo baseada no modelo de fluxo em redes para o escalonamento em máquinas paralelas (PI-STO-JIT), proposta neste trabalho, com o auxílio da biblioteca UFFLP. As soluções ótimas obtidas com o algoritmo *branch-and-cut* juntamente com o tempo de execução de cada instância também foram reportadas nas colunas das tabelas com o acrônimo ***PI-STO-JIT***.

Tabela 5.4: Resultados para 40, 50, 100, 150 e 200 tarefas em 2 máquinas paralelas idênticas ($P||\sum\alpha_j E_j + \sum\beta_j T_j$).

J_i	Inst.	PI-STO-JIT	Melhor Solução				Média das soluções				Iterações/Gerações				Melhor Tempo				Tempo Total				
			ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	PI-STO-JIT
40	1	26063	**26063**	**26063**	**26063**	**26063**	26063.0	26063.0	26064.0	26063.0	288	182	4	4	1.1	1.4	0.8	1.6	8.4	15.4	33.3	13.4	1302.87
	11	15451	**15451**	**15451**	**15451**	**15451**	15451.0	15451.0	15451.0	15451.0	12	8	1	1	0.1	0.1	0.1	0.1	7.5	13.9	31.5	13.3	157.71
	21	41054	**41054**	**41054**	**41054**	**41054**	41054.0	41054.0	41054.0	41054.0	2	4	1	1	0.0	0.0	0.0	0.0	7.1	13.1	33.6	13.2	3.85
	31	11679	**11679**	**11679**	**11679**	**11679**	11679.0	11679.0	11679.0	11679.0	3	2	1	1	0.0	0.0	0.0	0.0	8.1	14.9	32.8	13.1	17.61
	41	31678	**31678**	**31678**	**31678**	**31678**	31678.0	31678.0	31678.0	31678.0	5	2	1	1	0.0	0.0	0.0	0.0	7.6	13.6	32.1	12.7	10.19
	51	20534	21709	21709	21709	21709	21709.0	21709.0	21709.0	21709.0	20	3	1	1	0.1	0.0	0.0	0.3	8.6	16.3	31.5	13.3	9834.40
	61	12472	**12472**	**12472**	**12472**	**12472**	12472.0	12472.0	12478.0	12472.0	683	278	2	2	2.6	2.4	0.4	0.8	9.1	16.4	33.7	14.1	341.65
	71	47952	**47952**	**47952**	**47952**	**47952**	47952.0	47952.0	47952.0	47952.0	7	1	1	1	0.0	0.0	0.0	0.1	7.8	14.5	34.8	12.9	3.76
	81	5278	**5278**	**5278**	**5278**	**5278**	5278.0	5278.0	5278.7	5278.0	516	3	3	1	2.0	0.0	0.5	0.1	9.1	17.0	37.4	16.2	388.32
	91	26309	**26309**	**26309**	**26309**	**26309**	26309.0	26309.0	26309.0	26309.0	53	17	1	1	0.2	0.1	0.1	0.3	8.7	16.8	34.7	13.9	16.23
	101	–	40300	40300	40300	40300	40300.0	40300.0	40305.0	40300.0	263	34	1	1	1.0	0.3	0.3	0.5	8.8	16.9	32.4	13.7	–
	111	18493	**18493**	**18493**	**18493**	**18493**	18493.0	18493.0	18493.0	18493.0	28	5	1	1	0.1	0.0	0.1	0.1	8.8	17.9	34.3	14.0	93.03
	121	64584	**64584**	**64584**	**64584**	**64584**	64584.0	64584.0	64584.0	64584.0	96	23	1	1	0.3	0.2	0.1	0.1	8.2	15.1	33.2	13.6	13.95
50	1	62985	62989	**62985**	**62985**	**62985**	62990.3	62985.0	62989.7	62985.3	856	840	3	5	5.9	12.3	1.5	4.1	20.4	37.0	94.3	32.7	743.36
	11	27871	**27871**	**27871**	**27871**	**27871**	27871.0	27871.0	27871.0	27871.0	55	39	2	1	0.4	0.6	0.7	0.5	19.5	36.6	93.3	30.4	853.47
	21	111069	**111069**	**111069**	**111069**	**111069**	111069.0	111069.0	111069.0	111069.0	46	1	1	1	0.2	0.0	0.0	0.4	16.3	30.2	87.0	27.2	19.41
	31	18266	**18266**	**18266**	**18266**	**18266**	18266.0	18266.0	18266.0	18266.0	79	292	2	1	0.6	6.7	1.0	0.5	21.4	57.1	99.9	40.2	1966.51
	41	38491	**38491**	**38491**	**38491**	**38491**	38491.0	38491.0	38523.0	38491.0	416	23	2	1	2.4	0.4	0.6	0.4	17.1	47.5	97.7	30.3	131.37
	51	–	26152	26152	26152	26152	26152.0	26152.0	26152.0	26152.0	2044	1634	3	2	15.4	37.6	1.6	2.6	22.7	54.4	105.1	34.8	–
	61	17456	**17456**	**17456**	**17456**	**17456**	17456.0	17456.0	17480.0	17456.0	559	132	2	2	4.0	2.6	0.8	2.2	20.8	45.5	100.2	32.3	94.8
	71	78612	**78612**	**78612**	**78612**	**78612**	78612.0	78612.0	78612.0	78612.0	13	67	1	1	0.1	1.1	0.1	0.0	18.5	32.5	90.4	28.9	14.1
	81	–	7903	7903	7903	7903	7903.0	7903.0	7903.0	7903.0	92	24	1	1	0.7	0.4	0.3	0.4	23.0	42.2	109.8	33.6	–
	91	47513	**47513**	**47513**	**47513**	**47513**	47513.0	47513.0	47515.7	47513.0	913	9	2	2	7.3	0.2	0.8	1.7	20.5	37.4	98.4	31.8	662.2
	101	40623	42973	42973	42973	42973	42973.0	42973.0	42973.0	42973.0	429	71	2	3	3.3	1.2	1.2	2.9	23.4	40.1	98.6	34.1	9916.9
	111	17012	**17012**	**17012**	**17012**	**17012**	17012.0	17012.0	17012.0	17012.0	799	69	2	2	6.0	1.0	1.2	1.7	22.1	37.4	116.8	35.2	7424.6
	121	41989	**41989**	**41989**	**41989**	**41989**	41989.0	41989.0	41989.0	41989.0	3	14	1	1	0.0	0.2	0.1	0.1	21.9	36.7	96.4	29.5	547.5
100	1	–	198288	198291	198282	**198281**	198289.7	198292.3	198302.7	198281.7	4056	1233	9	7	223.4	143.0	63.7	102.1	330.6	575.7	2959.7	658.1	–
	11	–	127682	127676	**127666**	**127666**	127682.0	127678.3	127685.0	127666.0	1338	1187	6	4	66.9	123.6	42.0	63.3	299.4	562.1	2957.8	464.9	–
	21	–	457865	457865	457865	457865	457865.0	457865.0	457865.0	457865.0	18	29	1	1	0.8	2.5	1.5	0.5	240.4	417.9	2592.1	397.8	–
	31	–	95053	95051	**95038**	**95038**	95088.7	95051.0	95076.3	95038.0	5699	4805	8	6	357.2	607.7	73.1	107.0	376.2	632.0	3811.4	636.6	–
	41	–	238416	238416	238416	238416	238416.0	238416.0	238416.0	238416.0	4817	265	1	2	250.0	28.6	3.1	28.2	320.4	531.1	2953.4	487.0	–
	51	–	215500	**215498**	**215498**	**215498**	215501.3	215501.3	215498.0	215498.0	85	4614	5	4	6.1	595.4	36.4	69.2	383.3	645.5	2902.9	502.7	–
	61	–	52740	**52739**	52740	52740	52755.0	52749.0	52740.0	52740.0	467	4748	2	3	29.2	563.1	19.1	45.3	367.1	593.1	3090.3	491.4	–
	71	–	327358	327358	327358	327358	327360.7	327359.0	327359.0	327358.7	1568	2196	3	6	87.8	233.7	20.2	79.8	361.2	531.7	3311.1	531.1	–
	81	–	22740	22755	**22720**	**22720**	22773.3	22761.3	22730.7	22720.0	3653	1177	6	8	230.1	164.3	54.7	141.9	382.6	670.7	3393.8	616.8	–
	91	–	130168	130169	**130165**	**130165**	130172.0	130169.0	130165.0	130165.0	3482	979	5	8	204.3	118.6	37.9	115.8	347.9	677.0	2921.0	534.2	–
	101	–	172816	172815	**172796**	**172796**	172816.7	172819.7	172796.0	172799.7	5740	4330	6	7	344.1	524.6	48.0	107.6	359.5	604.8	3527.3	541.9	–
	111	–	85172	85164	**85160**	**85160**	85172.0	85164.7	85181.3	85161.3	2529	1068	7	8	163.7	137.6	71.6	138.0	389.1	642.7	3440.4	586.1	–
	121	–	242250	242250	**242234**	**242234**	242253.0	242252.7	242234.3	242234.0	4413	1988	4	5	253.3	228.8	29.5	75.1	334.3	576.5	2844.1	489.8	–
150	1	–	421150	421145	**421141**	**421141**	421150.0	421150.0	421141.0	421141.0	6738	4246	5	4	3160.5	4256.9	226.4	501.8	4215.8	7532.5	33043.1	13272.4	–
200	1	–	746289	746246	**746190**	**746190**	746296.3	746246.0	746190.0	746190.0	2680	999	13	14	3164.7	2694.7	1464.3	6215.5	14189.9	26843.6	45281.7	17394.9	–

Tabela 5.5: Resultados para 40, 50, 100, 150 e 200 tarefas em 2 máquinas paralelas idênticas ($P||\sum \alpha_j E_j + \sum \beta_j T_j$) - GA+PR e GA+LS+PR.

J_i	Inst.	Melhor Solução		Média das soluções		Iterações		Melhor tempo		Tempo total	
		GA+PR	GA+LS+PR	GA+PR	GA+LS+PR	GA+PR	GA+LS+PR	GA+PR	GA+LS+PR	GA+PR	GA+LS+PR
40	1	27618	26063	28716.7	26064.0	18	4	0.0	0.8	2.7	33.3
	11	17208	15451	18455.0	15451.0	18	1	0.0	0.1	2.7	31.5
	21	41479	41054	41788.3	41054.0	20	1	0.0	0.0	2.7	33.6
	31	15052	11679	15623.0	11679.0	17	1	0.0	0.0	2.7	32.8
	41	33635	31678	34182.7	31678.0	16	1	0.0	0.0	2.7	32.1
	51	23543	21709	24407.7	21709.0	17	1	0.0	0.0	2.7	31.5
	61	15607	12472	15884.3	12478.0	21	2	0.0	0.4	4.6	33.7
	71	48379	47952	48541.0	47952.0	20	1	0.0	0.0	2.8	34.8
	81	8416	5278	8742.0	5278.7	21	3	0.0	0.5	2.7	37.4
	91	28550	26309	28709.0	26309.0	15	1	0.0	0.1	2.8	34.7
	101	44390	40300	44732.3	40305.0	19	1	0.0	0.3	2.7	32.4
	111	20133	18493	21465.3	18493.0	16	1	0.0	0.1	2.8	34.3
	121	66299	64584	66692.7	64584.0	17	1	0.0	0.1	2.8	33.2
50	1	64157	62985	65046.7	62989.7	25	3	0.2	1.5	5.5	94.3
	11	29913	27871	30751.0	27871.0	24	2	0.0	0.7	5.1	93.3
	21	111696	111069	112583.3	111069.0	24	1	0.0	0.0	7.3	87.0
	31	21563	18266	23221.3	18266.0	20	2	0.0	1.0	5.1	99.9
	41	39852	38491	41013.3	38523.0	24	2	0.0	0.6	4.8	97.7
	51	28391	26152	30196.0	26152.0	27	3	0.1	1.6	5.0	105.1
	61	21645	17456	23662.7	17480.0	20	2	0.0	0.8	5.0	100.2
	71	79755	78612	80364.0	78612.0	25	1	0.0	0.1	4.8	90.4
	81	10520	7903	12385.7	7903.0	24	1	0.0	0.3	5.0	109.8
	91	49576	47513	50216.7	47515.7	19	2	0.0	0.8	7.3	98.4
	101	44856	42973	45382.7	42973.0	24	2	0.0	1.2	5.2	98.6
	111	20511	17012	21172.3	17012.0	20	2	0.0	1.2	5.1	116.8
	121	42652	41989	44042.3	41989.0	23	1	0.0	0.1	5.1	96.4
100	1	201984	198282	204359.7	198302.7	43	9	0.2	63.7	39.4	2959.7
	11	133990	127666	137521.7	127685.0	49	6	0.2	42.0	38.8	2957.8
	21	458662	457865	460534.7	457865.0	43	1	0.2	1.5	37.2	2592.1
	31	103397	95038	107093.3	95076.3	41	8	0.2	73.1	36.2	3811.4
	41	242606	238416	248163.0	238416.0	45	1	0.2	3.1	39.3	2953.4
	51	221060	215498	225274.0	215498.0	47	5	0.2	36.4	39.7	2902.9
	61	63089	52740	65742.3	52740.0	41	2	0.2	19.1	37.4	3090.3
	71	329432	327358	330643.7	327359.0	45	3	0.2	20.2	36.6	3311.1
	81	33862	22720	35559.3	22730.7	43	6	0.2	54.7	39.7	3393.8
	91	135148	130165	138527.0	130165.0	47	5	0.2	37.9	37.0	2921.0
	101	180606	172796	182098.3	172796.0	47	6	0.2	48.0	37.5	3527.3
	111	95474	85160	97889.3	85181.3	43	7	0.6	71.6	50.0	3440.4
	121	246566	242234	247854.0	242234.3	50	4	0.2	29.5	35.8	2844.1
150	1	429449	421141	429449.0	421141.0	68	5	2.9	226.4	173.9	33043.1
200	1	747385	746190	749350.7	746190.0	103	13	5.7	1464.3	556.5	45281.7

Tabela 5.6: Resultados para 40, 50 e 100 tarefas em 4 máquinas paralelas idênticas ($P||\sum \alpha_j E_j + \sum \beta_j T_j$).

J_i	Inst.	PI-STO-JIT	Best Solution				Average Solution				Iterations/Generations				Best Time				Total Time				
			ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	PI-STO-JIT
40	1	11985	**11985**	**11985**	**11985**	**11985**	11987.7	11985.0	11986.7	11986.7	478	60	2	2	6.1	1.6	0.9	6.2	60.9	97.4	187.6	63.7	126.67
	11	8206	**8206**	**8206**	**8206**	**8206**	8206.0	8206.0	8206.0	8206.0	22	3	1	1	0.2	0.1	0.1	0.1	32.7	61.3	212.3	40.1	5.89
	21	22793	**22793**	**22793**	**22793**	**22793**	22793.0	22793.0	22793.0	22793.0	14	4	1	1	0.2	0.2	0.0	0.1	33.6	62.5	183.8	42.9	1.20
	31	5950	**5950**	**5950**	**5950**	**5950**	5950.0	5950.0	5950.0	5950.0	18	9	1	1	0.2	0.2	0.1	0.2	35.7	64.0	233.4	42.3	4.20
	41	18020	**18020**	**18020**	**18020**	**18020**	18020.0	18020.0	18020.0	18020.0	40	2	1	1	0.3	0.0	0.0	0.1	33.5	59.8	192.3	42.3	5.34
	51	8360	9104	9104	9104	9104	9104.0	9104.0	9104.0	9104.0	38	53	1	1	0.4	0.9	0.1	0.8	39.9	72.4	200.1	30.7	306.67
	61	7714	**7714**	**7714**	**7714**	**7714**	7714.0	7714.0	7714.0	7714.0	42	5	1	1	0.5	0.2	0.2	0.8	37.9	69.7	196.3	30.3	9.36
	71	26740	**26740**	**26740**	**26740**	**26740**	26740.0	26740.0	26740.0	26740.0	43	1	1	1	0.3	0.0	0.1	0.1	33.1	59.6	178.4	25.8	0.44
	81	2181	**2181**	**2181**	**2181**	**2181**	2181.0	2181.0	2181.0	2181.0	385	859	2	2	3.2	15.7	0.9	1.9	40.3	71.9	209.6	30.5	3.51
	91	15678	**15678**	**15678**	**15678**	**15678**	15678.0	15678.0	15685.0	15678.0	210	144	1	1	1.8	2.7	0.4	0.3	39.7	69.8	198.4	31.0	6.87
	101	8782	17819	17819	17819	17819	17819.0	17819.0	17819.0	17819.0	365	160	2	2	3.0	2.9	1.2	1.4	39.0	69.4	193.4	29.3	844.72
	111	11505	**11505**	**11505**	**11505**	**11505**	11505.0	11505.0	11505.0	11505.0	23	9	1	2	0.3	0.2	0.7	1.3	38.8	67.9	195.4	28.7	6.82
	121	35783	**35783**	**35783**	**35783**	**35783**	35783.0	35783.0	35783.0	35783.0	44	8	1	1	0.4	0.2	0.2	0.0	36.3	64.0	225.2	35.5	3.28
50	1	29222	29243	29241	29229	29229	29246.3	29243.7	29230.7	29230.0	2312	3193	7	5	37.1	108.7	7.7	10.5	96.4	170.4	548.8	81.4	229237.49
	11	14828	**14828**	**14828**	**14828**	**14828**	14828.0	14828.0	14828.0	14828.0	270	167	2	1	4.1	5.0	1.8	1.6	84.8	151.3	474.1	67.5	43.77
	21	59441	**59441**	**59441**	**59441**	**59441**	59441.0	59441.0	59441.0	59441.0	30	16	1	1	0.4	0.6	0.2	0.1	80.9	145.1	419.5	62.0	2.25
	31	8709	**8709**	**8709**	**8709**	**8709**	8709.0	8709.0	8709.0	8709.0	4846	750	6	4	81.6	27.1	6.4	9.2	99.8	179.1	506.8	76.7	66.79
	41	22009	**22009**	**22009**	**22009**	**22009**	22009.0	22009.0	22009.0	22009.0	90	12	1	1	1.4	0.4	0.1	0.1	82.9	148.7	474.9	72.5	18.39
	51	–	11377	11377	11380	11377	11377.0	11378.0	11380.0	11377.0	908	738	5	2	16.7	28.4	6.1	5.9	107.4	191.0	559.6	77.5	–
	61	9376	**9376**	**9376**	**9376**	**9376**	9376.0	9376.0	9376.0	9376.0	69	25	2	2	1.3	0.9	2.3	3.5	97.9	167.6	586.0	75.1	15.76
	71	42660	**42660**	**42660**	**42660**	**42660**	42660.0	42660.0	42660.0	42660.0	17	3	1	1	0.3	0.1	0.6	0.1	85.0	145.5	560.7	69.8	8.99
	81	3835	**3835**	**3835**	**3835**	**3835**	3835.0	3835.0	3835.0	3835.0	1577	1253	3	2	27.1	45.5	4.3	5.9	101.9	180.1	631.1	78.7	592.86
	91	26659	**26659**	**26659**	26661	**26659**	26659.0	26659.0	26661.0	26659.0	992	170	3	2	16.3	5.5	5.1	2.8	91.6	161.6	576.4	70.5	29.99
	101	10669	19234	19205	19205	19205	19234.0	19213.7	19205.0	19211.0	1380	1268	3	4	24.0	45.8	4.7	10.7	104.4	181.2	609.7	79.0	959.31
	111	10694	**10694**	**10694**	10714	**10694**	10694.0	10694.0	10714.0	10694.0	5235	957	2	2	92.4	35.1	2.9	4.0	101.3	181.5	645.0	80.1	88.97
	121	23884	**23884**	**23884**	23886	23885	23884.7	23884.0	23886.0	23885.7	4241	2115	3	6	74.8	68.9	4.1	10.0	106.4	163.6	591.4	74.6	40.79
100	1	–	95013	95027	94963	**94951**	95013.0	95027.0	94969.7	94961.0	1574	1972	14	21	270.8	673.7	336.0	1520.1	1971.5	3407.5	20336.3	4685.8	–
	11	–	65779	65756	**65719**	**65719**	65779.0	65771.3	65719.0	65719.0	2599	9252	7	9	369.0	2739.0	168.8	314.3	1668.9	2956.3	16975.2	1282.1	–
	21	–	**237398**	**237398**	237399	237399	237398.0	237398.3	237399.0	237399.0	85	2006	2	1	9.4	464.9	31.4	17.2	1275.2	2296.5	19024.0	1115.4	–
	31	–	45797	45738	45656	**45644**	45798.3	45738.0	45656.0	45692.3	10431	4110	17	16	1717.3	1409.5	473.9	1149.6	1949.3	3396.2	19761.5	1974.6	–
	41	–	126409	**126408**	**126408**	**126408**	126409.0	126408.0	126408.0	126408.0	8829	8079	1	4	1080.5	2150.7	15.7	146.8	1447.4	2661.6	21013.6	1225.3	–
	51	–	100649	100643	100585	**100581**	100662.7	100648.3	100585.0	100582.3	2754	6069	16	16	429.4	1984.5	424.4	965.0	1861.0	3262.1	20247.0	1874.9	–
	61	–	**28273**	28293	28289	28274	28296.7	28295.7	28289.0	28277.0	6907	6885	9	6	1049.6	2234.9	232.3	276.7	1820.4	3246.7	19940.2	1499.1	–
	71	–	170705	170704	170700	**170696**	170715.0	170705.7	170700.0	170699.0	8168	304	13	15	980.5	79.8	405.2	730.3	1442.4	2602.5	19954.7	1494.0	–
	81	–	10571	10590	**10484**	10494	10598.7	10604.7	10484.0	10504.7	10843	1934	9	19	1798.2	681.7	329.9	1375.7	2000.8	3502.5	28502.3	2010.6	–
	91	–	70781	70770	**70755**	**70755**	70781.0	70774.0	70755.0	70755.0	6290	2019	7	5	884.9	600.0	308.9	193.8	1688.0	2994.3	19157.5	1374.5	–
	101	–	81206	81186	81086	**81082**	81221.0	81187.3	81086.0	81085.3	7869	2690	12	15	1161.5	827.5	328.6	908.6	1781.7	3088.2	18936.0	1659.1	–
	111	–	47416	47372	47324	**47323**	47416.0	47372.0	47324.0	47328.7	1007	3767	11	20	165.9	1314.2	324.8	1743.2	2001.5	3495.8	47324.0	2449.3	–
	121	–	127706	127747	127700	**127699**	127722.7	127747.0	127700.0	127699.0	11978	7660	14	13	1681.7	2265.0	315.7	688.2	1684.5	2958.5	17600.1	1530.2	–

Tabela 5.7: Resultados para 40, 50 e 100 tarefas em 4 máquinas paralelas idênticas ($P||\sum \alpha_j E_j + \sum \beta_j T_j$) - GA+PR e GA+LS+PR.

J_i	Inst.	Melhor Solução		Média das soluções		Iterações		Melhor tempo		Tempo total	
		GA+PR	GA+LS+PR	GA+PR	GA+LS+PR	GA+PR	GA+LS+PR	GA+PR	GA+LS+PR	GA+PR	GA+LS+PR
40	1	12302	11985	12467.0	11986.7	21	2	0.0	0.9	11.7	187.6
	11	8501	8206	8696.7	8206.0	16	1	0.0	0.1	12.6	212.3
	21	22822	22793	22830.7	22793.0	18	1	0.0	0.0	12.1	183.8
	31	6464	5950	6581.3	5950.0	18	1	0.0	0.1	12.0	233.4
	41	18108	18020	18220.0	18020.0	18	1	0.0	0.0	11.8	192.3
	51	10147	9104	10506.7	9104.0	16	1	0.0	0.1	14.0	200.1
	61	8037	7714	8106.0	7714.0	21	1	0.0	0.2	11.8	196.3
	71	26778	26740	26845.3	26740.0	18	1	0.0	0.1	11.8	178.4
	81	2559	2181	2877.0	2181.0	20	2	0.0	0.9	11.9	209.6
	91	15817	15678	16157.7	15685.0	19	1	0.1	0.4	11.8	198.4
	101	18094	17819	18289.0	17819.0	18	2	0.0	1.2	11.8	193.4
	111	11862	11505	11912.0	11505.0	20	1	0.0	0.7	12.0	195.4
	121	35904	35783	36017.3	35783.0	20	1	0.0	0.2	11.7	225.2
50	1	29503	29229	29782.7	29230.7	25	7	0.1	7.7	23.4	548.8
	11	15082	14828	15473.7	14828.0	23	2	0.1	1.8	21.7	474.1
	21	59482	59441	59514.0	59441.0	24	1	0.1	0.2	23.9	419.5
	31	9205	8709	9412.7	8709.0	22	6	0.1	6.4	25.2	506.8
	41	22111	22009	22301.7	22009.0	24	1	0.1	0.1	22.9	474.9
	51	12034	11380	12535.0	11380.0	27	5	0.1	6.1	22.2	559.6
	61	9933	9376	10418.3	9376.0	25	2	0.1	2.3	24.1	586.0
	71	42818	42660	42924.7	42660.0	26	1	0.1	0.6	22.3	560.7
	81	4686	3835	4929.3	3835.0	22	3	0.1	4.3	22.1	631.1
	91	26745	26661	27110.7	26661.0	24	3	0.1	5.1	24.8	576.4
	101	19873	19205	20151.0	19205.0	25	3	0.1	4.7	23.9	609.7
	111	11297	10714	11366.0	10714.0	25	2	0.1	2.9	21.8	645.0
	121	23973	23886	24059.3	23886.0	25	3	0.1	4.1	22.2	591.4
100	1	95597	94963	95972.0	94969.7	49	14	0.5	336.0	168.7	20336.3
	11	66592	65719	66846.3	65719.0	50	7	0.5	168.8	171.1	16975.2
	21	237425	237399	237459.7	237399.0	46	2	0.4	31.4	179.1	19024.0
	31	46623	45656	47593.7	45656.0	46	17	0.4	473.9	167.5	19761.5
	41	127087	126408	127283.3	126408.0	48	1	0.5	15.7	165.4	21013.6
	51	101643	100585	102307.3	100585.0	51	16	0.6	424.4	168.2	20247.0
	61	29173	28289	30411.0	28289.0	45	9	0.5	232.3	172.0	19940.2

Tabela 5.8: Resultados para 40, 50 e 100 tarefas em 10 máquinas paralelas idênticas ($P||\sum \alpha_j E_j + \sum \beta_j T_j$).

J_i	Inst.	PI-STO-JIT	Best Solution				Average Solution				Iterations/Generations				Best Time				Total Time				
			ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	ILS	ILS+PR	GA+LS+PR	ILS-M	PI-STO-JIT
40	1	3988	3988	3988	3988	3988	3988.0	3988.0	3988.0	3988.0	49	4	2	1	0.7	0.1	4.5	0.5	195.9	333.0	4229.0	47.5	0.14
	31	3558	3558	3558	3558	3558	3558.0	3558.0	3558.0	3558.0	8	7	1	1	0.1	0.2	4.9	0.0	200.1	337.4	5082.1	47.3	0.38
	61	5985	5985	5985	5985	5985	5985.0	5985.0	5985.0	5985.0	4	1	1	1	0.1	0.0	5.9	0.0	194.3	325.1	4749.9	47.5	0.40
	91	9818	9818	9818	9818	9818	9818.0	9818.0	9818.0	9818.0	41	76	1	1	0.6	2.5	8.6	0.0	191.0	320.4	4849.7	47.5	0.89
	121	18818	18818	18818	18818	18818	18818.0	18818.0	18818.0	18818.0	74	2	1	1	1.3	0.1	4.0	0.0	186.8	312.0	4991.4	44.9	0.67
50	1	9006	9171	9168	9154	9154	9171.0	9168.0	9157.3	9157.3	2216	4660	5	8	77.6	334.3	96.6	30.0	529.9	894.7	15211.1	145.9	17.01
	31	3839	**3839**	**3839**	**3839**	**3839**	3839.0	3839.0	3839.0	3839.0	189	341	3	1	6.5	23.8	53.8	2.9	524.3	881.7	11387.9	125.6	1.49
	61	5856	**5856**	**5856**	**5856**	**5856**	5856.0	5856.0	5856.0	5856.0	29	36	1	1	0.9	2.3	0.2	0.2	480.1	800.3	7480.1	115.1	1.60
	91	14527	**14527**	**14527**	**14527**	**14527**	14527.0	14527.0	14527.0	14527.0	89	22	1	1	3.2	1.5	5.7	1.3	512.7	857.2	9463.2	144.0	1.23
	121	13346	**13346**	**13346**	**13346**	**13346**	13346.0	13346.0	13346.0	13346.0	390	5	2	1	13.1	0.3	10.0	0.1	495.4	829.5	7646.5	119.6	0.85
100	1	31489	33362	33344	33278	33280	33365.3	33354.0	33278.0	33289.3	14449	18239	839	40	5026.6	12986.5	97462.8	7434.5	10440.7	17909.0	123686.2	19265.8	246.5
	31	17012	17127	17150	17022	17021	17149.7	17154.3	17022.0	17043.7	18965	7374	20	21	6388.0	5023.4	2499.6	4079.2	10146.1	17047.1	128010.8	9060.8	1724.9
	61	14634	14691	14684	14661	14663	14698.7	14684.0	14661.0	14664.0	26049	11799	4	13	8782.6	7984.5	636.4	1770.1	10143.7	16924.9	128529.3	3529.7	510.5
	91	35784	35813	35800	35788	35791	35816.0	35805.7	35788.0	35791.0	8320	855	15	12	2724.2	562.0	1857.4	1047.1	9877.4	16591.3	151380.8	3303.5	134.1
	121	59347	59389	59389	**59347**	59352	59393.7	59398.3	59347.0	59353.3	10665	4974	15	21	3377.5	3102.4	1992.0	3155.6	9494.8	15617.3	132072.4	6964.1	239.8

Tabela 5.9: Resultados para 40, 50 e 100 tarefas em 10 máquinas ($P||\sum \alpha_j E_j + \sum \beta_j T_j$) - GA+PR e GA+LS+PR.

Tarefas	Inst.	Melhor Solução		Média das Soluções		Iterações		Melhor Tempo		Tempo Total	
		GA+PR	GA+LS+PR	GA+PR	GA+LS+PR	GA+PR	GA+LS+PR	GA+PR	GA+LS+PR	GA+PR	GA+LS+PR
40	1	4025	3988	4074.0	3988.0	22	2	0.2	4.5	129.3	4229.0
	31	3570	3558	3587.3	3558.0	17	1	0.1	4.9	131.2	5082.1
	61	6039	5985	6040.7	5985.0	14	1	0.1	5.9	131.8	4749.9
	91	9852	9818	9859.7	9818.0	15	1	0.1	8.6	128.2	4849.7
	121	18818	18818	18825.7	18818.0	18	1	0.2	4.0	124.9	4991.4
50	1	9295	9154	9306.3	9157.3	25	5	0.3	96.6	234.9	15211.1
	31	3882	3839	3917.0	3839.0	23	3	0.3	53.8	248.9	11387.9
	61	5899	5856	5916.7	5856.0	19	1	0.2	0.2	254.4	7480.1
	91	14544	14527	14558.3	14527.0	23	1	0.3	5.7	243.9	9463.2
	121	13354	13346	13378.0	13346.0	23	2	0.3	10.0	237.6	7646.5
100	1	33342	33278	33384.3	33278.0	60	839	4.2	97462.8	1780.7	123686.2
	31	17205	17022	17305.7	17022.0	49	20	5.5	2499.6	1849.3	128010.8
	61	14785	14661	14836.0	14661.0	57	4	2.8	636.4	1863.2	128529.3
	91	35877	35788	35898.7	35788.0	47	15	2.5	1857.4	1778.7	151380.8
	121	59458	59347	59474.0	59347.0	47	15	2.0	1992.0	1834.2	132072.4

5.3.3 Análise do impacto do critério de desempate

A Tabela 5.10 apresenta a quantidade de soluções com empate, onde foram considerados somente os métodos ILS, ILS+PR e GA+LS+PR para fins de comparação e análise. Uma instância de 40 tarefas com datas de término arbitrárias foi utilizada, incluindo as funções objetivo que envolvem atraso ponderado ou não de tarefas ($\sum T_j$ e $\sum w_j T_j$), antecipação ponderada ou não de tarefas ($\sum E_j$ e $\sum w_j E_j$) e a forma combinada de ambos ($\sum E_j + \sum T_j$ e $\sum \alpha_j E_j + \sum \beta_j T_j$), onde pode-se observar que a versão não ponderada possui um número maior de soluções com empate do que a versão ponderada do problema na maioria dos casos. Assim, conclui-se que a função objetivo de atraso não ponderado é a que apresenta um número maior de soluções com empate, quando comparado com a função objetivo de antecipação não ponderada e antecipação e atraso não ponderados.

Isso pode ser explicado através de como a solução é gerada no algoritmo, ou seja, a medida que as tarefas vão sendo escalonadas nas máquinas, a probabilidade das primeiras tarefas do escalonamento estarem atrasadas é bem menor do que as tarefas posteriores do escalonamento. Assim, é provável que se tenha tarefas com valor de atraso igual a zero, logo, como o problema é de minimização, é interessante que o algoritmo guarde a solução que possui o menor número de tarefas atrasadas possível, não restando muitas possibilidades de melhores soluções que o algoritmo possa explorar.

Por outro lado, quando se tem uma função objetivo com antecipação de tarefas, quando a solução é gerada pelo algoritmo, vão aparecer muitas tarefas com valores de

antecipação, onde, a partir daí o algoritmo tende a procurar por diferentes ou melhores soluções que minimize a quantidade de tarefas antecipadas no escalonamento. Assim, tem-se uma grande quantidade de soluções diferentes a serem exploradas pelo algoritmo e, quando se consideram pesos (ou penalidades), a possibilidade de soluções é ainda maior. Dessa forma, pode-se concluir que quanto maior for a quantidade de soluções diferentes existentes no espaço de busca, menor será a quantidade de soluções com empate no escalonamento. E quanto mais se aumenta a número de máquinas, menores são as possibilidades de solução do escalonamento, pois apesar de se ter mais máquinas, a linha do tempo do escalonamento é diminuída, dessa forma, tem-se uma quantidade maior de soluções com empate.

A quantidade de soluções com empate varia de um algoritmo para outro devido a quantidade de buscas locais realizadas a cada iteração dos algoritmos. Assim, como no algoritmo GA+LS+PR são realizadas mais buscas locais que nos algoritmos ILS e ILS+PR, a quantidade de soluções com empate obtidas neste método é bem maior.

5.3.4 Comparação dos critérios de configuração dos métodos propostos

Esta seção apresenta a parametrização dos algoritmos deste trabalho. A Tabela 5.11 apresenta as configurações utilizadas nos algoritmos que começa pelo número de iterações/-gerações, a lista sequencial ou sequência única foi a representação da solução utilizada, as soluções iniciais geradas foram sequências aleatórias geradas a partir de uma determinada semente, os algoritmos foram executados três vezes para cada instância, sendo que cada execução contou com uma semente diferente, o critério de diversidade utilizado foi a geração de uma nova solução a cada iteração para o ILS, ILS+PR e ILS-M e, para os algoritmos GA+PR e GA+LS+PR, foi utilizada a mutação. A convergência considerada nos algoritmos ILS, ILS+PR e ILS-M foi a convergência monotônica, que é a convergência que guarda a melhor solução e trata critério de desempate. Esta convergência também é utilizada no GA+LS+PR juntamente com o Elitismo, no GA+PR é utilizado somente o Elitismo. O critério de parada utilizado é o número de iterações/gerações.

A busca por melhores soluções dos algoritmos ILS e ILS+PR baseia-se na busca simples (*single-start*), que é aquela busca em que uma solução aleatória é gerada, seguida de melhoria através da busca local. Já nos algoritmos GA+PR e GA+LS+PR, por serem

Tabela 5.10: Quantidade de soluções com empate obtidas pelos algoritmos ILS, ILS+PR e GA+LS+PR para o problema de escalonamento com antecipação e atraso ponderado de tarefas com datas de término arbitrárias para a instância de número 1 de 40 tarefas.

Função objetivo	Máquinas	ILS	ILS+PR	GA+LS+PR
$\sum T_j$	1	4982583	8294285	9744331
	2	7461200	12258312	24456336
	4	6189656	10302468	39702794
	10	23573716	39242995	296615545
$\sum w_j T_j$	1	5094888	8488568	4816394
	2	3979632	6624737	6457365
	4	2418612	4024735	9496217
	10	9796746	16268583	115939433
$\sum E_j$	1	769228	1271385	597709
	2	1807860	3045609	2136579
	4	4367476	7269019	15488380
	10	21749075	36284413	231039689
$\sum w_j E_j$	1	562984	914802	305979
	2	1635824	2745267	1774333
	4	4311023	7171821	12688689
	10	20381769	34114276	159409070
$\sum E_j + \sum T_j$	1	119267	198623	23801
	2	399826	661178	379473
	4	1351294	2260642	3186635
	10	11668627	19458863	85634541
$\sum \alpha_j E_j + \sum \beta_j T_j$	1	109408	181086	14208
	2	260608	432074	49423
	4	917908	1527337	1391734
	10	8314025	13810700	49588344

métodos baseados em população, eles fazem uso da busca múltipla (*multi-start*) de soluções, onde a cada iteração tem-se várias soluções que são seguidas de melhorias que são: Cruzamento, Busca Local e Reconexão de Caminhos para o algoritmo GA+LS+PR e Cruzamento e Reconexão de Caminhos para o algoritmo GA+PR. O algoritmo ILS-M, método de otimização global com busca múltipla, considera um conjunto de soluções a cada iteração e possui a Busca Local como critério de melhoria.

Além das configurações da Tabela 5.11, os algoritmos GA+PR e GA+LS+PR contaram com uma configuração extra, apresentada na Tabela 5.12, onde no primeiro $1/3$ de geração do algoritmo, considera-se a diversidade de soluções onde é aplicado 5% de Elitismo, 50% de Mutação, 30% de Cruzamento e 15% de Cruzamento com Busca Local como objetivo de se explorar a maior parte possível do espaço de soluções. Dessa forma,

no restante das gerações, $2/3$ da geração, é considerada a maior convergência possível de melhores soluções que o algoritmo possa encontrar no espaço de busca. Assim, foi considerado 10% de Elitismo, 30% de Mutação, 10% de Cruzamento, 20% de Cruzamento com Busca Local e 30% de Cruzamento com Busca Local e Reconexão de Caminhos.

Tabela 5.11: Configuração dos métodos propostos.

Configurações	**ILS**	**ILS+PR**	**GA+PR**	**GA+LS+PR**	**ILS-M**
Número de gerações/iterações	$30nm$	$25nm$	$40nm$	$2nm$	$12n$
Representação da solução	Lista sequencial	Lista sequencial	Lista sequencial	Lista sequencial	Lista sequencial
Critério de geração de soluções iniciais	Soluções aleatórias geradas a partir de uma semente inicial	Soluções aleatórias geradas a partir de uma semente inicial	Soluções aleatórias geradas a partir de uma semente inicial	Soluções aleatórias geradas a partir de uma semente inicial	Soluções aleatórias geradas a partir de uma semente inicial
Critério de diversidade	Geração de uma nova solução a cada iteração	Geração de uma nova solução a cada iteração	Mutação	Mutação	Geração de novas soluções a cada iteração
Critério de melhoria	Busca Local	Busca Local e Reconexão de Caminhos	Cruzamento e Reconexão de Caminhos	Cruzamento, Busca Local e Reconexão de Caminhos	Busca Local
Convergência	monotônica (guarda melhor solução e trata critério de desempate)	monotônica (guarda melhor solução e trata critério de desempate)	Elitismo	Elitismo, monotônica (guarda melhor solução e trata critério de desempate)	monotônica (guarda melhor solução e trata critério de desempate)
Critério de parada	Número de iterações	Número de iterações	Número de gerações	Número de gerações	Número de iterações
Estratégia de busca	Busca simples	Busca Simples	Busca múltipla	Busca múltipla	Busca múltipla

5.3.5 Análise dos métodos

Os gráficos apresentados nas Figuras 5.1, 5.2, 5.3 e 5.4 apresentam a curvas de crescimento relacionadas ao tempo de execução dos métodos *ILS*, *ILS+PR*, *GA+PR*, *GA+LS+PR* e *ILS-M* nos ambientes mono e multiprocessados, onde pode-se observar que o tempo de execução dos métodos aumenta a medida que se aumenta o número de máquinas.

A curva do método *GA+PR* apresentou um crescimento bem inferior em relação aos

Tabela 5.12: Configuração do algoritmo genético.

Configurações	Gerações	
	$1/3$ mais diversidade e menos convergência	$2/3$ menos diversidade e mais convergência
Tamanho da população	2nm	2nm
Elitismo	5%	10%
Mutação	50%	30%
Cruzamento	30%	10%
Cruzamento + Busca Local	15%	20%
Cruzamento + Busca Local + Reconexão de Caminhos	0%	30%

outros métodos mas, apesar disso, os resultados obtidos com esse método não foram muito satisfatórios. A curva do método *ILS* foi a segunda menor em relação aos outros métodos, mas apesar do baixo tempo de execução, o método não consegue atingir soluções ótimas quando se aumenta o número de tarefas (instâncias a partir de 100 tarefas). A terceira menor curva apresentada foi obtida pelo método *ILS-M*, este método foi o que obteve melhores resultados, atingindo soluções ótimas na maioria das instâncias testadas. O *ILS+PR* também conseguiu obter boas soluções, mas apesar disto, o método não consegue atingir soluções ótimas em algumas instâncias maiores, este método foi o que apresentou a quarta maior curva. A quinta maior curva foi apresentada pelo método *GA+LS+PR*, que também retornou soluções ótimas para a maioria dos casos testados, mesmo para instâncias maiores (assim como o ILS-M), mas este método foi o que apresentou o maior tempo de execução.

5.4 Resultados - branch-and-cut via CPLEX

Os resultados para o método exato utilizado são apresentados na Tabelas 5.4, 5.6 e 5.8 nas colunas com o acrônimo *PI-STO-JIT*, onde pode-se observar que os métodos aproximados apresentaram melhor desempenho em comparação com o algoritmo *branch-and-cut* do CPLEX, especialmente quando o número de tarefas é bem maior que o número de máquinas disponíveis, ou seja, o custo de transformar uma lista sequencial no escalona-

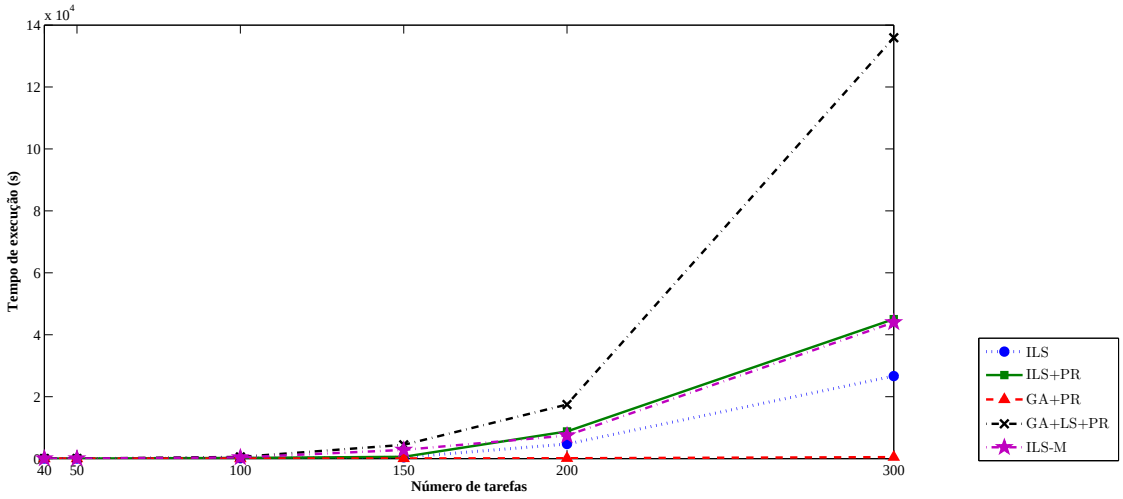


Figura 5.1: Comparação da média dos tempos de execução dos métodos ILS, ILS+PR, GA+PR, GA+LS+PR e ILS-M para 40, 50, 100, 150, 200 e 300 tarefas em ambiente monoprocessado.

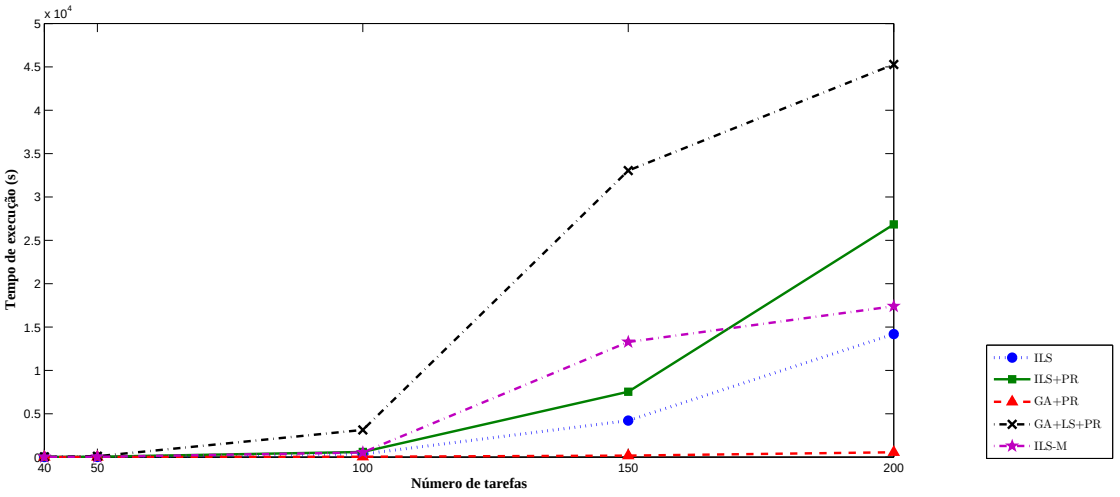


Figura 5.2: Comparação da média dos tempos de execução dos métodos ILS, ILS+PR, GA+PR, GA+LS+PR e ILS-M para 40, 50, 100, 150 e 200 tarefas em 2 máquinas.

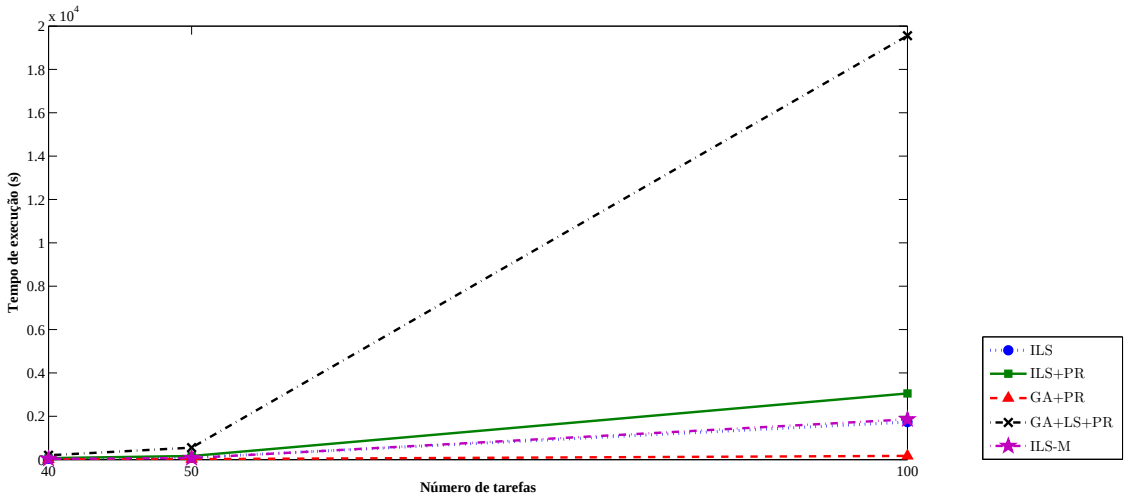


Figura 5.3: Comparação da média dos tempos de execução dos métodos ILS, ILS+PR, GA+PR, GA+LS+PR e ILS-M para 40, 50 e 100 tarefas em 4 máquinas.

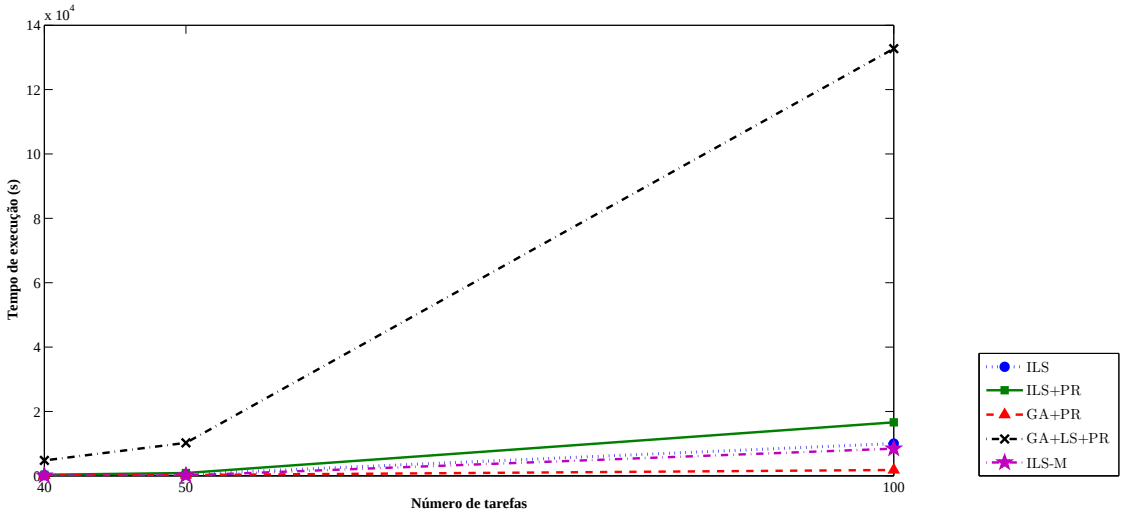


Figura 5.4: Comparação da média dos tempos de execução dos métodos ILS, ILS+PR, GA+PR, GA+LS+PR e ILS-M para 40, 50 e 100 tarefas em 10 máquinas.

mento propriamente dito utilizando regras de despacho (*dispatching*) é **proporcional** ao número de máquinas. Nos resultados obtidos, o número de variáveis é **inversamente proporcional** ao número de máquinas, ou seja, o parâmetro T (tempo total) diminui quando o parâmetro m (número de máquinas) aumenta. O parâmetro T é calculado da seguinte forma:

$$T = \left\lfloor \frac{\sum_{j=1}^{n} p_j - p_{max}}{m} \right\rfloor + p_{max}. \tag{5.1}$$

Onde p_{max} é o maior tempo de processamento do conjunto de tarefas. Isto explica o porquê do modelo matemático, implementado no CPLEX, se torna mais rápido que os algoritmos aproximados quando se aumenta o número de máquinas.

5.5 Considerações Finais

Este Capítulo apresentou os experimentos computacionais realizados para os métodos propostos neste trabalho, considerando a minimização das penalidades de antecipação e atraso de tarefas, nos ambientes mono e multiprocessado. Os resultados dos métodos foram reportados nas Tabelas 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8 e 5.9. Também foram apresentados: análise do impacto do critério de desempate de soluções no espaço de busca, comparação dos critérios de configuração dos métodos propostos e análise gráfica dos resultados obtidos.

Os experimentos computacionais mostram que as estratégias aproximadas são competitivas em relação aos resultados existentes na literatura para o escalonamento em ambiente monoprocessado, envolvendo as instâncias propostas por Tanaka [Tan12b], que foram baseadas no *benchmark* da *OR-Library* para 40, 50, 100, 150, 200 e 300 tarefas, onde todos os ótimos disponibilizados pelo autor foram obtidos para 40, 50 e 100 tarefas. Essas instâncias foram adaptadas para os experimentos em máquinas paralelas idênticas, onde as datas de término sugeridas das instâncias foram divididas pelo número de máquinas consideradas, sendo 2, 4 e 10 máquinas. Os métodos aproximados atingiram boas soluções, para os ambientes mono e multiprocessado, em um tempo razoável de execução.

Capítulo 6

Conclusões

Neste trabalho foram considerados problemas de escalonamento com penalidades de antecipação e atraso em ambiente monoprocessado e ambiente de máquinas paralelas idênticas, com tarefas independentes, de tempos de processamento arbitrários e pesos distintos ($P \mid\mid \sum \alpha_j E_j + \sum \beta_j T_j$ na notação de 3 campos). A fundamentação teórica envolvendo Otimização Combinatória, com ênfase em teoria de escalonamento, programação inteira e métodos heurísticos, juntamente com um detalhamento aprofundado do problema de escalonamento em enfoque no trabalho, envolvendo definições, notação clássica, discussão sobre estrutura de solução com empates e simetria, formulações matemáticas existentes e trabalhos relacionados. Foram apresentadas quatro abordagens algorítmicas aproximadas e uma exata para resolução do problema.

Dentre as estratégias aproximadas, foram considerados os métodos de **busca local iterada**, algoritmo inicialmente proposto por Rodrigues et al. [dFPUP08] e que foi adaptado para o problema considerado neste trabalho, o método de **reconexão de caminhos com busca local iterada**, cujo objetivo é encontrar melhores soluções no espaço de busca formado entre dois ótimos locais e, o terceiro método trata-se de um método de otimização global com busca múltipla baseada em população que foi desenvolvido através do **algoritmo genético com reconexão de caminhos** onde, a cada iteração, é gerada uma nova população com soluções cada vez melhores e diversificadas.

No quarto método proposto também foi utilizada a ideia de busca múltipla baseada em população, que foi proposto através do **algoritmo genético** utilizando **busca local** com **reconexão de caminhos**, cujo objetivo foi de explorar cada vez mais o espaço de busca entre duas soluções ótimas locais afim de convergir rapidamente para melhores soluções.

A quinta e última estratégia algorítmica apesentada envolveu o ILS com busca múltipla baseada em um conjunto de soluções, onde o objetivo desta estratégia foi de verificar se o algoritmo ILS consegue atingir melhores soluções para instâncias de tamanho maior, considerando um conjunto de soluções diversificadas a cada iteração.

O método exato considerado neste trabalho foi algoritmo *branch-and-cut* do CPLEX, onde foram consideradas duas formulações matemáticas, uma formulação para o ambiente monoprocessado, sem tempo ocioso entre as tarefas, proposta por Tanaka et al. [TFA09], e a outra formulação para o ambiente de máquinas paralelas, formulação clássica de escalonamento que, através dos experimentos computacionais apresentados, essa formulação não trata o tempo ocioso em algumas instâncias. A implementação contou com a utilização da biblioteca UFFLP para C++ onde, a partir de uma instância de teste, foi possível gerar o modelo matemático a ser executado pelo CPLEX.

Uma análise comparativa empírica mostrou que a estratégia baseada em população do algoritmo genético com busca local e reconexão de caminhos entre dois ótimos locais foi o melhor e mais promissor método proposto, atingindo todos os ótimos da literatura para o ambiente de monoprocessado. Os métodos apresentados também são adaptáveis ao ambiente de máquinas paralelas, onde é possível atingir boas soluções em um tempo razoável. Infelizmente ainda não existem *benchmarks* na literatura para máquinas paralelas para comparação com os nossos resultados, mais este trabalho apresentou uma comparação dos resultados dos métodos aproximados e exatos a fim de se tenha uma **referência de qual seria a melhor solução** ou **solução ótima** para o problema envolvendo **ambiente multiprocessado**, espera-se também que estes resultados possam ser utilizados como **referência em outras pesquisas** para fins de comparação com outros métodos.

Os resultados obtidos a partir da execução do método exato mostraram que, quando o mesmo consegue terminar, ou seja, encontrar a melhor solução, a execução do *branch-and-cut* do CPLEX é mais rápida para máquinas paralelas, ao contrário dos métodos aproximados propostos. Isso se deve ao fato de que o tempo total de processamento tende a diminuir a medida que o número de máquinas disponíveis aumenta, pois, para os métodos aproximados, considera-se o custo de distribuir as tarefas nas máquinas utilizando regras de despacho toda vez que uma solução tiver que ser avaliada. Mas, apesar disso, os métodos aproximados são extremamente promissores para o problema, apresentando bons resultados em um razoável espaço de tempo, além do mais, o tempo de processamento

dos métodos aproximados apresentados podem ser minimizados através da combinação de novas estratégias de solução.

6.1 Conferências e publicações

Os trabalhos apresentados/publicados estão listados abaixo em ordem cronológica do mais recente ao mais antigo:

- **Em processo de publicação para a revista *Expert Systems with Applications***.
  - *Single and multi-start methods based on local search and path-relinking technique for earliness-tardiness parallel machine scheduling problems*.
- **Trabalho no GECCO 2013 - Genetic and Evolutionary Computation Conference**.
  - *A hybrid genetic algorithm with local search approach for E/T scheduling problems on identical parallel machines* [ADdFU13].
- **Trabalho no WoPI 2012 - II Workshop de Pesquisa em Informática**.
  - *Minimização da Antecipação e Atraso em Escalonamento de Processos Produtivos Usando Máquinas Paralelas* [AdF12a].
- **Trabalho no EURO-INFORMS MMXIII - 26th European Conference on Operational Research**.
  - *Algorithmic strategies to single and parallel machine scheduling problems with earliness-tardiness penalties* [AdF13].
- **Trabalho no EURO 2012 -** 25^{th} **European Conference on Operation Research**.
  - *MIP models and algorithms for earliness/tardiness scheduling problems on parallel machines* [ADdF12b].
- **Trabalho no CLAIO/SBPO 2012 - Congreso Latino-iberoamericano de Investigación Operativa / Simpósio Brasileiro de Pesquisa Operacional**.
  - *ILS com reconexão de caminhos entre ótimos locais para um problema clássico de escalonamento com antecipação e atraso* [ADdF12a].

- **Participação na ELAVIO 2012 - XVI Escuela Latinoamericana de Verano en Investigación Operativa**.
  - *Problemas clássicos de escalonamento sob restrições de antecipação e atraso* [AdF12b].
  - Participação na dinâmica de grupo.

Além dos eventos e publicações supracitadas, foram realizadas visitas técnicas em instituições parceiras do Rio de Janeiro, incluindo o Departamento de Engenharia de Produção da Universidade Federal Fluminense, o Programa de Engenharia de Sistemas e Computação da COPPE/Universidade Federal do Rio de Janeiro e a sede do Instituto Nacional de Metrologia, Qualidade e Tecnologia (Inmetro), onde houve um encontro técnico com exposições tanto de pesquisas realizadas neste trabalho quanto de membros da instituição.

6.2 Trabalhos futuros

Para trabalhos futuros envolvendo problemas de escalonamento com penalidades de antecipação e atraso, considera-se importante serem propostas formulações matemáticas em programação inteira mais robustas para tais problemas, de tal forma a serem desenvolvidos métodos exatos mais eficientes. Métodos híbridos, envolvendo a resolução de formulações relaxadas para o problema em conjunto com métodos heurísticos, é um caminho interessante a seguir, no intuito de se obter soluções ótimas com tempos de execução mais eficientes.

Um aprofundamento no estudo e tratamento de soluções com empate e simétricas também faz-se necessário, de modo que procedimentos algorítmicos que sejam capazes de detectar tais soluções equivalentes sejam elaborados. Assim, tais soluções equivalentes não precisam ser consideradas durante o processo de busca por melhores soluções, permitindo uma redução significativa do tempo de processamento do algoritmo. Por outro lado, é interessante também investigar novos ambientes de processamento como máquinas de propósito geral (*mpm*) e diferentes tipos de restrições, tais como: tempos de preparação (*setups*), datas de término iguais (*common due dates*) e preempção (*pmtn*).

Referências

[ADdF12a] Rainer Xavier de Amorim, Bruno Raphael Cardoso Dias e Rosiane de Freitas. ILS com reconexão de caminhos entre ótimos locais para um problema clássico de escalonamento com antecipação e atraso. 1:1–12, 2012.

[ADdF12b] Rainer Xavier de Amorim, Bruno Raphael Cardoso Dias e Rosiane de Freitas. Mip models and algorithms for earliness/tardiness scheduling problems on parallel machines. *EURO 2012 - 25th European Conference on Operation Research*, 1:1, 2012.

[ADdFU13] Rainer Xavier de Amorim, Bruno Raphael Cardoso Dias, Rosiane de Freitas e Eduardo Uchoa. A hybrid genetic algorithm with local search approach for e/t scheduling problems on identical parallel machines. *ACM Proceedings - Genetic and Evolutionary Computation Conference, GECCO 2013*, 1:1–2, 2013.

[AdF12a] Rainer Xavier de Amorim e Rosiane de Freitas. Minimização da antecipação e atraso em escalonamento de processos produtivos usando máquinas paralelas. *WoPI 2012 - II Workshop de Pesquisa em Informática*, 1:4, 2012.

[AdF12b] Rainer Xavier de Amorim e Rosiane de Freitas. Problemas clássicos de escalonamento sob restrições de antecipação e atraso. *ELAVIO 2012 - XVI Escuela Latinoamericana de Verano en Investigación Operativa*, 1:1–2, 2012.

[AdF13] Rainer Xavier de Amorim e Rosiane de Freitas. Algorithmic strategies to single and parallel machine scheduling problems with earliness-tardiness

penalties. *EURO-INFORMS MMXIII - 26th European Conference on Operational Research*, 1:1, 2013.

[AMAY07] M. Arenales, R. Morabito, V. A. Armentano e H. Yanasse. *Pesquisa Operacional: As Disciplinas da Execução da Estratégia*. Elsevier, 2007.

[AY00] Vinícius A. Armentano e Denise S. Yamashita. Tabu search for scheduling on identical parallel machines to minimize mean tardiness. *Journal of Intelligent Manufacturing*, 11:453–460, 2000.

[Bap00] Philippe Baptiste. Scheduling equal-length jobs on identical parallel machines. *Discrete Appl. Math.*, 103:21–32, 2000.

[BM10] Andrew Bilyk e Lars Mönch. A variable neighborhood search approach for planning and scheduling of jobs on unrelated parallel machines. *Journal of Intelligent Manufacturing*, páginas 1–15, 2010.

[BPG11] M. Beyranvand, M. Peyghami e M. Ghatee. On the quadratic model for unrelated parallel machine scheduling problem with restrictive common due date. *Optimization Letters*, páginas 1–15, Agosto 2011.

[BPP04] Pascal Babu, Laurent Péridy e Eric Pinson. A branch and bound algorithm to minimize total weighted tardiness on a single processor. *Annals OR*, 129(1-4):33–46, 2004.

[Bru06] Peter Brucker. Scheduling algorithms. *Springer Publishing Company, Incorporated*, 5a ed.:1–104, 2006.

[BS90] Kenneth R. Baker e Gary D. Scudder. Sequencing with earliness and tardiness penalties: a review. *Oper. Res.*, 38:22–36, Fevereiro 1990.

[CGG12] F. Della Croce, T. Garaix e A. Grosso. Iterated local search and very large neighborhoods for the parallel-machines total tardiness problem. *Computers & Operations Research*, 39:1213–1217, 2012.

[CGK96] T.C.Edwin Cheng, Valery S. Gordon e Mikhail Y. Kovalyov. Single machine scheduling with batch deliveries. *European Journal of Operational Research*, 94:277–283, 1996.

[CP98] Richard K. Congram e Chris N. Potts. Or-library - benchmark instances the single-machine total weighted tardiness problem, 1998. Disponível em: http://people.brunel.ac.uk/∼mastjjb/jeb/info.html.

[CPW98] H. A. J. Crauwels, C. N. Potts e L. N. Van Wassenhove. Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS J. on Computing*, 10(3):341–350, Março 1998.

[DdFS10] Mitre C. Dourado, Rosiane de Freitas e Jayme Luis Szwarcfiter. Escalonamento em máquinas paralelas para minimizar o atraso ponderado de tarefas em tempos iguais. *Anais do XLII Simpósio Brasileiro de Pesquisa Operacional*, 1:1–10, 2010.

[dFPUP08] Rosiane de Freitas, Artur Pessoa, Eduardo Uchoa e Marcus Poggi. Heuristic algorithm for the parallel machine total weighted tardiness scheduling problem. *Relatórios de pesquisa em engenharia de produção*, 8:1–12, 2008.

[dFR96] Rosiane de Freitas Rodrigues. Times assíncronos para a resolução de problemas de otimização combinatória com múltiplas funções objetivo, Setembro 1996.

[dFR09] Rosiane de Freitas Rodrigues. *Caracterizações e algoritmos para problemas clássicos de escalonamento*. Tese de Doutorado, Rio de Janeiro, Maio 2009.

[dI11] Documentação da IBM. ILOG CPLEX optimization studio information center, 2011. Disponível em: http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r4/index.jsp.

[DL90] J. Du e Joseph Y.T. Leung. Minimizing total tardiness on one machine is NP-Hard. *Mathematics of Operations Research*, 15:483–495, 1990.

[DLLvdV90] M. I. Dessouky, B. J. Lageweg, J. K. Lenstra e S.L. van de Velde. Scheduling identical jobs on uniform parallel machines. *Statistica Neerlandica*, 44:115–123, 1990.

[DOcDT11] Tufan Demirel, Vildan Özkir, Nihan Çetin Demirel e Belgin Taşdelen. A genetic algorithm approach for minimizing total tardiness in parallel machine scheduling problems. *Proceedings of the World Congress on Engineering*, 2, 2011.

[DW90] M. E. Dyer e L. A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Appl. Math.*, 26(2-3):255–270, Fevereiro 1990.

[EC77] Samuel Eilon e I. G. Chowdhury. Minimising waiting time variance in the single machine problem. *Management Science*, 23:297–313, Fevereiro 1977.

[FHG12] Hamidreza Feili, Hamidreza Haddad e Payam Ghanbari. Two hybrid algorithms for single-machine total weighted tardiness scheduling problem with sequence-dependent setup. *American Journal of Scientific Research*, páginas 22–29, 2012.

[FMM01] Paulo M França, Alexandre Mendes e Pablo Moscato. A memetic algorithm for the total tardiness single machine scheduling problem. *European Journal of Operational Research*, 132:224 – 242, 2001.

[GDCT04] A Grosso, F Della Croce e R Tadei. An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Oper. Res. Lett.*, 32(1):68–72, Janeiro 2004.

[GJ79] Michael R. Garey e David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[GLLK79] R L Graham, E L Lawler, J K Lenstra e A H G Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[GLM00] Fred Glover, Manuel Laguna e Rafael Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

[GPC02] Valery Gordon, Jean-Marie Proth e Chengbin Chu. A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, 139:1–25, 2002.

[GS99] Valery S. Gordon e Vitaly A. Strusevich. Earliness penalties on a single machine subject to precedence constraints: SLK due date assignment. *Computers & OR*, 26:157–177, 1999.

[GS06] Skylab R. Gupta e Jeffrey S. Smith. Algorithms for single machine total tardiness scheduling with sequence dependent setups. *European Journal of Operational Research*, 175(2):722–739, 2006.

[HKS91] Nicholas G. Hall, Wieslaw Kubiak e Suresh P. Sethi. Earliness-tardiness scheduling problems, II: Deviation of completion times about a restrictive common due date. *Operations Research*, 39:847 – 856, 1991.

[HL93] Jeffrey W. Herrmann e Chung-Yee Lee. On scheduling to minimize earliness-tardiness and batch delivery costs with a common due date. *European Journal of Operational Research*, 70:272–288, 1993.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*, volume Ann Arbor. University of Michigan Press, 1975.

[HS05] Refael Hassin e Mati Shani. Machine scheduling with earliness, tardiness and non-execution penalties. *Computers & Operations Research*, 32(3):683 – 705, 2005.

[JB97] R.J.W. James e J.T. Buchanan. A neighbourhood scheme with a compressed solution space for the early/tardy scheduling problem. *European Journal of Operational Research*, 102(3):513 – 527, 1997.

[JC07] Aloísio Gomes Júnior e Carlos Carvalho. Um método heurístico híbrido para a resolução do problema de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção. *Anais do XXXIX Simpósio Brasileiro de Pesquisa Operacional*, 2007.

[Józ07] Joanna Józefowska. *Just-in-Time Scheduling: Models and Algorithms for Computer and Manufacturing Systems*. International Series in Operations Research & Management Science. Springer, 2007.

[KAK98] Murat Koksalan, Meral Azizoglu e Suna Koksalan Kondakci. Minimizing flowtime and maximum earliness on a single machine. *IIE Transactions*, 30:192–200, 1998.

[KL98] F. Kolahan e M. Liang. An adaptive TS approach to JIT sequencing with variable processing times and sequence-dependent setups. *European Journal of Operational Research*, 109(1):142–159, Agosto 1998.

[KS00] John J. Kanet e V. Sridharan. Scheduling with inserted idle time: problem taxonomy and literature review. *Operations Research*, 48:99–110, 2000.

[KSSS08] Safia Kedad-Sidhoum, Yasmin Rios Solis e Francis Sourd. Lower bounds for the earliness-tardiness scheduling problem on parallel machines with distinct due dates. *European Journal of Operational Research*, 189:1305–1316, 2008.

[KW09] S.A. Kravchenko e F. Werner. *Minimizing a separable convex function on parallel machines with preemptions*. Univ., Fak. für Mathematik, 2009.

[KW11] Svetlana A. Kravchenko e Frank Werner. Parallel machine problems with equal processing times: a survey. *J. Scheduling*, 14(5):435–444, 2011.

[LA04] Joseph Y-T. Leung e James H. Anderson. *Handbook of scheduling: algorithms, models and performance analysis*. Chapman and Hall/CRC, 2004.

[LAR05] Ning Liu, M. Abdelrahman e Srini Ramaswamy. A Genetic Algorithm for Single Machine Total Weighted Tardiness Scheduling Problem. *International Journal of Intelligent Control and Systems*, 10:218–225, Setembro 2005.

[Law01] Eugene L. Lawler. *Combinatorial Optimization : Networks and Matroids*. Dover Publications, Março 2001.

[LC06] Xiaochuan Luo e Feng Chu. A branch and bound algorithm of the single machine schedule with sequence dependent setup times for minimizing total tardiness. *Applied Mathematics and Computation*, páginas 575–588, 2006.

[Li97] George Li. Single machine earliness and tardiness scheduling. *European Journal of Operational Research*, 96(3):546 – 558, 1997.

[Luk09] Sean Luke. *Essentials of Metaheuristics*. Lulu, 2009. Disponível em://cs.gmu.edu/∼sean/book/metaheuristics/.

[LW04] V. Lauff e F. Werner. Scheduling with common due date, earliness and tardiness penalties for multimachine problems: a survey. *Mathematical and Computer Modelling*, 40(5-6):637 – 655, 2004.

[MFM02] A. Mendes, P. Franca e P. Moscato. Fitness landscapes for the total tardiness single machine scheduling problem. *Neural Network World*, 2:165–180, 2002.

[M'H07] Rym M'Hallah. Minimizing total earliness and tardiness on a single machine using a hybrid heuristic. *Computers and Operations Research*, 34(10):3126–3142, 2007.

[MM81] Heiner Müller-Merbach. Heuristics and their design: a survey. *European Journal of Operational Research*, 8(1):1 – 23, 1981.

[MR11] R. Martí e G. Reinelt. *The Linear Ordering Problem: Exact and Heuristic Methods in Combinatorial Optimization*. Applied Mathematical Sciences. Springer, 2011.

[NAG12] João Paulo Nogueira, José Elias Arroyo e Luciana Gonçalves. Uma heurística híbrida para o problema de sequenciamento de tarefas em máquinas paralelas não relacionadas. *Anais do XVI Simpósio Brasileiro de Pesquisa Operacional*, 1:1–12, 2012.

[OAV10] James Ostrowski, Miguel F. Anjos e Anthony Vannelli. Symmetry in scheduling problems. páginas 1–17, 2010.

[PBas] Artur Alves Pessoa e Eduardo Uchoa Barboza. UFFLP - an easy API for mixed, integer and linear programming, 2011, LOGIS - Núcleo de Logística Integrada e Sistemas. Disponível em: http://www.gapso.com.br/ufflp/.

[PE05] Supachai Pathumnakul e Pius J. Egbelu. Algorithm for minimizing weighted earliness penalty in single-machine problem. *European Journal of Operational Research*, 161(3):780–796, 2005.

[Pin12] Michael L. Pinedo. Scheduling: Theory, algorithms, and systems. *Springer Publishing Company, Incorporated*, 4a ed.:1–104, 2012.

[PR87] M. Padberg e G. Rinaldi. Optimization of a 532-city symsingle machine scheduling problems with anmetric traveling salesman problem by branch and cut. *Oper. Res. Lett.*, 6(1):1–7, Março 1987.

[PS82] Christos H. Papadimitriou e Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[PSdCAGO12] Puca Huachi Vaz Penna, Marcone Jamilson Freitas Souza, Frederico Augusto de Cezar Almeida Gonçalvez e Luiz Satoru Ochi. Uma heurística híbrida para minimizar custos com antecipação e atraso do sequenciamento da produção em uma máquina. *Produção*, 22(4):766–777, Dezembro 2012.

[PSK93] S.S. Panwalkar, M.L. Smith e C.P. Koulamas. A heuristic for the single machine tardiness problem. *European Journal of Operational Research*, 70(3):304 – 310, 1993.

[PUPdF10] Artur Pessoa, Eduardo Uchoa, Marcus Poggi e Rosiane de Freitas. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, 2:259–290, 2010.

[PWW68] A.A.B. Pritsker, L.J. Watters e P. Wolfe. *Multiproject Scheduling with*

Limited Resources: A Zero-one Programming Approach. Number N^{o} 3800 in P (Rand Corporation). Rand Corporation, 1968.

[RK08] M. Rebai e I. Kacem. Minimizing the earliness-tardiness costs on a single machine. Em *Service systems and service management, on 2008 international conference*, páginas 1 –5, Julho 2008.

[RMA04] Ghaith Rabadi, Mansooreh Mollaghasemi e Georgios C. Anagnostopoulos. A branch-and-bound algorithm for the early/tardy machine scheduling problem with a common due-date and sequence-dependent setup time. *Comput. Oper. Res.*, 31:1727–1751, Setembro 2004.

[RMAS06] Ghaith Rabadi, Reinaldo J. Moraga e Ameer Al-Salem. Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17:85–97, 2006.

[SKS03] Francis Sourd e Safia Kedad-Sidhoum. The one-machine problem with earliness and tardiness penalties. *Journal of Scheduling*, 6:533–549, 2003.

[SKS08] Francis Sourd e Safia Kedad-Sidhoum. A faster branch-and-bound algorithm for the earliness-tardiness scheduling problem. *J. of Scheduling*, 11(1):49–58, Fevereiro 2008.

[Sou06] Francis Sourd. Dynasearch for the earliness-tardiness scheduling problem with release dates and setup constraints. *Operations Research Letters*, 34(5):591 – 598, 2006.

[Sou09] Francis Sourd. New exact algorithms for one-machine earliness-tardiness scheduling. *INFORMS J. on Computing*, 21(1):167–175, Janeiro 2009.

[SS08] Dvir Shabtay e George Steiner. The single-machine earliness-tardiness scheduling problem with due date assignment and resource-dependent processing times. *Annals of Operations Research*, 159:25–40, 2008.

[SS12] Dvir Shabtay e George Steiner. Scheduling to maximize the number of just-in-time jobs: A survey. *Springer Optimization and Its Applications*, 60:3–20, 2012.

[ST12] Ameur Soukhal e Nguyen Huynh Toung. Just-in-time scheduling with equal-size jobs. Em Roger Z. Z. Ríos-Mercado e Yasmín A. A. Ríos-Solís, editors, *Just-in-Time Systems*, volume 60 of *Springer Optimization and Its Applications*, páginas 107–145. Springer New York, 2012.

[SV05] Shao Chin Sung e Milan Vlach. Maximizing weighted number of just-in-time jobs on unrelated parallel machines. *Journal of Scheduling*, 8:453–460, 2005.

[SW92] Jorge P. Sousa e Laurence A. Wolsey. A time indexed formulation of non-preemptive single machine scheduling problems. *Math. Program.*, 54:353–367, 1992.

[Tal09] El-Ghazali Talbi. *Metaheuristics : from design to implementation*, volume 10 of *The Sciences Po series in international relations and political economy*. John Wiley & Sons, 2009.

[Tan12a] Shunji Tanaka. Benchmark instances for the single-machine total weighted tardiness problem, 2012. Disponível em: http://turbine.kuee.kyoto-u.ac.jp/∼tanaka/index-e.html.

[Tan12b] Shunji Tanaka. An exact algorithm for the single-machine earliness-tardiness scheduling problem. *Springer Optimization and Its Applications*, 60:21–40, 2012.

[TF11] Shunji Tanaka e Shuji Fujikuma. A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time. *Journal of Scheduling*, páginas 1–15, Junho 2011.

[TFA09] Shunji Tanaka, Shuji Fujikuma e Mituhiko Araki. An exact algorithm for single-machine scheduling without machine idle time. *J. of Scheduling*, 12(6):575–593, Dezembro 2009.

[TN97] K.C. Tan e R. Narasimhan. Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach. *Omega*, 25:619–634, 1997.

[TNC06] Zhongjun Tian, C. T. Ng e T. C. E. Cheng. An $O(n^2)$ algorithm for scheduling equal-length preemptive jobs on a single machine to minimize total tardiness. *Journal of Scheduling*, 9:343–364, 2006.

[TNRR00] Keah-Choon Tan, Ram Narasimhan, Paul A. Rubin e Gary L. Ragatz. A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times. *Omega*, 28:313–326, 2000.

[TS08] Nguyen Huynh Tuong e Ameur Soukhal. Some new polynomial cases in just-in-time scheduling problems with multiple due dates. Em *2008 IEEE International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies, RIVF 2008, Ho Chi Minh City, Vietnam, 13-17 July 2008*, páginas 36–41. IEEE, 2008.

[TSA03] S. Tanaka, T. Sasaki e M. Araki. A branch-and-bound algorithm for the single-machine weighted earliness-tardiness scheduling problem with job independent weights. *Systems, Man and Cybernetics*, 2:1571–1577, 2003.

[TW12] Chi-Yang Tsai e Yi-Chen Wang. The sum of earliness and tardiness minimization on unrelated parallel machines with inserted idle time. *Latest Advances in Information Science and Applications*, páginas 125–130, Maio 2012.

[VD98] Sushil Verma e Dessouky. Single-machine scheduling of unit-time jobs with earliness and tardiness penalties. *Mathematics of Operations Research*, 23:930 – 943, 1998.

[vdAvHS99] J. M. van den Akker, C. P. M. van Hoesel e M. W. P. Savelsbergh. A polyhedral approach to single-machine scheduling problems. *Mathematical Programming*, 85(3):541, 1999.

[VR12] Eva Vallada e Rubén Ruiz. Scheduling unrelated parallel machines with sequence dependent setup times and weighted earliness-tardiness minimization. Em Roger Z. Z. Ríos-Mercado e Yasmín A. A. Ríos-Solís,

editors, *Just-in-Time Systems*, volume 60 of *Springer Optimization and Its Applications*, páginas 67–90. Springer New York, 2012.

[WL06] Chin-Chia Wu e Wen-Chiung Lee. Single machine scheduling to minimize the setup time and the earliness. *Journal of Information & Optimization Sciences*, 27(2):449 – 510, 2006.

[Wod08] Mieczyslaw Wodecki. A branch-and-bound parallel algorithm for single-machine total weighted tardiness problem. *The International Journal of Advanced Manufacturing Technology*, 37:996–1004, 2008.

[XL02] Wen-Qiang Xiao e Chung-Lun Li. Approximation algorithms for common due date assignment and job scheduling on parallel machines. *IIE Transactions*, 34:467–477, 2002.

[Yan00] Xiaoguang Yang. Scheduling with generalized batch delivery dates and earliness penalties. *IIE Transactions*, 32:735–741, 2000.

[YWW+12] Yunqiang Yin, Chin-Chia Wu, Wen-Hsiang Wu, Chou-Jung Hsu e Wen-Hung Wu. A branch-and-bound procedure for a single-machine earliness scheduling problem with two agents. *Applied Soft Computing*, páginas 1 – 12, 2012.

[YY12] Milad Yousefi e Rosnah Mohd Yusuff. *Minimizing Earliness and Tardiness Penalties in a Single Machine Scheduling Against Common Due Date using Genetic Algorithm*, volume 9. Maxwell Scientific Organization, Maio 2012.

[ZT07] Chuan-Li Zhao e Heng-Yong Tang. Single machine scheduling problemas with an aging effect. *J. Appl. Math and Computing*, 25:305–314, 2007.

Apêndice A

Apresentação da biblioteca UFFLP

Existem certos problemas para os quais tem-se uma formulação de programação matemática para o qual, a partir de uma instância do problema a ser resolvida, monta-se o modelo a ser resolvido pelo CPLEX. Geralmente, tais problemas costumam ter um número grande de variáveis, o que torna a montagem de instâncias para o otimizador interativo algo exaustivo. Para resolver isso, a ferramenta UFFLP ajuda a construir um modelo matemático particular, a partir de instâncias de teste, resolvê-lo utilizando o CPLEX e visualizar a solução no mesmo ambiente, o que torna o uso da UFFLP bem mais prático [PBas].

A UFFLP consiste em uma biblioteca de funções para integração entre softwares resolvedores de modelos de Programação Inteira Mista (PIM) e linguagens de programação tais como C/C++ e Visual Basic for Applications (VBA). Seu uso se tornou popular principalmente devido ao fato da linguagem VBA estar disponível dentro de Planilhas de Cálculo como o Microsoft Excel, o que facilita o tratamento dos dados utilizados pelos modelos matemáticos e a visualização dos resultados. As funções da UFFLP também podem ser chamadas de programas em C/C++ a partir do Windows ou do Linux. A UFFLP tem ainda as seguintes funcionalidades [PBas]:

- referência a variáveis e restrições de um modelo matemático apenas pelos seus nomes (*strings*);
- possibilidade de integração com heurísticas e rotinas de geração de cortes chamadas de *callbacks* em VBA ou em C++;
- pode utilizar como resolvedor tanto o *Coin-CBC*, que é gratuito, e o CPLEX, que gratuito apenas para fins acadêmicos;

- código fonte aberto e gratuito.

A UFFLP possui um conjunto de funções básicas que podem ser utilizados na criação do modelo matemático:

- ***UFFLP_CreateProblem*:** cria o problema de minimização ou maximização;
- ***UFFLP_AddVariable*:** insere uma nova variável no problema;
- ***UFFLP_SetCoefficient*:** insere um coeficiente de uma restrição;
- ***UFFLP_AddConstraint*:** insere uma restrição no problema;
- ***UFFLP_Solve*:** resolve o problema;
- ***UFFLP_GetObjValue*:** retorna o valor da função objetivo do problema;
- ***UFFLP_GetSolution*:** retorna o valor da variável na solução corrente;
- ***UFFLP_GetDualSolution*:** retorna o valor da variável dual na solução corrente;
- ***UFFLP_WriteLP*:** escreve o modelo no formato LP;
- ***UFFLP_SetLogInfo*:** define o nome do arquivo de *log*, nesse arquivo é gravada a saída do algoritmo;
- ***UFFLP_DestroyProblem*:** destrói um problema.

A concepção do algoritmo exato considerou o ambiente de máquinas paralelas idênticas, assim como nas heurísticas. A seguir é apresentado um exemplo de modelagem utilizando a biblioteca UFFLP e a formulação apresentada na seção 3.6.4. A construção do algoritmo começa na modelagem da função objetivo e suas restrições, sendo que para cada instância do problema, o tamanho da função objetivo e das restrições será diferente, o que torna necessário construir as equações de forma iterativa. Começando pela função objetivo, primeiramente deve-se começar com a criação do problema (minimização ou maximização):

```
/* Cria o problema de minimização */
UFFProblem* prob = UFFLP_CreateProblem( UFFLP_Minimize );
```

Para gerar a função objetivo, da formulação da seção 3.6.4, considera-se um somatório duplo, logo, são necessários dois laços de repetição para sua construção. Um dos laços percorrerá as j tarefas e o outro os $t, ..., T-1$ instantes possíveis, adicionando $f_j(t + p_j)y^t{}_j$ à expressão. Esse processo equivale ao seguinte código:

```
// procedimento para criação das variáveis binárias com os seus respectivos
// custos de antecipação e atraso
std :: string varName;
for (j = 0; j < n; j++)
{
    for (t = 0; t <= (T - proctime[j]); t++)
    {
        // variável "y_j_t" é 1 se a tarefa "j" é processada no instante "t"
        std :: stringstream s;
        s << "y_" << j << "_" << t;
        s >> varName;
        UFFLP_AddVariable(prob, /* problema de minimização criado */
                          (char*)varName.c_str(), /* nome da variável criada */
                           0.0, /* limite inferior da variável */
                           1.0, /* limite superior da variável */
                           /* função de define o valor do coeficiente das variáveis */
                           f(j, t, weight_e, weight_t, duedate, proctime),
                           UFFLP_Binary ); /* tipo de variável */
    }
}
```

Para a restrição 3.30, que a tarefa deve ser processada exatamente uma vez, tem-se 2 laços iguais aos da função objetivo para percorrer a matriz de variáveis, sendo que a soma das variáveis com mesmo índice de tarefas deve ser igual a 1. Existem n equações a serem adicionadas a respeito da primeira restrição. Com isso, tem-se o código mostrado a seguir:

```
// cria uma restrição onde cada tarefa deve ser executada exatamente em uma máquina
std :: string consName;
for (j = 0; j < n; j++)
{
    // define o nome da restrição " restr1_j "
    std :: stringstream s;
    s << " restr1_ " << j;
    s >> consName;
```

```
    //  define os coeficientes para a restrição
    for (t = 0; t < (T - proctime[j]); t++)
    {
   // adiciona a variável "y_j_t" a restrição
      std :: stringstream s;
      s << "y_" << j << "_" << t;
      s >> varName;
      UFFLP_SetCoefficient( prob, /* problema de minimização criado */
                            (char*)consName.c_str(), /* nome da restrição */
                            /* nome da variável que receberá o coeficiente */
                            (char*)varName.c_str(),
                            1 ); /* valor do coeficiente */
    }
    // cria a restrição
    UFFLP_AddConstraint( prob, /* problema de minimização criado */
                         (char*)consName.c_str(), /* nome da restrição */
                          1, /* valor que ficará do lado direito da restrição */
                          /* tipo de variável (UFFLP_Less, UFFLP_Equal ou UFFLP_Greater) */
                          UFFLP_Equal );
}
```

Para a restrição 3.31, que a máquina pode processar no máximo uma tarefa em um dado instante de tempo, é necessário que se tenha três laços de repetição, pois são dois somatórios por equação a respeito da restrição, sendo várias equações (no pior caso, T equações). Assim, para cada instante de tempo $t \in [0, T-1]$ percorrem-se todas as tarefas. Supondo-se que uma tarefa j é iniciada no instante t, se o tempo de término da tarefa não passar do tempo máximo considerado (ou seja, $t + p_j$), ela será considerada no somatório mais interno, no qual o valor inicial pode ser $t - p_j + 1$ se for maior que 0, caso contrário é assumido o valor 0 como valor inicial, e o valor final do somatório mais interno é o instante t. Assim, tem-se o código mostrado no código a seguir:

```
// cria a restrição que define que a máquina pode processar
// no máximo uma tarefa em um dado instante de tempo
for (t = 0; t <= (T - 1); t++)
{
    // define o nome da restrição " restr2_j "
    std :: stringstream s;
    s << " restr2_ " << t;
    s >> consName;
```

```
    for (j = 0; j < n; j++)
    {
    //   define os coeficientes para a restrição
    if ((t + proctime[j]) <= T)
    {
       if ((t - proctime[j] + 1) > 0)
       cont = t - proctime[j] + 1;
       else cont = 0;

       for (; cont <= t; cont++)
       {
      std :: stringstream st;
      st << "y_" << j << "_" << cont;
      st >> varName;
      UFFLP_SetCoefficient( prob,
                            (char*)consName.c_str(),
                            (char*)varName.c_str(),
                            1 );
       }
    }
    }
    // cria a restrição
    UFFLP_AddConstraint( prob, (char*)consName.c_str(), m, UFFLP_Less );
}
```

Para a restrição 3.33, que define que as tarefas devem iniciar no instante 0, evitando tempo ocioso no início do escalonamento. O código para essa restrição é apresentado a seguir:

```
// cria a restrição que define que as tarefas devem iniciar no instante 0
    std :: stringstream s;
    s << " restr0 ";
    s >> consName;

    for (j = 0; j < n; j++)
    {
    // adiciona a variável "y_j_0" a restrição
        std :: stringstream s1;
        s1 << "y_" << j << "_0";
```

```
        s1 >> varName;
        UFFLP_SetCoefficient( prob, (char*)consName.c_str(), (char*)varName.c_str(), 1 );

    }
    // cria a restrição
    UFFLP_AddConstraint( prob, (char*)consName.c_str(), m, UFFLP_Equal );
```

Os trechos de código mostrados acima fazem parte da modelagem de um modelo matemático utilizando a ferramenta de programação matemática UFFLP. Para exemplificar a execução deste modelo na prática, considere a instância apresentada na Figura 4.3 (a) e duas máquinas, onde, a partir desta instância, é gerado o modelo a seguir:

```
\ENCODING=ISO-8859-1
\Problem name:
Minimize
 obj: 9 y_0_0 + 6 y_0_1 + 3 y_0_2 + 5 y_0_4 + 10 y_0_5 + 15 y_0_6 + 20 y_0_7 + 25 y_0_8 +
      30 y_0_9 + 35 y_0_10 + 40 y_0_11 + 45 y_0_12 + 50 y_0_13 + 4 y_1_0 + 5 y_1_2 +
      10 y_1_3 + 15 y_1_4 + 20 y_1_5 + 25 y_1_6 + 30 y_1_7 + 35 y_1_8 + 40 y_1_9 +
      45 y_1_10 + 50 y_1_11 + 55 y_1_12 + 60 y_1_13 + 65 y_1_14 + 0 y_2_0 + 8 y_2_1 +
      16 y_2_2 + 24 y_2_3 + 32 y_2_4 + 40 y_2_5 + 48 y_2_6 + 56 y_2_7 + 64 y_2_8 +
      72 y_2_9 + 80 y_2_10 + 88 y_2_11 + 96 y_2_12 + 8 y_3_0 + 10 y_3_2 + 20 y_3_3 +
      30 y_3_4 + 40 y_3_5 + 50 y_3_6 + 60 y_3_7 + 70 y_3_8
Subject To
 restr1_0 : y_0_0 + y_0_1 + y_0_2 + y_0_3 + y_0_4 + y_0_5 + y_0_6 + y_0_7
            + y_0_8 + y_0_9 + y_0_10 + y_0_11 + y_0_12 = 1
 restr1_1 : y_1_0 + y_1_1 + y_1_2 + y_1_3 + y_1_4 + y_1_5 + y_1_6 + y_1_7
            + y_1_8 + y_1_9 + y_1_10 + y_1_11 + y_1_12 + y_1_13 = 1
 restr1_2 : y_2_0 + y_2_1 + y_2_2 + y_2_3 + y_2_4 + y_2_5 + y_2_6 + y_2_7
            + y_2_8 + y_2_9 + y_2_10 + y_2_11 = 1
 restr1_3 : y_3_0 + y_3_1 + y_3_2 + y_3_3 + y_3_4 + y_3_5 + y_3_6 + y_3_7 = 1
 restr2_0 : y_0_0 + y_1_0 + y_2_0 + y_3_0 <= 2
 restr2_1 : y_0_0 + y_0_1 + y_1_0 + y_1_1 + y_2_0 + y_2_1 + y_3_0 + y_3_1 <= 2
 restr2_2 : y_0_0 + y_0_1 + y_0_2 + y_1_0 + y_1_1 + y_1_2 + y_2_0 + y_2_1
            + y_2_2 + y_3_0 + y_3_1 + y_3_2 <= 2
 restr2_3 : y_0_0 + y_0_1 + y_0_2 + y_0_3 + y_1_1 + y_1_2 + y_1_3 + y_2_0
            + y_2_1 + y_2_2 + y_2_3 + y_3_0 + y_3_1 + y_3_2 + y_3_3 <= 2
 restr2_4 : y_0_1 + y_0_2 + y_0_3 + y_0_4 + y_1_2 + y_1_3 + y_1_4 + y_2_0
            + y_2_1 + y_2_2 + y_2_3 + y_2_4 + y_3_0 + y_3_1 + y_3_2 + y_3_3
            + y_3_4 <= 2
 restr2_5 : y_0_2 + y_0_3 + y_0_4 + y_0_5 + y_1_3 + y_1_4 + y_1_5 + y_2_1
```

+ y_2_2 + y_2_3 + y_2_4 + y_2_5 + y_3_0 + y_3_1 + y_3_2 + y_3_3
+ y_3_4 + y_3_5 <= 2

restr2_6 : y_0_3 + y_0_4 + y_0_5 + y_0_6 + y_1_4 + y_1_5 + y_1_6 + y_2_2
+ y_2_3 + y_2_4 + y_2_5 + y_2_6 + y_3_0 + y_3_1 + y_3_2 + y_3_3
+ y_3_4 + y_3_5 + y_3_6 <= 2

restr2_7 : y_0_4 + y_0_5 + y_0_6 + y_0_7 + y_1_5 + y_1_6 + y_1_7 + y_2_3
+ y_2_4 + y_2_5 + y_2_6 + y_2_7 + y_3_0 + y_3_1 + y_3_2 + y_3_3
+ y_3_4 + y_3_5 + y_3_6 + y_3_7 <= 2

restr2_8 : y_0_5 + y_0_6 + y_0_7 + y_0_8 + y_1_6 + y_1_7 + y_1_8 + y_2_4
+ y_2_5 + y_2_6 + y_2_7 + y_2_8 + y_3_0 + y_3_1 + y_3_2 + y_3_3
+ y_3_4 + y_3_5 + y_3_6 + y_3_7 + y_3_8 <= 2

restr2_9 : y_0_6 + y_0_7 + y_0_8 + y_0_9 + y_1_7 + y_1_8 + y_1_9 + y_2_5
+ y_2_6 + y_2_7 + y_2_8 + y_2_9 <= 2

restr2_10 : y_0_7 + y_0_8 + y_0_9 + y_0_10 + y_1_8 + y_1_9 + y_1_10 + y_2_6
+ y_2_7 + y_2_8 + y_2_9 + y_2_10 <= 2

restr2_11 : y_0_8 + y_0_9 + y_0_10 + y_0_11 + y_1_9 + y_1_10 + y_1_11 + y_2_7
+ y_2_8 + y_2_9 + y_2_10 + y_2_11 <= 2

restr2_12 : y_0_9 + y_0_10 + y_0_11 + y_0_12 + y_1_10 + y_1_11 + y_1_12 + y_2_8
+ y_2_9 + y_2_10 + y_2_11 + y_2_12 <= 2

restr2_13 : y_0_10 + y_0_11 + y_0_12 + y_0_13 + y_1_11 + y_1_12 + y_1_13 <= 2

restr2_14 : y_1_12 + y_1_13 + y_1_14 <= 2

restr0 : y_0_0 + y_1_0 + y_2_0 + y_3_0 = 2

Bounds

0 <= y_0_0 <= 1 0 <= y_0_1 <= 1 0 <= y_0_2 <= 1 0 <= y_0_3 <= 1 0 <= y_0_4 <= 1

0 <= y_0_5 <= 1 0 <= y_0_6 <= 1 0 <= y_0_7 <= 1 0 <= y_0_8 <= 1 0 <= y_0_9 <= 1

0 <= y_0_10 <= 1 0 <= y_0_11 <= 1 0 <= y_0_12 <= 1 0 <= y_0_13 <= 1 0 <= y_1_0 <= 1

0 <= y_1_1 <= 1 0 <= y_1_2 <= 1 0 <= y_1_3 <= 1 0 <= y_1_4 <= 1 0 <= y_1_5 <= 1

0 <= y_1_6 <= 1 0 <= y_1_7 <= 1 0 <= y_1_8 <= 1 0 <= y_1_9 <= 1 0 <= y_1_10 <= 1

0 <= y_1_11 <= 1 0 <= y_1_12 <= 1 0 <= y_1_13 <= 1 0 <= y_1_14 <= 1 0 <= y_2_0 <= 1

0 <= y_2_1 <= 1 0 <= y_2_2 <= 1 0 <= y_2_3 <= 1 0 <= y_2_4 <= 1 0 <= y_2_5 <= 1

0 <= y_2_6 <= 1 0 <= y_2_7 <= 1 0 <= y_2_8 <= 1 0 <= y_2_9 <= 1 0 <= y_2_10 <= 1

0 <= y_2_11 <= 1 0 <= y_2_12 <= 1 0 <= y_3_0 <= 1 0 <= y_3_1 <= 1 0 <= y_3_2 <= 1

0 <= y_3_3 <= 1 0 <= y_3_4 <= 1 0 <= y_3_5 <= 1 0 <= y_3_6 <= 1 0 <= y_3_7 <= 1

0 <= y_3_8 <= 1

Binaries

y_0_0 y_0_1 y_0_2 y_0_3 y_0_4 y_0_5 y_0_6 y_0_7 y_0_8 y_0_9 y_0_10

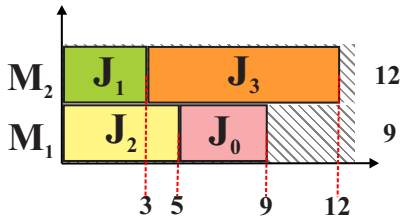
y_0_11 y_0_12 y_0_13 y_1_0 y_1_1 y_1_2 y_1_3 y_1_4 y_1_5 y_1_6

y_1_7 y_1_8 y_1_9 y_1_10 y_1_11 y_1_12 y_1_13 y_1_14 y_2_0 y_2_1

y_2_2 y_2_3 y_2_4 y_2_5 y_2_6 y_2_7 y_2_8 y_2_9 y_2_10 y_2_11

```
 y_2_12  y_3_0  y_3_1  y_3_2  y_3_3  y_3_4  y_3_5  y_3_6  y_3_7  y_3_8
End
```

Dessa forma, a solução ótima para o modelo acima é apresentada na Figura A.1 (a), que mostra a distribuição das tarefas nas máquinas e o valor das variáveis do modelo matemático, juntamente com o valor de função objetivo para antecipação e atraso ponderados na Figura A.1 (b).



(a)

$$y_2^0 = 1 \qquad y_3^3 = 1$$
$$y_1^0 = 1 \qquad y_0^5 = 1$$
$$FO = 0\,y_2^0 + 4\,y_1^0 + 10\,y_0^5 + 20\,y_3^3 = 34$$

(b)

Figura A.1: Apresentação da solução para 4 tarefas em 2 máquinas: (a) representação da solução no gráfico no gráfico de Gantt (b) valor das variáveis no modelo matemático e valor de função objetivo (FO).

Apêndice B

Apresentação da ferramenta CPLEX

O CPLEX é um conjunto de aplicações e bibliotecas de otimização para resolver problemas de programação linear, quadrática e inteira (mista) desenvolvido pela ILOG (agora parte da IBM). O CPLEX possui uma enorme utilidade para problemas de programação linear [dI11].

O CPLEX utiliza o algoritmo *branch-and-cut* para solucionar problemas de programação linear mista. O procedimento de *branch-and-cut* lida com vários nós de uma árvore de busca, onde cada nó representa um subproblema a ser processado, ou seja, a ser checado a sua integralidade e se esse nó pode ou não ser podado. O CPLEX processa os nós da árvore de busca até que não se tenham mais nós ativos disponíveis ou se algum limite foi atingido.

Primeiramente a árvore de *branch-and-cut* é inicializada a fim de que se tenha um nó raiz. O nó raiz da árvore representa o problema original, ignorando todas as restrições de integralidade. Os cortes são aplicados ao nó raiz com a finalidade de reduzir o espaço de busca. Se uma solução inteira for encontrada, ela será factível para o problema original e será considerada a melhor solução atualmente encontrada.

Um ramo (*branch*) é a criação de dois novos de um nó pai. Normalmente, um ramo ocorre quando os limites de uma única variável são modificados, esses novos limites são herdados pelos seus descendentes. Por exemplo, se uma ramo ocorre em uma variável binária, ou seja, variável que possui o limite inferior 0 (zero) e um limite superior 1 (um), então o resultado serão dois nós, um nó com o limite superior 0 (o limite inferior do ramo descendente dessa variável pode assumir somente o valor 0), e o outro nó com o limite superior modificado 1 (o limite superior do ramo descendente deve possuir somente o

valor 1). Os dois novos nós da árvore terão assim dois novos domínios distintos. Uma restrição de corte é adicionada ao modelo.

O objetivo de se adicionar cortes ao modelo matemático é limitar o tamanho do espaço de soluções contínuas (obtidas pela relaxação linear do método - onde é removida a restrição de integralidade do problema), mas sem eliminar soluções inteiras. Onde deseja-se reduzir o número de ramos necessários para solucionar o problema.

Quando processa um nó, o CPLEX começa o seu processamento resolvendo a relaxação contínua do subproblema (problema sem as restrições de integralidade). O CPLEX pode adicionar um ou vários cortes no subproblema até solucioná-lo. Neste processo de adição de cortes, o nó é removido (podado) da árvore quando o subproblema se torna infactível. Caso contrário, o CPLEX verifica se a solução do nó satisfaz as restrições de integralidade, caso essa solução é factível, e se o seu valor de Função Objetivo é melhor que a solução atual incumbente, a solução do nó é utilizada como nova solução incumbente. Senão, a ramificação irá ocorrer, mais primeiramente um método heurístico pode ser utilizado neste nó para verificar se uma nova solução incumbente pode ser inferida a partir deste nó. A ramificação, quando ocorre, é realizada em uma variável onde o valor da solução viola as restrições de integralidade. Esta prática resulta em dois nós que são adicionados na árvore para que seja processadas posteriormente.

As figuras B.1 e B.2 mostram a saída do algoritmo exato para a instância de número 21 de 40 tarefas em 1 máquina e 4 máquinas, respectivamente, onde é possível observar que no processamento para o ambiente de 1 máquina são necessárias mais ramificações na árvore de *branch-and-cut* para se atingir a solução ótima, por outro lado, no processamento para o ambiente de máquinas paralelas, o *branch-and-cut* do CPLEX apresenta menos ramificações na árvore para atingir a solução ótima do problema.

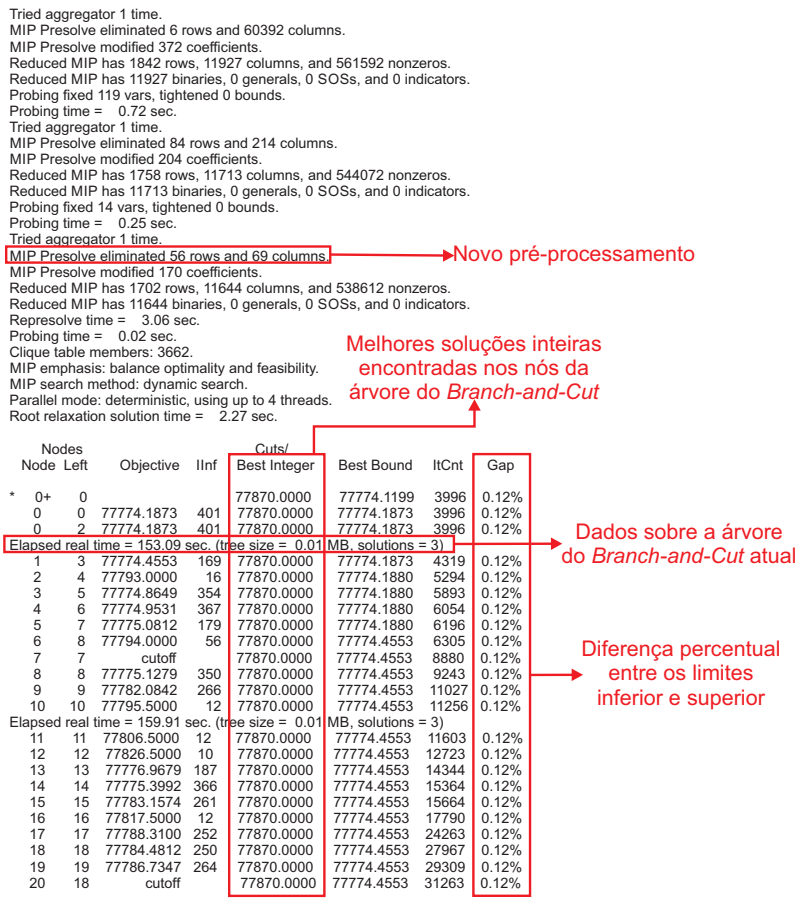


Figura B.1: Saída do CPLEX para 40 tarefas em ambiente monoprocessado.

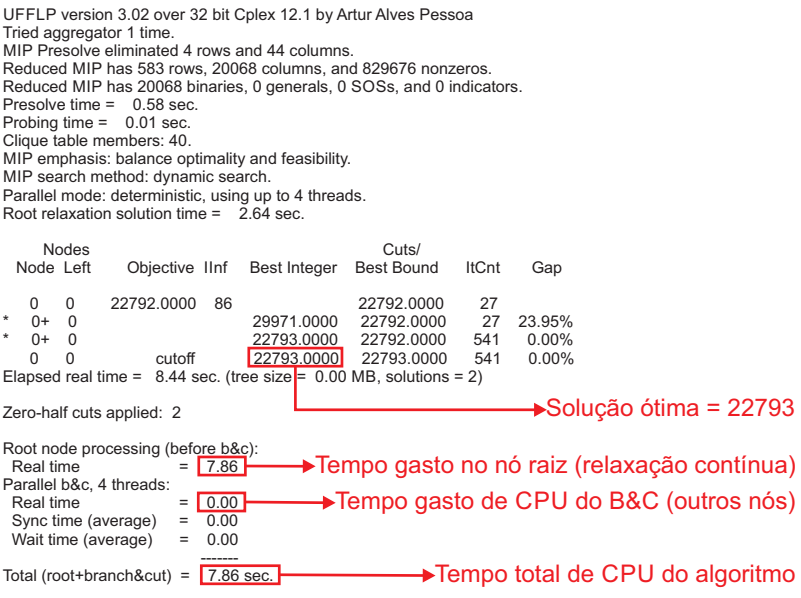


Figura B.2: Saída do CPLEX para 40 tarefas em 4 máquinas.