Pontifícia Universidade Católica
do Rio de Janeiro

Bruno Leonardo Kmita de Oliveira Passos

Estudo de heurísticas para problemas de escalonamento em um ambiente com máquinas indisponíveis

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática da PUC–Rio

Orientador: Prof. Eduardo Sany Laber

Rio de Janeiro
Junho de 2014

Pontifícia Universidade Católica
do Rio de Janeiro

Bruno Leonardo Kmita de Oliveira Passos

Estudo de heurísticas para problemas de escalonamento em um ambiente com máquinas indisponíveis

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC–Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Eduardo Sany Laber
Orientador
Departamento de Informática — PUC–Rio

Prof. Marcus Vinicius Soledade Poggi de Aragao
Departamento de Informática – PUC-Rio

Prof. David Sotelo Pinheiro da Silva
Departamento de Informática – PUC-Rio

Prof. Alexandre Roberto Renteria
Polo Capital

Prof. José Eugenio Leal
Coordenador Setorial do Centro Técnico Científico — PUC–Rio

Rio de Janeiro, 16 de Junho de 2014

Bruno Leonardo Kmita de Oliveira Passos

Graduou–se em Engenharia Eletrônica e de Computação pela Universidade Federal do Rio de Janeiro.

Ficha Catalográfica

Passos, Bruno Leonardo Kmita de Oliveira

Estudo de heurísticas para problemas de escalonamento em um ambiente com máquinas indisponíveis : / Bruno Leonardo Kmita de Oliveira Passos; orientador: Eduardo Sany Laber — 2014.

90 f: il. (color.); 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2014.

Inclui bibliografia.

1. Informática – Teses. 2. Escalonamento. 3. Sequenciamento. 4. Indisponibilidade. 5. Quebra de máquinas. 6. NP-difícil. 7. Mercado Financeiro. 8. Precificação de Ativos. 9. Sistema de Risco. I. Laber, Eduardo Sany. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

À Pontifícia Universidade Católica do Rio de Janeiro, pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado;

Ao professor Eduardo Laber, orientador da dissertação, por todo suporte e ensinamentos durante o período da dissertação;

À Polo Capital, pelo incentivo para elaboração desta pesquisa;

Aos amigos Alexandre, André e Marcelo pela colaboração no tema desta dissertação;

Aos meus pais, por todo esforço, oração, incentivo e apoio necessários;

À minha esposa, Mariana Martins, pela paciência e apoio incondicional durante toda esta jornada.

Resumo

Passos, Bruno Leonardo Kmita de Oliveira; Laber, Eduardo Sany. **Estudo de heurísticas para problemas de escalonamento em um ambiente com máquinas indisponíveis**. Rio de Janeiro, 2014. 90p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Grande parte da literatura de problemas de escalonamento assume que todas as máquinas estão disponíveis durante todo o período de análise o que, na prática, não é verdade, pois algumas das máquinas podem estar indisponíveis para processamento sem aviso prévio devido a problemas ou a políticas de utilização de seus recursos. Nesta tese, exploramos algumas das poucas heurísticas disponíveis na literatura para a minimização do makespan para este tipo de problema NP-difícil e apresentamos uma nova heurística que utiliza estatísticas de disponibilidade das máquinas para gerar um escalonamento. O estudo experimental com dados reais mostrou que a nova heurística apresenta ganhos de makespan em relação aos demais algoritmos clássicos que não utilizam informações de disponibilidade no processo de decisão. A aplicação prática deste problema está relacionada a precificação de ativos de uma carteira teórica de forma a estabelecer o risco de mercado da forma mais rápida possível através da utilização de recursos tecnológicos ociosos.

Palavras–chave

Escalonamento. Sequenciamento. Indisponibilidade. Quebra de máquinas. NP-difícil. Mercado Financeiro. Precificação de Ativos. Sistema de Risco.

Abstract

Passos, Bruno Leonardo Kmita de Oliveira; Laber, Eduardo Sany. **Scheduling algorithms application for machine availability constraint**. Rio de Janeiro, 2014. 90p. MsC Thesis — Department of Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Most literature in scheduling theory assumes that machines are always available during the scheduling time interval, which in practice is not true due to machine breakdowns or resource usage policies. We study a few available heuristics for the NP-hard problem of minimizing the makespan when breakdowns may happen. We also develop a new scheduling heuristic based on historical machine availability information. Our experimental study, with real data, suggests that this new heuristic is better in terms of makespan than other algorithms that do not take this information into account. We apply the results of our investigation for the asset-pricing problem of a fund portfolio in order to determine a full valuation market risk using idle technological resources of a company.

Keywords

Scheduling. Sequencing. Unavailability period. Breakdown. NP-hard. Finance. Pricing. Risk System.

Sumário

1
Introdução

Escalonamento de processos ou agendamento de tarefas pode ser entendido como o processo de decisão de alocação de recursos para execução de tarefas (ou atividades) durante um período de tempo de forma a otimizar um ou mais objetivos.

Trata-se de um processo comum que utilizamos no dia a dia para nos organizar e planejar. Os recursos e as tarefas podem tomar diferentes formas, dependendo do problema abordado. Recursos podem ser máquinas ou processadores, mesas de jantar, salas de aula, pistas de avião, empregados numa construção, dentre outros. As tarefas podem ser vistas como programas a serem executados, pessoas com horário para jantar, aulas numa universidade, pousos e decolagens nas pistas de um aeroporto ou estágios de construção numa obra.

Cada uma destas tarefas podem ter diferentes prioridades, requisitos para começar o mais rápido possível ou uma data limite para sua entrega ou execução. Além disso, problemas podem ocorrer durante a execução do planejamento, como por exemplo, quebra de máquinas, atraso na entrega de um prato no jantar que pode resultar em um atraso em outras reservas ou atraso na decolagem de um avião. Quando estes eventos inesperados ocorrem podemos ter mudanças no planejamento de forma a minimizar os danos que podem ser causados, como por exemplo redirecionar a execução de um programa para outra máquina, avisar aos clientes o atraso ou redirecionar uma aeronave para outro aeroporto.

Como podemos perceber, o escalonamento de tarefas como processo de decisão, possui uma importância muito grande na maioria dos sistemas de produção, nos processos de manufatura e no processamento de informações. Ele também é crítico para questões de transporte, sistemas de distribuição e em outros tipos de indústrias. Como a motivação deste ramo de estudo surgiu de problemas de processos de manufatura, os termos e vocabulários utilizados seguem ainda a modelagem feita no passado. Desta forma, temos que os recursos são chamados de máquinas e as tarefas de trabalhos[1].

[1] nesta tese utilizaremos para as tarefas a terminologia em inglês "jobs"

Segundo Baker (3), a teoria de escalonamento preocupa-se principalmente com modelagens matemáticas para representar problemas de escalonamento. O desenvolvimento de bons modelos, que geram técnicas de solução e *insights* práticos, tem sido a interface entre teoria e prática. A perspectiva teórica é em geral uma abordagem quantitativa que fazemos para capturar a estrutura de um problema na sua forma matemática, que começa com a descrição dos recursos e tarefas e com a tradução do objetivo do processo decisório em uma função objetivo explícita.

Idealmente, a função objetivo deve representar todos os custos que dependem do escalonamento e geralmente levam em consideração 3 tipos de objetivos: *turnaround* (tempo requerido para completar uma tarefa), *timeliness* (mede a conformidade de uma tarefa ser completada ou iniciada em um tempo limite) e *throughput* (mede a quantidade de trabalho completado num período de tempo)(3).

A modelagem de processos de escalonamento é categorizada através da configuração dos recursos e da natureza das tarefas. Por exemplo, uma modelagem de um problema de escalonamento pode conter apenas uma máquina ou várias máquinas capazes de processar tarefas. No caso de muitas máquinas, elas podem ser divididas em estágios, onde cada máquina é responsável por um estágio do processamento (como numa linha de produção), ou podem todas participar em paralelo do processamento das tarefas. No caso das tarefas, elas podem ser conhecidas a priori, tornando o sistema estático, ou podem ir aparecendo ao longo do tempo tornando o sistema dinâmico.

Quando temos todas as informações sobre o problema, dizemos que este é determinístico, do contrário, se qualquer informação relevante não for conhecida a priori, mas for representada por uma distribuição de probabilidade, dizemos que é estocástico.

Os problemas de escalonamento tem sido estudado por muitos pesquisadores. No entanto, a grande maioria dos modelos consideram que as máquinas estão sempre disponíveis para o processamento, o que não é real do ponto de vista prático.

Por exemplo, no escalonamento de processos de produção, ordens são fixas em termos de princípio e fim. Se novas ordens chegam, dado que os recursos estão sendo consumidos pelo processamento das ordens antigas, elas precisam ser processadas utilizando o restante de tempo livre dos intervalos de tempo. Neste caso, as ordens antigas que estão sendo processadas podem ser entendidas como intervalos de tempo onde as máquinas não estão disponíveis para processamento.

1.1 Motivação e Objetivos

Segundo Lee (24), problemas clássicos de escalonamento consideram que o ambiente de máquinas está continuamente disponível para processamento de tarefas, o que pode ser razoável em alguns casos, mas não satisfaz uma parte dos problemas práticos. Um exemplo disto é a quebra de máquinas após ou durante o processo de escalonamento que pode implicar na mudança do sequenciamento enviado para as máquinas de forma a conseguir finalizar a tarefa sem perdê-la.

O problema motivador desta tese tem como base uma empresa do mercado financeiro, chamada Polo Capital, que possui um parque tecnológico bem avançado onde em grande parte do dia estas máquinas são sub utilizadas em termos de memória e processamento. Existem vários processos que exigem poder computacional intensivo e que são executados por apenas uma única máquina.

Dentre estes vários processos existe o sistema para cálculo de risco de mercado, cujo processo de cálculo é facilmente paralelizável. Deseja-se então, utilizar os recursos disponíveis deste parque tecnológico, porém com a restrição de não afetar o dia a dia dos usuários destas estações de trabalho.

Isto pode ser modelado como um problema de escalonamento onde nosso objetivo é executar um conjunto muito grande de tarefas no menor tempo possível, paralelizando ao máximo os cálculos necessários para sua conclusão em máquinas com diferentes velocidades de processamento. No entanto, neste caso, temos que considerar que as máquinas podem ficar indisponíveis em qualquer instante, dado que o usuário pode estar utilizando intensamente sua máquina para a realização de suas tarefas diárias, ou pode até mesmo ter reiniciado a mesma durante o dia.

Nossos objetivos nesta tese são:

- Estudar um problema de escalonamento com máquinas em paralelo com restrições de disponibilidade;
- Propor e implementar heurísticas que levem em consideração estatísticas de disponibilidade das máquinas para auxiliar o processo de alocação de tarefas;
- Avaliar os resultados destas heurísticas utilizando um conjunto de instâncias coletadas de um ambiente real.

Os resultados obtidos com as novas heurísticas apresentaram melhorias em relação aos algoritmos clássicos, porém estas não foram suficientemente

significativas para concluirmos que elas devem ser utilizadas numa aplicação real, dado o tempo gasto na computação do escalonamento.

Esta tese está organizada da seguinte maneira. Apresentaremos no Capítulo 2 conceitos básicos de mercado financeiro, mapeamento de ativos em fatores primitivos de risco, cálculos para composição do risco e sua importância para uma instituição financeira. Ainda neste capítulo apresentaremos a notação adotada nesta tese para problemas de escalonamento e a formulação oficial do problema.

O Capítulo 3 possui uma breve revisão bibliográfica sobre algoritmos e complexidade computacional, suficiente para o entendimento sobre a dificuldade do problema e o porque ele é classificado como NP-completo. Em seguida, temos a revisão de alguns problemas e algoritmos propostos até hoje para ambientes com indisponibilidade de máquinas.

O Capítulo 4 apresenta os algoritmos implementados nesta tese e o Capítulo 5 o ambiente de simulação utilizado nos testes, detalhando tanto o simulador de algoritmos como os programas de monitoramento utilizados para obter os dados reais dos jobs e do ambiente de máquinas.

Finalmente, temos no Capítulo 6 os resultados obtidos com os algoritmos implementados e a apresentação de duas análises para diferentes grupos de jobs, considerando um ambiente com muita disponibilidade e um com muita indisponibilidade. Concluiremos a tese no Capítulo 7, apresentando também algumas propostas de trabalhos futuros.

2
Conceitos Básicos

Este capítulo tem como objetivo apresentar os conceitos básicos de escalonamento, algumas técnicas disponíveis na literatura, a notação que será adotada e como associamos os problemas de escalonamento à motivação inicial que é a otimização de um sistema para cálculo de risco de mercado. No entanto, para contextualizar o problema abordado, introduziremos inicialmente uma visão geral sobre cálculo de margem de garantia e fatores primitivos de risco. Os detalhes dos modelos de precificação por marcação a mercado de ativos no mercado financeiro mais relevantes para esta tese podem ser encontrados no Apêndice B.

O capítulo está organizado da seguinte forma. Na Seção 2.1 apresentaremos uma breve introdução da metodologia para cálculo de margem de garantia utilizada no Brasil e uma das metodologias utilizadas para o cálculo de risco de mercado. Na Seção 2.2, apresentaremos a notação e definiremos formalmente os problemas mais comuns de escalonamento disponíveis na literatura e finalmente, na Seção 2.3, apresentaremos a formulação do problema estudado nesta tese.

2.1
Risco de mercado e precificação de ativos

Nesta seção apresentamos alguns conceitos básicos sobre mercado financeiro no Brasil, começando com o conceito de carteira de investimentos e a razão da necessidade de um controle de risco sobre eles. Em seguida, abordamos modelos de precificação.

O conceito de carteira de investimentos, pode ser definido como um conjunto de ativos pertencentes a um investidor, pessoa física ou jurídica. Estes ativos podem ser ações, fundos, títulos públicos, debentures, aplicações imobiliárias, entre outros, o que permite tanto a diversificação de ativos como de risco.

A gestão de investimentos tem incorporada uma relação clara entre risco e retorno. O retorno esperado pelo gestor de uma carteira está relacionado com o risco que pretende correr. No caso do gestor ser avesso ao risco irá optar

por uma carteira com menor risco, logo menor possibilidade de retorno. Se for propenso ao risco irá optar por uma carteira de maior risco, logo maior possibilidade de retorno.

Uma carteira de investimentos pode ser dividida internamente combinando compra e venda de títulos em diferentes estratégias. Desta forma, é interessante para o gestor da carteira separar a análise do PnL[1] de cada uma das estratégias de investimento para entender quais são àquelas mais promissoras em termos de risco e retorno.

A Tabela 2.1 exemplifica a composição de uma carteira de investimentos com duas estratégias distintas A e B. Neste exemplo, temos que o gestor optou na estratégia A vender 150 ações da Petrobrás (PETR4) e comprar 300 ações da companhia Vale do Rio Doce (VALE5) e numa estratégia B comprar 200 ações da Petrobrás e comprar 200 opções de venda de VALE5 com vencimento em janeiro de 2015 (VALEM28).

Estratégia	PETR4 (Qtd)	VALE5 (Qtd)	VALEM28 (Qtd)
A	-150	300	0
B	200	0	200

Tabela 2.1: Exemplo de uma composição de ativos para uma carteira de investimentos

A diminuição de risco de uma carteira pode ser conseguida pela sua diversificação, que reduz o risco único, mas não necessariamente reduz o risco do mercado. Assim, é importante saber o efeito que cada título poderá ter na carteira, tanto no nível das estratégias, como no nível da carteira em si (no exemplo da Tabela 2.1 teríamos, para a carteira, um total de 50 ações de PETR4, 300 VALE5 e 200 VALEM28).

Para saber qual a contribuição de um ativo no risco de uma carteira diversificada, é necessário medir o seu risco de mercado, o que implica em avaliar a sua sensibilidade às variações do mercado. Uma medida utilizada é o desvio padrão dos resultados que chamaremos de volatilidade (σ).

Além disso, alguns ativos requerem garantias junto à Câmaras de Liquidação[2], como veremos na Seção 2.1.1. As Seções 2.1.1, 2.1.2 e 2.1.4 são baseadas no documento oficial da BM&FBOVESPA que pode ser encontrado em (6).

[1]Notação para Profit and Loss que pode ser entendida também como resultado ou retorno financeiro.

[2]Câmaras de Liquidação atuam como uma contraparte central nas negociações de compra e venda de títulos. Desta forma, a instituição se interpõe entre operações e contratos, tornando-se a compradora para todos os vendedores e a vendedora para todos os compradores.

2.1.1
Cálculo de margem de garantia

O cálculo de margem de garantia é uma ferramenta importantíssima de controle de risco de mercado, pois os mercados de Bolsa e de balcão no Brasil tem na figura da Câmara de Liquidação uma contraparte central garantidora para fins de liquidação e, portanto, exige de seus participantes o depósito de garantias.

Margem, é a denominação para valor em garantia, com as seguintes variações:

Margem de garantia requerida: É o valor mínimo que o participante deve depositar junto à Câmara de Liquidação para garantir a liquidação das obrigações decorrentes das operações a ele atribuídas.
Margem de garantia depositada: É o valor que o participante mantém depositado junto à Câmara de Liquidação, para garantir a liquidação das obrigações decorrentes das operações a ele atribuídas.
Chamada de margem de garantia: É a diferença negativa entre a margem de garantia requerida e a margem de garantia depositada, ou seja, é o valor que o participante deve depositar junto à Câmara a fim de atender ao requerimento de margem.

O valor da margem de garantia requerida do participante deve ser suficiente para cobrir o custo total de encerramento das posições de sua carteira em caso de inadimplência. Até que a carteira do participante faltoso seja integralmente liquidada, os preços, taxas e indicadores de mercado podem sofrer alterações, modificando o valor do correspondente custo de liquidação.

Por este motivo, o valor da margem de garantia requerida da carteira deve ser suficiente para cobrir seu custo de liquidação a valor de mercado e a potencial elevação deste custo, definida como risco de mercado da carteira e avaliada por meio de metodologias de teste de cenários de estresse.

A margem de garantia requerida pode então ser representada, de forma bastante geral, pela equação 2-1.

$$Margem = CL - RM + TA \tag{2-1}$$

Onde:

- CL: Custo de liquidação (valor envolvido no encerramento da posição)
- RM: Risco de mercado
- TA: Termos adicionais definidos pela Câmara

Segundo (6), a margem de garantia requerida dos participantes é atualizada com frequência diária após a compensação dos negócios do dia. A chamada de margem resultante deve ser atendida com o depósito de garantias em dinheiro ou, a critério da Câmara, em ativos e/ou outros instrumentos financeiros. Por meio do acompanhamento de risco intradiário, a Câmara é capaz de antecipar a chamada de margem, tantas vezes quantas forem necessárias ao longo do dia, com base nas operações realizadas pelos participantes e em suas posições em aberto nos mercados de Bolsa e de Balcão com garantia, atualizadas.

Os critérios para constituição, movimentação e utilização de garantias, bem como para atendimento à antecipação de chamada de margem estão descritos no Manual de Procedimentos Operacionais da Câmara em (5). Um exemplo detalhado de utilização desta equação pode ser encontrado em (4)

Sistemas que utilizam essa metodologia de forma rápida e eficiente, podem gerar simulações de exposição a risco de mercado para auxiliar diversas decisões, tal como o bloqueio de novas operações, desalavancagem ou alavancagem da carteira de investimentos, evitando uma eventual liquidação de seus ativos pela Câmara.

O custo de liquidação e os termos adicionais definidos pela Câmara são variáveis que dependem apenas dos ativos da carteira de investimentos e não serão considerados na análise desta tese. Já o risco de mercado depende de uma análise complexa que envolve múltiplos fatores e conceitos, tais como o conceito de fatores primitivos de risco, cenário de variação para um fator primitivo de risco, área de cenários e variação financeira sob cenário. Esses conceitos formarão as bases da metodologia de *full valuation* de cenários de estresse (16).

2.1.2 Fatores primitivos de risco

Fatores primitivos de risco (FPR) associados a um contrato derivativo é a denominação dada às variáveis financeiras relevantes à formação do valor, ou preço, do contrato. Ao se definir o valor de um contrato derivativo por meio de uma relação matemática envolvendo um conjunto de variáveis econômicas, fica implicitamente estabelecido que tais variáveis representam os fatores primitivos de risco do contrato.

Desta forma, para a formação do preço de um determinado ativo teremos, muitas vezes, um conjunto de N_{FPR} fatores primitivos de risco, os quais representaremos por $C_{FPR} = \{FPR^1, FPR^2, \ldots, FPR^{N_{FPR}}\}$. Cada fator FPR^i corresponde a um fator primitivo de risco, por exemplo, $FPR^{Ibovespa}$

está relacionado ao índice Ibovespa.

Além disso, um FPR pode representar também uma série temporal quando associada a prazos de referência (denominados vértices). Um exemplo é o fator primitivo de risco da taxa de juros pré fixada que é representada pelo vértice de 1 dia e pelos vértices múltiplos de 21 dias úteis. Desta forma, no caso da taxa de juros pré fixada, teríamos o fator primitivo de risco representado por $FPR^{Pre} = \{FPR^{Pre,1}, FPR^{Pre,21}, FPR^{Pre,42}, \dots\}$ (referentes, neste caso, a 1, 21 e 42 dias).

Podemos então afirmar que a identificação de FPRs é a definição do conjunto de FPRs associados a um contrato e que a decomposição, ou mapeamento, de uma posição em FPRs consiste em, identificados os FPRs a ela associados, expressar sua exposição a risco como função das exposições aos FPRs. A Tabela 2.2 apresenta um exemplo de mapeamento em fatores de risco para LTN, NTN-F, LFT, NTN-D, NTN-A1 e NTN-C.

É possível observar nesta tabela que títulos diferentes podem apresentar fatores em comum. Desta forma, a combinação de posições compradas e vendidas pode implicar na eliminação total ou parcial da exposição a um ou mais fatores de risco. Tal é o caso, por exemplo, da compra de NTN-D e da venda de NTN-A1 em que a exposição ao cupom cambial e ao dólar podem ser eliminadas.

	termo ou taxa de deságio				à vista ou índices	
	Pré	CC	CupIGPM	Deságio	Dólar	IGPM
LTN, NTNF	X					
LFT				X		
NTND,NTNA1		X			X	
NTNC			X			X

Tabela 2.2: Exemplo de mapeamento em fatores primitivos de risco

2.1.3 Precificação de ativos

No ano de 2002, o Banco Central do Brasil instituiu uma série de normas para controlar a marcação de ativos financeiros (2). Com isso, tornou-se obrigatória a marcação a mercado de ativos que compõem a carteira dos portfólios administrados por instituições financeiras que atuam como gestoras de fundos de investimento, fundos de pensão, empresas de seguro, etc. Esta normalização fez com que diversos modelos de precificação, para ativos mais comuns, começassem a ser padronizados por instituições como ANBIMA, B&MF Bovespa, etc. Cada uma responsável por classes específicas de ativos.

O conceito de marcação a mercado está relacionado ao provável preço de venda de um investimento realizado num ativo financeiro, isto é, consiste em estabelecer o valor presente (ou valor atual) de um ativo de tal forma que sua reposição permita ao adquirente os mesmos resultados de uma nova operação com características de fluxos de caixa e prazos remanescentes, iguais aos da operação original. Uma vez definidos e implementados, estes modelos podem ser utilizados tanto para a precificação de carteiras de ativos para indicar a valorização diária (ou em tempo real) da sua carteira, como para a construção de sistemas de risco onde, dadas perturbações em determinadas componentes que compõem o preço, conseguimos identificar a maior perda dessa carteira com um determinado grau de confiança.

Já o conceito de marcação na curva é mais utilizado quando não se tem o objetivo de negociar um determinado título até a sua data de vencimento. Em geral, marcação na curva é o valor da aquisição do título acrescido da atualização pelo indexador vinculado ao papel em questão e dos juros correspondentes ao período analisado. Ambos os cálculos devem ser realizados sobre o valor de emissão do título (também conhecido como valor de face).

O Apêndice B apresenta alguns dos modelos de precificação mais comuns que são cobertos pelos documentos de precificação das instituições e detalha uma classe de ativos, considerada exótica, que foi utilizada nos testes e que não é coberta por nenhuma documentação de instituição.

2.1.4 Cenário para um fator primitivo de risco

Um cenário para um fator primitivo de risco representa uma variação hipotética (choque) do valor do fator a ocorrer ao longo de determinado período, expressa como variação (Δ) relativa ou absoluta sobre um valor de referência do FPR (usualmente o seu valor de mercado).

O período associado à variação definida pelo cenário é denominado horizonte de tempo. Ao utilizar o cenário na avaliação de risco de uma posição, o horizonte de tempo do cenário representa o prazo necessário ao encerramento total da posição sob avaliação. Deste modo, em função das características dos contratos, podem ser definidos, para um FPR comum a contratos distintos, cenários para horizontes de tempo distintos.

Um cenário pode ser neutro, de alta ou de baixa, conforme Δ assuma valor nulo, positivo ou negativo, respectivamente. O cenário neutro é denominado cenário de referência, uma vez que, sob tal cenário, o fator não sofre variação.

Denote por FPR_{Ref} o valor de referência do fator e por FPR_{Cen} seu valor

sob cenário, ou seja, supondo a ocorrência da variação por ele definida sobre o valor de referência. Variações relativas são definidas para fatores expressos em forma de preço e são dadas pela equação $\Delta_{Cen} = \frac{FPR_{Cen} - FPR_{Ref}}{FPR_{Ref}}$. Já variações absolutas referem-se aos fatores do tipo taxa de juro, bem como para volatilidades (quando aplicável) e são definidas pela equação $\Delta_{Cen} = FPR_{Cen} - FPR_{Ref}$.

O conjunto de nc_j cenários para o j-ésimo fator primitivo de risco, FPR^j, é denotado por $C_{Cen}^{FPR^j} = \{Cen_1^j, Cen_2^j, \ldots, Cen_{nc_j}^j\}$, onde um cenário Cen_k^j para um fator FPR_j pode ser, do tipo estrutura temporal e, portanto, representar um grupo de cenários, com um cenário $Cen_k^{j,v}$ para cada vértice v do fator FPR_j. Na Tabela 2.3 estão dispostos, em cada linha, os cenários atribuídos a um mesmo fator primitivo de risco.

FPR	Cenários de Variação
FPR^1	$Cen_1^1\ Cen_2^1 \ldots Cen_{nc_1}^1$
FPR^2	$Cen_1^2\ Cen_2^2 \ldots Cen_{nc_2}^2$
$\vdots$	
FPR^j	$Cen_1^j\ Cen_2^j \ldots Cen_{nc_j}^j$
$\vdots$	
FPR^N	$Cen_1^N\ Cen_2^N \ldots Cen_{nc_N}^N$

Tabela 2.3: Cenários de Variação de FPRs

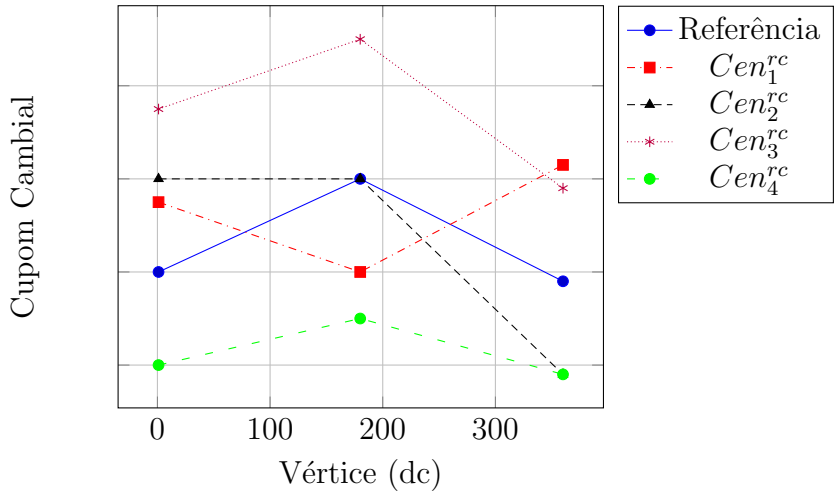
Como exemplo para este último caso, podemos dizer que o FPR^{rc} (fator primário de risco para o Cupom Cambial) é um fator do tipo estrutura temporal. Considere que ele seja definido nos vértices $v_1 = 1$, $v_2 = 180$ e $v_3 = 360$, onde os prazos são dados por dias corridos, e que sejam estabelecidos 4 cenários de variação sobre o valor de referência, $C_{Cen}^{rc} = \{C_1^{rc}, C_2^{rc}, C_3^{rc}, C_4^{rc}\}$. Cada cenário C_k^{rc} (onde $k = 4$) corresponde a um conjunto de 3 variações entre os fatores temporais, isto é, uma variação para cada vértice. A Tabela 2.4 apresenta um exemplo de cenários e a Figura 2.1 o efeito dessas variações sobre os valores de mercado de referência do fator.

Tabela 2.4: Exemplo de cenário para um FPR temporal

FPR^{rc}	Cen_1^{rc}	Cen_2^{rc}	Cen_3^{rc}	Cen_4^{rc}
$FPR^{rc,1}$	150	200	350	-200
$FPR^{rc,180}$	-200	0	300	-300
$FPR^{rc,360}$	250	-200	200	-200

Estes cenários de variação são utilizados para estimar o risco de mercado de uma carteira. Faz-se esta estimação por meio da avaliação da perda potencial

Figura 2.1: Cenários para o FPR^{rc}



da carteira, decorrente de variações nos valores dos $FPRs$, expressas através dos cenários definidos para eles.

Para fixar os conceitos vistos nesta seção, vamos considerar um exemplo de uma posição comprada em 1 unidade ($q = 1$) do contrato futuro de câmbio real por dólar, de tamanho igual a 50.000 dólares, cotado em $\frac{R\$}{U\$1000}$. A Tabela 2.5 apresenta os valores de referência dos três fatores de risco que devem ser aplicados ao contrato, bem como os cenários de variação definidos para eles e seus valores sob o cenário. A última linha da tabela nos dará qual será o valor final P_f a ser utilizado na precificação deste ativo para este cenário de referência.

Fatores de Risco	**Referência (P_i)**	**Cenário**	**Valor final (P_f)**
FPR^{DOL} ($\frac{R\$}{U\$}$)	2,300	-7%	2,139
FPR^{r}	19%	-100 pb	18,00
PU_r ($R\$$)	0,8403	0,86 %	0,8475
FPR^{rc}	9,48	0 pb	9,48
$PU_{rc}(U\$)$	0,9134	0%	0,9134
Preço DOLFUT	**2.500,00**	**-7,78 %**	**2.305,46**

Tabela 2.5: Exemplo de precificação com choque

Sob os cenários de referência dos FPRs, o valor futuro da taxa de câmbio vale 2.500 e a posição, 125.000. Já sob os cenários de estresse definidos para os FPRs, o contrato e a posição valem, respectivamente 2.305,46 e 115.273,11. Desta forma, temos que a carteira variou -7,78 % em relação ao valor de referência da posição o que significaria uma perda de R$ 9,726,89.

Esta seção buscou apresentar conceitos básicos sobre carteiras de investimentos, risco de mercado e precificação de ativos, necessários para o enten-

dimento do problema abordado por esta tese. Para maior aprofundamento do assunto, sugerimos as leituras de (17), (18), (16), (12), (2), (1, ANB05) e (6).

2.2 Escalonadores

Nesta seção introduzimos em 2.2.1 a notação de Graham et al (15), que é largamente utilizada em toda a literatura, estendendo-a de acordo com (28), (25), (20) e (30) para adaptá-la às restrições do problema abordado nesta tese. A notação consiste em três campos básicos $\alpha|\beta|\gamma$. O primeiro campo α indica o ambiente de processamento (ou configuração do ambiente de máquinas) e só pode receber um único valor, o segundo campo β trata das características e restrições do ambiente, podendo apresentar mais de um valor e o último campo γ apresenta a função objetivo do sistema analisado.

2.2.1 Definição de Jobs, Máquinas e Ambiente de Processamento

Inicialmente definiremos que a quantidade de jobs e máquinas [3] são finitas e definidas por n e m, respectivamente, e onde o índice de cada job será definido como j e o índice de cada máquina como i. O par (i, j) indicará um job j sendo processado por uma máquina i. Desta forma teremos os conjuntos:

$$J = \underbrace{\{J_1, J_2, \ldots, J_j, \ldots, J_n\}}_{\text{Conjunto de } n \text{ Jobs}}$$

$$M = \underbrace{\{M_1, M_2, \ldots, M_i, \ldots, M_m\}}_{\text{Conjunto de } m \text{ máquinas}}$$

Onde cada job pode possui uma série de atributos básicos como:

p_{ij}: Tempo de processamento de um job j numa máquina i, onde i pode ser omitido se o tempo de processamento for o mesmo para todas as máquinas envolvidas no processo ou se o job j só puder ser processado numa única máquina.

r_j: Ready/Release Date: Define quando o job j estará disponível para ser processado.

Já para as máquinas, temos que alguns atributos podem ser dependentes do job escolhido para ser processado naquele momento. Portanto, teremos:

v_{ij}: Velocidade da máquina i para o job j.

[3] O termo máquina aplica-se a toda entidade capaz de processar um job, seja ela composta por apenas um único processador, ou por múltiplos processadores. Nesta tese trataremos máquinas com múltiplos *cores* como várias máquinas *single core*

Dada a velocidade, definiremos que o tempo gasto por cada job numa máquina é dado pela expressão $p_{ij} = p_j/v_{ij}$.

O ambiente de processamento definido por α que pode assumir os seguintes cenários:

Uma única máquina (1): Caso mais simples onde apenas uma única máquina está disponível para o processamento.

Máquinas idênticas em paralelo (P_m): Existem m máquinas idênticas em paralelo disponíveis para processamento. Neste cenário, um job j pode ser processado com tempo p_j em qualquer uma das máquinas disponíveis no ambiente.

Máquinas em paralelo com velocidade diferentes (Q_m): Existem m máquinas com velocidades v_i diferentes em paralelo disponíveis para processamento. Neste cenário, um job j pode ser processado numa máquina i com tempo $p_{ij} = p_j/v_i$ (p_j representa o tempo de processamento normalizado, ou independente da máquina). Este ambiente também é conhecido como uniforme e é uma generalização do caso anterior i.e., $v_i = 1$ para todo i e $p_{ij} = p_j$.

Máquinas em paralelo com velocidade diferentes para cada tipo de job (R_m): Generalização dos ambientes apresentados anteriormente. Nele existem m máquinas com velocidades definidas por v_{ij}, definindo que uma máquina i processa um job j com velocidade v_{ij}. Desta forma, o tempo p_{ij} que o job j gasta na máquina i é igual a p_j/v_{ij}.

Diferentes tipos de problemas podem ser modelados com a notação $\alpha|\beta|\gamma$ e o método a ser selecionado para solucionar esses problemas pode depender da quantidade de informação que temos sobre eles. Isto quer dizer, por exemplo, que em alguns problemas, podemos não ter informações sobre a duração do período de indisponibilidade do sistema, ou até mesmo do seu início. Um bom exemplo deste caso é a quebra inesperada de uma máquina que pertence a um pool de processamento, onde podemos ou não ter informações sobre seu início ou sua duração. Para estes casos, Schmid (30) define que, com base no tipo de informação que temos sobre um problema, são possíveis três tipos de algoritmos:

- On-Line: Algoritmo procede sequencialmente e só possui no tempo t informações sobre quantidade de jobs disponíveis para processamento naquele instante e a quantidade de máquinas disponíveis para processamento, também naquele instante.
- Nearly On-Line: Algoritmo necessita saber no instante de decisão t o instante $t + 1$ do próximo evento de indisponibilidade.
- Off-Line: Possui toda informação a priori.

Além disso, o tipo de indisponibilidade apresentado no campo α pode seguir algum tipo padrão. Sejam $0 = t_1 < t_2 < ... < t_j < ... < t_q$ os pontos no tempo onde a disponibilidade de uma máquina muda e seja $m(t_j)$ o número de máquinas disponíveis durante o intervalo de tempo $[t_j, t_{j+1})$ com $m(t_j) > 0$, Schmid (30) define seis padrões diferentes:

1. Constante (0): Máquinas sempre disponíveis;
2. Zig-zag (NC_{zz}): Se existem apenas m ou $m-1$ máquinas disponíveis em cada intervalo;
3. Crescente (decrescente) (NC_{inc}, NC_{dec}): O número de máquinas do intervalo $[t_{j-1}, t_j)$ é menor (maior) que o número de máquinas no intervalo $[t_j, t_{j+1})$;
4. Crescente (decrescente) zig-zag (NC_{inczz}, NC_{deczz}): Se para todo j positivo, $m(t_j) >= \max_{1 \leq u \leq j-1} m(t_u) - 1$, $(m(t_j) >= \min_{1 \leq u \leq j-1} m(t_u) + 1)$;
5. Escada (NC_{sc}): Para todos os intervalos, a disponibilidade de uma máquina M_i implica na disponibilidade de uma máquina $P_{M_{i-1}}$;
6. Arbitrário (NC_{win}): Onde nenhum dos casos acima podem ser aplicados.

Ele ainda apresenta que um sistema com padrão arbitrário que permite preempção pode sempre ser reduzido a uma composição de máquinas que forma um padrão escada em $O(qm)$ onde q é a quantidade de posições no tempo que uma máquina muda a sua disponibilidade.

Existem ainda uma série de outros tipos de configurações apresentadas por Pinedo em (28) e Graham em (15) que estão fora do escopo desta tese como flow shop, job shop e open shop.

2.2.2 Restrições

O segundo campo β apresenta os diversos tipos de restrições que podem ser impostas a modelagem do problema. Destacamos aqui aqueles mais relevantes a tese, enquanto outras restrições podem ser encontradas em (28).

Definiremos por r_j a data de liberação (ou data de início) a qual o job não pode começar a ser processado antes desta data. Caso esta restrição não precise ser satisfeita o job poderá ser iniciado a qualquer momento.

Definiremos por preempção ($prmp$) a possibilidade do escalonador parar a execução de um job durante o seu processamento para colocar outro job no lugar. O job que foi interrompido poderá ser transferido e ter o restante do seu

processamento em outra máquina ou na mesma máquina onde estava sendo processado.

Jobs podem também apresentar restrições de precedência (*prec*). Isto significa que um job só pode ser iniciado após um outro job específico tenha sido finalizado ou uma sequência de jobs tenham o seu processamento finalizado. As restrições de precedência podem ser classificadas da seguinte forma:

- Cadeias: Jobs possuem pelo menos um predecessor e pelo menos um sucessor.
- Intree: Cada job possui no máximo um ou nenhum sucessor.
- Outtree: Cada job possui no máximo um ou nenhum predecessor.

Uma máquina pode também ser responsável por realizar um processamento em batch de b jobs simultaneamente ($batch(b)$), onde o tempo de processamento de cada um dos b jobs pode ser diferente. O processo batch só é finalizado após o último job ter sido finalizado Para estes processos, temos dois casos especiais que são do nosso interesse. Se a capacidade de processamento paralelo do processo batch for igual a 1 ($b = 1$), o problema é reduzido a um novo problema de escalonamento num ambiente com uma única máquina. O outro caso que é interessante ser apresentado é quando $b = \infty$, nesse caso não há limite para a quantidade de jobs que podem ser processados simultaneamente e implica no tempo total de processamento do batch ser definido pelo job com o maior tempo de execução.

A restrição de maior interesse nesta tese é dada por *brkdwn* que indica períodos predeterminados onde uma máquina i pode não estar disponível por conta de, por exemplo, manutenção ou troca de turnos entre os operadores. Ela foi intensamente estudada por Kacem (20), Lee (24) e Schmidt (30). Nela, cada período de indisponibilidade q será representados por $a_{iq} = [s_{iq}, f_{iq}]$ para uma máquina M_i para todo i, onde $0 \leq s_{iq} \leq f_{iq}$ (Ex: a_{21} representa a primeira indisponibilidade da máquina 2). Esta restrição geralmente é apresentada na literatura como fixa, conhecida a priori e durante qualquer momento o escalonador possui $m(t)$ máquinas idênticas em paralelo disponíveis para processamento. No nosso caso, esta restrição será estendida para representar períodos de indisponibilidade inesperada, o que faz o problema de escalonamento deixar de ser determinístico e passar a ser estocástico, por conta da falta de todas as informações disponíveis no ambiente para um algoritmo determinístico tomar uma decisão (24).

Um outro ponto importante desta restrição é a forma de interrupção e reinício do job interrompido. Quando um job é interrompido pela entrada em período de indisponibilidade, ele pode ter seu processamento reiniciado do

início (24) (*Nonresumable*) ou ponto onde foi interrompido (25) (*Resumable*). Lee (25) ainda define uma terceira forma de reinicio, chamada de *Semiresumable* que insere um tempo de setup para o reinicio do job a partir de um determinado ponto de parada. Definiremos como *nr* o tipo *Nonresumable*, *r* *Resumable* e *sr* *Semiresumable*.

2.2.3 Funções Objetivo

O último campo γ a ser apresentado define a função objetivo do escalonador, que será sempre minimizar a função do tempo (ou custo) para completar a execução de todos os n jobs, respeitando o escalonamento predeterminado. Esta definição segue a mesma linha daquela apresentada por Karger em (11), que diz que a teoria de escalonamento preocupa-se com a alocação ótima de recursos escassos para realizar atividades ao longo do tempo.

Para isso, adotaremos que o tempo necessário para completar um job j numa máquina i é dado por C_{ij}. O tempo que o job leva para completar todo o seu percurso dentro do sistema será dado por C_j. Na literatura existem várias funções objetivos que são largamente utilizadas como: o menor tempo de atraso; maior atraso observado; tempo de finalização com pesos e finalmente o Makespan, que será a função objetivo de maior interesse desta tese.

O makespan C_{max} é definido como $\max(C_1, ..., C_n)$, que é equivalente ao tempo de finalização do último job a sair do sistema. Outra forma de vermos o makespan é como o tempo total de processamento máquina. Geralmente, ao minimizarmos o makespan, temos que as máquinas estão sendo bem utilizadas pelo escalonador. Definiremos como C^* o makespan ótimo de um problema.

2.3 Formulação do Problema

Como apresentado na Seção 2.1, para que o cálculo de risco de uma carteira de investimentos seja realizado da maneira correta, deveremos reprecificar esta carteira aplicando uma série de cenários de estresses para escolher aquele cenário que nos forneça o pior resultado. No entanto, é de importância para o gestor de risco desta carteira, saber quais são os piores cenários, para que ele possa realizar uma análise macro econômica e verificar se este cenário faz sentido ou não.

Baseado na notação apresentada na Seção 2.2, definimos o problema a ser estudado como:

$$Q_m|prec, brkdwn, Nonresumable|C_{max}$$

A definição Q_m no campo α denota máquinas em paralelo com velocidades diferentes de processamento, mas sem diferença de velocidade por job i.e. $p_{ij} = p_j/v_i$.

Como o campo β não apresenta a restrição *prmp*, temos que preempções não são permitidas devido a natureza do problema de precificação. Isto acontece, pois uma vez iniciado um processo de precificação ele deve ser finalizado ou abortado (a introdução de continuidade pode ser inviável em alguns casos).

Prec indica que existem restrições de precedência que podem ser classificadas como do tipo *intree* por conta dos jobs do tipo I (precificação) que precedem os jobs do tipo II (cálculo de PnL).

Brkdwn indica a existência de períodos de indisponibilidade das máquinas, que a priori apresentam-se como arbitrários, mas que para questões de estudo podem fornecer algum tipo de look ahead para algoritmos nearly on-line.

Nonresumable determina que os jobs interrompidos por uma indisponibilidade devem ser reiniciados. Finalmente, o campo γ nos mostra que a função objetivo será minimizar o makespan.

Finalmente o campo γ define que a função objetivo de interesse é a minimização do makespan.

Assumiremos que as informações dos tempos dos jobs são conhecidas e, apesar de termos os períodos de indisponibilidade das máquinas, analisaremos algoritmos que não sabem quando será a próxima indisponibilidade e algoritmos que conhecem apenas estatísticas de tempo médio e variância da disponibilidade de cada máquina.

Como exemplo, considere inicialmente a precificação de uma carteira de investimentos para um conjunto de ativos $A = \{A_1, A_2, \ldots, A_z\}$ alocados nas estratégias $P = \{P_1, P_2, \ldots, P_y\}$, onde cada item A_i do conjunto A está associado a pelo menos uma estratégia do conjunto P, tal que o resultado desta carteira de investimentos seja dado pelo somatório dos resultados individuais de PnL representados por cada item do conjunto $PnL = \{P_1A_1, P_2A_1, P_3A_2, P_4A_3, \ldots, P_yA_z\}$ onde cada item P_iA_j do conjunto PnL é dado por $P_iA_j = Qtd_i(A_j) * Res(A_j)$ representado pela quantidade de um ativo j na estratégia i multiplicado pelo seu resultado ($Res(A_j)$) proveniente da precificação.

Isto quer dizer que existe uma dependência óbvia entre os conjuntos A e PnL, dado que o item P_xA_i não pode ser iniciado antes que A_i seja finalizado e esteja disponível para consumo do cálculo de resultado P_xA_i.

Nosso problema resume-se, então, a precificar esta mesma carteira

de investimentos n vezes para n cenários de stress e escolher aquele que nos fornece o pior resultado. Ou seja, dado que um cenário de stress w nos fornece um conjunto de preços de ativos $A_w = \{A_{w1}, A_{w2}, \ldots, A_{wz}\}$, temos para este mesmo cenário um conjunto de resultados $PnL_w = \{P_1A_{w1}, P_2A_{w1}, P_3A_{w2}, P_4A_{w3}, \ldots, P_yA_{wz}\}$ onde o resultado final é dado pelo somatório de todos os itens deste conjunto. Para encontrarmos o menor resultado de todos os cenários, basta percorrer a lista de resultados e escolher o menor deles $Risco = \min(PnL_1, PnL_2, \ldots, PnL_w, \ldots, PnL_n)$.

A Figura 2.2 mostra um exemplo do mapeamento de três jobs de precificação para duas estratégias em uma carteira de investimentos, onde o job de precificação A_2 é utilizado para computar o resultado tanto na estratégia P_1 como na estratégia P_2. O resultado do risco desta carteira será o somatório dos resultados indicados pelas arestas A_1P_1, A_2P_1, A_2P_2, A_3P_2.

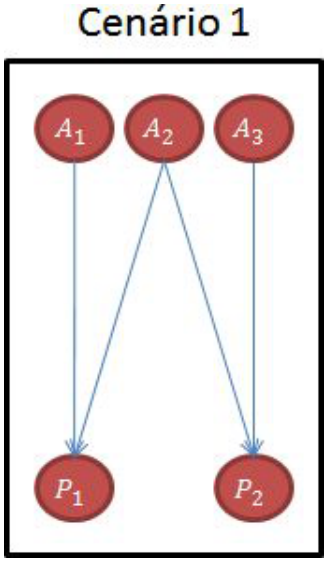


Figura 2.2: Mapeamento de jobs de precificação para cálculo de PnL para 1 cenário de stress

A Figura 2.3 mostra o mesmo exemplo da Figura 2.2, porém nela temos n cenários de stress. Neste caso, o resultado do risco desta carteira será o menor PnL (calculados como no exemplo anterior) dentre todos os n cenários de stress.

Como exemplo para um único cenário de stress, temos o seguinte conjunto de ativos A para a Tabela 2.1:

$$A = \{A_1, A_2, A_3\}$$

onde,

$$A_1 = PETR4$$

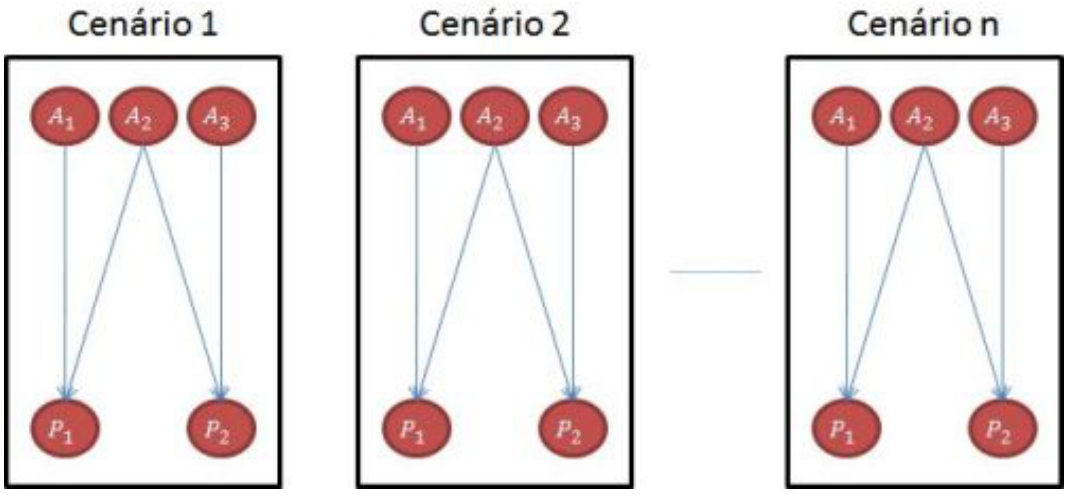


Figura 2.3: Mapeamento de jobs de precificação para cálculo de PnL para n cenário de stress

$$A_2 = VALE5$$

$$A_3 = VALEM28$$

Cada ativo está distribuído em compras e vendas em duas diferentes estratégias da carteira definidas por:

$$PnL = \{P_1A_1, P_1A_2, P_2A_2, P_2A_3\}$$

Onde,

$$P_1 = Estrategia_A$$

$$P_2 = Estrategia_B$$

Esta divisão em estratégias nos fornece o conjunto de resultados de PnL.

Para o cálculo final do risco e do PnL, estamos interessados inicialmente nos 3 cálculos de precificação para os ativos $A_1 = PETR4$, $A_2 = VALE5$, $A_3 = VALEM28$ que definiremos como jobs de precificação ou jobs do tipo I. Em seguida, teremos 4 cálculos de PnL para as posições em duas estratégias diferentes P_1 e P_2 que definiremos como jobs do tipo II. Para a estratégia A representada $P_1 = \{P_1A_1, P_1A_2\}$, temos as seguintes posições (quantidades de ativos na estratégia) $P_1 = Qtd_1(PETR4) = 200$ e $P_1 = Qtd_1(VALE5) = -150$, já na estratégia B representada por $P_2 = \{P_2A_2, P_2A_3\}$ temos as seguintes quantidades, $P_2 = Qtd_2(VALE5) = 300$, $P_2 = Qtd_2(VALEM28) = 200$.

Desta forma, supondo que os resultados da precificação sejam iguais a $Res(A_1) = 2,5$, $Res(A_2) = -1,25$, $Res(A_3) = 1$ o cálculo de PnL é dado pela

Equação 2-2.

$$\begin{aligned} PnL &= \{Est_A PETR4, Est_A VALE5, Est_B VALE5, Est_B VALEM28\} \\ &= \{200 * 2.5, -150 * -1.25, 300 * -1.25, 200 * 1\} \\ &= \{500.00, 187.50, -375.00, 200.00\} \\ Risco &= 512,50 \end{aligned} \tag{2-2}$$

Isto significa que para chegarmos ao resultado do risco para a carteira de investimentos para um único cenário de stress temos um total de:

- z Jobs de precificação (jobs do tipo I)
- k Jobs de PnL (jobs do tipo II), onde $k \geq z$
- 1 Job para calcular o resultado do risco fazendo o somatório dos k jobs de PnL.

Isto totaliza $n = z + k + 1$ jobs a serem processados que podem ser multiplicados pela quantidade de cenários de stress envolvidos na análise. O ambiente de processamento será dado por m máquinas que são consideradas como ativas se estiverem ligadas e se sua capacidade de processamento e utilização de memória pelo usuário logado não estiverem ultrapassando 50% do total disponível da máquina, caso contrário, será considerada como indisponível para processamento dos jobs.

O próximo capítulo, apresentará uma introdução básica de escalonamento em máquinas paralelas com restrições de disponibilidade através de uma revisão bibliográfica de estudos já realizados sobre o tópico.

3
Revisão Bibliográfica

Neste capítulo apresentamos na Seção 3.1 uma breve introdução sobre algoritmos e complexidade computacional e na Seção 3.2 uma discussão sobre a complexidade computacional de problemas relacionados com escalonamento. Finalmente, na Seção 3.3 apresentamos uma revisão bibliográfica das pesquisas e estudos de algoritmos nessa área, apresentando aqueles já existentes e os resultados conhecidos até hoje.

3.1
Algoritmos e complexidade computacional

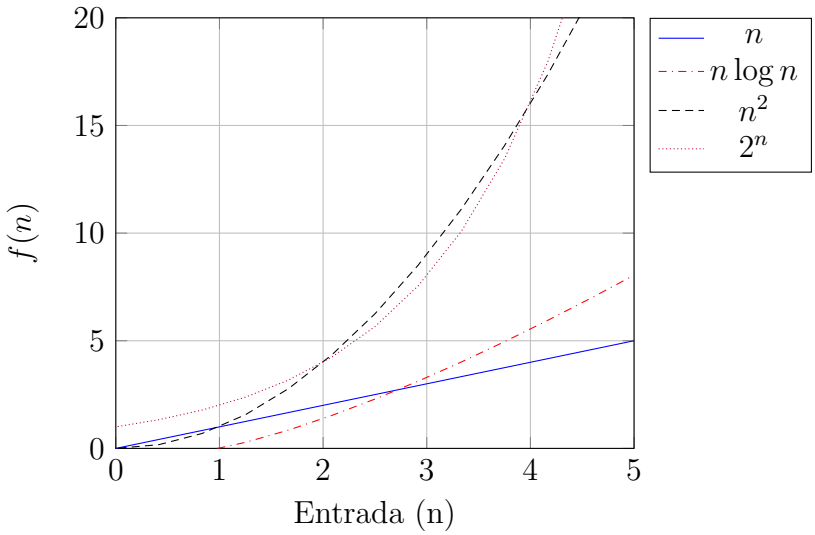
Cormen (10) define que um algoritmo é um procedimento computacional que recebe um valor, ou uma série de valores, como entrada e produz um valor, ou uma série de valores, na saída. Por procedimento computacional, entende-se uma sequência de passos computacionais que transforma uma entrada numa saída. Podemos entender também que uma entrada ou um conjunto de entradas para um problema é equivalente a uma instância ou a um conjunto de instâncias do problema. Finalmente, diremos que um algoritmo é correto se, para qualquer instância, ele produz uma saída correta que soluciona o problema.

Para o tamanho da entrada, a notação deve ser definida para cada tipo de problema específico. Em geral, a forma natural de representar o tamanho da entrada é a contagem de itens a serem processados, como por exemplo uma quantidade n de números num array que deve ser ordenado. Porém, existem problemas, como a multiplicação de dois números inteiros, onde é necessário analisar a quantidade de bits da representação binária de uma entrada (Ex: 101 (nº5), 1000011101 (nº541)). Outros tipos de representações podem ser encontradas em (10).

Em geral, estamos interessados na eficiência dos algoritmos. Diferentes algoritmos desenvolvidos para resolver um mesmo problema, podem ter eficiências completamente diferentes. A Figura 3.1 mostra a taxa de crescimento (para diferentes funções) em passos computacionais para uma entrada de tamanho n. Para definirmos essa função de eficiência do algoritmo (também conhecido como o seu tempo de execução), precisamos definir a medida destes

passos computacionais que será dada por comparações, operações algébricas, ou qualquer manipulação de dados, onde é comum assumir que estas operações acontecem em tempo constante (14).

Figura 3.1: Crescimento de funções



Uma prática comum em análises de algoritmos é a sua classificação em termos de taxa de crescimento da função, ou ordem da função. Esta análise nos fornece um limite superior $O(.)$, que Cormen (10) define pela Equação 3-1.

$$O(g(n)) = \{f(n) : \text{onde existe constantes positivas } c \text{ e } n_0 \text{ tal que } 0 \leq f(n) \leq c.g(n) \text{ para todo } n \geq n_0\} \quad (3\text{-}1)$$

Tardos (14) propõe que um algoritmo eficiente é aquele que consegue ser executado em tempo polinomial. Leung (26) define a classe dos problemas NP (*Non-deterministic Polynomial-time*) como a classe de problemas de decisão que possuem um certificado de tamanho limitado por uma função polinomial do tamanho da entrada e que pode ser verificado em tempo polinomial. Por problemas de decisão, entende-se um problema que levanta uma questão resolvida por uma resposta sim ou não.

No entanto, quando tratamos com problemas de otimização, que necessitam minimizar ou maximizar alguma função objetivo, temos que encontrar uma maneira de relacioná-los a problemas de decisão para incluí-los na análise da classe NP. Conforme Pinedo (28) todo problema de otimização pode ser associado a um problema de decisão e caso exista um algoritmo que resolva o problema de decisão em tempo polinomial, então existe um algoritmo que resolva o problema de otimização em tempo polinomial e vice versa. Da mesma forma, se não existe um algoritmo que resolva o problema de decisão em tempo

polinomial então não existe um algoritmo que resolva o problema de otimização em tempo polinomial.

Para reforçarmos a análise de complexidade, apresentaremos o conceito de redução de problemas. Pinedo (28) afirma que um problema L se reduz a um problema L' se para qualquer instância de L, uma instância equivalente de L' puder ser construída. Na teoria da complexidade computacional, um problema L se reduz polinomialmente a um problema L' se um algoritmo polinomial no tempo para L' implica num algoritmo polinomial no tempo para L ($L \propto L'$, ou $L \leq_p L'$).

Desta forma, podemos agora afirmar que um problema de decisão L é dito *NP-completo* se (1) L está na classe de problemas NP e (2) todos os problemas da classe NP podem ser reduzidos polinomialmente para o problema L (26). Um problema Q é dito *NP-hard* se satisfizer apenas a condição (2), isto é, todos os problemas da classe NP são reduzidos a Q. Resumidamente, Tardos (14) afirma que quando um problema é NP-completo significa dizer que trata-se de um problema computacionalmente difícil.

Porém, não é claro uma forma plausível de se estabelecer uma prova para a condição (2), dado que existe um número muito grande de problemas na classe NP. Cook (9) provou em 1971 que o problema conhecido como *SATISFABILITY* (*SAT*) é NP-Completo através de uma redução de uma máquina de Turing para o problema *SAT*. Desta forma, a partir deste problema, podemos agora mostrar que outros problemas são NP-completos reduzindo estes problemas ao *SAT*.

Embora não seja uma tarefa fácil encontrar uma solução ótima para um problema da classe NP, é interessante desenvolver algoritmos polinomiais no tempo que sejam capazes de entregar soluções, com algum tipo de garantia, próximas às ótimas. Isto gerou uma série de pesquisas numa área denominada algoritmos aproximativos. Cormen (10) define que um algoritmo possui um fator de aproximação $\rho(n)$ se para qualquer entrada de tamanho n, o custo C da solução produzida pelo algoritmo está dentro de um fator $\rho(n)$ do custo C^* da solução ótima como apresentado pela Equação 3-2.

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n) \tag{3-2}$$

Desta forma, se um algoritmo atinge o fator de aproximação $\rho(n)$, chamaremos este algoritmo de $\rho(n)$-aproximado. Esta definição pode ser aplicada tando para problemas de maximização onde $0 < C \leq C^*$ tal que C^*/C é o fator pelo qual o custo da solução ótima é maior que o custo da solução aproximada, como para problemas de minimização onde $0 < C^* \leq C$ tal que C/C^* é o fator pelo qual o custo da solução aproximada é maior que o

custo ótimo.

Temos também que um esquema de aproximação para um problema de otimização é um algoritmo aproximativo que toma como entrada não só a instância do problema, mas também um valor $\epsilon > 0$ tal que para qualquer ϵ fixo, o esquema resulta num algoritmo $(1 + \epsilon)$-aproximado.

3.2 Complexidade computacional em problemas de escalonamento

Em geral, o escalonamento requer que um determinado algoritmo decida qual será a sequência de jobs a ser executada e quais recursos deverão ser consumidos para que esta execução seja ótima em relação ao objetivo pré determinado. Baker (3) define que quando existe apenas um recurso, sua alocação é completamente determinada pelas decisões de sequenciamento dos jobs. Como consequência, num ambiente que possui apenas uma máquina, não existe distinção entre sequenciamento e alocação de recursos. Para avaliar melhor essa distinção, nós precisamos examinar modelos com mais de uma máquina.

O modelo mais simples que podemos analisar os efeitos do paralelismo é o problema de minimizar o makespan de n jobs descorrelacionados que estão disponíveis no tempo zero em m máquinas em paralelo que também estão disponíveis para processamento no tempo zero ou, resumidamente, $P_m|prmp|C_{max}$. Neste modelo assumimos que qualquer máquina pode processar qualquer um dos jobs. Pinedo (28) afirma que em geral, quando lidamos com problemas de máquinas em paralelo, o makespan se torna o principal objetivo, pois, na prática, trata-se do problema de balancear a carga em todas as máquinas disponíveis e a minimização do makespan nos garante um bom balanceamento desta carga.

Se o modelo permitir preempção, qualquer processamento de job pode ser interrompido e continuado em outra máquina, desta forma é possível mostrar que o makespan ótimo é dado por:

$$C^*_{max} = \max \left[\sum_{j=1}^{n} (p_j/m), \max_j (p_j) \right] \tag{3-3}$$

A Equação 3-3 define que ou os jobs serão alocados igualmente entre as máquinas, ou o job com maior tempo para ser finalizado definirá o makespan. Um algoritmo simples para a construção de um escalonamento ótimo para este caso seria selecionar qualquer job numa máquina i no tempo zero e em seguida colocar outros jobs que não foram sequenciados na mesma máquina até completar o tempo igual a C^*_{max}, ou até todos os jobs serem escalonados, repetindo

o segundo passo passando por todas as demais máquinas. Se proibirmos a preempção, o problema $P_m||C_{max}$, torna-se *NP-hard*.

Para analisarmos a complexidade de problemas de escalonamento, primeiramente precisamos definir como deve ser representado o tamanho da entrada. Pinedo (28) afirma que para estes problemas na prática, o tamanho da entrada de uma instância pode ser dada pela contagem do número de jobs, e que isto é suficiente para fazermos distinções entre a complexidade de diferentes variações de problemas. Além disso, claramente temos que estes são problemas de otimização, dado que sempre tentaremos minimizar ou maximizar alguma função objetivo.

No entanto, nem todos os problemas NP-hard são igualmente difíceis para uma grande parte de instâncias práticas. Existem, por exemplo, alguns problemas que podem ser resolvidos por programação dinâmica ou por *branch and bound*. No entanto, para os problemas de escalonamento Baker (3) afirma que mesmo estes métodos não conseguem obter muito sucesso para problemas pequenos, pois requerem muita memória e poder computacional.

Finalmente, Pinedo (28) apresenta uma hierarquia da complexidade de problemas de escalonamento segregando-as pelos campos $\alpha|\beta|\gamma$ conforme apresentado nas Figuras 3.2 para o campo α, 3.3 para o campo β e 3.4 para o campo γ. Esta hierarquia nos permite criar uma linha entre problemas similares de escalonamento para identificar se ao alterarmos algum parâmetro do problema, ele ficará mais difícil ou mais fácil. Isto também ajuda a definirmos que se um problema é *NP-hard* na hierarquia, e subimos sua a complexidade, o novo problema também será *NP-hard*. Isto significa que se $P_2||C_{max}$ é *NP-hard*, o problema $Q_m|prec, brkdwn|C_{max}$, abordado nesta tese, também será.

Como vimos, embora não seja uma tarefa fácil encontrar o makespan ótimo, é interessante desenvolver algoritmos aproximativos, polinomiais no tempo, que sejam capazes de entregar soluções próximas às ótimas com algum

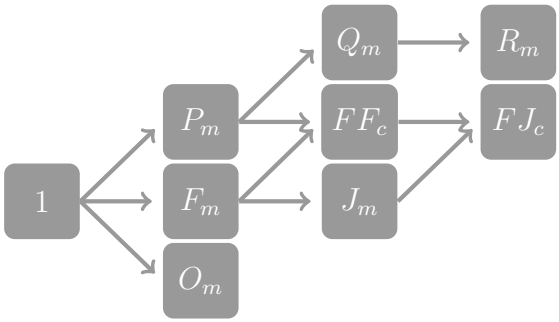


Figura 3.2: Hierarquia de complexidade para ambiente de máquinas (cresce da esq. para dir.)

tipo de garantia,.

Destes algoritmos, um dos mais conhecidos para o problema base $P_m||C_{max}$, é chamado de *List Scheduling* (LS) e tem como objetivo ordenar os jobs de uma determinada maneira e enviá-los para as máquinas disponíveis no momento. A solução para o makespan depende de como os jobs serão ordenados. Baker (3) prova que ao aplicar uma política de LS para um problema de máquinas em paralelo temos uma garantia de desempenho ao estabelecer um limite superior dado pela Equação 3-4.

$$\frac{C_{max}(LS)}{C_{max}(OPT)} \leq 2 - 1/m \tag{3-4}$$

Além disso, quando temos uma quantidade de jobs, muito grande no sistema ($n \rightarrow$ inf) tal que $p_{max}/\sum p_j \rightarrow 0$ (ie. nenhum job domina o tempo de processamento), o LS é assintoticamente ótimo para o makespan (3). Isto nos dá um resultado importante para uma cadeia muito grande de jobs. No entanto, para cadeias pequenas ou médias, outros métodos heurísticos foram desenvolvidos que fornecem resultados melhores, como o método *Longest Processing Time* (LPT) que ordena os jobs em ordem decrescente de tempo de processamento. Este método nos dá uma garantia de desempenho (para $P_m||C_{max}$) dada pela Equação 3-5. A prova desta equação pode ser encontrada em (28).

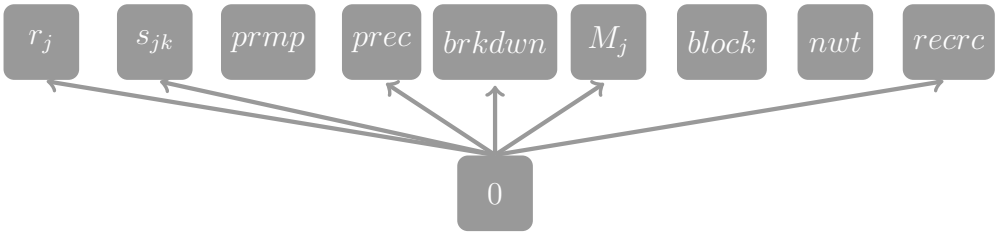


Figura 3.3: Hierarquia de complexidade para restrições (cresce de baixo para cima)

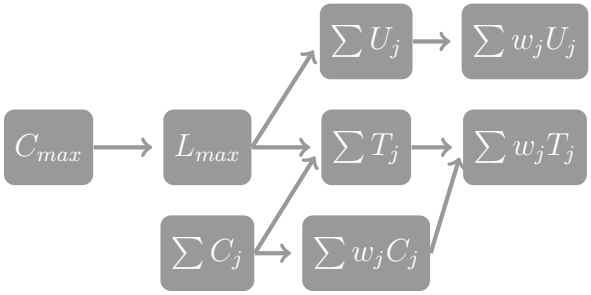


Figura 3.4: Hierarquia de complexidade para funções objetivo (cresce da esq. para dir.)

$$\frac{C_{max}(LPT)}{C_{max}(OPT)} \leq 4/3 - 1/3m \tag{3-5}$$

Muitos outros métodos heurísticos foram propostos na literatura para endereçar o problema clássico $P_m||C_{max}$ como em (19). Isso nos indica que métodos heurísticos produzem resultados satisfatórios dado uma garantia de desempenho, o que nos motiva ainda mais a explorar outros métodos para nosso problema. A Seção 3.3 apresentará uma breve revisão dos estudos realizados até hoje para problemas similares.

3.3 Algoritmos de escalonamento para máquinas com indisponibilidades

Esta seção tem como objetivo revisar modelos que lidam com indisponibilidades num ambiente de processamento. Existem diversos artigos na literatura que tratam do problema de escalonamento num ambiente com máquinas em paralelo, embora poucos são aqueles que lidam com indisponibilidades. Iniciaremos revisando artigos que lidam com indisponibilidades em um ambiente com uma única máquina. Em seguida, apresentaremos resultados encontrados para máquinas em paralelo que não estão disponíveis no tempo zero e finalmente a última seção revisa artigos que lidam com múltiplas indisponibilidades durante o período de execução.

Lee (24) mostrou que o problema $1|nr|C_{max}$ é *NP-Completo* quando existe um ou mais períodos de indisponibilidade mostrando que é um caso especial do problema $3 - PARTITION$. Wu e Lee (32) estudaram uma variação deste problema, onde incluíram uma função linear de deterioração no tempo de execução dos jobs. Eles resolveram este problema utilizando uma técnica de programação inteira e em seguida mostraram que o problema poderia ser resolvido de maneira ótima utilizando o algoritmo SPT (*Shortest Processing Time* onde jobs são enviados para processamento de acordo com a ordem crescente de tempo de processamento).

Para $1|batch|C_{max}$, Wang and Cheng (31) propuseram um método heurístico com um fator de aproximação de 1/2. Chen (7) considerou o problema de manutenção periódica numa única máquina para uma companhia têxtil. Neste caso os períodos de indisponibilidades são conhecidos e os valores dos tempos $\in \mathbb{N}$. Ele desenvolveu uma heurística quase ótima e um algoritmo de branch and bound ótimo para minimizar o atraso máximo. Os resultados apresentados e a proposta heurística eram altamente acuradas e eficientes.

Lee (23) apresentou dois algoritmos para o caso $P_m|nr|C_{max}$ onde uma das máquinas está sempre disponível e as demais possuem no máximo um

período de indisponibilidade. Estes períodos poderiam ser diferentes, mas todos tinham que começar no tempo zero.

No primeiro algoritmo ele aplicou o LPT clássico que roda em $O(n.\log n)$, em seguida provou que o makespan deste algoritmo é sempre $\leq 3/2 - 1/2m$ do ótimo. Anos depois ele analisou o caso onde o período de indisponibilidade é maior que o makespan e viu que o limite reportado não era mais válido. Ele então limitou o problema e garantiu aquele limite quando o número de maquinas ativas ($m' < m$) passa a ser $\leq 3/2 - 1/2m'$ do ótimo.

No segundo algoritmo ele considerou que os períodos de indisponibilidade eram jobs com prioridade máxima e ordenou os jobs de em ordem decrescente de tempo de processamento (MLPT) considerando que os "jobs"de indisponibilidade deveriam ser escalonados antes dos demais e que cada máquina só poderia receber no máximo um destes jobs especiais. O algoritmo roda em $O((n+m)\log(n+m) + m(n+m))$ e ele provou que o makespan deste algoritmo é $C_{MLPT}/C^* \leq 4/3$.

Kellerer (21) apresentou um algoritmo aproximativo para o problema clássico $P_m|.|C_{max}$ (onde as máquinas possuem um tempo inicial de indisponibilidade) utilizando uma abordagem semelhante ao *bin packing* que resultou num fator de aproximação de 5/4.

Schmidt (30) apresenta que alguns algoritmos podem ser ótimos dependendo do padrão de disponibilidade apresentado. Coffman e Graham (8) apresentaram um algoritmo ótimo para duas máquinas num padrão NC e jobs com restrição de precedência. O algoritmo ordena os jobs pela maior quantidade de sucessores (MSF). Dolev e Warmuth (13) demonstraram um algoritmo que ordena os jobs pelo maior nível de um grafo (HLF) do tipo (*intree* ou *outtree*) operando num padrão de disponibilidade de máquinas do tipo NC_{inczz} ou NC_{deczz} nos resulta em $O(n\log m)$

Listamos aqui alguns os algoritmos mais relevantes na literatura até então para os casos onde não temos preempção e que o job não pode continuar o seu processamento após uma indisponibilidade (ie. deve reiniciar o seu processamento). Outros algoritmos e resultados para casos onde o job pode continuar o seu processamento podem ser encontrados em (28).

4
Algoritmos de Escalonamento

Neste capítulo apresentamos os algoritmos propostos que serão utilizados nas simulações do problema definido na Seção 2.3. A Seção 4.1 introduz os modelos de *List Scheduling* selecionados da literatura que se adequam ao problema $Q_m|prec, brkdwn|C_{max}$ e em seguida, na Seção 4.2, apresentamos o novo modelo heurístico proposto nesta tese.

Para todas as seções seguintes, assumimos que as disponibilidades das máquinas são representadas por uma estrutura de Heap, onde a primeira máquina disponível encontra-se na raiz. Além disso, todos os eventos de observação de disponibilidade nas máquinas são separados por um intervalo mínimo de observação (*sample time* ou *pooling time*) definido para esta tese como um segundo.

Desta forma, para nosso problema, utilizaremos para função de cálculo de mínimo do heap, o tempo de início de disponibilidade s_{iq} de uma Máquina i e uma caso os tempos $s_i = s_{(i+1)}$ sejam iguais, utilizaremos a máquina com maior v_i.

Além disso, todos os modelos assumem que caso um job entre num período de indisponibilidade da máquina durante sua execução, seu processamento será abortado e ele voltará para a lista de jobs a serem processados. Isto significa que o job abortado só poderá ser processado novamente após este início do intervalo de indisponibilidade. Definiremos este como tempo de liberação do job ou *release time*.

A função $VerificaHeapDeProcessamento$ utilizada nos algoritmos deste capítulo representa o processamento descrito acima, retornando os jobs incompletos para a lista de processamento principal caso o seu *release time* já tenha sido ultrapassado.

É importante ressaltar que a apresentação dos algoritmos trata a informação de indisponibilidade como uma variável aleatória que influencia na probabilidade do job ser finalizado ou não.

4.1
Algoritmos da Literatura

4.1.1 Service In Random Order (SIRO)

Este algoritmo (1), tem como sua principal característica a rapidez de execução e a simplicidade de implementação. O algoritmo recebe uma fila de jobs (FIFO) sem nenhuma ordenação e envia para processamento em qualquer máquina disponível no ambiente.

Algorithm 1 SIRO

```
Job ← FilaDeJobs.Dequeue()
while Job ≠ null do
    Maquina ← PegaQualquerMaquinaDisponivel()
    Resultado ← RodaJob(Maquina, Job)
    if Resultado.Finalizado = false then FilaDeJobs.Enqueue(Job)
    end if
    Job ← FilaDeJobs.Dequeue()
end while
```

4.1.2 Longest Processing Time (LPT)

Este algoritmo (2), também tem como principal característica a rapidez de execução e a simplicidade de implementação. O algoritmo recebe um Heap-max (em função do tempo de processamento) de jobs e envia para qualquer máquina disponível no ambiente processar.

Algorithm 2 LPT

```
Job ← HeapDeJobs.GetMax()
while Job ≠ null do
    Maquina ← PegaQualquerMaquinaDisponivel()
    Resultado ← RodaJob(Maquina, Job)
    if Resultado.Finalizado = false then HeapDeJobs.Insere(Job)
    end if
    Job ← HeapDeJobs.GetMax()
end while
```

Para exemplificar seu funcionamento, considere um conjunto de 4 Jobs $J = \{2, 2, 1, 1\}$, onde 2 jobs custam 1 minuto e 2 jobs custam 2 minutos e um ambiente com 4 máquinas idênticas com o diagrama de disponibilidade apresentado pela Figura 4.1.

LPT inicia o processo ordenando os jobs pelo tempo de processamento, colocando os jobs que demoram mais para serem processados assim que possível. Desta forma temos que os jobs 1, 2, 3 e 4 são alocados nas máquina M_1, M_2, M_3 e M_4 respectivamente (Figura 4.2).

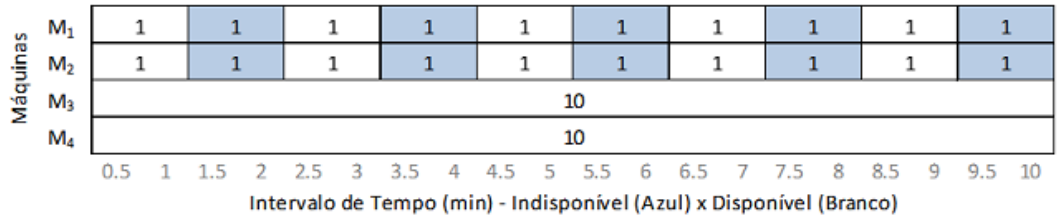


Figura 4.1: Diagrama de disponibilidades para 4 máquinas

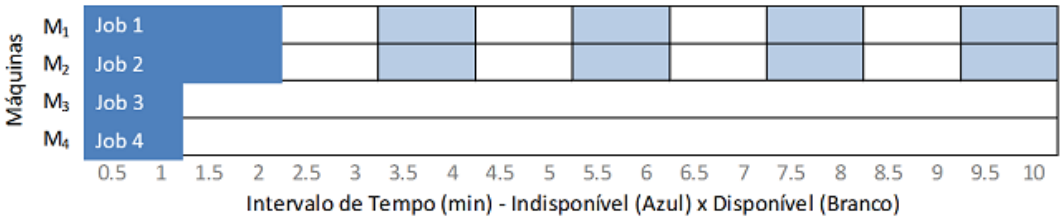


Figura 4.2: Alocação inicial dos jobs no instante zero

Como os jobs 1 e 2 ultrapassam os períodos de disponibilidade das máquinas M_1 e M_2, eles serão abortados no segundo 1 como mostra a Figura 4.3.

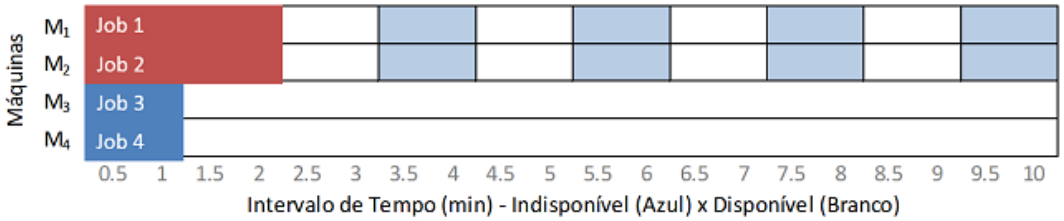


Figura 4.3: Jobs 1 e 2 encontram indisponibilidades no instante 1

Em seguida eles serão reinseridos na lista de processamento e serão alocados nas próximas máquinas com disponibilidade como ocorre na Figura 4.4.

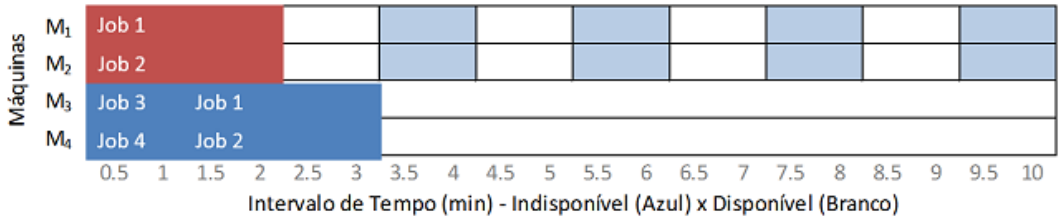


Figura 4.4: Realocação dos jobs interrompidos

4.1.3 Shortest Processing Time (SPT)

Este algoritmo é idêntico àquele apresentado na Seção 4.1.2, porém a estrutura de Heap ordena os jobs pelo menor tempo estimado para o processamento.

4.2 Algoritmo Proposto

Nesta seção introduzimos a nova heurística que será testada nesta tese. A ideia por trás dela é combinar métodos de *list scheduling* com informações estocásticas relativas às disponibilidades históricas das máquinas.

4.2.1 Longest Probable Processing Time (BLPPT)

Este algoritmo, tem como objetivo utilizar um intervalo histórico T de observação sobre disponibilidade de cada uma das máquinas para estimar o maior tempo de execução de um job para um determinado nível de confiança α.

Nossa intenção é evitar o envio de jobs custosos, neste caso jobs mais longos, para máquinas com um histórico recente de intermitência. Isto quer dizer, que queremos evitar este envio para máquinas que estejam alterando o seu status de disponível e indisponível muito frequentemente, pois isso resultará no reprocessamento destes jobs. Dado que estamos lidando com estações de trabalho de uma empresa, espera-se que as máquinas sejam utilizadas mais intensamente durante partes do dia de trabalho, que por sua vez pode penalizar as heurísticas clássicas que não tem nenhuma informação sobre o ambiente e por isso acreditamos que este algoritmo nos ajudará a obter um makespan melhor.

O algoritmo (3) mantém os jobs numa árvore de busca cuja chave é o tempo estimado de processamento. Primeiramente ele atualiza as probabilidades condicionais de execução de jobs de cada máquina e seleciona dentre as máquinas disponíveis àquela que tiver a menor probabilidade condicional de ficar indisponível dado que está disponível $P(I|D)$ para uma determinada janela de observação T. Em seguida, o algoritmo utiliza a informação de probabilidade condicional da máquina continuar disponível dado que ela está disponível $P(D|D)$ para calcular o maior tempo que esta máquina provavelmente consegue executar um job. Finalmente buscamos na árvore o job com maior tempo estimado de execução tal que a probabilidade de ser executado seja maior que o nível de confiança α pré definido.

7	5	4	8	3	4	2	5

Tabela 4.1: Intervalos: Indisponível (marcado) x Disponível (branco)

Para exemplificar o funcionamento do algoritmo, vamos considerar o cenário de disponibilidade da Tabela 4.1 de uma máquina qualquer, considerando uma janela de observação de $T = 38$. Os valores indicam a quantidade de eventos de disponibilidade e indisponibilidade observados e por simplicidade consideraremos que estes eventos são equivalentes ao sample time (t_{st}) que foi definido como um segundo. Desta forma, para atualizarmos as probabilidades desta máquina teremos as Equações 4-1 e 4-2.

$$P(I|D) = \frac{3}{5+8+4+4} = 14.29\% \tag{4-1}$$

$$P(D|D) = \frac{4+7+3+4}{4+7+3+4+3} = 85.71\% \tag{4-2}$$

A Equação 4-1 indica a probabilidade da máquina ficar indisponível dado que ela está disponível. Para chegarmos nesta probabilidade, contamos os eventos de mudanças de estado disponível para indisponível que aconteceram na janela T em questão e dividimos este valor pela quantidade total de eventos analisados. Para o exemplo acima, temos 3 eventos de mudanças de estado disponível para indisponível (numerador) e 18 eventos de disponibilidade (denominador). As demais equações apresentadas seguem o mesmo raciocínio.

Uma vez atualizadas as probabilidades condicionais das máquinas, escolhemos a máquina com menor probabilidade $P(I|D)$ dentre as máquinas disponíveis naquele instante. Como já calculamos a probabilidade da máquina continuar disponível dado que ela está disponível, podemos calcular para o job com maior tempo na fila, a sua probabilidade $P(S)$ de ser executado com sucesso através da Equação 4-3, onde t é a o tempo de execução do job naquela máquina em relação ao sample time t_{st} (no caso em segundos). Se esta probabilidade ultrapassar o nível de aceitação α definido para o algoritmo, admitimos que esse job pode ser executado com sucesso e alocaremos tempo desta máquina para sua execução.

$$P(S) = P(D|D)^{t-1} \tag{4-3}$$

Porém, para a implementação do algoritmo, dado que o nível de aceitação α é configurado no início da análise e que as probabilidades condicionais

$P(D|D)$ são atualizadas a cada iteração, podemos estimar o tempo t como o maior tempo que essa máquina pode processar um job através da Equação 4-4. Uma vez encontrado este tempo, o algoritmo fará uma busca na árvore para encontrar o job com o maior tempo estimado de processamento p_{ij} tal que $p_{ij} \leq t$.

$$t = \frac{t_{st}}{v_j} * \left[1 + \left(\frac{\log \alpha}{\log P(D|D)}\right)\right], \tag{4-4}$$

onde v_j é a velocidade da máquina selecionada. Para a implementação do algoritmo, calibraremos os dois parâmetros de configuração T e α para um conjunto pequeno de instâncias. Os parâmetros escolhidos serão aqueles que obtiverem o melhor resultado médio de makespan e serão aplicados para os demais testes em diferentes instâncias.

Algorithm 3 BLPPT

```
Job ← Jobs.TakeOne()
while Job ≠ null do
    AdicionaNo(JobTree, Job)()
    Job ← Jobs.TakeOne()
end while
while JobTree.Count > 0 do
    VerificaHeapReprocessamento(HeapDeJobs, HeapDeReproc)
    ListaMaquinas ← AtualizaProbabilidades(IntervaloDaJanela)
    for each Machine in ListaDeMaquinas do
        Job = FindBestJobForMachine(Machine, JobTree, NivelDeTolerancia)
        Resultado ← RodaJob(Maquina, Job)
        if Resultado.Finalizado = false then
            HeapDeReproc.Insere(Job)
        end if
    end for
end while
```

Como segundo exemplo, considere o mesmo conjunto de 4 Jobs $J = \{2, 2, 1, 1\}$, apresentado na Seção 4.1.2. Assumiremos um sample time de $t_{st} = 1s$, um nível de confiança $\alpha = 60\%$ e uma janela de observação de $T = 4min$.

BLPPT inicia o processo calculando as probabilidades condicionais de cada uma das máquinas. Desta forma temos:

$$P_1(D|D) = P_2(D|D) = \frac{59 + 59}{60 + 59} = 99.16\%$$

$$P_3(D|D) = P_4(D|D) = \frac{119}{119} = 100\%$$

Em seguida, o algoritmo procura na árvore de busca o primeiro job que tenha uma probabilidade de execução na máquina maior que a o nível de confiança definido para o algoritmo (no caso deste exemplo, 60%).

A Figura 4.5 ilustra o caminho percorrido na árvore. Como o job 1 possui uma probabilidade de execução menor que 60 %, o algoritmo verifica o próximo nó com valor mais baixo (no caso o job 3).

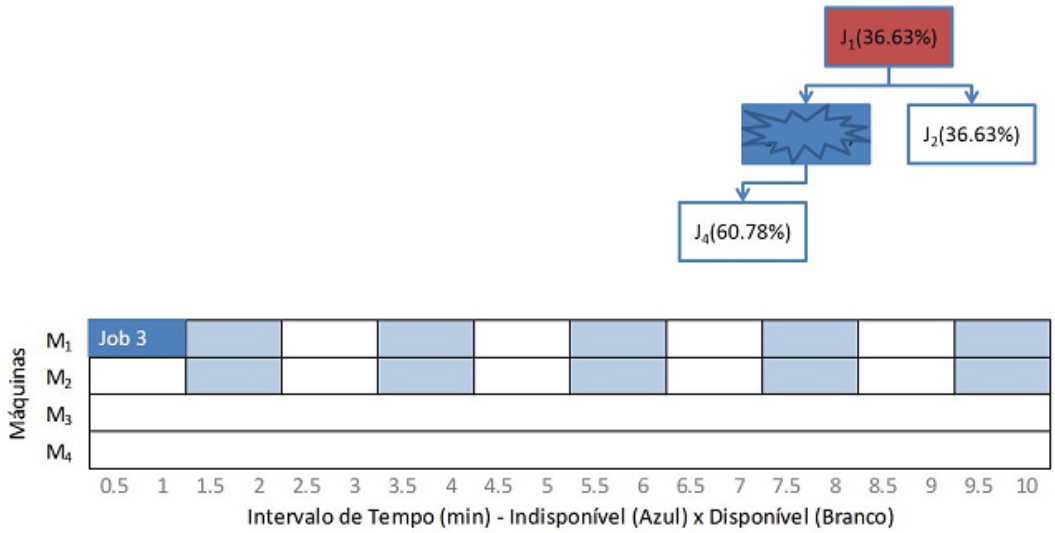


Figura 4.5: Alocação do 1º job

Como o job 3 possui uma probabilidade de execução maior que 60 %, ele será removido da árvore e será processado na máquina M_1, como mostra a Figura 4.6. O mesmo raciocínio se repete nas Figuras 4.6, 4.7, 4.8, até que todos os jobs tenham sido processados.

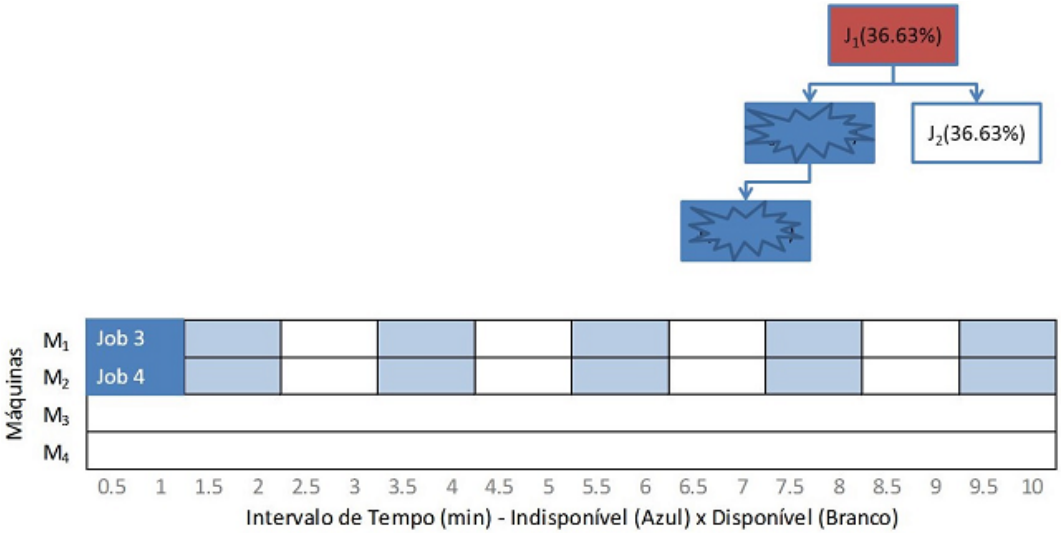


Figura 4.6: Alocação do 2º job

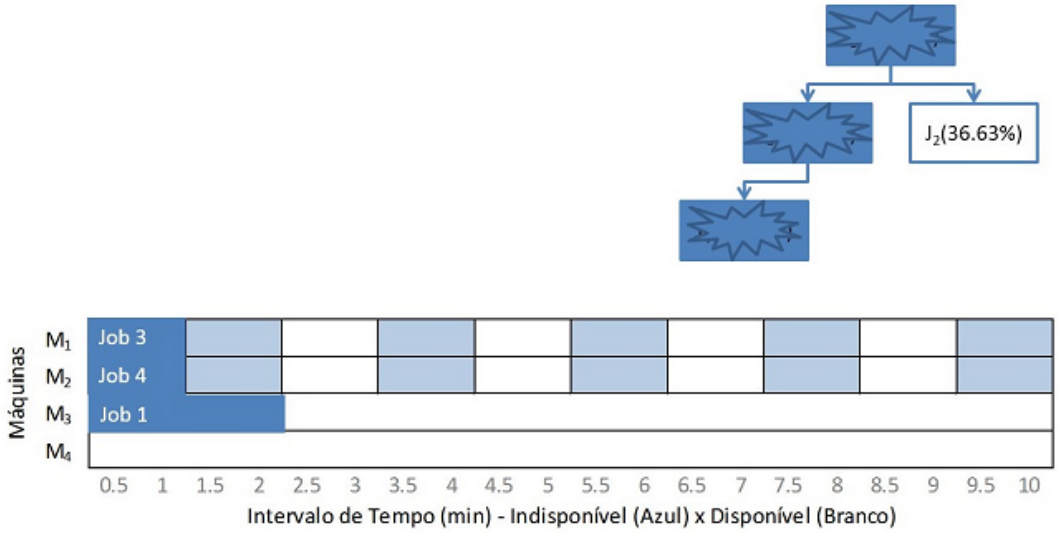


Figura 4.7: Alocação do 3º job

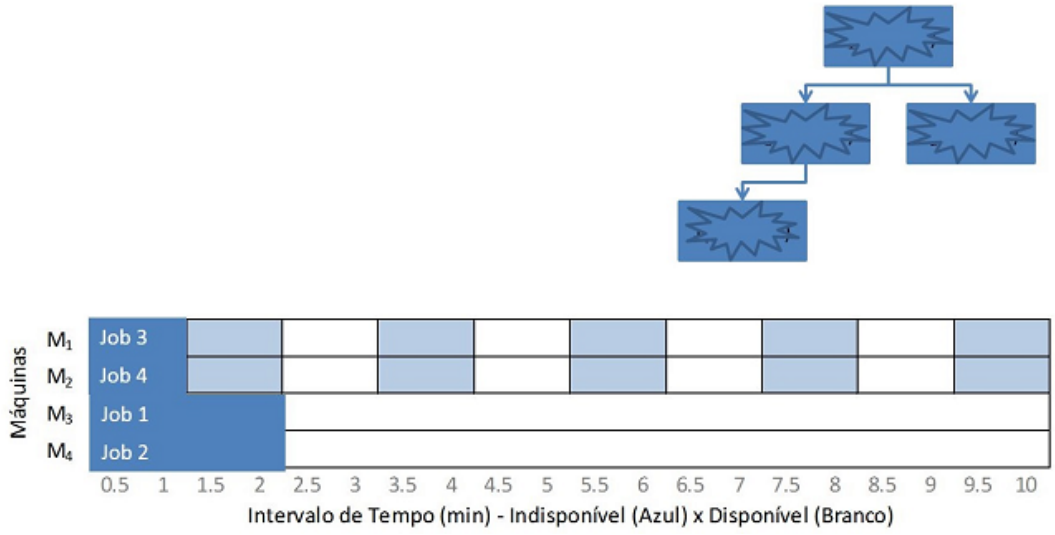


Figura 4.8: Alocação do 4º job

5
Ambiente de Simulação

Para testar os algoritmos apresentados no Capítulo 4, necessitamos de um ambiente de simulação capaz de comparar o makespan gerado por cada um dos algoritmos. Porém, antes de entramos nos detalhes do ambiente, precisaremos definir quais serão as instâncias de entrada para estes algoritmos de forma a estabelecer uma comparação justa em diferentes cenários de execução.

Como um dos nossos objetivos é utilizar um ambiente real, iniciaremos a construção das instâncias através da extração de dados reais relacionados a quantidade de Jobs, máquinas, tipos de Jobs, e intervalos de disponibilidade, para gerar instâncias que possam ser utilizadas no ambiente de simulação.

Apresentamos na Seção 5.1 o método de criação das instâncias e em seguida na Seção 5.2 o método de simulação dos algoritmos implementados.

Toda a codificação foi realizada em C# utilizando o framework .NET 4.5 com o ambiente de desenvolvimento Microsoft Visual Studio 2012 Ultimate. Os componentes gráficos pertencem a DevExpress e foram utilizados com a licença e autorização da Polo Capital. A interface gráfica foi desenvolvida parte em WinForms e parte em WPF (*Windows Presentation Foundation*)

5.1
Criação das Instâncias

Nesta seção discutiremos o processo de criação das instâncias utilizadas no problema. Para isto, devemos lembrar que o objetivo final do processo é gerar a informação de risco o mais rápido possível. Isto será realizado com sucesso se o Makespan for minimizado.

Para o processo ser concluído, o sistema de risco deve precificar todos os ativos pertencentes a sua carteira para todos cenários do conjunto $Cen = \{Cen_1, Cen_2, ..., Cen_z\}$. Uma vez feito isso, escolhe-se o cenário com pior resultado para a indicação da exposição ao risco.

Relembrando as definições da Seção 2.3, para o cenário onde há apenas uma máquina o tempo total de execução será dado pelo somatório dos tempos de processamento dos Jobs de tipo I e II, mais o tempo necessário para somar

todos os p_j. Isto deverá ser realizado z vezes para os contemplar todos os cenários de risco.

Nas seções seguintes, analisaremos o processo de obtenção das instâncias de disponibilidade das máquinas e em seguida analisaremos a distribuição dos tempos de processamento dos Jobs.

5.1.1 Disponibilidade de Máquinas

Para o processo de obtenção dos dados de disponibilidades das máquinas, desenvolvemos um serviço de monitoramento de desempenho e instalamos em todos os terminais participantes do experimento.

Ao ser iniciado, o serviço pega dados estáticos da máquina como: nome da máquina, nome do processador, quantidade de cores, velocidade do processador, versão do sistema operacional, total de memória e total de espaço em disco disponível. Estes dados são armazenados num banco de dados distribuído de alto desempenho chamado MongoDb (29).

Uma vez que os dados estáticos são carregados, o serviço monitora a cada segundo os dados referentes a utilização da máquina, salvando cada uma das observações no banco de dados. Estes dados são:

- Dt: Data e tempo que a informação foi registrada
- CPU: Percentual de tempo que o processador está efetivamente gastando trabalhando em threads produtivas e quão frequente está ocupado servindo a requisições;
- CPUds: Percentual de tempo que o processador está gastando com o processo de precificação;
- CPUUsr: Percentual de tempo que o processador está gastando com processos e threads de aplicações iniciadas pelo usuário na sua estação de trabalho. Esta é uma medida importante de utilização dos recursos da máquina;
- CPUQueue: Quantidade de tarefas na fila de processamento do processador. O resultado desta medida nos dá uma boa noção da utilização dos recursos da máquina, pois se apresentar um valor maior que durante um período longo de tempo, estamos com problemas de recursos na máquina;
- MemMB: Quantidade de memória disponível em MB;
- MemP: Percentual de memória disponível na máquina;
- IsEnding: Assume o valor "verdadeiro"se o sistema operacional registrou um pedido de desligar a máquina.

Processador	Clock (GHz)	Mem (GB)	Qtd
Intel(R) Core(TM)2 Quad Q840	2.66	8	3
Intel(R) Core(TM)2 Quad Q9650	3.00	8	1
Intel(R) Core(TM) i7-2600	3.40	8	5
Intel(R) Core(TM) i7-3770	3.40	8	8
Intel(R) Core(TM) i7-3770	3.40	16	6
Intel(R) Xeon(R) E5530	2.40	64	1
Intel(R) Xeon(R) E5620	2.40	64	1

Tabela 5.1: Ambiente de máquinas

Com estes dados, atribuiremos a uma máquina o status de "Indisponível"se para um determinado segundo a máquina apresentar uma observação de desempenho no banco de dados tal que:

$$CPUQueue > 10 \vee CPUUsr > 20 \vee MemP < 40$$

Isto é, se o processador estiver com uma fila de tarefas a serem processadas maior que 10, ou se o usuário estiver trabalhando na máquina e se suas aplicações exigirem mais que 20% do tempo do processador, ou se a utilização total de memória estiver acima dos 60%, consideraremos que a máquina não está disponível para o escalonador enviar jobs para processamento.

O monitoramento foi realizado em 25 máquinas diferentes. A Tabela 5.1 apresenta as variações de configuração do ambiente de máquinas.

A carga de dados foi obtida durante 1 mês de trabalho normal dos usuários nas suas estações de trabalho. Com posse destes dados, foi introduzida uma funcionalidade para gerar as instâncias de ambiente de máquinas. Estas instâncias serão agrupadas por dia e armazenadas num arquivo XML.

O XML contém uma entrada para cada intervalo de disponibilidade ou indisponibilidade observada. Estes intervalos são construídos olhando a série temporal de dados de disponibilidade armazenada no banco de dados, levando em consideração que caso não haja alguma entrada para algum segundo do dia em questão consideraremos, por falta de comunicação, a máquina como indisponível neste instante.

Para exemplificarmos o processo de criação destas instâncias, a Tabela 5.2 apresenta um conjunto de 10 segundos de observações para uma máquina. Para cada segundo só teremos um único status observado (Disponível, Indisponível, ou sem observação armazenada). O sistema de geração de instâncias determinará 3 linhas no XML, um primeiro intervalo marcado como disponível (0 a 2 segundos), um segundo intervalo como indisponível (2 a 6 segundos) e um último intervalo disponível (6 a 10 segundos).

Status—Segundos	1	2	3	4	5	6	7	8	9	10
Disponível										
Indisponível										
Sem observação										

Tabela 5.2: Intervalos de disponibilidade de máquinas

O processo de escolha das instâncias utilizadas no experimento será discutido no Capitulo 6.

A Figura 5.1 apresenta o sistema de geração de instâncias descrito acima. O gráfico gerado facilita a visualização do volume de indisponibilidades das máquinas e permite uma melhor análise das instâncias (as partes mais escuras do gráfico indicam períodos de indisponibilidade).

Por fim, o sistema classifica o ambiente das máquinas utilizadas na instância como *Instável* se o percentual de segundos classificados como indisponíveis ultrapassar 50% do tempo total ($brkdwn > 50\%$), *Regular* se a instabilidade estiver entre 25% e 50% ($25\% < brkdwn \leq 50\%$) e como *Estável* se a instabilidade for menor que 25% ($brkdwn \leq 25\%$). Em seguida ele escreve num arquivo XML todas estas informações apresentadas nesta seção de forma estruturada para ser utilizado no problema como apresentaremos na Seção 5.2.

5.1.2 Tempo de execução dos Jobs

Para o processo de obtenção dos dados de tempo de execução dos Jobs de precificação (Jobs do tipo I), foram introduzidos nos sistemas de precificação da empresa, métodos para armazenar o tempo gasto na precificação de cada ativo separando o tempo para carregar os dados necessários ao cálculo (*load time*) e o tempo efetivo de precificação (*calculation time*). Todos os dados de tempo de processamento foram escritos num arquivo de log local na máquina e foram reunidos e analisados manualmente para a composição das instâncias utilizadas pelo simulador.

A Figura 5.2 apresenta o histograma da distribuição do tempo total em milissegundos (*load + calculation times*) observado durante 20 precificações em 7 das máquinas intel i7 (consideradas as mais rápidas do ambiente). Nele podemos perceber que existe uma concentração muito grande de um tempo de processamento próximo entre 10 e 100 ms.

Foram disponibilizados pela empresa $n = 14,533$ diferentes ativos para precificações. Estes ativos estão separados em 4 grandes grupos: ações, renda fixa (fixinc), opções e índices. Além de pertencer a um grupo, cada ativo também pertence a uma classe dentro de um grupo.

A Tabela 5.3 apresenta a separação em grupos e classes destes ativos. É importante perceber que todos os grupos de ativos (exceto opções) possuem uma distribuição de tempo de processamento p_j muito similar à distribuição apresentada na Figura 5.2 como podemos ver para o grupo de ações na Figura 5.3, renda fixa na Figura 5.4 e índices na Figura 5.5). Além disso, dadas estas distribuições, temos que o sistema de precificação é dominado por um tempo de processamento curto e com um desvio padrão pequeno.

EQUITY	FIXINC		INDEX	OPTION
DEBENTURE	BRADY	LTN	N/A	DOL
N/A	CASADO	N/A		EQTY OPTION
	CCB	NDF		FRA DI1
	CCI	NPL		IBOV
	CDB PÓS	NTNB		IDI
	CDS	NTNC		SWAPTION
	CLN	NTNF		
	CPR FIS	OVER		
	CRI	PAYROLL LOAN		
	DEBENTURE	PRÉ		
	DIFUT	PRECATORIO		
	DOLFUT	PRIVATE CREDIT		
	EQTY FWD	RECEIVABLES		
	EQY FWD T	REPO		
	EURO BOND	SP500FUT MINI		
	FWD	SWAP		
	FRA	T05FUT		
	FUND	T10FUT		
	ZERAGEM	TAG ALONG		
	FUTURE	TBILL		
	FX	TES		
	INDFUT	TNOTE		
	LFT			

Tabela 5.3: Grupos e Classes de Jobs

No entanto, se olharmos no detalhe algumas das classes de ativos, principalmente para o grupo de opções, como podemos ver na Figura 5.6, veremos que a distribuição difere do padrão apresentado na Figura 5.2. Esta classe possui uma função lenta de cálculo que introduz uma variação grande no tempo de precificação.

Os Jobs do tipo II são calculados após a finalização da precificação de cada Job do tipo I e são responsáveis pelo cálculo efetivo do PnL de cada ativo para um cenário de risco para a carteira do fundo, consideraremos que o tempo gasto por cada Job é equivalente a 20 ms. Vale lembrar que cada um

destes Jobs precisa ser alimentado pelo preço de fechamento (onde este preço é o resultado da precificação realizada pelo Job do tipo I) para ter o seu PnL computado.

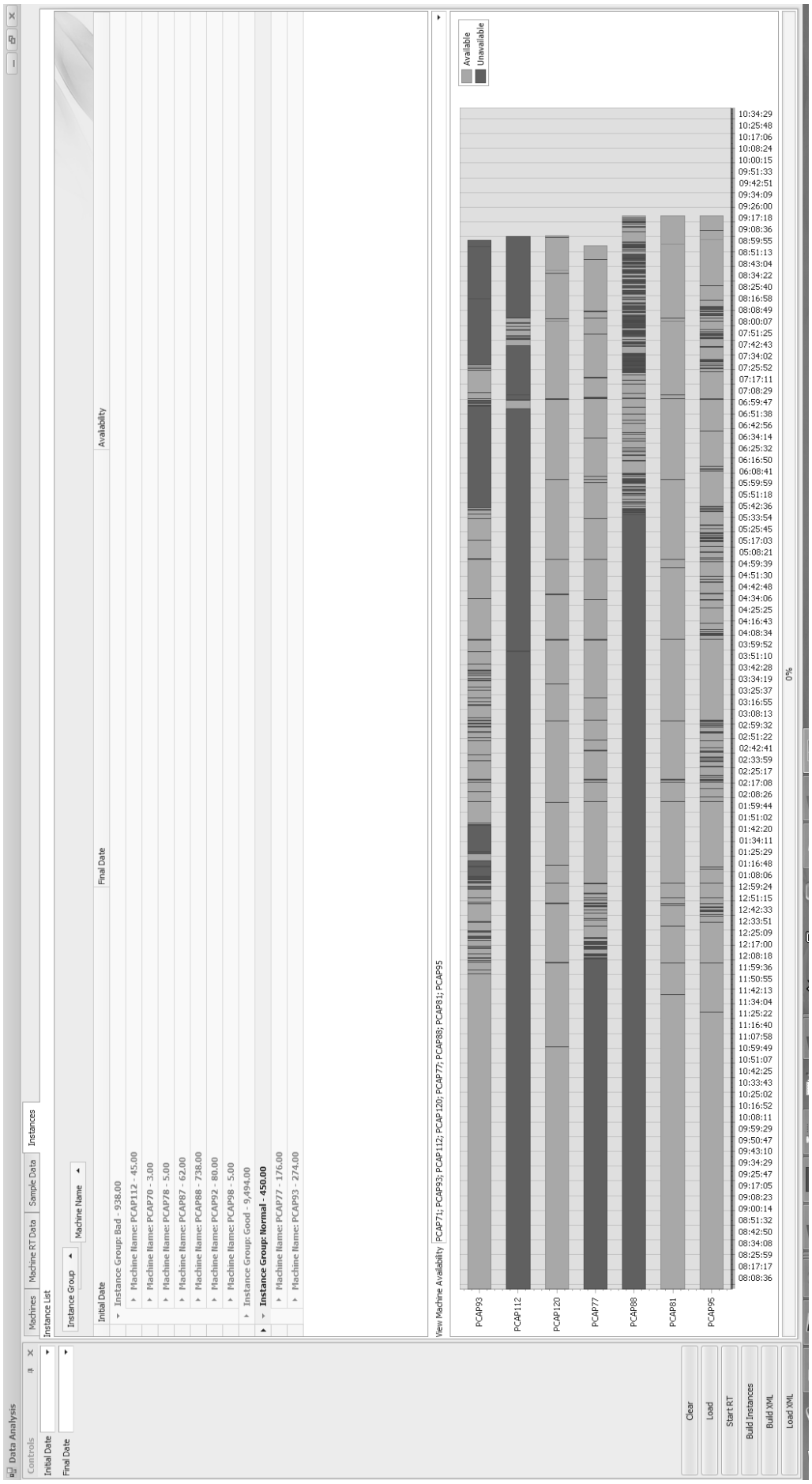


Figura 5.1: Sistema para criação de instâncias para o ambiente de máquinas

Figura 5.2: Histograma da distribuição de tempo de todos os ativos precificados (ms)

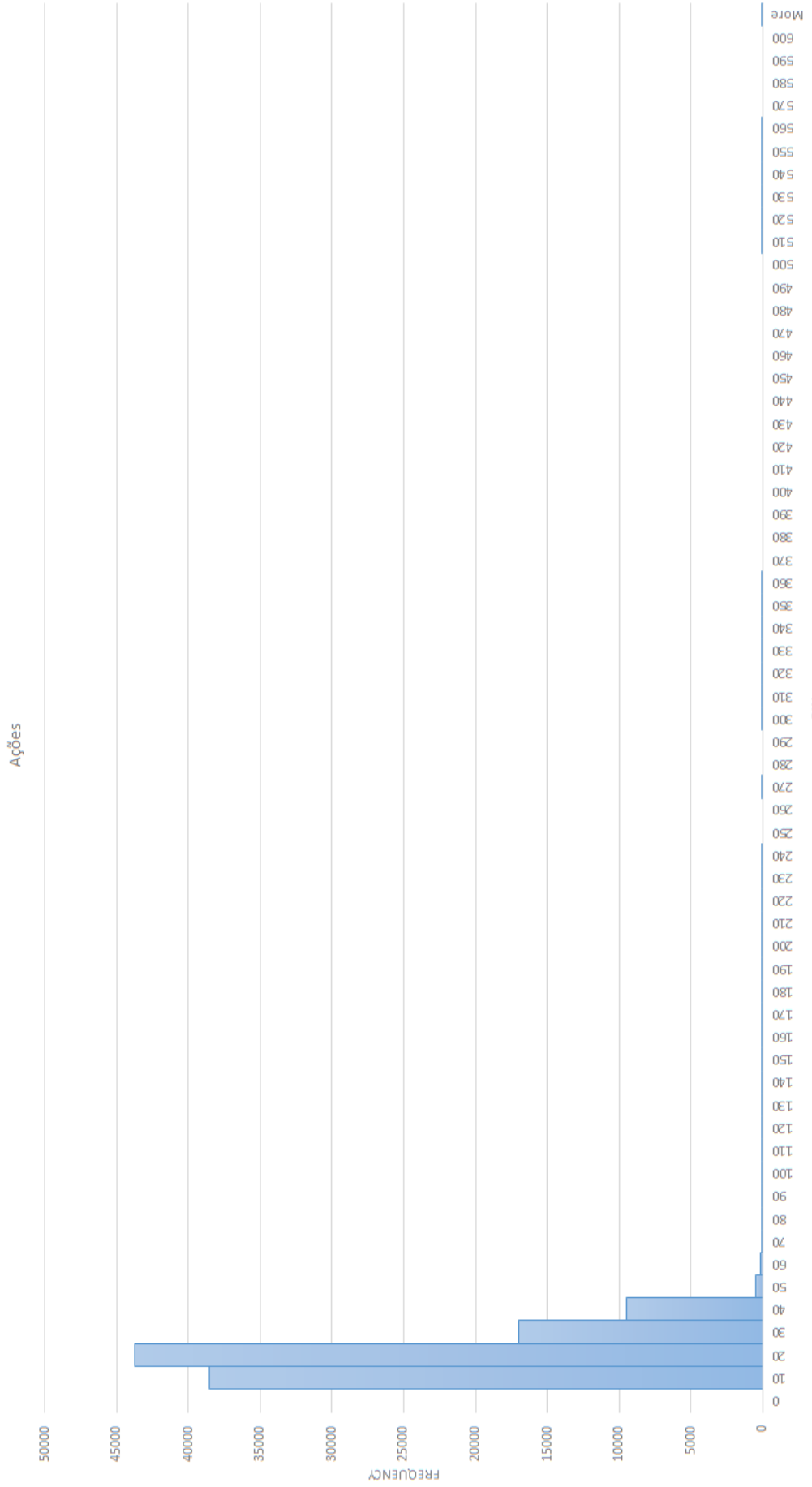


Figura 5.3: Ações

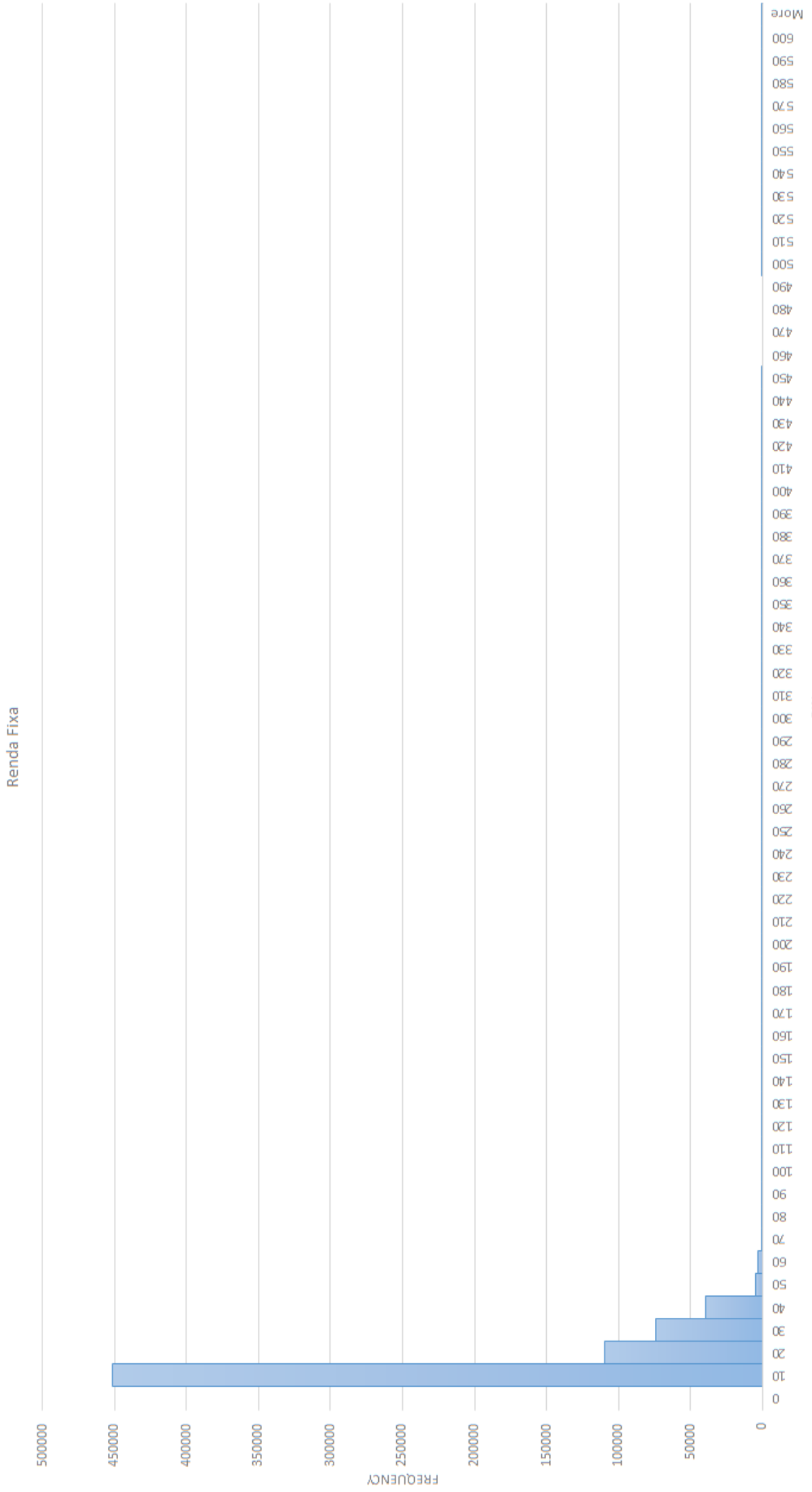


Figura 5.4: Renda Fixa

Figura 5.5: Índices

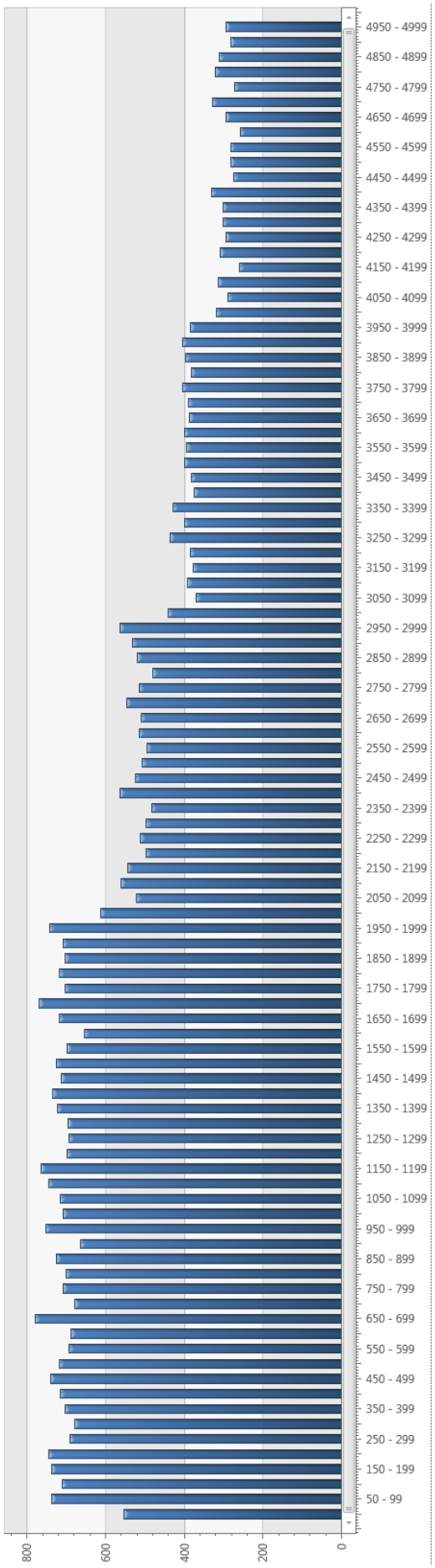


Figura 5.6: Histograma da distribuição de tempo Opções (ms)

5.2 Modelo de Simulação

O sistema para simular a execução dos algoritmos foi desenvolvido em C# e sua interface em WPF utilizando o padrão de projeto MVVM para separar a lógica de programação (*View Model*) da interface do usuário (*View*) e do modelo de dados (*Model*). Além disso, o projeto utiliza o framework MEF (27) para desacoplar os componentes utilizando o conceito de inversão de controle e injeção de dependência (*dependency injection*).

O *View Model* no padrão MVVM encapsula a lógica de apresentação e os dados apresentados na View. Sua característica principal e fundamental é que ele não tem e não deve ter nenhum conhecimento da implementação da *View*. Sua responsabilidade é simplesmente implementar propriedades e comandos os quais a *View*, via binding, irá se ligar.

Com isso, conseguimos abstrair toda a lógica da *View*, permitindo assim, testes unitários de interação, sem a necessidade de ferramentas de automação de testes visuais, que geralmente não são eficientes. Isto também permite que toda a lógica dos algoritmos seja implementada sem a preocupação com a apresentação dos resultados.

O *View Model* principal da aplicação está representado na Figura 5.7, em seguida apresentaremos o diagrama de sequência para a execução da função principal do aplicativo (Execução do teste síncrono):

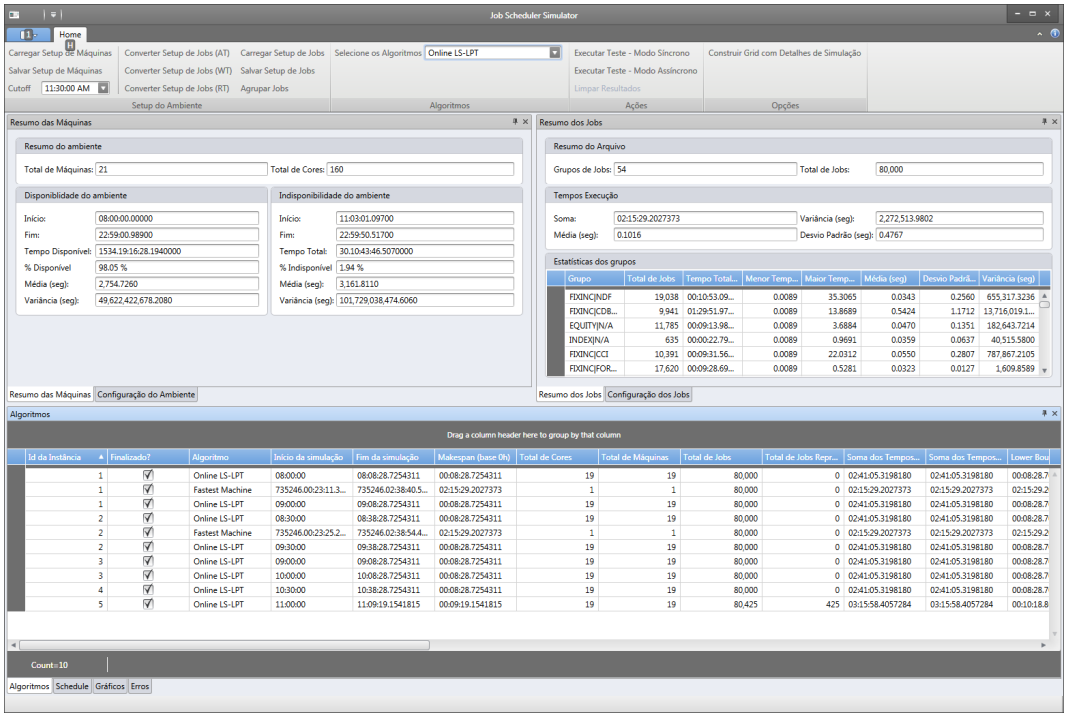


Figura 5.7: VM principal da aplicação de simulação

A aplicação permite que o usuário carregue as informações das instâncias do ambiente de máquinas. Uma vez que a informação foi carregada, o sistema fornece um resumo sobre o ambiente, indicando a quantidade de máquinas, o tempo total de disponibilidade, o tempo total de indisponibilidade e a média em segundos que as máquinas estão disponíveis. Além desta informação resumida, o sistema também apresenta o detalhamento, por máquina, destas informações.

Isto nos permite qualificar as instâncias de entrada e nos fornece mais informações para obtermos uma melhor análise dos resultados dos algoritmos.

Outra funcionalidade é a capacidade de converter os arquivos de log com as informações de tempo de processamento dos jobs (apresentada em 5.1.2), tomando como parâmetro todos os Jobs (AT), somente os n Jobs com pior tempo (WT) e Jobs selecionados de forma aleatória por uma distribuição uniforme (RT).

Uma última funcionalidade que precisa ser mencionada é a capacidade de agrupamento de pequenos Jobs em Jobs maiores. A ideia é semelhante a preparar um processamento em *batch*(1) (sem paralelização de execução), conforme comentado no Capítulo 2. O objetivo principal do agrupamento é reduzir o número de envios de jobs muito curtos às estações de trabalho no ambiente real para não sofrermos com fatores externos ao algoritmo, como a latência da rede.

Ambas conversões de Jobs podem ser custosas no tempo e portanto o sistema oferece a possibilidade de salvar cada uma destas em arquivos XML estruturados.

Uma vez definido o ambiente de máquinas e os Jobs a serem executados, o usuário seleciona os algoritmos para comparação e em seguida executa o teste.

Conforme discutido anteriormente, o sistema foi desenvolvido para desacoplar a parte de interface do usuário (ie. telas) do desenvolvimento dos algoritmos. Para que isto se torne uma tarefa produtiva, foi necessário criar interfaces de testes que devem ser implementadas por todos os algoritmos. Assim, pode-se garantir que o sistema não precisa ser alterado para que um novo algoritmo seja testado com as instâncias nele carregadas.

Em linhas gerais a simulação funcionará da seguinte forma. O algoritmo define qual é o melhor conjunto Job - Máquina a ser executado segundo sua política de escalonamento e simula a execução do job adicionando o seu tempo de execução à máquina selecionada. Caso este tempo ultrapasse o período atual de disponibilidade, consideramos que o Job não foi executado até o fim e o reinserimos na lista de Jobs pendentes. Para a máquina, este tempo de execução até o período de indisponibilidade é perdido e a máquina só voltará a

poder ser escolhida para execução de Jobs quando apresentar um novo período de disponibilidade.

A seguir exemplificaremos dois casos de envio de jobs para a simulação, no primeiro caso, apresentaremos um cenário onde os Jobs possuem tempo de execução maior que a frequência de observações (sampling) dos intervalos de disponibilidade, ou seja, consideraremos que os Jobs levam segundos para serem executados. O segundo caso, apresenta Jobs com tempo de execução menor que a frequência de observações de intervalos de disponibilidade, ou seja, consideraremos que os Jobs demoram milissegundos para serem executados.

Como primeiro exemplo temos dois Job que demoram 20 segundos e uma única máquina que apresenta apenas 1 período de indisponibilidade entre 2 e 5 segundos. O primeiro Job é enviado para a máquina e como sua execução ultrapassa primeiro período de disponibilidade da máquina (0 a 2 segundos), o primeiro Job é reinserido na lista de Jobs pendentes. No sexto segundo o segundo Job (ou o primeiro, dependendo da política de escalonamento do algoritmo) é enviado novamente para a máquina e como não há mais períodos de indisponibilidade é considerado como executado com sucesso, assim como o próximo Job. O sistema então considera o escalonamento como completo e retorna para o usuário um resumo da execução apresentando 1 reprocessamento e o Makespan de 46 segundos.

Como segundo exemplo temos 50 Jobs que demoram 15 milissegundos com a mesma configuração de disponibilidade de máquinas do primeiro exemplo. O primeiro Job é enviado para a máquina e como sua execução não ultrapassa o primeiro período de disponibilidade da máquina (0 a 2 segundos), o primeiro Job é executado com sucesso e o simulador é avisado instantaneamente que o Job foi executado, fazendo com que o próximo Job seja enviado. Assim, os demais 13 passam pela mesma análise e são executados com sucesso. Quando o 14º Job é enviado, o simulador acusa que sua execução ultrapassa o primeiro período de indisponibilidade de dois segundos, isto faz com que o este Job seja reinserido na lista de Jobs pendentes. No sexto segundo o processo é reiniciado e um novo Job é enviado novamente para a máquina e como não há mais períodos de indisponibilidade é considerado como executado com sucesso. O sistema então considera o escalonamento como completo e retorna para o usuário um resumo da execução apresentando 1 reprocessamento e o Makespan de 11.5 segundos.

O capítulo seguinte apresentará as instâncias escolhidas para os testes e os resultados experimentais obtidos com os algoritmos desenvolvidos.

6
Resultados Experimentais e Discussão

O estudo empírico apresentado neste capítulo foi conduzido de acordo com as orientações propostas por Kitchenham em (22) e tem como objetivo testar a hipótese de que algoritmos de *List Scheduling* que utilizam informações do histórico recente de disponibilidade das máquinas produzem melhores resultados de makespan (onde o menor tempo é o melhor resultado) que algoritmos que não utilizam estas informações.

O propósito deste estudo é verificar sob a perspectiva do programador se a adição de informações estocásticas de disponibilidade das máquinas pode melhorar o processo de decisão do escalonamento dos jobs. Nosso objeto de análise serão os algoritmos e o foco da qualidade será a redução do makespan e a média de tempo dos jobs reprocessados. Este último elemento de qualidade nos fornecerá uma medida complementar ao makespan, cujo racional é se jobs muito longos forem constantemente reprocessados, teremos provavelmente um makespan maior.

Conforme apresentado no Capítulo 1, temos que a indústria estudada é o mercado de finanças e o problema é a redução do makespan para obter o resultado do risco de uma carteira de investimentos no menor tempo possível utilizando diversas estações de trabalho ociosas (ou com baixa utilização de recursos de memória e processamento) de uma empresa para realizar os cálculos de risco, sem afetar o trabalho do dia a dia do usuário da estação de trabalho.

Em geral, usuários desta indústria possuem um perfil que preza a qualidade da informação e a velocidade na sua obtenção para tomada de decisão. No entanto, os recursos de infra estrutura necessários para obter estes resultados geralmente ficam em segundo plano e se tornam um problema quando este não consegue obter a informação desejada num tempo razoavelmente curto (geralmente na escala de minutos). Desta forma, para atender esta demanda de pico, a maioria destes usuários requerem estações de trabalho superdimensionadas para sua demanda de processamento, deixando estas estações ociosas em boa parte do dia útil, podendo ser usadas por sistemas distribuídos para realizarem tarefas que necessitem de poder computacional elevado.

Conforme apresentado no Capítulo 5, as instâncias referentes ao ambiente

das máquinas foram obtidas através do monitoramento de trinta dias de utilização destas estações. Testaremos os algoritmos propostos no Capítulo 4 escolhendo aleatoriamente dois dias de observações quaisquer e dois dias para cada grupo de ambiente definido como instável, regular e estável, totalizando 8 dias de observações para o ambiente de máquinas. O objetivo desta separação em grupos foi gerar uma maior diversidade de ambiente de máquina o que nos permite uma melhor análise do desempenho de cada algoritmo para diferentes distribuições de disponibilidade.

Para cada teste, o dia de observação será decomposto em inícios de 30 minutos a partir das oito horas da manhã até nove horas da noite de forma a representar o horário normal de trabalho e de utilização das estações analisadas. Isto significa que para cada arquivo de disponibilidade das máquinas, teremos 27 instâncias de entrada para os ambientes (pex. início da simulação às 8:00, 8:30, 9:00, ...) totalizando 216 (27 x 8) diferentes configurações de ambientes que serão utilizados para cada um dos algoritmos testados.

Para o experimento, consideraremos que cada posição só aparece uma única vez na carteira. Esta restrição não representa perda de generalidade, pois sua introdução só resulta na diminuição de jobs do tipo II (ie. uma quantidade de jobs relativamente maior que não representa redução ou aumento do nível da árvore de precedência, somente apresenta um aumento no número de arestas). Os jobs de tipo I serão representados por uma entrada de ativo do arquivo de configuração de jobs. Os diferentes cenários de precificação serão representados pela repetição destes ativos no arquivo de configuração.

Os experimentos foram executados em um computador com processador Intel Core i7-2600k 3.4GHz e 16GB de RAM com o sistema operacional Windows 7 Professional SP1 x64. O modo utilizado para a tomada de tempo foi total de milissegundos. Foi imposto um corte de tempo final limitado pela própria instância, fixa em nove horas da noite, significando que qualquer simulação que ultrapasse este limite será considerada como não finalizada. Não foi estabelecido um tempo limite para a execução dos testes e reportaremos a duração da execução da simulação de cada algoritmo. Esta duração está diretamente relacionada ao tempo que o algoritmo leva para gerar o escalonamento.

Em relação a lista de jobs para execução, separaremos as análises em 2 categorias: 80.000 jobs com pior tempo de execução (WT.80k) e jobs agrupados em 10.500 jobs batch(1), onde cada job batch(1) contém uma quantidade de jobs referentes a uma média de 10 segundos de execução (Batch.10k). Desta forma, teremos 2 classes de simulações para todas as instâncias de disponibilidade de máquinas. Na primeira categoria os jobs são, em geral, menores que o sample time (variando entre 0.0090 e 3 segundos) e na segunda

categoria eles são maiores que o sample time (variando entre 2 e 20 segundos)

Os algoritmos serão indicados nas tabelas de resultado por sua abreviação. Para simplificar a análise dos resultados, a Tabela 6.1 resume os algoritmos que analisaremos indicando a sua abreviação, sua seção nesta tese e o seu nome completo.

Tabela 6.1: Resumo dos Algoritmos

Algoritmo	Seção	Descrição
LS-SIRO	4.1.1	Service In Random Order
LS-LPT	4.1.2	Longest Processing Time
LS-SPT	4.1.3	Shortest Processing Time
LS-BLPPT	4.2.1	Longest Probable Processing Time

Como o algoritmo proposto LS-BLPPT é sensível à escolha dos seus parâmetros α e T, realizamos uma análise de sensibilidade do algoritmo (detalhada na Seção 6.4) e escolhemos os parâmetros $\alpha = 0.8$ e $T = 4$ minutos. A escolha foi feita analisando o resultado da análise de sensibilidade de um dia de observação (27 instâncias). É importante ressaltar que o dia escolhido para a escolha dos parâmetros coincide com um dos dias envolvidos na análise do ambiente com alta disponibilidade, mas obviamente não coincide com nenhum outro dia das demais análises.

6.1 Definição das Análises

Nesta seção descrevemos as análises que serão realizadas nas simulações dos algoritmos. Inicialmente descrevemos o ambiente de simulação para cada resultado gerado, cujo os detalhes estão descritos na Seção 6.1.1.

Como nosso objetivo é a minimização do makespan, definimos na Seção 6.1.2 as medidas relacionadas ao Makespan e detalhamos processos comparativos com o limite inferior.

Em seguida apresentamos na Seção 6.1.3 as análises relacionadas ao desempenho dos algoritmos de escalonamento que foram propostos. Para facilitar a análise final dos resultados cada uma destas seções corresponde a uma tabela de análise. Chamamos a tabela relacionada à Seção 6.1.2 de "Tabela de Makespan"e a tabela relacionada às análises da Seção 6.1.3 de "Tabela estatísticas e desempenho".

6.1.1 Análises relacionadas ao ambiente de simulação

Como os resultados podem apresentar alguma dependência relacionada ao ambiente de simulação, iniciaremos cada sub seção de análise descrevendo características do ambiente para o qual os resultados da simulação serão apresentados. Estamos interessados nas seguintes medidas:

- $\overline{m}$: Média da quantidade de núcleos de processamento utilizados na simulação de cada algoritmo para todas as instâncias;
- $\overline{D\%}$: Média do percentual de disponibilidade para o processamento dos jobs nas máquinas para todas as instâncias da simulação de cada algoritmo. A Equação 6-1 apresenta a forma matemática para o cálculo de $D\%$ para uma instância e um algoritmo, onde $D_i(C_{max})$ é a soma do tempo disponível para processamento na máquina i até o makespan C_{max}. O valor $\overline{D\%}$ é obtido pela média destes resultados para todas as instâncias e algoritmos.

$$D\% = \frac{1}{m}\sum_{i=1}^{m}\frac{D_i(C_{max})}{C_{max}} \tag{6-1}$$

- $\overline{D_t}$ e $\overline{I_t}$: Média do conjunto de intervalos de tempo disponível ($D_i(C_{max})$) e indisponível ($I_i(C_{max})$) de todas as máquinas para todas as instâncias da simulação de cada algoritmo;
- $\overline{P(I|D)}$: Média da probabilidade condicional de mudança de estado disponível para indisponível. O cálculo para cada $P(I|D)$ foi mostrado na Seção 4.2.1, o valor $\overline{P(I|D)}$ é obtido através da média de $P(I|D)$ de cada algoritmo e cada instância que compõem o ambiente analisado.

Enquanto os itens $\overline{D\%}$ e $\overline{D_t}$ nos fornecem uma medida relacionada a disponibilidade do ambiente em função do tempo, o último item $\overline{P(I|D)}$ nos oferece uma ideia da frequência de mudança de estados neste ambiente.

Como exemplo, suponha que existam duas máquinas, onde a disponibilidade de cada uma está representada pelos intervalos marcados em branco na Figura 6.1. Nela, vemos que a primeira máquina está disponível no intervalo 0 a 5 segundos e indisponível de 5 a 20 segundos e segunda máquina tem alternância de estado (disponível para indisponível e vice versa) a cada 2 segundos.

Considerando que um algoritmo atinja um makespan igual a 10 segundos para uma instância qualquer, temos um ambiente com $\overline{D\%} = 55\%$, pois

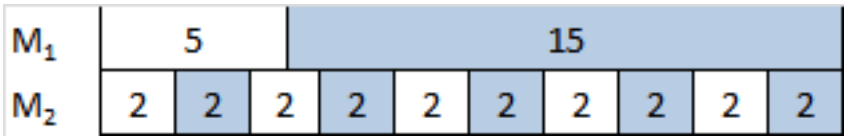


Figura 6.1: Duas máquinas e suas disponibilidades

$D_1(10) = 50\%$ e $D_2(10) = 60\%$. $\overline{D_t}$ será dado pela média do conjunto de intervalos de disponibilidade de todas as máquinas da instância $\{5, 2, 2, 2\}$ resultando em $\overline{D_t} = 2.75$ segundos. Já para $\overline{P(I|D)}$, calculamos as probabilidades condicionais de cada máquina onde:

$$P_{M_1}(I|D) = \frac{1}{5} = 20\%$$

e

$$P_{M_2}(I|D) = \frac{2}{2+2+1} = 40\%,$$

para finalmente calcularmos a média da probabilidade de alternância de estados no ambiente $\overline{P(I|D)} = 30\%$.

6.1.2 Análises relacionadas ao Makespan

Nesta tabela, apresentaremos tanto dados de makespan para cada um dos algoritmos como o seu percentual em relação ao limite inferior médio. Os dados analisados para cada conjunto de instâncias da tabela serão:

- $\min C_{max}$: Menor makespan observado;
- $\max C_{max}$: Maior makespan observado;
- $\overline{C_{max}}$: Média dos makespans observados;
- $LB(\%)$: Percentual do makespan médio sobre o limite inferior médio.

Para calcularmos um limite inferior (LB) de cada instância consideramos que as máquinas aceitam preempção. Nosso objetivo é encontrar o menor T^*, como indicado na Figura 6.2, tal que $T_p \leq D(T^*)$, onde:

- $T_p = \sum_{j=1}^{n} p_{xj}$: O somatório dos tempos de processamento de todos os n jobs na máquina mais lenta M_x disponível no ambiente,
- $D(T^*)$ é dado pela Equação 6-2. Trata-se do somatório de disponibilidades das máquinas, em função de um tempo limite T^*, ponderadas pela sua velocidade v_j relativa a pior velocidade de processamento v_x de todas as máquinas.

$$D(T^*) = \sum_{i=1}^{m} D_i(T^*)\frac{v_x}{v_i}, \tag{6-2}$$

onde m é a quantidade de máquinas e $D_i(T^*)$ o tempo disponível até o instante T^* na máquina i.

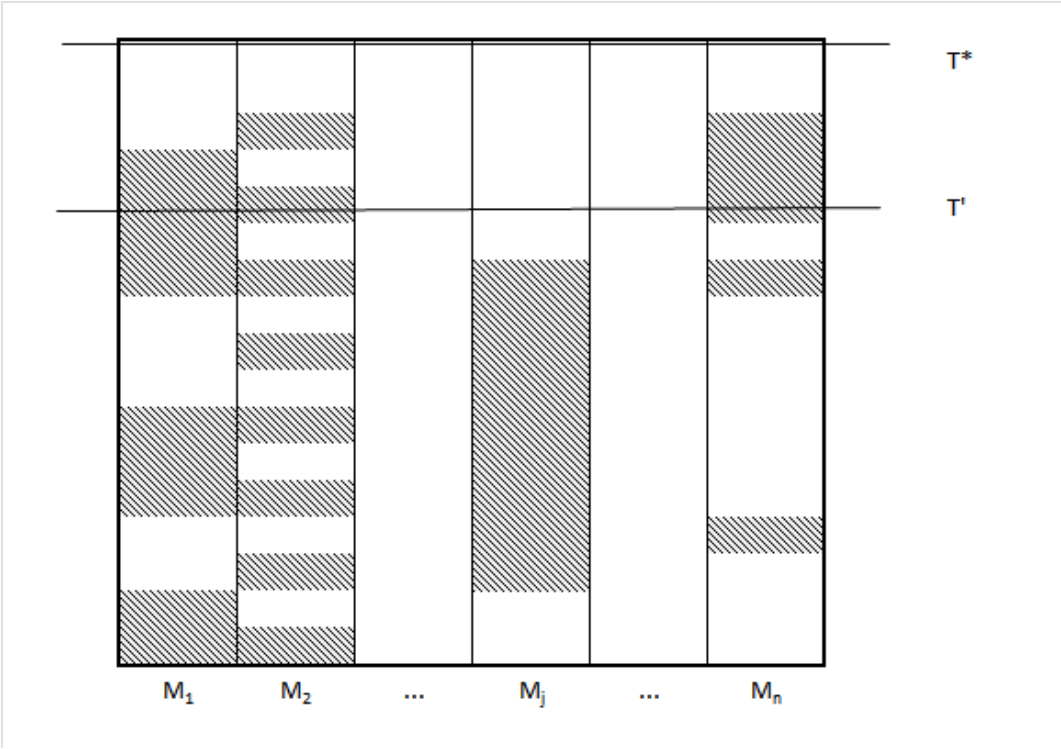


Figura 6.2: Diagrama de disponibilidades das máquinas

Para entender o limite inferior, considere uma estrutura que apresenta um conjunto de 4 máquinas com seus respectivos intervalos de disponibilidade pré-definidos com uma alocação arbitrária de jobs tal como a seguir:

$$M_1 = \{p_1, p_2, p_3\}$$

$$M_2 = \{p_4, p_5\}$$

$$M_3 = \{p_6, p_7\}$$

$$M_4 = \{p_8, p_9, p_{10}\}$$

Temos que $T_p = \sum_{j=1}^{10} p_{xj}$ como o somatório do tempo de processamento de cada um dos jobs na máquina mais lenta disponível no ambiente. Para este exemplo, consideraremos que máquina mais lenta do ambiente é a máquina M_1 com velocidade de processamento v_1.

Além disso, temos que $D_i(T')$ é a disponibilidade de tempo de processamento (até um tempo limite T') de cada máquina i do ambiente e que o

tempo p_{ij}, necessário para o processamento do job j na máquina i, é dado por $p_{ij} = p_i/v_j$. Desta forma, como todo p_j foi dividido por v_x relativo a velocidade da máquina mais lenta, teremos as 4 Inequações 6-3, as quais reorganizaremos os parâmetros como em 6-4, para chegarmos a Inequação 6-5.

$$\begin{aligned} p_1\left(\frac{v_1}{v_1}\right) + p_2\left(\frac{v_1}{v_1}\right) + p_3\left(\frac{v_1}{v_1}\right) &\leq D_1(T') \\ p_4\left(\frac{v_1}{v_2}\right) + p_5\left(\frac{v_1}{v_2}\right) &\leq D_2(T') \\ p_6\left(\frac{v_1}{v_3}\right) + p_7\left(\frac{v_1}{v_3}\right) &\leq D_3(T') \\ p_8\left(\frac{v_1}{v_4}\right) + p_9\left(\frac{v_1}{v_4}\right) + p_{10}\left(\frac{v_1}{v_4}\right) &\leq D_4(T') \end{aligned} \tag{6-3}$$

$$\begin{aligned} p_1 + p_2 + p_3 &\leq D_1(T')\left(\frac{v_1}{v_1}\right) \\ p_4 + p_5 &\leq D_2(T')\left(\frac{v_2}{v_1}\right) \\ p_6 + p_7 &\leq D_3(T')\left(\frac{v_3}{v_1}\right) \\ p_8 + p_9 + p_{10} &\leq D_4(T')\left(\frac{v_4}{v_1}\right) \end{aligned} \tag{6-4}$$

$$\sum_{j=1}^{10} p_j \leq D_1(T')\left(\frac{v_1}{v_1}\right) + D_2(T')\left(\frac{v_2}{v_1}\right) + D_3(T')\left(\frac{v_3}{v_1}\right) + D_4(T')\left(\frac{v_4}{v_1}\right) \tag{6-5}$$

O lower bound T^* é obtido buscando o menor T' tal que a Inequação 6-5 é satisfeita. Esta busca pode ser realizada de diversas formas, nós implementamos um método de cálculo numérico onde, definimos como T' inicial o tempo de execução dos jobs na máquina mais rápida dividido pela quantidade de máquinas no ambiente. O processo inicia incrementando T' por um delta grande (por exemplo, uma hora). Caso este valor não satisfaça mais a inequação, reduzimos o delta e diminuímos de T', fazemos isso até um delta muito pequeno, ou até que os termos da Inequação 6-5 sejam equivalentes.

6.1.3 Análises relacionadas ao desempenho dos algoritmos

Nesta tabela, estamos interessados em análises relacionadas ao desempenho do algoritmo tanto em função do seu tempo de execução como em relação ao reprocessamento dos jobs durante a simulação. Desta forma, analisaremos

os seguintes resultados:

- $T(\%)$: Percentual do tempo total de execução do algoritmo em relação ao tempo total de execução de todos os algoritmos;
- $\overline{n}$: Média da quantidade de jobs reprocessados;
- $n_{\overline{t}}$: Média dos tempos de jobs reprocessados.

6.2 Análise de 80k jobs com pior tempo

A Tabela 6.2 apresenta o resumo dos tempos de execução dos 80.000 jobs que apresentaram pior tempo de execução. Como podemos perceber, apesar do processo em apenas uma máquina durar 2 horas e 53 minutos, a média de tempos dos jobs é muito baixa (0.13 segundos) com um desvio padrão não ultrapassando 1 segundo. A Figura 6.3 apresenta o histograma do tempo estimado de processamento destes jobs.

Grupos de jobs	54
Total de jobs	80,000
Soma dos Tempos de Execução	02:53:10
Média dos Tempos de Execução (seg)	0.13
Desvio Padrão dos Tempos de Execução (seg)	0.59

Tabela 6.2: Resumo do grupo de 80k jobs com pior tempo

6.2.1 Ambiente com alta disponibilidade no tempo

Esta seção apresenta um ambiente de máquinas com uma disponibilidade média no tempo de 98.50% e com características dadas pela Tabela 6.3. Nela, podemos perceber que temos intervalos de indisponibilidade muito curtos. Além disso, observamos uma baixa probabilidade de mudança de estado disponível para indisponível e uma alta probabilidade da máquina permanecer disponível.

Os resultados da simulação neste ambiente podem ser analisados na Tabela 6.4.

Para os algoritmos da literatura, podemos perceber que o algoritmo LS-LPT obteve o melhor resultado de makespan.

Apesar do LS-BLPPT superar marginalmente em 0.53% o melhor algoritmo da literatura, ele consome um tempo de processamento muito alto como

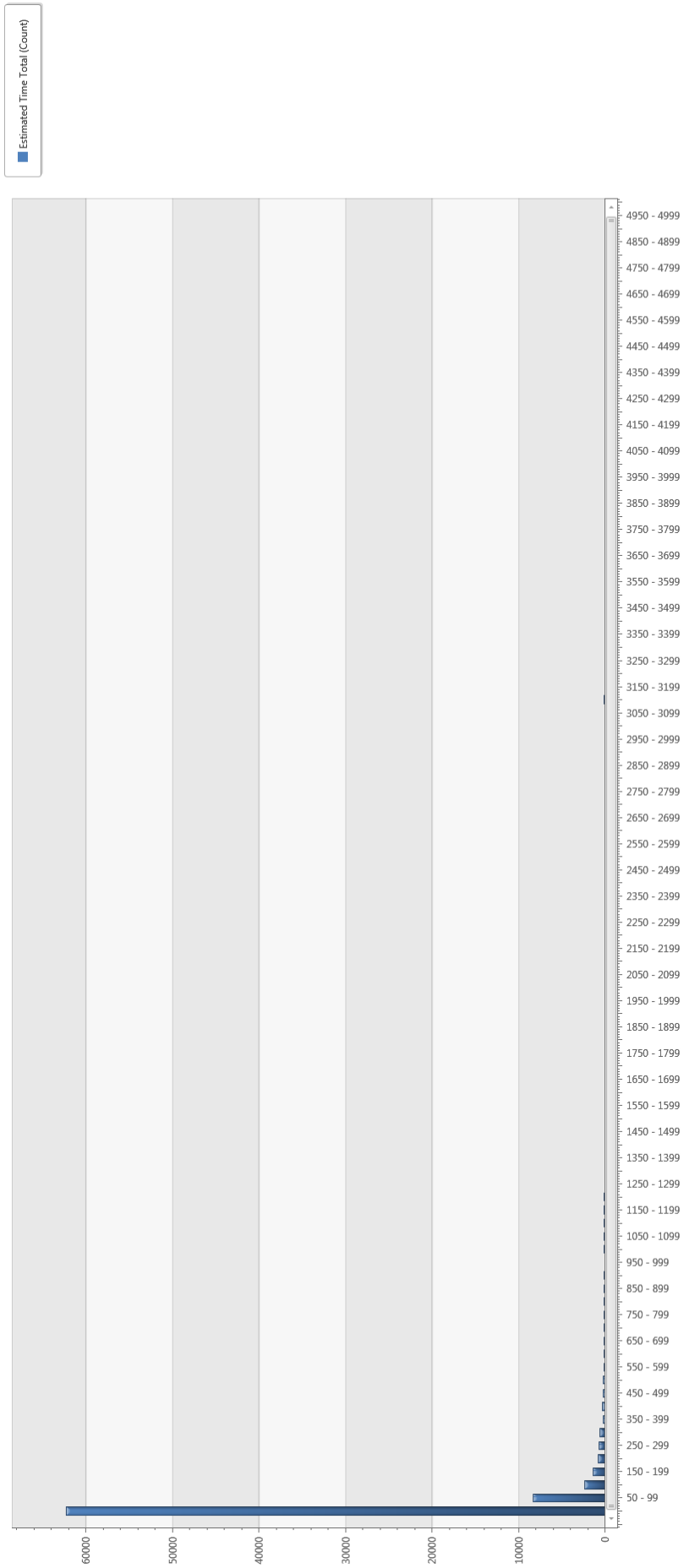


Figura 6.3: Histograma do tempo estimado de execução dos jobs

Medida	**Valor**
$\overline{m}$	13.88
$\overline{D_t}$	00:10:25
$\overline{I_t}$	00:00:52
$\overline{P(I\|D)}$	0.07
$\overline{P(D\|D)}$	0.93

Tabela 6.3: 80k - Ambiente de alta disponibilidade

pode ser visto na Tabela 6.5. Isto poderia inviabilizar a utilização deste algoritmo na prática, pois estaríamos perdendo tempo de execução de job para processar o escalonamento.

Porém, utilizando uma ferramenta de análise de desempenho de código, verificamos que a função de atualização das probabilidades condicionais da máquinas é uma operação muito custosa. Desta forma, se retirarmos esta função, movendo-a para um processo paralelo independente do algoritmo, ou se otimizássemos esta função, o seu desempenho tornar-se-ia muito superior a implementação original, custando um tempo semelhante ao LS-LPT.

Algoritmo	$\min C_{max}$	$\max C_{max}$	$\overline{C_{max}}$	$LB(\%)$
LS-BLPPT	0:10:50	1:01:39	0:26:59	301.65 %
LS-LPT	0:10:50	1:01:39	0:27:02	302.18 %
LS-SIRO	0:10:55	1:01:43	0:27:13	303.87 %
LS-SPT	0:11:15	1:02:07	0:27:30	306.90 %

Tabela 6.4: 80k - Tabela de Makespan em alta disponibilidade

Os resultados de duração dos jobs reprocessados apresentados na Tabela de Desempenho 6.5 apresentam baixa dispersão, pois os jobs desta categoria são muito semelhantes.

Nesta tabela, podemos observar que o algoritmo LS-BLPPT conseguiu obter uma redução de reprocessamento de jobs em relação ao LS-LPT o que colabora com uma dos seus objetivos que é reduzir a quantidade de reprocessamentos.

Algoritmo	$T(\%)$	$\overline{n}$	$n_{\overline{t}}$
LS-BLPPT	60.60 %	80.84	0:00:01
LS-LPT	14.34 %	81.84	0:00:02
LS-SIRO	10.18 %	81.88	0:00:02
LS-SPT	14.88 %	82.95	0:00:02

Tabela 6.5: 80k - Tabela de Desempenho em alta disponibilidade

6.2.2 Ambiente com alta indisponibilidade no tempo

Nesta seção, analisaremos os resultados obtidos para um ambiente de máquinas com disponibilidade média no tempo menor que 90% e com características dadas pela Tabela 6.6. Nela, podemos perceber que temos uma indisponibilidade mais alta no tempo e que este ambiente na média não muda de estados disponíveis para indisponíveis (e vice versa) com frequência.

Medida	**Valor**
$\overline{m}$	11.53
$\overline{D_t}$	00:15:27
$\overline{I_t}$	00:14:58
$\overline{P(I\|D)}$	0.04
$\overline{P(D\|D)}$	0.96

Tabela 6.6: 80k - Ambiente de alta indisponibilidade

Os resultados da simulação neste ambiente podem ser analisados na Tabela 6.7. Nela, podemos perceber que o algoritmo LS-BLPPT consegue superar o melhor algoritmo da literatura (LS-LPT) por uma margem muito pequena. A melhoria observada de 0.66 % vem, também, com um custo maior de processamento, pois como podemos ver na Tabela 6.8 este algoritmo consome 60 % do tempo total da simulação enquanto o LPT consome 14%.

Em relação ao tempo consumido pelo LS-BLPPT, as mesmas otimizações citadas na Seção 6.2.1 podem ser aplicadas nesta seção.

A quantidade de jobs reprocessados também foi reduzida na média, assim como na Seção 6.2.1.

Algoritmo	$\min C_{max}$	$\max C_{max}$	$\overline{C_{max}}$	$LB(\%)$
LS-BLPPT	0:10:27	0:49:36	0:30:51	390.68 %
LS-LPT	0:10:27	0:49:36	0:30:54	391.34 %
LS-SIRO	0:10:34	0:49:45	0:31:02	392.93 %
LS-SPT	0:11:19	0:49:52	0:31:22	397.02 %

Tabela 6.7: 80k - Tabela de Makespan (MS) alta indisponibilidade

6.2.3 Demais instâncias

Nesta seção apresentaremos as tabelas com os resultados referentes as demais instâncias simuladas cujo resumo pode ser visto na Tabela 6.9.

Algoritmo	$T(\%)$	$\overline{n}$	$n_{\overline{t}}$
LS-BLPPT	60.67 %	49.20	0:00:01
LS-LPT	14.26 %	50.70	0:00:01
LS-SIRO	10.43 %	50.42	0:00:01
LS-SPT	14.64 %	50.80	0:00:01

Tabela 6.8: 80k - Tabela de Desempenho em alta indisponibilidade

Medida	**Valor**
$\overline{m}$	14.66
$\overline{D_t}$	00:08:43
$\overline{I_t}$	00:02:33
$\overline{P(I\|D)}$	0.05
$\overline{P(D\|D)}$	0.95

Tabela 6.9: 80k - Demais instâncias

Os resultados seguem a mesma linha do ambiente com alta disponibilidade e servem para ilustrar que o algoritmo LS-BLPPT consegue obter um desempenho, ligeiramente melhor que o LS-LPT na média.

Algoritmo	$\min C_{max}$	$\max C_{max}$	$\overline{C_{max}}$	$LB(\%)$
LS-BLPPT	0:11:43	0:44:37	0:19:10	178.06 %
LS-LPT	0:11:43	0:44:37	0:19:11	178.25 %
LS-SIRO	0:11:52	0:44:42	0:19:23	179.93 %
LS-SPT	0:12:17	0:44:56	0:19:42	182.72 %

Tabela 6.10: 80k - Tabela de Makespan (MS) - Demais ambientes

Algoritmo	$T(\%)$	$\overline{n}$	$n_{\overline{t}}$
LS-BLPPT	58.23 %	57.32	0:00:01
LS-LPT	15.61 %	57.83	0:00:01
LS-SIRO	10.28 %	57.84	0:00:01
LS-SPT	15.88 %	57.83	0:00:02

Tabela 6.11: 80k - Tabela de Desempenho demais instâncias

6.3 Análise de 10k jobs em batch(1)

A Tabela 6.12 apresenta o resumo dos tempos de execução da segunda categoria de jobs analisada (aproximadamente 10.500 grupos de jobs de todos os jobs disponíveis para execução). Como podemos perceber, o processo em apenas uma máquina dura aproximadamente um dia e a média dos tempos dos jobs é muito mais alta que caso anterior (8.23 segundos) seguindo uma distribuição apresentada na Figura 6.4 com um desvio padrão de 3.14 segundos.

Grupos de jobs	9
Total de jobs	10.498
Soma dos Tempos de Execução	1d.00:00:03
Média dos Tempos de Execução (seg)	8.23
Desvio Padrão dos Tempos de Execução (seg)	3.14

Tabela 6.12: Resumo do grupo de 10k grupos de jobs em batch(1)

6.3.1 Ambiente com alta disponibilidade

Nesta seção, apresentamos os resultados obtidos para dois dias de análise, totalizando 54 instâncias, cujas máquinas apresentaram uma disponibilidade média de 97% e com características dadas pela Tabela 6.13. Como os jobs são mais longos, passamos a observar mais eventos de mudança de estado que na análise feita na Seção 6.2.

Medida	**Valor**
$\overline{m}$	16.91
$\overline{D_t}$	00:32:59
$\overline{I_t}$	00:19:52
$\overline{P(I\|D)}$	0.07
$\overline{P(D\|D)}$	0.93

Tabela 6.13: B.10k - Ambiente de alta disponibilidade

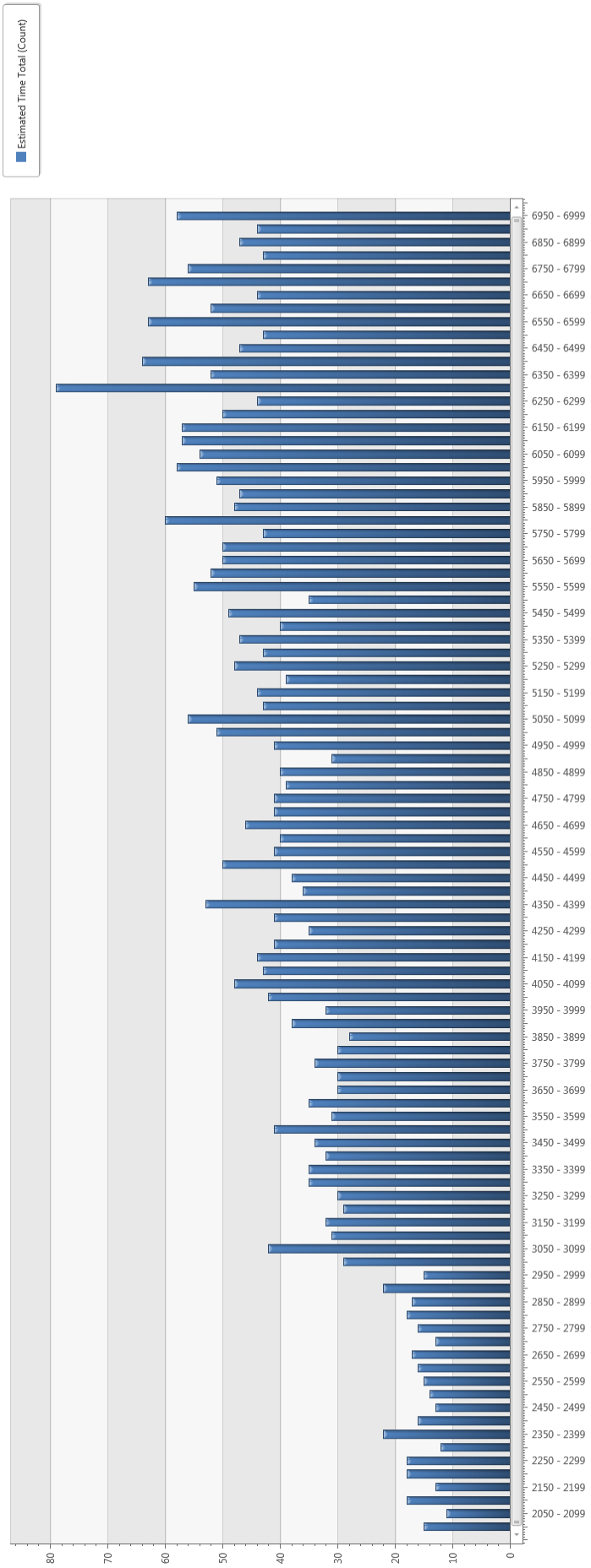


Figura 6.4: Histograma do tempo estimado de execução dos jobs

Os resultados da simulação podem ser vistos na Tabela 6.14. Nela, podemos perceber que os algoritmos baseados na ordenação por ordem decrescente de execução (LPTs) apresentam um desempenho sutilmente melhor que àqueles que utilizam ordenação crescente (1.7% melhor em relação a $LB(\%)$).

O melhor algoritmo neste ambiente foi novamente o LS-BLPPT, apresentando uma diferença de 6% em relação ao limite inferior $LB(\%)$ do melhor algoritmo da literatura. O consumo excessivo do tempo pode ser diminuído assim como nas seções anteriores.

Algoritmo	$\min C_{max}$	$\max C_{max}$	$\overline{C_{max}}$	$LB(\%)$
LS-BLPPT	1:30:10	3:22:38	2:15:21	173.29 %
LS-LPT	1:30:10	3:22:38	2:16:40	174.98 %
LS-SIRO	1:30:20	3:22:50	2:16:41	174.98 %
LS-SPT	1:30:32	3:22:47	2:17:14	175.70 %

Tabela 6.14: B.10k - Tabela de Makespan em alta disponibilidade

Assim como nas seções anteriores os resultados da Tabela 6.15 mostram uma redução na quantidade de reprocessamentos do algoritmo LS-BLPPT em relação aos demais. Além disso, como os jobs nesta seção são maiores que àqueles da Seção 6.2, podemos observar uma dispersão maior dentre a média dos tamanhos dos jobs reprocessados, tendo o algoritmo LS-BLPPT o melhor resultado.

Algoritmo	$T(\%)$	$\overline{n}$	$n_{\overline{t}}$
LS-BLPPT	41.49 %	833.86	0:00:05
LS-LPT	19.70 %	858.20	0:00:07
LS-SIRO	18.90 %	857.66	0:00:08
LS-SPT	19.90 %	869.96	0:00:09

Tabela 6.15: B.10k - Tabela de Desempenho em alta disponibilidade

6.3.2 Ambiente com alta indisponibilidade

Nesta seção, apresentamos os resultados obtidos para dois dias de análise, totalizando 54 instâncias, cujas máquinas apresentaram um ambiente com disponibilidade média de 80% e com características dadas pela Tabela 6.16.

Apesar da indisponibilidade no tempo ser grande, a quantidade de eventos de mudança de estados é baixa. Isto torna a decisão da heurística ainda mais importante, pois quanto mais rápido o algoritmo for finalizado, menor é a chance de enfrentarmos um período longo de indisponibilidade.

Medida	Valor
$\overline{m}$	17.63
$\overline{D_t}$	00:49:25
$\overline{I_t}$	00:47:26
$\overline{P(I\|D)}$	0.04
$\overline{P(D\|D)}$	0.96

Tabela 6.16: B.10k - Ambiente de alta indisponibilidade

Algoritmo	$\min C_{max}$	$\max C_{max}$	$\overline{C_{max}}$	$LB(\%)$
LS-BLPPT	1:26:55	3:02:40	2:23:33	217.43 %
LS-SIRO	1:27:09	3:02:49	2:25:04	219.68 %
LS-LPT	1:26:55	3:02:41	2:25:05	219.75 %
LS-SPT	1:27:04	3:03:03	2:25:53	220.96 %

Tabela 6.17: B.10k - Tabela de Makespan em alta indisponibilidade

Assim como nas seções anteriores o algoritmo LS-BLPPT superou os demais algoritmos em todos os critérios. Neste caso a melhoria foi superior a 2.25% em relação ao melhor algoritmo clássico desta simulação (LS-SIRO).

Unindo esta à melhoria observada na Tabela 6.18, vemos que o algoritmo está realizando seus objetivos, pois ele melhora o desempenho do LS-LPT e no pior caso ele se iguala ao LS-LPT como podemos ver na coluna de $\max C_{max}$.

Algoritmo	$T(\%)$	$\overline{n}$	$n_{\overline{t}}$
LS-BLPPT	37.76 %	499.94	0:00:03
LS-LPT	21.16 %	521.68	0:00:05
LS-SPT	19.95 %	522.46	0:00:05
LS-SIRO	21.14 %	533.95	0:00:06

Tabela 6.18: B.10k - Tabela de Desempenho em alta indisponibilidade

6.3.3 Demais instâncias

Nesta seção apresentaremos as tabelas com os resultados referentes as demais instâncias simuladas cujo resumo pode ser visto na Tabela 6.19.

Os resultados seguem a mesma linha do ambiente com alta disponibilidade e servem para ilustrar que o algoritmo LS-BLPPT consegue obter um desempenho melhor que os algoritmos clássicos na média.

Medida	**Valor**
$\overline{m}$	17.11
$\overline{D_t}$	00:45:19
$\overline{I_t}$	00:31:40
$\overline{P(I\|D)}$	0.07
$\overline{P(D\|D)}$	0.93

Tabela 6.19: 80k - Demais instâncias

Algoritmo	$\min C_{max}$	$\max C_{max}$	$\overline{C_{max}}$	$LB(\%)$
LS-BLPPT	1:30:10	3:22:38	2:28:57	223.20 %
LS-SIRO	1:30:20	3:22:50	2:30:38	225.42 %
LS-LPT	1:30:10	3:22:38	2:30:27	225.54 %
LS-SPT	1:30:32	3:22:47	2:30:54	226.62 %

Tabela 6.20: 80k - Tabela de Makespan (MS) - Demais ambientes

Algoritmo	$T(\%)$	$\overline{n}$	$n_{\overline{t}}$
LS-BLPPT	40.17 %	748.83	0:00:04
LS-LPT	19.47 %	782.72	0:00:07
LS-SIRO	20.13 %	781.56	0:00:06
LS-SPT	20.23 %	803.04	0:00:08

Tabela 6.21: 80k - Tabela de Desempenho demais instâncias

6.4 Análise de sensibilidade do algoritmo proposto

Na Seção 4.2.1, apresentamos a nova heurística defendida nesta tese e afirmamos que o algoritmo proposto depende de dois parâmetros: nível de aceitação α e intervalo histórico T. Nesta seção, apresentamos a metodologia utilizada para a escolha destes parâmetros que foram utilizados nas simulações das seções deste capítulo.

Iniciaremos o processo escolhendo, algumas instâncias dentre o universo total de instâncias disponíveis para a simulação. Nosso objetivo é computar o makespan do algoritmo LS-BLPPT variando tanto o nível de aceitação como o intervalo histórico. Os parâmetros escolhidos serão aqueles que apresentarem melhor resultado de makespan na média.

Escolhemos aleatoriamente um dia de instâncias (27 instâncias no total). Vimos que este dia representava um dia comum de trabalho, com alguns eventos de instabilidade, mas com bons períodos de disponibilidade na média. Em seguida começamos a variar os parâmetros e calcular o makespan, utilizando os jobs em batch da Seção 6.3 como entrada. Os dados para o resultado são

apresentados na Figura 6.5.

Realizamos a mesma análise para um outro dia de instâncias que possuía um período do dia onde houve uma grande instabilidade nas máquinas da empresa e vimos que o resultado de makespan para os parâmetros escolhidos estava dentre os melhores na média. O objetivo desta segunda análise foi verificar se os parâmetros escolhidos no primeiro dia estariam treinados apenas para aquele dia. O resultado pode ser visto na Figura 6.6.

Como podemos ver pela Figura 6.5, à medida que aumentamos muito o intervalo histórico T, observamos que o algoritmo tende a piorar o resultado de makespan obtido, independente do α escolhido.

Porém, isto não quer dizer que quanto menor este intervalo, melhor o resultado, pois se este intervalo de tempo para observação de eventos for muito pequeno, o algoritmo não conseguirá computar uma probabilidade $P_i(D|D)$ que represente o histórico recente de mudança de estado.

Esta habilidade de capturar a probabilidade de uma máquina i mudar de estado é exatamente o que queremos obter para estimar o melhor job que a máquina consegue executar.

Quanto ao nível de aceitação α, parece claro que à medida que α vai crescendo até um determinado valor (no caso 80%), melhor é o resultado de makespan. No entanto, a partir destes 80%, esperamos que ele aproxime-se muito do LPT, tendo em vista que os Jobs escolhidos por conta da Equação 4-4 necessitarão ser cada vez mais longos.

Assim, da Figura 6.5, temos que o menor makespan médio é obtido com os parâmetros $\alpha = 80\%$ e $T = 4$, Desta forma, eles foram escolhidos para as simulações deste capítulo e aplicados nos seis demais dias de análise realizadas nesta dissertação.

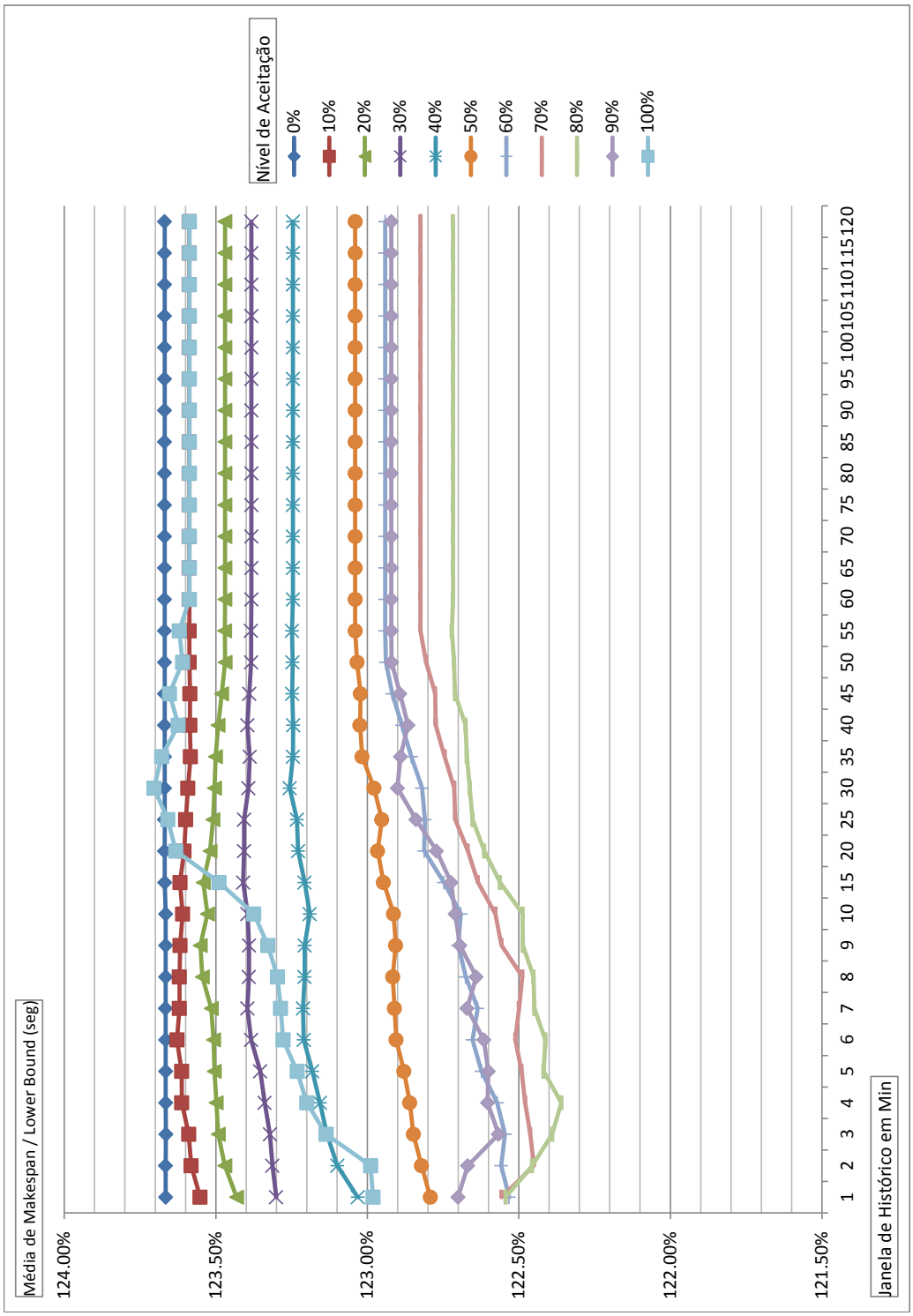


Figura 6.5: Gráfico da média dos resultados para o dia 2

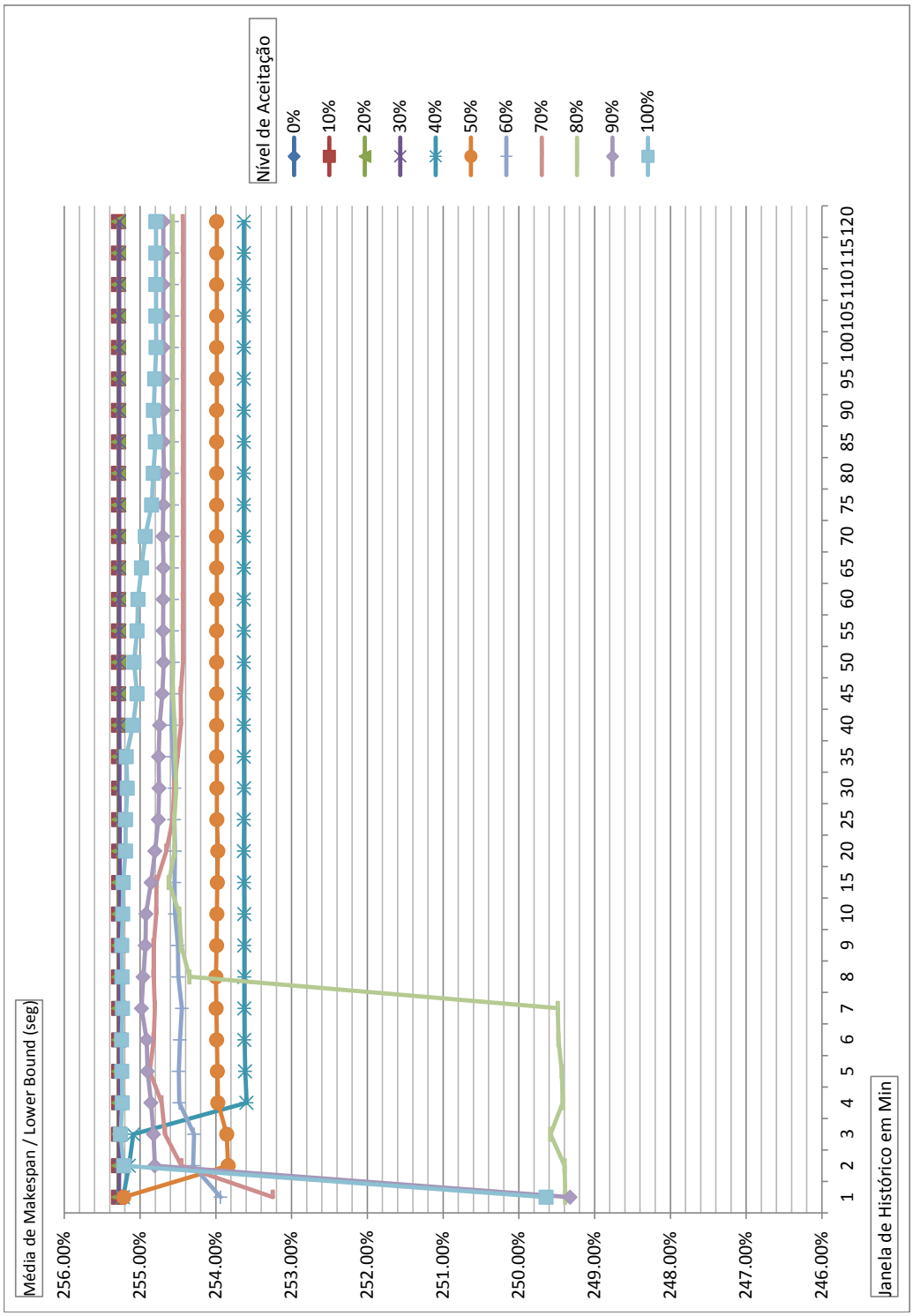


Figura 6.6: Gráfico da média dos resultados para o dia 1

6.5 Conclusões gerais sobre os experimentos

Pelos resultados apresentados, podemos concluir que o algoritmo LS-BLPPT nos fornece melhorias de Makespan em relação aos algoritmos clássicos de *List scheduling* da literatura.

Testamos diversas outras heurísticas que utilizam informações estatísticas. Porém, apenas o LS-BLPPT obteve um resultado satisfatório e apenas seu resultado foi reportado.

Como mencionamos, o algoritmo implementado, no entanto, possui uma ineficiência grande em relação ao tempo de execução dos demais, pois atualiza a probabilidade de execução de todas as máquinas do ambiente a cada processo de decisão. Apresentamos na Seção 6.2.1 uma otimização do algoritmo que sugere uma redução de 80% do tempo apresentado.

Os parâmetros de calibragem utilizados para algoritmo foram uma janela histórica T de 4 minutos e um nível de aceitação α de 80%. Numa análise paralela, verificamos que o algoritmo é muito sensível ao nível de confiança pre-definido.

Desta forma, apesar de tentarmos mitigar este risco através da diversificação tanto de ambientes como dos cenários de simulação, vale atentar que existe um risco, o qual acreditamos ser pequeno, do algoritmo ter apresentado bons resultados para o conjunto de simulações realizadas nesta tese, para o qual o algoritmo foi treinado.

7
Conclusão

Na literatura, a maioria dos problemas de escalonamento não consideram que algumas máquinas podem não estar disponíveis para processamento continuamente e que muitas vezes estas indisponibilidades não são conhecidas o que na prática deve ser considerado.

Nesta tese consideramos o problema de escalonamento em máquinas paralelas com velocidades diferentes que podem ficar indisponíveis para processamento a qualquer instante, fazendo com que o Job que estava em processamento seja reinserido na lista de jobs a serem processados. Resumidamente, nós estudamos a minimização do Makespan do problema $Q_m|prec, brkdwn, Nonresumable|C_{max}$.

Além de desenvolvemos um ambiente de simulação e comparação de algoritmos por um modelo comparativo relacionado ao Makespan, propusemos novas heurísticas que levam em consideração o histórico recente de disponibilidade das máquinas para alocação dos Jobs e comparamos com algoritmos clássicos de *list scheduling* como SIRO, LPT e SPT.

Uma aplicação de monitoramento de máquinas foi desenvolvida e 30 dias de dados de utilização das máquinas foram armazenados. Os experimentos foram conduzidos com um conjunto grande de dados onde tínhamos aproximadamente um milhão de Jobs variando entre 10 ms e 20 s. Além disto os experimentos foram executados utilizando diferentes dias de observação dos ambientes para garantir uma melhor análise dos dados.

Os resultados obtidos com a heurística Longest Probable Processing Time (BLPPT) apresentaram, na média, melhorias consistentes de makespan em relação aos algoritmos clássicos. Porém, o custo de execução do algoritmo pode ser algo relevante no processo de decisão da heurística a ser utilizada.

Para mitigar este custo, sugerimos uma variação deste algoritmo que elimina a necessidade de operações custosas de atualização da matriz de probabilidades das máquinas. Indicamos que esta variação, pode apresentar melhorias significativas em relação ao tempo de processamento.

Numa análise paralela, verificamos que o novo algoritmo é muito sensível ao nível de confiança pre-definido. Desta forma, apesar de tentarmos mitigar

este risco através da diversificação de ambientes e cenários de simulação, vale atentar que o algoritmo pode ter apresentado bons resultados para as simulações realizadas nesta tese. Porém, dito isto, podemos afirmar que tivemos sucesso com a heurística de buscar o melhor job possível para ser executado por uma máquina com limitações de disponibilidade.

Estudos futuros podem estender os conceitos apresentados nesta tese e buscar evoluções relacionadas ao método de treinamento e adaptação dos níveis de confiança e da janela de análise do histórico de eventos, através de machine learning ou outro método qualquer.

Referências Bibliográficas

[1] ANBIMA. **Manual de referencia - precificacao de ativos financeiros**. cartilha nova metodologia, 2005. 2.1.4

[2] BACEN. **Circular 3.086**. `www4.bcb.gov.br/gci/Focus`, 2002. 2.1.3, 2.1.4

[3] BAKER, K. R.; TRIETSCH, D. **Principles of Sequencing and Scheduling**. Wiley Publishing, 2009. 1, 3.2, 3.2, 3.2, 3.2

[4] BOVESPA, B. **Manual de procedimentos operacionais**. /Manual-do-Sistema-de-Administracao-de-Risco.pdf, 2011. 2.1.1

[5] BOVESPA, B. **Manual do sistema de administração de risco**. BMFBOVESPA-Manual-de-Procedimentos-Operacionais-Acoes.pdf, 2011. 2.1.1

[6] BOVESPA, B. **Manual de administracao de risco da camara de derivativos**. MAR-Derivativos-110318-Em-vigor.pdf, 2013. 2.1, 2.1.1, 2.1.4, B.3

[7] CHEN, W. **The International Journal of Advanced Manufacturing Technology**. Scheduling of jobs and maintenance in a textile company, journal, v.31, n.7-8, p. 737–742, 2007. 3.3

[8] COFFMAN, E.G., J.; GRAHAM, R. **Acta Informatica**. Optimal scheduling for two-processor systems, journal, v.1, n.3, p. 200–213, 1972. 3.3

[9] COOK, S. A. **The complexity of theorem-proving procedures**. In: PROCEEDINGS OF THE THIRD ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, STOC '71, p. 151–158, New York, NY, USA, 1971. ACM. 3.1

[10] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. ; STEIN, C. **Introduction to Algorithms, Third Edition**. 3rd. ed., The MIT Press, 2009. 3.1, 3.1, 3.1

[11] D. KARGER, C. STEIN, J. W. **Scheduling algorithms survey**, 1900. 2.2.3

[12] DIRETO, T. **Metodologia de calculo dos titulos publicos federais ofertados nos leiloes primarios**. Bonds Versao portugues atualizado Revisado.pdf. 2.1.4, B.2, B.2

[13] DOLEV, D.; WARMUTH, M. **SIAM Journal on Computing**. Scheduling flat graphs, journal, v.14, n.3, p. 638–657, 1985. 3.3

[14] E. TARDOS, J. K. **Algorithm Design**. 2nd. ed., Pearson, 2005. 3.1, 3.1

[15] GRAHAM, R.; LAWLER, E.; LENSTRA, J. ; KAN, A. **Optimization and approximation in deterministic sequencing and scheduling: a survey**. In: P.L. Hammer, E. J.; Korte, B., editors, DISCRETE OPTIMIZATION II PROCEEDINGS OF THE ADVANCED RESEARCH INSTITUTE ON DISCRETE OPTIMIZATION AND SYSTEMS APPLICATIONS OF THE SYSTEMS SCIENCE PANEL OF NATO AND OF THE DISCRETE OPTIMIZATION SYMPOSIUM CO-SPONSORED BY IBM CANADA AND SIAM BANFF, AHA. AND VANCOUVER, volume 5 de **Annals of Discrete Mathematics**, p. 287 – 326. Elsevier, 1979. 2.2, 2.2.1

[16] HULL, J. **Options, Futures, and Other Derivatives**. 8th. ed., Prentice Hall, 2011. 2.1.1, 2.1.4

[17] JORION, P. **Financial Risk Manager**. 2nd. ed., John Wiley & Sons, 2003. 2.1.4

[18] JORION, P. **Value at Risk: The New Benchmark for Managing Financial Risk**. 3rd. ed., McGraww-Hill, 2006. 2.1.4

[19] JR., E. G. C.; GAREY, M. R. ; JOHNSON, D. S. **SIAM J. Comput.** An application of bin-packing to multiprocessor scheduling, journal, v.7, n.1, p. 1–17, 1978. 3.2

[20] KACEM, I. **Journal of Combinatorial Optimization**. Approximation algorithms for the makespan minimization with positive tails on a single machine with a fixed non-availability interval, journal, v.17, n.2, p. 117–133, 2009. 2.2, 2.2.2

[21] KELLERER, H. **IIE Transactions**. Algorithms for multiprocessor scheduling with machine release times, journal, v.30, n.11, p. 991–999, 1998. 3.3

[22] KITCHENHAM, B. A.; PFLEEGER, S. L.; PICKARD, L. M.; JONES, P. W.; HOAGLIN, D. C.; EMAM, K. E. ; ROSENBERG, J. **IEEE Trans. Softw. Eng.** Preliminary guidelines for empirical research in software engineering, journal, v.28, n.8, p. 721–734, Ago. 2002. 6

[23] LEE, C.-Y. **Discrete Applied Mathematics**. Parallel machines scheduling with nonsimultaneous machine available time, journal, v.30, n.1, p. 53 – 61, 1991. 3.3

[24] LEE, C.-Y. **Journal of Global Optimization**. Machine scheduling with an availability constraint, journal, v.9, n.3-4, p. 395–416, 1996. 1.1, 2.2.2, 3.3

[25] LEE, C.-Y. **European Journal of Operational Research**. Two-machine flowshop scheduling with availability constraints, journal, v.114, n.2, p. 420 – 429, 1999. 2.2, 2.2.2

[26] LEUNG, J.; KELLY, L. ; ANDERSON, J. H. **Handbook of Scheduling: Algorithms, Models, and Performance Analysis**. Boca Raton, FL, USA: CRC Press, Inc., 2004. 3.1

[27] **Managed extensibility framework**. http://mef.codeplex.com/, 2008. 5.2

[28] PINEDO, M. L. **Scheduling: Theory, Algorithms, and Systems**. 4th. ed., Springer Publishing Company, Incorporated, 2012. 2.2, 2.2.1, 2.2.2, 3.1, 3.2, 3.2, 3.2, 3.3

[29] PLUGGE, E.; HAWKINS, T. ; MEMBREY, P. **The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing**. 1st. ed., Berkely, CA, USA: Apress, 2010. 5.1.1

[30] SCHMIDT, G. **Scheduling with limited machine availability**. In: EUROPEAN JOURNAL OF OPERATIONAL RESEARCH, p. 1–15. Elsevier, 2000. 2.2, 2.2.1, 2.2.2, 3.3

[31] WANG, X.; CHENG, T. C. E. **Naval Research Logistics (NRL)**. Machine scheduling with an availability constraint and job delivery coordination, journal, v.54, n.1, p. 11–20, 2007. 3.3

[32] WU, C.-C.; LEE, W.-C. **Information Processing Letters**. Scheduling linear deteriorating jobs to minimize makespan with an availability constraint on a single machine, journal, v.87, n.2, p. 89 – 93, 2003. 3.3

A
Definições, siglas e abreviações

São listadas a seguir algumas definições, siglas e abreviações utilizadas nesta tese.

Abreviação	Significado
MtM	Marcação a mercado
BM&FBOVESPA	BM&FBOVESPA S.A. - Bolsa de Valores, Mercadorias e Futuros
Bacen	Banco Central do Brasil
Câmara	Câmara de Registro, Compensação e Liquidação de Operações de Derivativos da BM&FBOVESPA
Comitente	Investidor titular das operações realizadas e/ou registradas por sua conta e ordem nos mercados da BM & FBOVESPA com liquidação garantida pela Câmara
Entrega	Liquidação das obrigações decorrentes de uma Operação por meio da entrega, pela Câmara ou pelo Comitente vendedor, conforme o caso, dos ativos ou mercadorias negociados
Especificação	Procedimento por meio do qual são indicados o Comitente de uma Operação e o Membro de Compensação responsável por seu registro e liquidação
Inadimplemento	Descumprimento de obrigação, por um Membro de Compensação, Liquidante, Negociador, Intermediário por Conta ou Comitente, perante a Câmara ou perante os demais Participantes
Intermediário por Conta	Participante que opera por conta e ordem de terceiros, transmitindo ordens de negociação a um Negociador
Liquidação	Cumprimento, perante a Câmara ou perante os Membros de Compensação e os demais participantes, de obrigações decorrentes de uma ou mais Operações

Abreviação	**Significado**
Membro de Compensação	Participante ao qual é facultado registrar, compensar e liquidar Operações registradas nos sistemas registro, compensação e liquidação da Câmara
Negociador	Participante com acesso direto aos sistemas de negociação e de registro da BM&FBOVESPA, que recebe ordem e executa a Operação em pregão e/ou a registra nos sistemas de registro
Operação	Negócio realizado em qualquer dos pregões ou sistemas de negociação da BM&FBOVESPA e/ou registrado em seus sistemas, cuja liquidação se dá por meio do serviço de liquidação da Câmara
Posição	Saldo de contratos resultante das Operações de um Comitente
d_c	Abreviação para dias corridos
d_u	Abreviação para dias úteis
FPR	Abreviação para fator primitivo de risco
bps	Abreviação para pontos-base (100 pb = 1%)
PnL	Profit and Loss que pode ser entendida também como resultado ou retorno financeiro

B
Precificação de ativos

B.1
Ações

As ações são marcadas a mercado diariamente pelas cotações de fechamento do pregão da BM&F BOVESPA. Não havendo negociação no dia, é mantido o preço do último pregão em que houve negociação.

B.2
Títulos Públicos

Segundo (12, TES00), o investimento em títulos públicos é um investimento em renda fixa, onde entre a data de compra e de vencimento, o preço do título flutua em função das condições do mercado e das expectativas quanto ao comportamento das taxas de juros futuras. Uma redução nas taxas de juros de mercado em relação à taxa de compra do título provocará aumento no valor do título. Já um aumento nas taxas de juros proporciona o efeito contrário.

Os títulos tem diferentes características, podendo ser indexados a índices de preços, à Selic ou títulos prefixados. Além disto, podem fazer pagamentos (cupons) semestrais ou não, a Tabela B.1 mostra as principais diferenças na composição dos preços destes títulos.

Tabela B.1: Componentes do Preço

Título	LTN	NTN-F	NTN-B	NTN-C	LFT
Tipo	Pré	Pré	Pós	Pós	Pós
Taxa de juros	Nominal	Nominal	Real	Real	Prêmio
Indexador	Não tem	Não tem	IPCA	IGP-M	Selic
VNA	Não tem	Não tem	15-Jul-00	1-Jul-00	1-Jul-00
Cupom Anual	Não tem	10 %	6 %	6 %	Não tem

O preço do Título é o resultado da multiplicação da sua cotação vezes o VNA. Assim, o preço unitário do título é dado por $PU = (VP * VNA)/100$, onde a VP denota o valor presente do fluxo do título, descontado pela taxa de juros informada ou pelo prêmio de deságio e o VNA é o valor que corrige

o fluxo pelo indexador ao qual o papel é atrelado. Para títulos prefixados (LTN e NTN-F) o VNA não é corrigido por nenhum indexador, sendo sempre R$1.000,00.

Como exemplo, digamos que uma LTN com vencimento 01-01-07 estivesse a 440 *du* do vencimento e que a taxa de juros ao ano era de 19 % em 31-03-05. Sabendo que o valor de face deste título é 100,000, o valor presente deste título pode ser calculado por $VP = \frac{100}{(1+Taxa)(\frac{du}{252})}$, o que fará que o preço deste título valha 73.8061. Exemplos da precificação dos demais tipos de títulos podem ser encontrados em (12).

B.3 Futuros

Contratos futuros são aqueles onde as partes assumem compromisso de compra e venda de um ativo financeiro ou mercadoria, representadas por contratos padronizados para liquidação (física e/ou financeira) numa data futura. Como vimos na Seção 2.1 a BMF é a Câmara responsável pela garantia de liquidação destes contratos e utiliza o mecanismo das margens depositadas em garantia para evitar a acumulação de perdas decorrentes da falta de pagamento de ajustes diários negativos.

No entanto, estamos interessados no processo de formação destes preços e o site da Câmara (6) detalha a precificação de cada um dos contratos. Como exemplo, apresentaremos o modelo de precificação de um contrato de futuro de dólar dado pela equação B-1.

$$PU = S * (1 + r) * (1 + rc^{-1}) \tag{B-1}$$

Onde:

- S: Valor à vista da taxa de câmbio;
- r: Taxa de juro pré-fixada em moeda nacional para o prazo até o vencimento do contrato;
- rc: Taxa de juro pré-fixada na moeda estrangeira (cupom cambial limpo) para o prazo até o vencimento do contrato.

B.4 Ativos de Crédito

Nesta seção, apresentamos uma classe exótica em termos de precificação que não se encontra no manual de nenhuma instituição pública.

B.4.1 CCIs

As cédulas de crédito imobiliário são títulos de dívida emitidos por um credor para o parcelamento de um pagamento de natureza imobiliária. Geralmente esses títulos são indexados a inflação (IGPM [1], IPCA [2], INCC [3], etc) mais algum spread negociado no momento da emissão.

Apesar desde título possuir um modelo de marcação a mercado quando indexado por IGPM e IPCA, muitas vezes ele acaba sendo marcado na curva por falta de componentes primitivas que permitam a marcação a mercado quando o índice do título é INCC. Para a marcar esse ativo na curva, precisamos trazer o fluxo de pagamentos futuros a valor presente.

A equação B-4 apresenta o modelo utilizado para a marcação na curva das CCIs sem atualização monetária (ie. sem apropriação do índice associado ao título emitido).

$$Taxa_i = (1 + tx)^{\frac{du_i}{252}} \tag{B-2}$$

$$PVPmt_i = \sum_{i=1}^{n} \frac{Pmt_i}{Taxa_i} \tag{B-3}$$

$$NPV = \sum_{i=1}^{n} PVPmt_i \tag{B-4}$$

$$PU = NPV * VNAFactor \tag{B-5}$$

Onde:

- tx: TIR [4] do título.
- du_i: Quantidade de dias úteis entre a data desejada e a data de vencimento da parcela i.
- Pmt_i: Valor futuro da parcela i a ser paga.
- NPV: Valor presente do título.
- $VNAFactor$: Fator de atualização monetária calculado como pró-rata entre a data de início do título e a data desejada.

[1] Índice Geral de Preços do Mercado
[2] Índice Nacional de Preços ao Consumidor Amplo
[3] Índice Nacional de Custo da Construção
[4] Taxa Interna de Retorno