Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Programa de Pós-Graduação em Engenharia de Produção

Problema de movimentação do carro *tripper*: uma abordagem via programação dinâmica aproximada

Mayra Cristina Silva Santos

Número de Ordem PPGEP: M001

João Monlevade-MG, Maio de 2021

Mayra Cristina Silva Santos

Problema de movimentação do carro *tripper*: uma abordagem via programação dinâmica aproximada

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Produção da UFOP (Linha de Pesquisa: Modelagem de Sistemas Produtivos e Logísticos), como parte dos requisitos necessários para a obtenção do Título de Mestre em Engenharia de Produção.

Universidade Federal de Ouro Preto – UFOP

Instituto de Ciências Exatas e Aplicadas

Programa de Pós-Graduação em Engenharia de Produção

Orientador: Thiago Augusto de Oliveira Silva

Coorientador: Maurício Cardoso de Souza

João Monlevade-MG

Maio de 2021

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

S237p Santos, Mayra Cristina Silva.
Problema de movimentação do carro tripper: uma abordagem via programação dinâmica aproximada. [manuscrito] / Mayra Cristina Silva Santos. - 2021.
108 f.

Orientador: Prof. Dr. Thiago Augusto de Oliveira Silva.
Coorientador: Prof. Dr. Maurício Cardoso Souza.
Dissertação (Mestrado Acadêmico). Universidade Federal de Ouro Preto. Departamento de Engenharia de Produção. Programa de Pós-Graduação em Engenharia de Produção.

1. Otimização combinatória. 2. Correias transportadoras - Carro Tripper. 3. Programação dinâmica. I. Silva, Thiago Augusto de Oliveira. II. Souza, Maurício Cardoso. III. Universidade Federal de Ouro Preto. IV. Título.

CDU 658.5

Bibliotecário(a) Responsável: Maristela Sanches Lima Mesquita - CRB-1716

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE ENGENHARIA DE PRODUCAO - ICEA

FOLHA DE APROVAÇÃO

Mayra Cristina Silva Santos

Problema de movimentação do carro *tripper*: uma abordagem via programação dinâmica aproximada

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Mestre em Engenharia de Produção

Aprovada em 06 de junho de 2021

Membros da banca

Prof. Dr. Thiago Augusto de Oliveira Silva - Orientador (Universidade Federal de Ouro Preto)
Prof. Dr. Maurício Cardoso de Souza - coorientador (Universidade Federal de Minas Gerais)
Prof. Dr. Alexandre Xavier Martins- (Universidade Federal de Ouro Preto)
Prof. Dr. Sérgio Ricardo de Souza - (Centro Federal de Educação Tecnológica de Minas Gerais)

Prof. Dr. Thiago Augusto de Oliveira Silva, orientador do trabalho, aprovou a versão final e autorizou seu depósito no Repositório Institucional da UFOP em 09/03/2022

Documento assinado eletronicamente por **Thiago Augusto de Oliveira Silva**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 09/03/2022, às 17:19, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0290111** e o código CRC **EC158D47**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: (31)3808-0817 - www.ufop.br

Agradecimentos

A Deus, por iluminar meu caminho.

Aos meus pais, pelo apoio e incentivo em toda e qualquer jornada.

À minha irmã, por estar sempre presente.

Aos meus familiares e amigos, pelo companheirismo.

Aos professores Alexandre Xavier Martins, Maurício Cardoso de Souza e Thiago Augusto de Oliveira Silva por todos os conselhos, pela paciência e pela dedicação neste trabalho.

À CAPES, ao CNPq, à FAPEMIG e à UFOP pelo apoio ao desenvolvimento deste projeto.

Resumo

Devido à sua importância, o setor mineral é alvo de estudos constantes, visando aprimoramentos ao longo de sua cadeia produtiva. Nesse sentido, o presente trabalho aborda o problema de movimentação do carro *tripper*, um problema de sequenciamento que visa determinar os movimentos que o equipamento deve realizar para descarregar o minério sobre os silos. Foram propostos métodos de solução para o problema determinístico proposto por Caldas e Martins (2018) e para uma versão estocástica desenvolvida para representar a natureza dinâmica do problema. Para a realização dos testes, foram utilizadas adaptações de uma instância presente na literatura. A partir dos resultados obtidos, verifica-se que, tanto no problema determinístico quanto no estocástico, alguns métodos apresentaram resultados satisfatórios em relação ao tempo de execução e à performance do algoritmo, sendo a performance dependente da combinação de funções utilizada no método de aproximação de programação dinâmica. Ademais, ainda no que diz respeito ao método de aproximação de programação dinâmica, o desempenho do modelo estocástico também se mostrou dependente do estado inicial utilizado e da realização do treinamento para cada novo estado.

Palavras-chaves: otimização combinatória. carro *tripper*. programação dinâmica.

Abstract

Due to its importance, the mineral sector is subject to constant studies, aiming at improvements throughout its supply chain. In this regard, this work addresses the movement problem of the tripper car, a scheduling problem that aims to determine the movements that the equipment must carry out to unload the ore on the silos. Solution methods were recommended for the deterministic problem proposed by Caldas e Martins (2018) and for a stochastic version developed to represent the dynamic nature of the problem. To perform the tests, adaptations of an instance found in the literature were used. From the results obtained, in both deterministic and stochastic problem, some methods displayed satisfactory results in relation to the execution time and the performance of the algorithm, where the performance is dependent on the combination of the functions used in the approximate dynamic programming method. In addition, with regard to the dynamic programming approximation method, the performance of the stochastic model also proved to be dependent on the initial state used and the training conducted for each new state.

Keywords: combinatorial optimization. tripper car. dynamic programming.

Lista de ilustrações

Lista de tabelas

Sumário

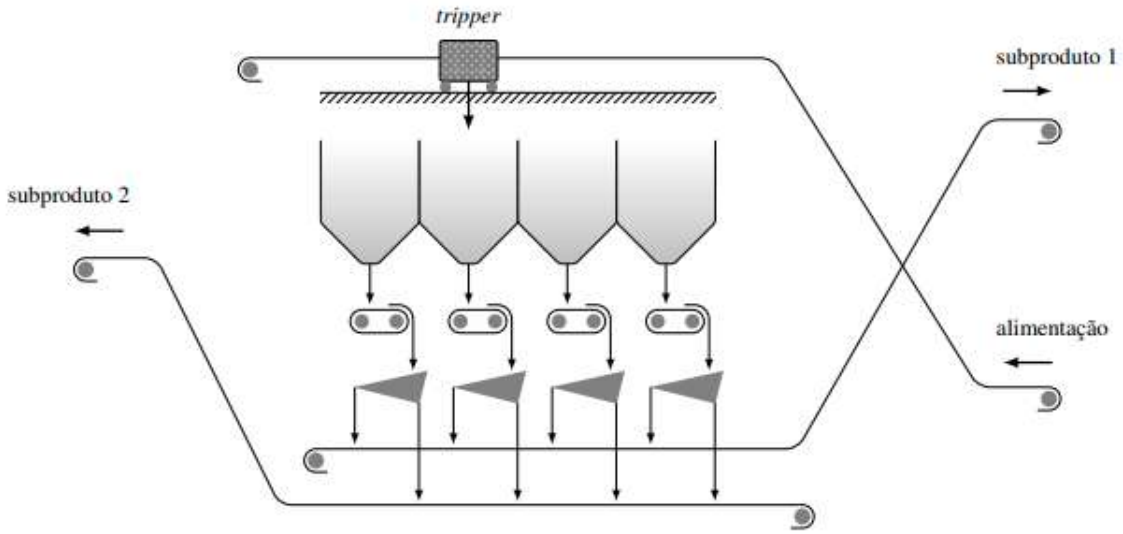
1 Introdução

A mineração possui extrema relevância no cenário econômico nacional e global, sendo uma das principais atividades responsáveis pelo abastecimento de matérias-primas a diversos setores, como civil e industrial, e pela geração de emprego e renda por meio da instalação de indústrias. Segundo o MME (2020), a mineração é responsável por 1,4% do Produto Interno Bruto (PIB) brasileiro e 12% das exportações brasileiras. Considerando que o minério é uma *commodity* e o mercado global é competitivo, estratégias que tornem seus processos mais eficientes, elevem a produtividade e reduzam os custos operacionais são necessárias para a sobrevivência das mineradoras.

Dentre as possibilidades de otimização do processo de beneficiamento do minério de ferro, que consiste basicamente em realizar operações no minério desde a sua extração para transformá-lo em um produto final comercial (WILLS; FINCH, 2015), destaca-se a aplicação de técnicas de controle em sua estocagem. Ainda de acordo com os mesmos autores, o estoque de minério é realizado em silos quando é necessária a formação de estoques intermediários entre as etapas do processo de beneficiamento.

Durante o processo de estocagem, o minério é depositado nos silos pelo carro *tripper*, um equipamento móvel que recebe o minério proveniente da correia transportadora e o distribui ao longo dos silos (NÚÑEZ; SOLEDAD, 2013). A Figura 1, extraída de Caldas (2018), apresenta o fluxograma do processo de peneiramento de minério, em que o estoque de minério é realizado por meio do *tripper*. Como se pode observar, o *tripper*, alimentado pela correia de alimentação, se locomove ao longo do trilho distruibuindo o minério entre os quatro silos. Cada silo, por sua vez, possui sob ele um alimentador e uma peneira, representada pelo triângulo cinza. O minério, após ser retirado dos silos pelos alimentadores e ser peneirado, é então transportado pelas correias transportadoras subproduto1 e subproduto2 para as linhas de processamento subsequentes.

Figura 1 – Processo de peneiramento de minério granulado



Fonte: Caldas (2018)

Logo, o problema de armazenagem de minério pode ser abordado sob a perspectiva da movimentação do *tripper*, cujo objetivo consiste na determinação dos movimentos a serem realizados pelo equipamento, uma vez que seu posicionamento define qual silo será alimentado. Como na maioria das vezes o estoque dos silos se encontra desbalanceado, a tomada de decisão deve ser realizada visando atingir o equilíbrio dos níveis dos silos ao longo do tempo.

De forma geral, a escolha dos movimentos que o *tripper* deve realizar é feita por um operador humano ou por meio de uma programação simples. Buscando-se aprimorar a eficiência do processo, torna-se interessante que sejam implementadas novas tecnologias para que a tomada de decisão seja realizada de forma autônoma com o objetivo de maximizar o desempenho da planta.

Nesse sentido, Caldas e Martins (2018), Pedrosa (2019) e Morais et al. (2020) estudaram o problema de otimização da estocagem de minério com o objetivo de determinar o posicionamento do *tripper* sobre os silos ao longo do tempo, de forma a manter os níveis de minério armazenado dentro de limites pré-estabelecidos. No entanto, Caldas e Martins (2018) e Pedrosa (2019) encontraram como principal barreira o alto tempo de execução necessário para que a solução seja encontrada, inviabilizando a aplicação de seus modelos em aplicações reais. Já os resultados encontrados por Morais et al. (2020) foram obtidos em um tempo de execução satisfatório, entretanto, o comportamento do nível dos silos não se enquadrou totalmente dentro do esperado, mantendo uma certa variabilidade e provocando penalizações por extrapolação dos limites.

Visando solucionar as lacunas identificadas nos trabalhos anteriores, e considerando-se que o problema se assemelha a problemas de sequenciamento de máquinas e *robot*

scheduling, considerados NP-completos, torna-se interessante que seja estudado através de outras abordagens com o objetivo de encontrar uma solução que seja capaz de sanar as lacunas existentes e resolver o problema para instâncias reais.

Sendo assim, o presente trabalho se propõe a desenvolver métodos de resolução para dois problemas de sequenciamento da movimentação do *tripper*, sendo o primeiro o problema determinístico definido por Caldas e Martins (2018), e o segundo um problema estocástico proposto, criado com o objetivo de representar a dinâmica de uma planta de beneficiamento real por meio da incorporação de incertezas no processo e da tomada de uma nova decisão, a duração da alimentação do silo escolhido.

Para tanto, o primeiro problema foi modelado como um Processo de Decisão de Markov, e foi desenvolvido um algoritmo de aproximação de programação dinâmica baseado no Algoritmo de Iteração de Política. Além disso, foram criadas funções indicadoras visando uma melhor representação do estado do problema. Já para o problema estocástico, foram desenvolvidos um método para cálculo do tempo de alimentação do silo, um método de programação linear inteira mista, um método que determina a movimentação do *tripper* de acordo com um zigue-zague, e um método de aproximação de programação dinâmica que utilizada o Algoritmo de Iteração de Política e funções indicadoras. Os resultados obtidos mostraram que, exceto pelo método zigue zague, os métodos desenvolvidos foram eficientes na resolução dos problemas.

1.1 Motivação

A motivação deste trabalho partiu da pesquisa realizada por Caldas e Martins (2018), que iniciou o estudo da otimização em plantas de beneficiamento de minério sob a perspectiva da movimentação do carro *tripper*. Além de sua relevância, dada a escassez de estudos na literatura a respeito do tema, a pesquisa ainda apresenta um *gap*, proporcionando que novas frentes de trabalho sejam exploradas.

Além disso, trata de um problema frequentemente encontrado em plantas de beneficiamento mineral, e devido à importância do setor na economia mundial e a busca constante pelo seu crescimento, torna-se interessante o seu estudo para que a otimização de seus processos possa ser realizada.

1.2 Objetivos

1.2.1 Objetivos Gerais

Desenvolver métodos e algoritmos que sejam capazes de gerar novas políticas para a resolução do problema de movimentação do *tripper* proposto por Caldas e Martins (2018)

e para a resolução de um novo modelo estocástico desenvolvido para o problema.

1.2.2 Objetivos Específicos

- Compreender o problema de movimentação do carro *tripper* e os modelos já desenvolvidos;
- Modificar o modelo proposto por Morais et al. (2020), visando resolvê-lo através da aproximação da programação dinâmica;
- Propor um novo modelo estocástico para o problema;
- Desenvolver três métodos de resolução para o problema estocástico;
- Desenvolver políticas para solucionar o problema;
- Realizar testes computacionais e verificar a eficiência dos métodos propostos.

1.3 Justificativa

No Brasil, o setor mineral representa um dos setores básicos da economia. Entretanto, no decorrer dos anos, tem passado por altos e baixos, apresentando um crescimento de 2001 a 2013 e uma desaceleração a partir de 2014, impulsionada pela queda mundial da atividade e dos preços das *commodities* minerais e também pela crise econômica e política enfrentada pelo Brasil. Já em 2020, de acordo com o IBRAM (2020), o setor apresentou um crescimento de 36% em relação ao ano anterior, e criou-se a expectativa de que a pesquisa mineral seja impulsionada, auxiliando na retomada do crescimento. Dessa forma, buscando-se a recuperação do setor, faz-se necessário que se busque realizar aprimoramentos ao longo de toda sua cadeia, justificando assim a realização deste trabalho.

Do ponto de vista científico, o presente trabalho se justifica pela sua complexidade, devido à sua natureza dinâmica. Além do modelo determinístico possuir uma restrição na movimentação do *tripper*, dificultando o processo de homogeneização dos níveis dos silos, e do modelo estocástico considerar incertezas durante o processo, o problema possui um amplo espaço de estados e ações, o que dificulta a sua resolução, podendo até mesmo inviabilizá-la de acordo com o método de resolução escolhido.

Essa é uma das principais causas que fazem com que a programação dinâmica seja pouco utilizada, conhecida como maldição da dimensão, que segundo Powell (2007) consiste no rápido crescimento do tamanho do problema, dificultando sua resolução. O que significa, na prática, que as necessidades computacionais necessárias para a resolução do problema crescem exponencialmente à medida que o número de dimensões cresce. Para contornar tal impasse, faz-se necessário o desenvolvimento de métodos que possibilitem

a resolução de problemas de grande porte por programação dinâmica, conhecidos como métodos de aproximação.

1.4 Organização do Trabalho

Este trabalho está organizado da seguinte forma: após o presente capítulo realizar a apresentação da visão geral do problema e de seus objetivos e justicativa, o Capítulo 2 traz uma revisão de literatura sobre os problemas de *Single Machine Scheduling*, *Robot Scheduling*, *Dynamic Scheduling* e sobre o problema do *tripper*, em que além de suas definições, foram apresentados alguns trabalhos relevantes acerca dos temas.

O Capítulo 3 apresenta o referencial teórico sobre programação dinâmica, focando nos fundamentos necessários ao desenvolvimento e ao entedimento do trabalho.

Já no Capítulo 4, o problema determinístico de movimentação do *tripper* é descrito, seguido pela apresentação de seu modelo e dos métodos utilizados em sua resolução.

No Capítulo 5 são apresentados os resultados obtidos pela implementação dos métodos de solução propostos para resolver o problema determinístico de sequenciamento do *tripper*. De posse dos resultados, foi realizada uma análise comparando-os com os resultados encontrados na literatura.

O Capítulo 6 apresenta o modelo estocástico proposto para o problema, por meio da descrição do problema e dos métodos de solução desenvolvidos, baseados em programação linear inteira mista, aproximação de programação dinâmica e em um algoritmo que simula um zigue-zague.

O Capítulo 7 apresenta os resultados obtidos pelos testes realizados para todos os métodos de solução. Em seguida, foi realizada uma comparação dos resultados para que a validação dos métodos pudesse ser realizada.

Por fim, são apresentadas as conclusões obtidas pela realização do presente trabalho e as sugestões de trabalhos futuros.

2 Revisão da Literatura

2.1 *Single Machine Scheduling*

No âmbito da pesquisa operacional, o problema de programação ou sequenciamento da produção é vastamente abordado na literatura e aplicado no ambiente industrial. Além de seu próprio valor intrínseco, Gupta e Kyparisis (1987) destacam que sua importância está diretamente relacionada à sua flexibilidade, já que pode ser aplicado tanto em problemas mais complexos, como os que envolvem gargalos, como também é capaz de simplificar alguns problemas, como os que consideram uma linha de produção como uma única máquina.

Segundo Pinedo (2012), um problema de sequenciamento consiste em alocar recursos para o processamento de um conjunto de tarefas ao longo do tempo, satisfazendo um conjunto de restrições temporais e de recursos, e visando otimizar um ou mais objetivos. Os recursos são os bens ou serviços, que podem ter disponibilidade limitada, como por exemplo equipamentos, matérias-primas, mão-de-obra, dentre outros.

Os problemas de sequenciamento podem ter muitas variações, baseadas em características básicas dos problemas como ambiente de produção, objetivo e restrições. No que diz respeito aos ambientes de produção, os trabalhos de Johnson (1954) e Wagner (1959) foram pioneiros ao abordar o problema de sequenciamento *flow shop*, que consiste em um conjunto de tarefas que deve ser processado por um conjunto de máquinas na mesma ordem de processamento, enquanto que Giffler e Thompson (1960) estudaram o problema *job shop*, caracterizado pela possibilidade das tarefas serem executadas sem uma ordem pré-estabelecida.

Os objetivos mais comuns são minimizar o *makespan* (tempo total para a conclusão das tarefas), o tempo de fluxo, os atrasos e as antecipações, podendo ainda estar sujeito a restrições como penalidades por antecipação ou atraso, preempção, etc., como mostram os trabalhos de Allahverdi et al. (2008), Gordon, Proth e Chu (2002), Lei (2009), e Koulamas (2010), que apresentam uma revisão sobre problemas de sequenciamento envolvendo restrições de configuração de máquinas, de data de vencimento das tarefas, problemas multiobjetivo e penalização pelo atraso total, respectivamente.

Estudos mais recentes seguem a tendência de explorar problemas de sequenciamento em aplicações reais, motivados pelo *gap* entre a teoria e a prática. Nesse sentido, há o notório interesse em problemas de sequenciamento estocásticos, em que há a presença de incertezas e os parâmetros não são conhecidos a priori, como mostra o compilado de técnicas de sequenciamento sob incertezas realizado por Li e Ierapetritou (2008). Ademais,

vem sendo estudadas novas abordagens para a busca da solução ótima em tempo viável, como aplicações de inteligência artificial, como mostram Akyol e Bayhan (2007), Metaxiotis, Askounis e Psarras (2002) e Çaliş e Bulkan (2015).

A maioria dos problemas de sequenciamento pertencem à classe NP, e portanto, são resolvidos em tempo exponencial, possuindo elevado custo computacional. Visando encontrar soluções para o problema em tempo aceitável, frequentemente propõe-se o uso de técnicas para a resolução dos problemas, como heurísticas e metaheurísticas, em detrimento da utilização de métodos exatos, como programação linear inteira mista, devido ao elevado tempo de execução.

2.1.1 *Robot Scheduling*

Como dito anteriormente, os problemas de sequenciamento da produção possuem ampla aplicabilidade, como na produção industrial, apresentando variações conforme algumas de suas características. Dentro dessas variações, destaca-se uma generalização do problema de sequenciamento clássico, em que as tarefas são transportadas entre as máquinas por um ou mais robô(s) móvel(is), problema conhecido na literatura como *Robot Scheduling*.

Hurink e Knust (2002) definem o problema basicamente como um conjunto de tarefas, formadas por uma cadeia de operações, que devem ser processadas em um conjunto de máquinas, em que um robô móvel é responsável por realizar o transporte das tarefas entre as máquinas. Segundo Zacharia e Aspragathos (2005), o problema é considerado uma generalização do problema do Caixeiro Viajante, sendo que o objetivo mais comum é encontrar uma rota ótima que minimize o *makespan*, e as restrições mais comuns são as de capacidade, tanto da máquina quanto do robô, e as de movimentação do robô.

O trabalho de Alatartsev, Stellmacher e Ortmeier (2015) apresenta uma visão geral sobre abordagens utilizadas no sequenciamento de tarefas de um robô, como o planejamento de rotas livres de colisões, casos em que há mais de um robô, etc. Já Raja e Pugazhenthi (2012) apresentam uma visão geral sobre o planejamento de rotas de um robô tanto em ambientes *on-line* como *off-line* por meio de abordagens clássicas e evolutivas.

Algumas aplicações do *Robot Scheduling* valem ser ressaltadas, como o estudo de Dang et al. (2014) que visa resolver o problema de sequenciamento de um robô em uma indústria de fabricaçao de bombas, em que o robô transporta pequenos transportadores de carga contendo peças (SLCs) de uma área de armazenagem central até os alimentadores.

He et al. (2015) estudaram o problema de sequenciamento de peças com auxílio de um robô em um sistema de manufatura flexível, e desenvolveram uma abordagem baseada em conjunto de segmentos para resolver o problema. Já He, Stecke e Smith (2016) utilizaram o modelo de He et al. (2015), e aplicaram regras de scheduling às máquinas e a

um robô para a resolução do problema em ambiente de customização / personalização em massa.

Os trabalhos de Caumond et al. (2009) e Hurink e Knust (2005) abordam o problema em que as tarefas precisam ser transportadas entre as máquinas por um robô de transporte considerando-se os movimentos realizados quando o robô não está transportando alguma tarefa. Enquanto o primeiro modela o problema por programação linear inteira mista, o segundo apresenta um algortimo de busca local para a resolução do problema.

A Tabela 1 apresenta o objetivo, a decisão e as restrições dos trabalhos da revisão de literatura acima mencionados.

Os problemas de *Robot Scheduling* são pertences à classe NP-difícil e possuem ampla complexidade, já que dois problemas NP-difíceis são considerados simultaneamente: o problema de sequenciamento de máquinas e o de roteirização de robôs.

Tabela 1 – Principais características dos artigos abordados sobre *Robot Scheduling*

Referência	**Objetivo**	**Decisão**	**Restrições**
(DANG et al., 2014)	Minimizar o tempo total de viagem do robô	Se o robô, na rota r, viaja do local da tarefa i com subtarefa k para o local da tarefa j com subtarefa l	O horário de início de qualquer subtarefa de uma tarefa deve satisfazer à janela de tempo dessa subtarefa; No estágio inicial, o robô deve partir do armazém; Não são permitidas sub-rotas entre subtarefas; Cada subtarefa deve ser concluída exatamente uma vez; A capacidade do robô deve ser respeitada; Uma subtarefa de uma tarefa deve ser atribuída à uma rota.
(HE et al., 2015)	Maximizar o total de peças produzidas	Se a peça α_i é processada ou não no periodo t	A precedência deve ser respeitada; Uma peça que acabou de chegar não pode ser processada imediatamente; O tempo de fluxo é definido pela diferença entre o horário de início e o de conclusão da tarefa; A soma de todos os tempos de processamento de uma peça e dos tempos de movimentação do robô deve ser menor ou igual ao tempo de fluxo parcial.
(HE; STECKE; SMITH, 2016)	Minimizar o tempo total de fluxo e maximizar o total de peças e a utilização do robô	Se a peça α_i é processada ou não no periodo t	A precedência deve ser respeitada; Uma peça que acabou de chegar não pode ser processada imediatamente; O tempo de fluxo é definido pela diferença entre o horário de início e o de conclusão da tarefa; A soma de todos os tempos de processamento de uma peça e dos tempos de movimentação do robô deve ser menor ou igual ao tempo de fluxo parcial. As máquinas possuem capacidade limitada
(CAUMOND et al., 2009)	Minimizar o makespan	Se a tarefa i é processada ou não no instante	Número máximo de tarefas; O robô pode realizar viagens vazias, ou seja, sem estar transportando nenhuma tarefa; A máquina pode executar apenas uma operação de uma tarefa por vez; O robô só pode fazer uma viagem por vez, e transportando apenas uma tarefa; As máquinas possuem capacidade limitada; Nenhuma viagem vazia pode começar antes que uma tarefa chegue ao buffer de saída da máquina; As operações de uma tarefa seguem uma ordem de precedência.
(HURINK; KNUST, 2005)	Minimizar o makespan	Se a tarefa i é processada ou não no instante	A ordem de precedência das operações de cada tarefa deve ser respeitada; cada operação deve ser processada em uma máquina; Não é permitida preempção; Cada máquina pode processar no máximo uma operação por vez; O robô só pode transportar uma tarefa por vez; O tempo de transporte de uma tarefa entre duas máquinas é considerado; O tempo que robô se move sem estar transportando uma tarefa é considerado;

2.1.2 *Dynamic Scheduling*

Os problemas de sequenciamento clássicos abordam a programação da produção em ambientes estáticos. Entretanto, como ambientes reais de manufatura são dinâmicos, em que a ocorrência de eventos inesperados resulta em alterações no planejamento, surge o sequenciamento dinâmico para tratar o problema de programação da produção na presença de eventos em tempo real.

Um problema de sequenciamento dinâmico realiza a programação da produção a partir da chegada de tarefas urgentes. Com base nas novas informações, é realizada uma revisão no planejamento atual e, se necessário, são feitas alterações. Segundo Ouelhadj e Petrovic (2009), o problema de reagendamento consiste em decidir quando e como reagir às informações, ou seja, quais estratégias seguir e quando seguir, e verificar os impactos da estratégia no planejamento original.

Suresh e Chaudhuri (1993) apresentaram uma revisão dos estudos publicados sobre sequenciamento dinâmico, classificando os trabalhos em três abordagens de sequenciamento: convencional, baseada em conhecimento e solução distribuída. Já o trabalho de Ouelhadj e Petrovic (2009), além de apresentar o problema de sequenciamento dinâmico e categorizar os eventos em tempo real, apresenta uma visão geral sobre abordagens de reagendamento e sobre as diferentes técnicas utilizadas para a resolução do problema.

Belhe e Kusiak (1997) abordam o problema de sequenciamento dinâmico de atividades de processos simultâneos de design, sujeitos a restrições de precedência e disponibilidade de recursos, visando maximizar a recompensa total. O modelo matemático é formulado como um problema da mochila multidimensional, e foram propostos como métodos de solução as heurísticas gulosa e *beam search*. A partir de testes computacionais, pode-se verificar que o método *beam search* alcançou soluções quase ótimas.

Cowling e Johansson (2002) apresentam um *framework* para a utilização das informações em tempo real no processo de tomada de decisão em problemas de sequenciamento, baseado na melhoria no objetivo e nas perturbações causadas pela revisão do planejamento. Para a ilustração da utilização do *framework*, considerou-se um sistema com uma única máquina, e estudou-se como o modelo poderia ser utilizado em um sistema complexo de lingotamento contínuo de aço.

Fattahi e Fallahi (2010) abordam o sequenciamento dinâmico em um ambiente *job shop* flexível por meio de um modelo matemático multiobjetivo, visando minimizar o *makespan* e as penalizações causadas pelas informações em tempo real. Foi desenvolvido um algoritmo genético para a resolução do problema, que mostrou-se, por meio de testes, capaz de encontrar a solução ótima para problemas pequenos, e soluções quase ótimas para problemas de tamanho médio.

No trabalho desenvolvido por Chryssolouris e Subramaniam (2001), foi abordado

o problema de sequenciamento dinâmico multiobjetivo em um ambiente *job shop*, cujos objetivos consistiam em minimizar o atraso e o custo das tarefas. Para resolução do problema, foi desenvolvido um algoritmo genético, e os resultados obtidos foram comparados aos resutados encontrados pelas regras de despacho *shortest processing time (SPT), first-in-first-out(FIFO), last-in-first-out(LIFO)*, sendo que os resultados obtidos pelo algoritmo genético proposto foram significativamente superiores.

2.2 Problema do *tripper*

O problema de movimentação do *tripper* consiste em determinar o posicionamento que o *tripper* deve assumir enquanto despeja material sobre os silos ao longo do tempo, a partir das informações disponíveis no estado atual e de previsões de informações futuras, a fim de otimizar um ou mais objetivos (CALDAS, 2018). Apesar de sua importância em plantas de beneficiamento de minério, o problema ainda é pouco abordado pela literatura, e os trabalhos encontrados acerca do tema são sintetizados a seguir.

O trabalho de Caldas e Martins (2018) modelou e solucionou o problema sob a forma de programação linear inteira mista, visando atingir a estabilidade dos níveis do silo ao longo do tempo. Além disso, em Caldas (2018) foram desenvolvidos dois métodos metaheurísticos, *GRASP* e *Simulated Annealing*, para a resolução do problema e analisou-se a viabilidade de implantação dos métodos desenvolvidos em sistemas reais.

Já Pedrosa (2019) realizou modificações no modelo de Caldas e Martins (2018) e desenvolveu uma nova política de movimentação para o *tripper*, utilizando-se de programação linear inteira mista. Os resultados obtidos foram comparados com os resultados encontrados por Caldas e Martins (2018), a fim de avaliar a viabilidade da nova política proposta.

Ainda baseado no trabalho de Caldas e Martins (2018), Morais (2019) redefiniu o problema como um processo de decisão de Markov, construiu um modelo em Programação Dinâmica e desenvolveu três regras de despacho para a resolução do problema. Foram utilizadas algumas instâncias criadas por Pedrosa (2019) para a realização de testes e validação do modelo.

No trabalho de Morais et al. (2020), os modelos de Morais (2019) e Caldas e Martins (2018) foram executados para todas as instâncias de Pedrosa (2019), e a partir dos resultados obtidos, realizou-se a comparação da aplicabilidade dos dois métodos.

No estudo realizado por Karelovic, Putz e Cipriano (2015) foi desenvolvido um sistema de controle preditivo baseado em modelo híbrido (HMPC, do inglês *Model Predictive Controller*) para minimizar os movimentos feitos pelo *tripper* e evitar o transbordo dos silos. O HPMC desenvolvido controla toda a planta de britagem, e por tal motivo, não há

uma descrição detalhada do controle do *tripper* e de sua implementação.

Viana (2018) desenvolveu um sistema especialista baseado em PLC (*Programmable Logic Controller*) para a resolução do problema do *tripper*, buscando uniformizar os níveis dos silos e minimizar os movimentos do *tripper*. A modelagem foi realizada por formulações matemáticas, e a implementação em linguagem ladder. Os resultados mostraram uma melhoria no desempenho ao se comparar ao *tripper* operado de forma manual.

O trabalho de Albuquerque et al. (2019) visa resolver o problema do controle do nível médio em silos por meio de um controlador baseado em regras, a partir da velocidade do carro *tripper* e do número de silos em uso. O controlador proposto, além de otimizar a distribuição de minério nos silos, evita que algum silo transborde por acúmulo de material ou que falte material em um silo.

Por fim, Santos et al. (2020) propuseram um algoritmo baseado em *Simheuristic* para realizar o dimensionamento de um circuito de britagem de minério, visando otimizar a quantidade de equipamentos ativos a fim de maximizar a produção. Apesar de ser um equipamento pertencente ao circuito, o *tripper* não foi considerado na simulação, pois assumiu-se que a distribuição de minério nos silos foi realizada de forma homogênea. A partir do estudo realizado, pode-se concluir que o algoritmo proposto foi eficiente tanto em maximizar a produção quanto em reduzir o consumo energético.

O problema do *tripper* pode ser considerado como um problema de sequenciamento da movimentação do carro *tripper*, e é análogo à um problema *Robot Scheduling*, em que o *tripper* é considerado como o robô, e ao invés da tomada de decisão ser baseada na sequência em que o robô deve visitar as máquinas, ela se baseia na sequência de movimentos que o carro *tripper* deve realizar para visitar os silos.

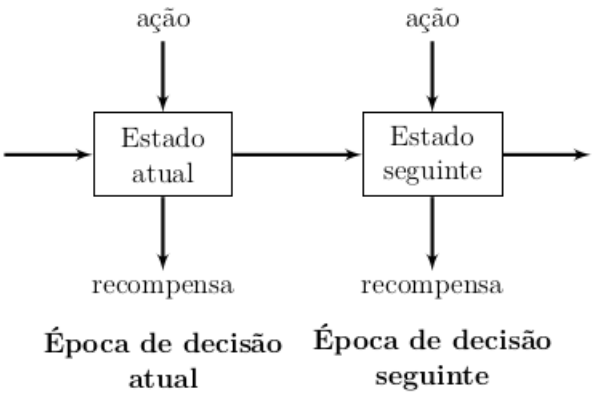
Quando são consideradas informações em tempo real da planta, como por exemplo o defeito em alguma máquina que receberia o minério armazenado em algum silo, o problema se torna análogo à um problema de sequenciamento dinâmico, em que as decisões de planejamento são afetadas e devem ser replanejadas baseadas nas novas informações em tempo real, à medida que chegam.

Dessa forma, fatores como a complexidade do problema e sua aplicabilidade em sistemas reais, visto que pode ser encontrado em diversas plantas de beneficiamento de minério, fundamentam o seu estudo.

3 Fundamentação Teórica

Segundo Arenales et al. (2007), pode-se definir pesquisa operacional como o desenvolvimento de métodos científicos para auxiliar na tomada de decisões. Nos problemas em que as decisões são feitas ao longo do tempo a partir de tomadas de decisão anteriores, chamados de problemas de decisão sequencial, representados de forma simbólica pela Figura 2, encontrar uma solução pode ser uma tarefa difícil.

Figura 2 – Representação de um problema de decisão sequencial



Fonte: Puterman (2014)

A programação dinâmica se apresenta como uma técnica clássica de resolução de problemas de tomada de decisão sequencial. Desenvolvida por Bellman (1957), consiste em transformar um problema de decisão sequencial em uma série de subproblemas. A resolução é feita utilizando-se da recursividade, em que os resultados obtidos são armazenados para evitar recálculos dos subproblemas em comum, e a solução do problema é definida por meio da solução dos subproblemas.

Powell (2007) apresenta os principais elementos necessários à modelagem de um sistema dinâmico. São eles:

- Estágio: instante de tempo em que as decisões são tomadas;
- Estado: representa todas as informações necessárias para descrever o sistema, auxiliando na tomada de decisão;
- Ação: ação tomada pelo tomador de decisões ao se observar o estado do sistema a cada estágio, representando o controle processo. São as variáveis de decisão do problema;

- Informações exógenas: são os dados que se tornam conhecidos a cada estágio do problema, representando as incertezas do processo;
- Função de transição: responsável por determinar como o sistema evolui de um estágio atual para o estágio posterior, a partir das informações do estado atual, da ação tomada e das informações exógenas;
- Função objetivo: responsável por especificar os resultados das escolhas de uma ação a cada estágio;
- Política: regra ou função que determina, a partir de um estado, a ação a ser tomada. As políticas podem ser agrupadas em quatro categorias, sendo elas:

  i) Políticas Míopes: escolhem a próxima ação, visando otimizar o objetivo, sem considerar quaisquer tipos de informações futuras;

  ii) Políticas Antecipadas*(Lookahead)* : as decisões são tomadas levando em consideração estimativas de informações futuras e ações futuras;

  iii) Políticas Baseadas em Funções de Aproximação *(Policy Function Approximations)*: consistem na aplicação de uma regra que retorna a ação, a partir de um estado, sem se recorrer a qualquer forma de otimização ou a previsão de informações futuras;

  iv) Políticas Baseadas em Aproximação de Valor *(Value Function Approximations)*: para a tomada de decisão no estado atual, consideram o impacto dessa decisão em estados futuros, por meio de uma aproximação do valor de se estar no estado futuro.

Sendo assim, a cada estágio k, uma ação a_k é tomada, de acordo com a política π escolhida, levando o sistema de um estado s_k à um novo estado s_{k+1} conforme a função de transição $s_{k+1} = f(s_k, a_k, \omega_{k+1})$, em que ω representa as incertezas do processo. Esse processo vai sendo executado ao longo do horizonte de tempo, calculando-se o custo ou recompensa $C_k(s_k, a_k)$ de cada ação tomada. Portanto, o objetivo é encontrar uma política ótima, ou seja, a sequência de ações ótimas que minimizam o custo ou maximizam o retorno do problema.

A programação dinâmica desponta como uma técnica de resolução eficiente pois trata de problemas de várias naturezas, como discretos, contínuos, determínísticos, estocásticos, e de horizonte finito e infinito. De acordo com Powell (2007), a programação dinâmica aborda os problemas discretos sob a perspectiva dos processos de decisão de Markov. Nos *Processos de Decisão de Markov*, o próximo estado e a recompensa resultante da ação tomada dependem apenas do estado e da ação atuais, uma vez que o estado atual armazena as informações relevantes ao longo do tempo (PUTERMAN, 2014).

A programação dinâmica é fundamentada no *Princípio da Otimalidade de Bellman*, que diz que em uma política ótima, as decisões futuras devem constituir uma política ótima independentemente das decisões tomadas anteriormente (BELLMAN, 1957). A utilização desse princípio implica em uma forma de se encontrar a solução de um problema de programação dinâmica, que consiste em resolvê-lo recursivamente do estágio final até se chegar ao estado atual, utilizando a saída do estágio atual como a entrada do próximo estágio.

Para fins de ilustração, considere um problema determinístico de horizonte infinito definido em Powell (2007), em que a ação a ser tomada é definida por

$$a_k^*(S_k) = arg \max_{a_k \in A_k} \left(C_k\left(S_k, a_k\right) + \gamma J_{k+1}\left(S_{k+1}\right)\right) \tag{3.1}$$

em que "arg max"indica que a ação a ser escolhida deve ser aquela que apresenta o maior resultado obtido pela resolução da equação, γ é o fator de desconto, utilizado em problemas de horizonte infinito para auxiliar na convergência, e o valor de se estar no estado S_k, conhecido como função de valor, é dado por:

$$\begin{aligned} J_k(S_k) &= \max_{a_k \in A_k} \left(C_k\left(S_k, a_k\right) + \gamma J_{k+1}\left(S_{k+1}\left(S_k, a_k\right)\right)\right) \\ &= C_k\left(S_k, a_k^*\left(\left(S_k\right)\right) + \gamma J_{k+1}\left(S_{k+1}\left(S_k, a_k^*(S_k)\right)\right)\right) \end{aligned} \tag{3.2}$$

Considerando um problema estocástico, em que $\mathbb{E}$ representa o valor esperado da função de valor, a equação pode ser escrita como:

$$J_k(S_k) = \max_{a_k \in A_k} \left(C_k\left(S_k, a_k\right) + \gamma \mathbb{E}\left\{J_{k+1}\left(S_{k+1}\left(S_k, a_k, W_{k+1}\right)\right) | S_k\right\}\right) \tag{3.3}$$

A equação 3.3 é conhecida como a equação de Bellman. Ela expressa a função valor, ou seja, o valor de se estar em um estado S_k, como a soma da recompensa do estado atual com a expectativa das recompensas a partir do estado atual até o final. Sendo assim, a função valor avalia o valor de uma política por meio do valor esperado para a soma das recompensas.

Como visto anteriormente, para se obter a política ótima do sistema é necessário calcular a função valor ótima. Entretanto, na maioria dos casos, métodos analíticos não são suficientes, e é necessária a utilização de algoritmos aproximados.

Em problemas de horizonte finito, assume-se que a função valor é dada ou o próprio valor é conhecido, e a resolução do problema pode ser obtida calculando-se o valor de se estar em cada estado possível do último estágio para o primeiro, método conhecido como *Backward Dynamic Programming Algorithm*. Já em problemas de horizonte infinito,

considera-se que o problema seja estacionário, ou seja, seus parâmetros não variam ao longo do tempo, ou que os parâmetros variem em ciclos, e há alguns algoritmos para sua resolução.

Como o foco deste trabalho são os problemas de horizonte infinito, a seguir serão abordados os dois algoritmos mais utilizados na resolução de problemas estacionários de horizontes infinitos: Algoritmo de Iteração de Valor e Algoritmo de Iteração de Política, sendo que o primeiro consiste na busca pela função valor ótima, e então a utiliza para a obtenção da politica ótima, e o segundo consiste na busca pela política ótima a partir de uma política inicial (ERNST; GEURTS; WEHENKEL, 2005). Serão apresentandos também o método de resolução *Q-Learning*, as maldições da dimensão e uma visão geral sobre aproximação da programação dinâmica.

3.1 Algoritmo de Iteração de Valor

O algoritmo de Iteração de Valor (*Value Iteration Algorithm*) calcula iterativamente, por meio da equação de Bellman, uma função valor ótima, que é utilizada para se obter uma política ótima.

A cada iteração, é calculado o valor de se estar no dado estágio baseado no valor de se estar no estágio anterior, até que o critério de parada seja satisfeito.

O pseudocódigo do algoritmo, baseado na descrição realizada por Powell (2007), é apresentado a seguir.

Algoritmo 1: *Algoritmo de Iteração de Valor*

Saída: J_ϵ, π^ϵ

inicialize $J^0(s) = 0, \forall s \in S$;
inicialize n $= 1$;
enquanto $\|J^n - J^{n-1}\| \geq \frac{\epsilon(1-\gamma)}{2(\gamma)}$ **faça**
    $n = n + 1$;
    para todo $s \in S$ **faça**
        $J^n(s) = \max_{a \in A}(C(S,a) + \gamma \sum_{s' \in S} \mathbb{P}(s'|s,a) J^{n-1}(s'))$;
    fim
fim
$J^\epsilon = J^n$;

No pseudocódigo acima, γ é o fator de desconto, ϵ representa o erro máximo permitido a cada estágio e é utilizado como critério de convergência, $||J||$ é o maior valor absoluto de um vetor v de elementos, e n é um contador que é incrementado até que o critério de parada $\frac{\epsilon(1-\gamma)}{2(\gamma)}$ seja atingido.

O algoritmo inicializa o valor de se estar no estágio inicial como nulo, e a cada

iteração, enquanto a maior mudança no valor de se estar no estado corrente for maior do que o critério de parada, o valor de se estar no estágio corrente é calculado. Quando finalizado, o procedimento retorna a política e o vetor de valores.

3.2 Algoritmo de Iteração de Política

O algoritmo de Iteração de Política (*Policy Iteration Algorithm*) parte de uma política inicial arbitrária e busca refiná-la a cada iteração, retornando uma política ótima após o critério de parada ser satisfeito. Todavia, quando o número de estados do problema é muito grande, pode se tornar inviável devido ao fato de ter que atualizar a função valor a cada iteração (POWELL, 2007).

Cada iteração possui duas etapas: avaliação de política e aperfeiçoamento de política. Na avaliação da política, sua função valor é calculada, e no aperfeiçoamento da política, uma nova política é gerada a partir da função de valor obtida.

O pseudocódigo do algoritmo, baseado em Powell (2007), é mostrado a seguir.

Algoritmo 2: *Algoritmo de Iteração de Política*

Saída: π^n

selecione uma política inicial π^0;
inicialize n = 1;
calcule a matriz de transição $P^{\pi^{n-1}}$;
calcule o vetor custo $c^{\pi^{n-1}}$ por meio de $c^{\pi^{n-1}}(s) = C(s, A^{\pi^{n-1}})$;
encontre $J^{\pi,n}$ através de $(I - \gamma P^{\pi^{n-1}})J = c^{\pi^{n-1}}$;
encontre a política π^n definida por $a^n = arg\max_{a \in A}(C(a) + \gamma P^{\pi} J^n)$;
se $a^n(s) = a^{n-1}(s)$ **então**
    $a^* = a^n$;
fim
senão
    n = n+1;
    retorne ao passo 3;
fim

É escolhida uma política inicial arbitrária π, e são calculados o vetor custo c^{π}, a matriz de transição P^{π} e o valor J^n de se estar em um determinado estado de acordo com a política. A atualização da política se dá pela aplicação de uma decisão *a(s)* em cada estado s do sistema. Se a decisão aplicada não resulta em uma melhoria na política, isto é, se a política não consegue mais ser melhorada, ela é uma política ótima e então é retornada.

3.3 *Q-learning*

O algortimo *Q-learning* foi desenvolvido por Watkins e Dayan (1992), e se configura como uma das principais técnicas de aprendizagem por reforço. Segundo Sutton e Barto (2018) a aprendizagem por reforço se baseia em problemas em que o sistema deve aprender, de forma autônoma e através de sua interação com o ambiente, a tomar a decisão que proporcione a maior recompensa.

Ao executar uma ação, o estado do sistema é atualizado, e é representado por um valor conhecido como reforço. Através de tentativa e erro, deve-se descobrir quais ações resultam no maior retorno, a soma dos retornos ao longo do tempo, para produzir uma política ótima.

De acordo com Teixeira (2016), o algoritmo é semelhante a um Algoritmo de Iteração de Valor livre de modelo, e é baseado na função de valor de ação $Q^{\pi}(s,a)$, que representa o valor esperado do reforço ao se tomar a ação a no estado s.

Ao executar a ação, o sistema vai do estado s_k para o estado s_{k+1}, e recebe o reforço imediato r_{k+1}. Então, a função de valor de ação Q(s,a) é atualizada, de acordo com a equação 3.4, baseada na diferença entre a função valor máxima do estado seguinte e a função de valor atual.

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha \left[r_{k+1} + \gamma \max_a Q_k(s_{k+1}, a) - Q_k(s_k, a_k) \right] \tag{3.4}$$

em que γ representa o fator de desconto, e α a taxa de aprendizagem.

O pseudocódigo do algoritmo é apresentado a seguir.

Algoritmo 3: *Q-learning*

Saída: $Q(s, a)$

1 inicialize $Q_0 = 0$;
2 **para cada** *período* **faça**
3 inicialize o estado s;
4 escolha a ação dada por: $a = arg\max_{a \in A} Q_k(S_k, a)$;
5 observe a recompensa r_{k+1} e o próximo estado S_{k+1};
6 atualize o valor de Q por meio de
$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha([r_{k+1} + \gamma \max_a Q_k(s_{k+1}, a) - Q_k(s_k, A_k)])$;
7 **fim**

De acordo com Watkins (1989), a função valor de ação Q converge para a função de valor de ação ótima Q^*, e a política definida pela execução da ação a converge para a política ótima se cada par estado-ação for testado um número de vezes suficientemente grande.

3.4 Maldições da Dimensionalidade

A programação dinâmica resolve por si só, de forma analítica, uma parcela muito pequena de problemas. As maldições da dimensão justificam porque a Programação Dinâmica não pode ser utilizada de forma mais ampla. Bellman (1961) definiu como "maldição da dimensionalidade"o fato da resolução de um problema estar diretamente relacionada ao seu número de estados e estágios. Na prática, as necessidades computacionais necessárias para a resolução do problema crescem exponencialmente à medida que o número de dimensões cresce.

Powell (2007) descreve as três maldições da dimensionalidade como:

- Variável de estado do sistema: se a variável de estado S_k possui I dimensões, e se cada elemento pode assumir L valores, então o número de estados possíveis é dado por L^I;
- Conjunto de saídas: se a variável W_k, que representa as incertezas, possui J dimensões, e se cada elemento pode assumir M saídas, então o número de saídas é dado por M^J;
- Conjunto de ações: se o vetor ações x_k possui H dimensões, e se cada elemento pode assumir N ações, então o número de ações é dado por N^H;

Portanto, pode-se perceber que, para problemas muito grandes, o uso da programação dinâmica como método de resolução se torna inviável, devido ao elevado tempo de processamento necessário. Para contornar tal problema, a aproximação da programação dinâmica desponta como método de solução de problemas reais, e é apresentada a seguir.

3.5 Aproximação da Programação Dinâmica

Segundo Powell (2007), a aproximação da programação dinâmica proporciona a solução não somente de problemas grandes, mas também de problemas em que não há a descrição formal de todas as suas informações, como a função de transição por exemplo.

A aproximação da programação dinâmica consiste em resolver o problema do início para o fim, técnica conhecida como *Forward Dynamic Programming* (BERTSEKAS, 1995). Entretanto, não é possível a utilização da equação de Bellman (equação 3.3), uma vez que ela exige informações de estágios futuros. Dessa forma, como a função valor não pode ser calculada, a aproximação da programação dinâmica utiliza um aproximador de função, que estima a função de valor sobre o espaço de estados.

A programação dinâmica aproximada simula caminhos aleatórios para os estados futuros, e estima o valor de tomar uma ação. O valor $J_k(S_k)$ de se estar em um estado

então é aproximado por $\bar{J}_k(S_k)$. Dessa forma, não é necessário visitar todos os estados do problema, evitando que as maldições da dimensão ocorram.

Powell (2007) apresenta um *framework* básico para o desenvolvimento de um algoritmo de aproximação de programação dinâmica, mostrado no Algoritmo 4.

A partir de uma aproximação inicial e de um caminho aleatório gerado, a ação a ser tomada é escolhida, e então o próximo estado é calculado. Com base nessas informações, a aproximação da função valor é então atualizada, sendo a eficiência do método proporcional ao número de iterações realizadas.

A aproximação da função valor consiste em uma etapa crucial no processo de solução de problemas via programação dinâmica, uma vez que a maneira como é desenvolvida impacta diretamente nos gastos computacionais.

Algoritmo 4: *Algoritmo de Aproximação de Programação Dinâmica*

Saída: $\bar{J}_k^n(s_k)$

inicialize o estado inicial S_0^1;
inicialize a função de aproximação $\bar{J}_k^0(s_k), \forall s_k \in S_k \forall k \in K$;
inicialize n = 1;
enquanto *critério de parada não for satisfeito* **faça**
    escolha um caminho aleatório ω^n;
    para todo $k \in K$ **faça**
        Otimização: encontre a decisão $a_k^n = A_k^\pi S(k)$;
        Simulação: encontre o próximo estado por meio de $S_k^n = S^M(S_k^n, a_k^n, W_{k+1}(\omega^n))$;
    fim
    atualize a função de aproximação $\bar{J}_k^n(s_k), \forall k \in K$;
    n = n+1;
fim

3.5.1 Método de Aproximação *Least Squares Temporal Differences (LSTD)*

A resolução de problemas por meio da aproximação da programação dinâmica busca estimar a função valor do sistema, também conhecida como o valor de se estar em um estado, por meio do valor esperado das recompensas acumuladas a partir do determinado estado.

Um método amplamente difundido para estimar a função valor é o método de diferenças temporais (*temporal differences (TD)*). Apresentado por Sutton (1985), consiste em obter novas estimativas da função valor por meio de estimativas já realizadas, a partir de informações do estado atual, do próximo estado, e da recompensa imediata obtidas do próprio sistema, não necessitando de um modelo.

A atualização da estimativa da função valor pode ser realizada quando um estado intermediário é visitado, não havendo a necessidade de que se alcance o estado final de um episódio. Dessa forma, a cada passo no episódio, no instante de tempo *k+1* a estimativa da função valor é atualizada a partir da recompensa imediata r_{k+1} e da estimativa da função valor do próximo estado $J(S_{k+1})$, conforme mostra a equação 3.5:

$$J(S_k) = J(S_k) + \alpha[r_{k+1} + \gamma J(S_{k+1}) - J(S_k)] \tag{3.5}$$

em que α representa o fator de aprendizagem, e γ o fator de desconto. Um instante de tempo após a execução da ação, a estimativa da função valor é atualizada a partir da diferença entre a estimativa atualizada $r_{k+1} + \gamma J(S_{k+1})$ associada à transição do estado x_k para x_{k+1}, e a estimativa atual $J(S_k)$ da função valor, que dá o nome de diferenças temporais ao método.

Entretanto, a convergência do método pode ser muito lenta, uma vez que as modificações realizadas são muito pequenas. Uma alternativa para evitar tal comportamento seria aplicar a equação 3.5 sobre todos os estados já visitados ao invés de aplicá-la somente ao último, como é feito. Dessa forma, o histórico de dados é utilizado de forma mais eficiente, sendo possível escolher em quais estados a modificação deve ser mais intensa, característica controlada pelo parâmetro λ (BARRETO, 2008).

Nesse sentido, Bradtke e Barto (1996) propuseram o algoritmo de diferenças temporais baseado nos mínimos quadrados (*least squares temporal differences (LSTD)*), que consiste em minimizar a soma dos quadrados dos resíduos, isto é, da diferença entre os valores reais observados e os valores estimados, ao longo das experiências.

4 Modelo 1 - Modelo Determinístico para o Problema de Movimentação do *tripper*

4.1 Descrição do Problema

Em plantas de beneficiamento mineral, o armazenamento temporário de minério é realizado em silos, equipamentos vastamente utilizados no setor industrial que armazenam materiais secos a granel, como o minério e grãos (ROTTER, 2009). Os silos desempenham um importante papel durante o processamento de minério, pois ao realizarem o armazenamento, são capazes de minimizar os impactos causados por algum problema durante a operação da mina, como a falha de algum equipamento, por exemplo.

Os silos recebem o minério proveniente das correias transportadoras, responsáveis pelo tráfego do minério pela planta transportando o minério oriundo das linhas de processamento para o armazenamento, e a retirada do minério para que ele possa ser enviado as linhas de processamento seguintes é feita pelos alimentadores, localizados em suas extremidades inferiores.

Durante o beneficiamento, o minério conduzido pelas correias transportadoras é descarregado nos silos por meio do carro *tripper*, um equipamento móvel que se locomove longitudinalmente sobre um trilho para direcionar a descarrega do minério à qualquer ponto ao longo de seu trajeto, distribuindo o minério conforme o proposto (NÚÑEZ; SOLEDAD, 2013).

Frequentemente, há uma variabilidade entre os níveis de minério armazenados por cada cada silo. Tal comportamento é acarretado pelo fato de que cada linha de processamento da planta, que deposita e retira minério dos silos, possui sua própria demanda, além do fato do minério ser depositado sobre um silo de cada vez. Sendo assim, é necessário que os níveis de minério armazenados nos silos sejam controlados, para que estejam dentro da faixa desejada e se possível equilibrados.

O controle dos níveis de minério pode ser realizado por meio do controle da movimentação do *tripper*, pois o silo é alimentado somente quando o *tripper* está localizado sobre ele. Dessa forma, o objetivo do problema é determinar o posicionamento do equipamento ao longo do tempo.

Os movimentos do *tripper* podem ser realizados de forma manual ou automática. No controle manual, um operador humano determina o próximo movimento do *tripper* baseado em sua percepção sobre informações do processo, como posição atual do *tripper*, nível atual de minério nos silos e as taxas de entrada e saída de minério nos silos. Entretanto,

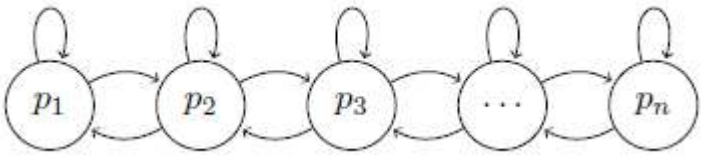
por depender de um operador humano, está sujeito à falhas, que podem afetar a produção e causar diversos problemas, como a necessidade de manutenção em um equipamento (ALBUQUERQUE et al., 2019).

Desse modo, surge a necessidade de que a tomada de decisão do próximo movimento a ser realizado pelo *tripper* seja feita de forma automática, sem participação humana. Em suma, o problema de movimentação do carro *tripper* visa, a partir de informações do estado atual como nível de material armazenado e posição atual do *tripper*, controlar os níveis de minério nos silos, mantendo-os dentro do limite e buscando que o nível seja o mais uniforme possível, por meio da determinação do posicionamento que o *tripper* deve assumir sobre os silos durante o horizonte de tempo. O presente modelo se propõe a resolver o problema através do desenvolvimento de um método de aproximação da programação dinâmica.

O modelo determinístico desenvolvido para o problema de movimentação do carro *tripper* abordado sob a ótica da aproximação da programação dinâmica se baseia no modelo desenvolvido por Morais et al. (2020), que trataram o problema como um processo de decisão de Markov em que a variável de estado armazena toda as informações necessárias à tomada de decisão. Dessa forma, foi possível utilizar o Princípio da Otimalidade de Bellman, que assegura que as decisões tomadas pela política ótima sejam independentes dos estágios anteriores, possibilitando a resolução do problema.

No modelo, considera-se que o *tripper* possui como característica uma limitação na realização de seus movimentos, que faz com que seja possível apenas que ele se movimente em direção as posições imediatamente adjacentes à sua posição atual, ou que ele não se movimente e permaneça na posição em que está. Esta particularidade é apresentada na Figura 3 proposta por Caldas (2018).

Figura 3 – Possíveis movimentos realizados pelo *tripper*



Fonte: Caldas (2018)

Além disso, estipula-se que as taxas de entrada e saída de minério nos silos são constantes e definidas previamente, e que a cada estágio o *tripper* deposita toda a taxa de entrada de minério sobre o silo. A velocidade do *tripper* é considerada constante, e os movimentos a serem realizados devem ser determinados de forma que uniformize o nível de minério armazenado nos silos. Cada silo é caracterizado pelas capacidades mínima e máxima, suas capacidades físicas, e pelos limites inferior e superior estabelecidos, que

representam a faixa de segurança em que os níveis devem ser mantidos. Considera-se também que, de acordo com o movimento realizado pelo *tripper*, há a possibilidade de que ocorra a falta ou o excesso de minério em algum silo, caracterizado pelo extrapolamento dos limites inferior ou superior pré-estabelecidos. Tal situação é representada no modelo pelas variáveis de folga positiva e negativa, que, ao serem utilizadas de forma a penalizar a função objetivo, contribuem para que esses eventos ocorram com a menor frequência possível.

Sendo assim, para a resolução do problema, a cada estágio deve-se escolher o melhor dentre os três movimentos possíveis no quesito custo, isto é, o que resulte na menor folga possível. Para tanto, as regras de despacho propostas por Morais et al. (2020) foram utilizadas como funções indicadoras, foram desenvolvidas outras três e um método de aproximação de programação dinâmica para a resolução.

O modelo do problema, formado pelas variáveis de estado, função objetivo, equação de Bellman, função de transição e restrições é apresentado a seguir. Posteriormente, são apresentados os métodos de solução desenvolvidos.

4.2 Modelo

4.2.1 Parâmetros

Os parâmetros de entrada, que representam as informações do sistema cujos valores são conhecidos, e as variáveis do problema são apresentados na Tabela 2 a seguir.

Tabela 2 – Parâmetros do problema determinístico

Parâmetro	**Descrição**
J	Conjunto de silos
γ_1	Coeficiente folga de excesso
γ_2	Coeficiente folga de falta
t_e	Taxa de entrada
t_s	Taxa de saída
l_{max}	Limite máximo do silo
l_{min}	Limite mínimo do silo
cap_{max}	Capacidade máxima do silo
cap_{min}	Capacidade mínima do silo
P_{ini}	Posição inicial
V_{ini}	Volume inicial
Variável de Estado	**Descrição**
P^k	Posição do *tripper* no k-ésimo estágio
V_j^k	Volume dos j silos no k-ésimo estágio
Variável de Decisão	**Descrição**
x^k	Movimento a ser realizado pelo *tripper* no k-ésimo estágio

Sendo assim, em cada instância há j silos, em que $j \in J$, que possuem capacidade limitada entre cap_{max} e cap_{min}, níveis limitados entre l_{max} e l_{min} e níveis iniciais determinados por V_{ini}. t_e e t_s representam a taxa de entrada e saída de minério em cada silo a cada estágio, e P_{ini} representa a posição em que o *tripper* se encontra inicialmente. P^k e V_j^k, a posição atual do *tripper* e os níveis atuais dos silos respectivamente, são responsáveis por descrever o estado do sistema, enquanto que x^k, o movimento que o *tripper* deve realizar, representa a ação a ser tomada.

4.2.2 Variável de Estado

A variável de estado, que armazena as informações básicas do sistema, é apresentada na Equação 4.1.

$$S^{(k)} = \begin{pmatrix} P^{(k)} \\ V_j^{(k)}, \forall j \in J \end{pmatrix} \tag{4.1}$$

É constituída pela posição atual do *tripper* no k-ésimo estágio e pelos níveis $V_j^{(k)}$ de cada silo $j \in J$.

4.2.3 Função de Transição

A função de transição, representada por $S^{(k)}\left(S^{(k-1)}, x^{(k)}\right)$, realiza a transição do estado $S^{(k-1)}$ para o estado $S^{(k)}$ a partir do movimento $x^{(k)}$ realizado pelo *tripper*.

Na transição entre os estados, são atualizados os valores da posição do *tripper*, dos níveis dos silos e das folgas positiva e negativa. Tais atualizações são realizadas pelas equações a seguir.

$$P^{(k)} = P^{(k-1)} - x_E + x_D \tag{4.2}$$

$$V_j^{(k)} = V^{(k-1)_j} + I_{[P_j]}T_e - T_s \tag{4.3}$$

$$A_j^{(k)} = \max\left\{V_j^{(k)} - l_{max}, 0\right\} \tag{4.4}$$

$$B_j^{(k)} = \max\left\{l_{min} - V_j^{(k)}, 0\right\} \tag{4.5}$$

A equação 4.2 define a nova posição do *tripper*. Na equação 4.3, os níveis $V_j^{(k)}$ de cada silo são atualizados pelo decreścimo da taxa de saída e pelo acréscimo da taxa de

entrada se o *tripper* se encontra sobre o silo corrente, situação representada por $I_{[P_j]}$. Já as folgas $A_j^{(k)}$ e $B_j^{(k)}$, a quantidade de minério que ultrapassou os limites superior e inferior, são calculadas pelas equações 4.4 e 4.5, respectivamente.

4.2.4 Função Objetivo

O presente trabalho busca controlar o nível de minério armazenado nos silos ao longo do tempo, penalizando quando ocorre a falta ou o extrapolamento de minério, isto é, quando há ocorrência de folgas. Dessa forma, o problema visa minimizar o custo gerado pelas folgas.

Em um problema de minimização, o objetivo é minimizar o custo de cada estágio, também conhecido como função retorno $R^{(k)}$, que por sua vez é dado em função do estado atual do sistema e da decisão tomada. Neste trabalho, o custo do movimento realizado pelo *tripper* é dado pelo somatório das penalizações γ_1 e γ_2 devido as folgas positivas e negativas $A_j^{(k)}$ e $B_j^{(k)}$ produzidas por cada silo, conforme mostra a Equação 4.6, e a função objetivo é apresentada pela Equação 4.7 a seguir.

$$R^{(k)}\left(S^{(k)}\left(S^{(k-1)}, x^{(k-1)}\right)\right) = \sum_{j \in J}\left(\gamma_1 A_j^{(k)} + \gamma_2 B_j^{(k)}\right) \tag{4.6}$$

$$\min_{x^{(k)} \in \chi^{(k)}} \sum_{k=1}^{\infty} \sum_{j \in J}\left(\gamma_1 A_j^{(k)} + \gamma_2 B_j^{(k)}\right) \tag{4.7}$$

4.2.5 Restrições

Como dito anteriormente, os possíveis movimentos a serem realizados pelo *tripper* são limitados, e podem ser definidos pelo conjunto $\chi = \{E, C, D\}$, em que E representa movimentar-se para o silo imediatamente à sua esquerda, C faz com que o *tripper* não se movimente e continue na posição em que está, e D representa movimentar-se para o silo imediatamente à sua direita.

A cada k-ésimo estágio, o *tripper* só pode estar posicionado sobre um determinado silo, situação representada pela Equação 4.8, que assegura que a soma dos valores referentes as movimentações a cada estágio seja um. Já a Equação 4.9 mostra que a variável $x_p^{(k)}$, que representa o movimento $p \in \chi^{(k)}$, é binária.

$$\sum_{p \in \chi^{(k)}} x_p^{(k)} = 1 \tag{4.8}$$

$$x_p^{(k)} \in \{0, 1\}\, \forall p \in \chi \tag{4.9}$$

4.2.6 Equação de Bellman

Conforme visto anteriormente, a política ótima do sistema é obtida a partir de sua função valor, apresentada pela Equação 3.2, que representa o valor de se estar em um determinado estado a cada k-ésimo estágio como o valor da série de recompensas acumuladas a partir do estado atual.

Como o presente trabalho se trata de um problema de minimização, a função valor, apresentada pela Equação 4.10 a seguir, é dada pela soma dos custos a partir do estado atual até o estado final,

$$J^{(k)}_{(S^k)} = \sum_{n=k}^{\infty} R^{(k)}_{(s^k)} \tag{4.10}$$

em que $R^{(k)}_{(s^k)}$ representa o custo do k-ésimo estágio, conforme apresentado pela Equação 4.6.

A equação de Bellman, que consiste em calular de forma recursiva a função valor do estágio atual considerando-se o próximo estado, é mostrada na equação 4.11.

$$J^{(k)}_{(S^k)} = \min \left(R^{(k)}_{(s^k)} + \lambda^{(k)} J^{(k+1)}_{(S^{k+1})} \right) \tag{4.11}$$

4.2.6.1 Subproblemas

Devido à recursividade do problema, ele é formado por subproblemas menores a cada estágio.

Como a resolução dos subproblemas será realizada de maneira *forward*, ou seja, de frente pra trás, não é possível o cálculo da função valor, uma vez que ela depende de informações de estágios futuros. Dessa forma, é necessário que seja feita uma aproximação da função de valor dos próximos estágios, estimando-se $\bar{J}^{(k+1)}$ a partir de $S^{(k)}$ e $x^{(k)}$.

Portanto, para a resolução do subproblema, a equação 4.11 pode ser reescrita substituindo-se $J^{(k+1)}$ por $\bar{J}^{(k+1)}$, resultando na equação 4.12 apresentada a seguir.

$$\bar{J}^{(k)}_{(S^k)} = \min \left(R^{(k)}_{(s^k)} + \lambda^{(k)} \bar{J}^{(k+1)}_{(S^{k+1})} \right) \tag{4.12}$$

4.3 Métodos de Solução

4.3.1 Aproximação de Programação Dinâmica

Como já mencionado anteriormente, a aproximação da programação dinâmica permite que sejam resolvidos, além de problemas cujo modelo completo não é conhecido, problemas de dimensões reais, evitando que as maldições da dimensão aconteçam.

Para tanto, confome mostra a equação 4.12, deve-se somar o custo do estágio atual com o custo dos estágios seguintes, dado pela aproximação da função valor $\bar{J}$.

Existem vários métodos de aproximação da função valor, como por exemplo regressão linear e redes neurais. O método escolhido para ser utilizado neste trabalho foi proposto por Bertsekas e Tsitsiklis (1996) e consiste em aproximar a função por meio de uma combinação linear de funções indicadoras ϕ, representada pela equação 4.13, apresentada por Teixeira (2016),

$$\bar{J}(S) = \sum_{i=1}^{m} \theta_i \phi_i(S) \tag{4.13}$$

em que $\theta \in \mathbb{R}^m$ representa o vetor de parâmetros, e ϕ: S $\rightarrow \mathbb{R}^m$ representa um mapeamento do espaço de estados em um escalar em um espaço característico m-dimensional.

De acordo com Powell (2007), as funções indicadoras são uma representação do estado que possuem a função de extrair informações relevantes sobre ele e transformá-las em um escalar, e podem ser construídas de diferentes maneiras, já que dependem do problema e também de quem as modela. Dessa maneira, a qualidade das indicadoras determina a eficiência da aproximação da função valor.

O presente método utiliza as regras de despacho Gulosa, Passos e Desvio Padrão desenvolvidas por Morais et al. (2020) como funções indicadoras, cria outras três, Passos up, Caminho Ideal e Custo Mínimo, e resolve o problema por meio do Algoritmo de Iteração de Política Aproximado para horizontes infinitos usando diferença temporal por mínimos quadrados (*LSTD*).

O Algoritmo de Iteração de Política consiste em estimar o valor da função valor e então utilizá-lo na obtenção de uma nova política melhorada. A partir da nova política obtida, são escolhidas as ações que devem ser tomadas ao longo do horizonte de tempo.

Os algoritmos e as funções desenvolvidas para cada modelo são apresentados a seguir.

4.3.2 Funções Indicadoras

Nesta seção, são apresentadas as funções indicadoras utilizadas na resolução proposta para o problema determinístico de movimentação do *tripper*.

4.3.2.1 Função Gulosa

Um algoritmo guloso possui um comportamento míope, em que somente as informações disponíveis no estágio atual são utilizadas na tomada de decisão, desconsiderando o impacto que a ação escolhida causaria em estágios futuros. Dessa forma, não requer nenhum tipo de aproximação por $\bar{J}$.

No presente trabalho, as informações necessárias à função gulosa estão contidas na variável de estado. Considerando-se o k-ésimo estado do sistema como o estado atual, por meio da variável de estado expressa por 4.1 e da resolução das equações 4.2 e 4.3, as folgas $A_j^{(k)}$ e $B_j^{(k)}$ são calculadas pelas equações 4.4 e 4.5.

O retorno da função é dado pelas folgas produzidas, e é calculado pela Equação 4.14 a seguir, em que γ_1 e γ_2 representam os coeficientes das folgas positiva e negativa, respectivamente.

$$\phi_{gulosa} = \sum_{j \in J} \left(\gamma_1 A_j^{(k)} + \gamma_2 B_j^{(k)} \right) \tag{4.14}$$

4.3.2.2 Função Passos

Consiste em calcular, para cada silo j, o número NP_j de passos (movimentos) do silo vigente até o silo sobre o qual o *tripper* está posicionado, e utilizar essa informação na atualização dos níveis dos silos, realizada pela Equação 4.15.

$$V_j = V_j - T_s(NP_j) \tag{4.15}$$

De acordo com a Equação 4.15, quanto maior o número de passos, maior o declínio do nível de cada silo. A partir dos níveis atualizados, as folgas positiva e negativa dos silos são calculadas pelas equações 4.16 e 4.17 a seguir:

$$F1_j = \max\{V_j - l_{max}, 0\} \tag{4.16}$$

$$F2_j = \max\{l_{min} - V_j, 0\} \tag{4.17}$$

Por fim, o retorno da função passos é dado por:

$$\phi_{passos} = \sum_{j \in J} \left(\gamma_1 F1_j + \gamma_2 F2_j \right), \tag{4.18}$$

4.3.2.3 Função Desvio Padrão

Para que a tomada de decisão seja baseada também na homogeneidade dos níveis, a fim de que os níveis dos silos sejam os mais equilibrados possível, calcula-se o desvio padrão dos níveis, que mede o grau de dispersão dos valores, e então o utiliza no cálculo do retorno local.

O cálculo do desvio padrão é realizado pela Equação 4.19, em que V_j representa o nível do silo j, $\bar{V}$ representa a média dos níveis dos silos e |J| representa o número de silos da instância utilizada.

$$\sigma = \sqrt{\frac{\sum_{j \in J} \left(V_j - \bar{V}\right)^2}{|J|}} \tag{4.19}$$

A Equação 4.20 mostra a utilização do desvio padrão σ no cálculo do retorno local da função desvio padrão.

$$\phi_{desvio} = -\sigma \tag{4.20}$$

4.3.2.4 Função Passos Up

A função Passos Up busca analisar o comportamento do silo sobre o qual o *tripper* está localizado, indicando e penalizando caso o seu transbordo esteja próximo a ocorrer, ou seja, caso a quantidade de minério depositada pelo *tripper* ultrapasse o seu limite superior.

Se o nível do silo não ultrapassar o valor de seu limite superior, a função não retorna nenhum valor. Entretanto, se o limite for excedido, a folga positiva $F1_p$ é calculada através da atualização no nível do silo.

A função Passos Up pode ser considerada semelhante à função Passos, diferindo-se pelos fatos de considerar apenas o silo onde o *tripper* está e analisar apenas o extrapolamento de seu limite superior. Comparando-se as duas funções, enquanto na função Passos o nível dos silos é atualizado de acordo com o número de passos do silo atual até o silo onde está o *tripper*, na função Passos Up o nível do silo cujo o *tripper* está posicionado é atualizado de acordo com a quantidade que excedeu o limite superior.

Primeiramente, é calculada a quantidade q_u que ultrapassa o limite superior do silo por:

$$q_u = \frac{V_p - l_{max}}{t_s}, \tag{4.21}$$

e o total de iterações a serem realizadas para a atualização do nível é igual à quantidade q_u. A cada iteração $i = 0, ..., q_u$, o nível do silo então é atualizado pela Equação 4.22:

$$V_{pi} = V_p - T_s q_a, \quad p/ \quad q_a = 0...q_u \tag{4.22}$$

e a folga $F1_i$ é calculada por:

$$F1_i = V_{pi} - l_{max} \tag{4.23}$$

A folga ao final das iterações é dada por:

$$F1_p = \sum_{i=0}^{q_u} F1_i \tag{4.24}$$

e o retorno da função indicadora é calculado conforme a Equação 4.25, em que γ_1 é o coeficiente da folga.

$$\phi_{up} = (\gamma_1 F1_p) \tag{4.25}$$

4.3.2.5 Função Caminho Ideal

A função faz alusão ao padrão geométrico zigue-zague, em que o considera como o caminho ideal a ser realizado pelo *tripper*, dependendo das condições iniciais do sistema.

Neste caminho, há duas possibilidades: iniciar movimentando o *tripper* para a esquerda ou para a direita. Dessa forma, calcula-se para cada silo j dois estados, esquerda e direita, a partir da posição atual p do *tripper* pelas equações 4.26 e 4.27 a seguir:

$$Esp\_direita(p) = \begin{cases} (n-p) * ts + vol_{medio}, & p/ \quad j = 1...p \\ (-p) * ts + vol_{medio}, & p/ \quad j = p+1...n \end{cases} \tag{4.26}$$

$$Esp\_esquerda(p) = \begin{cases} (p-1) * ts + vol_{medio}, & p/ \quad j = p...n \\ (p-1-n) * ts + vol_{medio}, & j/ \quad k = 1...p-1 \end{cases} \tag{4.27}$$

em que vol_{medio} representa a média dos niveis dos silos.

A soma das diferenças entre os níveis de cada silo e o seu estado equivalente em uma política zigue-zague controlada é calculada por:

$$Ind\_direita(p) = \sum_{i=1}^{n} (nivel(i) - Esp\_direita(p)[i]) \tag{4.28}$$

$$Ind\_esquerda(p) = \sum_{i=1}^{n} (nivel(i) - Esp\_esquerda(p)[i]) \tag{4.29}$$

O retorno da função é definido como a menor soma das diferenças, como mostra a equação 4.30.

$$\phi_{zigue} = \min(Ind\_direita, Ind\_esquerda) \tag{4.30}$$

4.3.2.6 Função Ciclo mínimo

Dada a posição em que o *tripper* se encontra, consiste em calcular o custo de se realizar uma volta completa nos silos, para a esquerda e para a direita, e escolher o ciclo que apresentar o menor custo.

Ou seja, no ciclo da esquerda, consiste em calcular o custo do *tripper* partir de sua posição, se locomover até o primeiro silo, em seguida se locomover até o último silo, e finalmente voltar à sua posição inicial.

Já o ciclo da direita consiste em calcular o custo do caminho realizado pelo *tripper*, partindo de sua posição inicial em direção ao último silo, em que em seguida locomove-se até o primeiro silo, e por fim desloca-se até sua posição inicial.

Em cada ciclo, a atualização dos níveis a cada movimento realizado pelo *tripper* é feita pela Equação 4.3, e cálculo das folgas é realizado pelas equações 4.4 e 4.5, respectivamente.

O retorno de cada ciclo é dado por:

$$ciclodir = \sum_{j \in J} (\gamma_1 A_j + \gamma_2 B_j) \tag{4.31}$$

$$cicloesq = \sum_{j \in J} (\gamma_1 A_j + \gamma_2 B_j) \tag{4.32}$$

A função indicadora retorna o menor custo dos dois caminhos percorridos, dado por:

$$\phi_{ciclo\_min} = \min(ciclodir, cicloesq) \tag{4.33}$$

4.3.3 Algoritmo de Iteração de Política Aproximado LSTD

Tendo em vista que o presente trabalho utiliza como método de resolução o Algoritmo de Iteração de Política Aproximado, e utiliza as funções indicadoras da Seção 4.3.2 e o método de diferenças temporais de mínimos quadrados ($LSTD$) apresentado na Seção 3.5.1 como aproximadores da função valor, a aproximação é feita conforme mostra a Equação 4.13, reapresentada pela Equação 4.34.

$$\bar{J}(S) = \sum_{i=1}^{m} \theta_i \phi_i(S) \tag{4.34}$$

De acordo com a Equação 4.34, a aproximação da função valor é dada pelas funções indicadoras ϕ e seus respectivos coeficientes θ. Para o cálculo do vetor de coeficientes θ, são realizadas várias simulações para convergência do valor, e durante as simulações o método de mínimos quadrados é utilizado para atualizar os valores dos coeficientes. Dessa

forma, o objetivo do algoritmo consiste em estimar os valores dos coeficientes θ para a criação de uma nova política.

Seu pseudocódigo, adaptado de Powell (2007), é apresentado no Algoritmo 5. A partir de uma política inicial e seu coeficiente θ, a cada iteração m para a convergência de valor, são calculados a ação a ser tomada, o próximo estado do sistema, o custo, o erro, a matriz B e o vetor θ. Após terem sido realizadas as m iterações, a política e seu coeficiente θ são atualizados. Essa operação é repetida n vezes, e retorna a política atualizada e o θ correspondente.

Algoritmo 5: *Algoritmo de Iteração de Política Aproximado LSTD*

Saída: A^{pi}, θ^n

inicialize a política A^{π};
inicialize θ^0;
incialize o estado inicial $S^{0,0}$;
para cada $iteraçãon \in N$ **faça**
    inicialize θ^n;
    para cada $iteraçãom \in M$ **faça**
        inicialize o estado $S^{n,m}$;
        encontre a decisão a dada por $a^{n,m} = A^{\pi}(S^{n,m}|\theta^{n-1})$;
        encontre o próximo estado por meio de $S^{n,m+1} = S^{M}(S^{n,m}, a^{n,m}, W^{m+1})$;
        calcule o custo $\hat{C}^m = C(S^{n,m}, a^{n,m}, W^{m+1})$;
        calcule o erro $\epsilon^m = \hat{C}^m - (\phi^m - \gamma\phi^{m+1})\theta^{m-1}$;
        calcule B por $B^m = B^{m-1} - \frac{B^m\phi^m(\phi^m - \gamma\phi^{m+1})^T B^{m-1}}{\lambda + (\phi^m - \gamma\phi^{m+1})^T B^{m-1}\phi^m}$;
        calcule θ através de $\theta^m = \theta^{m-1} + \frac{\epsilon^m B^{m-1}\phi^m}{\lambda + (\phi^m - \gamma\phi^{m+1})^T B^{m-1}\phi^m}$;
    fim
    atualize $\theta^{n+1} = \theta^{n,M}$;
    atualize a política;
fim

O parâmetro λ funciona como um parâmetro de ajuste, possibilitando a escolha de quais observações devem ser priorizadas. Por definição, quando λ é igual a 1, todas as observações recebem o mesmo peso. Já quando λ é maior que 1, as observações anteriores recebem um peso maior, enquanto que λ menor que 1 significa que as observações mais recentes recebem maior peso.

No presente trabalho, foram desenvolvidas três funções para determinar o valor de λ. São elas:

- stpsze_cte1: não prioriza nenhuma observação, já que todas recebem o mesmo peso;
- stpsze_ln50up100: retorna um valor crescente ao longo das iterações, que varia entre

0,5 e 1,0 a uma taxa de 0,5%, ou seja, o histórico de observações recebe maior peso;

- stpsze_ln100dn80: o valor retornado decresce de 1,0 a 0,8 a uma taxa de 0,2% ao longo das iterações, o que indica que as observações mais recentes possuem um peso maior.

5 Experimentos Computacionais para o Problema Determinístico

A implementação do algoritmo de Iteração de Política Aproximado *LSTD* desenvolvido foi realizada em linguagem *Python*, e os testes foram realizados em um computador com processador Intel Core i7-4790, 3,60GHz.

Para a realização dos testes, adaptou-se uma instância utilizada por Pedrosa (2019), resultando na instância *16_60_1_teste.xml*. Seus parâmetros são apresentados na Tabela 3 a seguir, em que J representa o conjunto de silos, γ_1 e γ_2 são os coeficientes das folgas positiva e negativa, t_e representa a taxa de entrada de minério nos silos e t_s a taxa de saída, cap_{max} e cap_{min} representam as capacidades máxima e mínima dos silos, l_{max} e l_{min} representam os limites superior e inferior dos silos, P_{ini} é a posição inicial do *tripper* e V_{ini} é a lista de níveis iniciais de cada silo.

Tabela 3 – Parâmetros Instância *16_60_1_teste.xml*

Parâmetro	**Valor**
J	16
γ_1	0.95
γ_2	0.95
t_e	16
t_s	1
cap_{max}	100
cap_{min}	0
l_{max}	80
l_{min}	20
P_{ini}	13
V_{ini}	[39, 79, 13, 63, 47, 70, 53, 84, 86, 34, 11, 21, 67, 15, 94, 15]

Foram realizados dezesseis testes, divididos em duas categorias. Na primeira categoria, foram utilizadas todas as funções indicadoras implementadas, e testou-se a combinação entre uma das três funções lambda implementadas, stpsze_ln50up100, stpsze_ln100dn80 e stpsze_cte1, e a utilização ou não da função resetB, que quando utilizada reseta a matriz B como uma matriz identidade a cada nova iteração do algoritmo. Já na segunda categoria, testou-se diferentes combinações de utilização das funções indicadoras, utilizando a função lambda cte1 e não utilizando a função resetB. Em todos os testes, os coeficientes das funções indicadoras utilizadas foram inicializados com o valor 0, e o fator de desconto γ assumiu o valor 0.99.

Os testes realizados são apresentados, de forma sucinta, a seguir.

- Categoria 1 - Teste de combinações das funções lambda e da função resetB:
  - Teste resetB-cte1: habilita a função resetB e utiliza a função stpsze_cte1;
  - Teste resetB-ln100dn80: habilita a função resetB e utiliza a função stpsze_ln100dn80;
  - Teste resetB-ln50up100: habilita a função resetB e utiliza a função stpsze_ln50up100;
  - Teste B-cte1: desabilita a função resetB e utiliza a função stpsze_cte1;
  - Teste B-ln100dn80: desabilita a função resetB e utiliza a função stpsze_ln100dn80;
  - Teste B-ln50up100: desabilita a função resetB e utiliza a função stpsze_ln50up100;
- Categoria 2 - Teste de combinações das funções indicadoras:
  - Teste Custo Míope: utiliza apenas a função indicadora Custo Míope;
  - Teste Custo Míope e Desvio Padrão: utiliza as funções indicadoras Custo Míope e Desvio Padrão;
  - Teste Custo Míope e Ciclo Mínimo: utiliza as funções indicadoras Custo Míope e Ciclo Mínimo;
  - Teste Custo Míope e Passos: utiliza as funções indicadoras Custo Míope e Passos;
  - Teste Custo Míope, Passos e Desvio Padrão: utiliza as funções indicadoras Custo Míope, Passos e Desvio Padrão;
  - Teste Custo Míope, Passos, Desvio Padrão e Ciclo Mínimo: utiliza as funções indicadoras Custo Míope, Passos, Desvio Padrão e Ciclo Mínimo;
  - Teste Custo Míope, Passos, Desvio Padrão, Ciclo Mínimo e Caminho Ideal: utiliza as funções indicadoras Custo Míope, Passos, Desvio Padrão, Ciclo Mínimo e Caminho Ideal;
  - Teste Custo Míope, Passos, Desvio Padrão e Caminho Ideal: utiliza as funções indicadoras Custo Míope, Passos, Desvio Padrão e Caminho Ideal;
  - Teste Custo Míope e Passos Up: utiliza as funções indicadoras Custo Míope e Passos Up;
  - Teste Custo Míope e Caminho Ideal: utiliza as funções indicadoras Custo Míope e Caminho Ideal.

Em cada teste, foram realizadas duas etapas. A primeira, de treinamento, consiste em executar 15 iterações do Algoritmo de Iteração de Política Aproximado *LSTD* para gerar a nova política, em que foram realizadas 1000 simulações para convergência do

valor a cada iteração. A segunda, de simulação, consiste em realizar duas simulações para a nova política gerada pelo treinamento: na primeira simulação, são realizadas 100 iterações do algoritmo de solução do problema utilizando como estado inicial a instância *16_60_1_teste.xml*, e na segunda são gerados 50 estados de forma aleatória e o algortimo de solução é executado 100 vezes para cada estado.

Para fins de exemplificação, serão abordados o treinamento e os gráficos gerados pela primeira simulação de dois testes realizados, o Teste Custo Míope e Ciclo Mínimo e o Teste Custo Míope e Passos, que obtiveram resultados satisfatórios e insatisfatórios, respectivamente. Já os resultados das duas simulações para todos os testes realizados serão apresentados sob as formas de tabela e gráfica, cujas medidas permitem que uma análise comparativa possa ser realizada. Por fim, será apresentada uma análise comparativa entre os resultados obtidos por este trabalho e os resultados encontrados por Caldas e Martins (2018).

5.1 Etapa de Treinamento

Como dito anteriormente, a etapa de treinamento neste trabalho consiste em encontrar uma nova política para o sistema a partir de uma política inicial, por meio da execução de 15000 iterações do Algoritmo de Iteração de Política Aproximado *LSTD*.

Os tempos gastos para a realização do treinamento de cada teste são apresentados na Tabela 4 a seguir, os quais são exibidos em segundos. Na tabela, os testes resetB-cte, resetB-dn, resetB-up, B-cte, B-dn e B-up referem-se aos testes resetB-cte1, resetB-ln100dn80, resetB-ln50up100, B-cte1, B-ln100dn80 e B-ln50up100 da Categoria 1, enquanto que os testes CM, CM-DP, CM-CMin, CM-P, CM-P-DP, CM-P-DP-CMin, CM-P-DP-CMin-CI, CM-P-DP-CI, CM-Pup e CM-CI representam, respectivamente, os testes Custo Míope, Custo Míope e Desvio Padrão, Custo Míope e Ciclo Mínimo, Custo Míope e Passos, Custo Míope, Passos e Desvio Padrão, Custo Míope, Passos, Desvio Padrão e Ciclo Mínimo, Custo Míope, Passos, Desvio Padrão, Ciclo Mínimo e Caminho Ideal, Custo Míope, Passos, Desvio Padrão e Caminho Ideal, Custo Míope e Passos Up, e Custo Míope e Caminho Ideal, que formam a Categoria 2. Por meio da análise dos dados, constata-se que, de forma geral, o tempo aumenta à medida que o número de funções indicadoras utilizadas cresce, sendo que o teste CM apresentou o menor tempo, de 5416,81 segundos, e o teste B-dn apresentou o maior tempo, de 25045,26 segundos.

Tabela 4 – Tempo de Treinamento dos Testes - Problema Determinístico

Teste	**Tempo de Treinamento (s)**
resetB-cte	24109,51
resetB-dn	24178,23
resetB-up	23447,20
B-cte	24943,63
B-dn	25045,26
B-up	22987,60
CM	5416,81
CM-DP	10246,60
CM-CMin	10700,77
CM-P	9822,13
CM-P-DP	13777,50
CM-P-DP-CMin	19826,09
CM-P-DP-CMin-CI	19419,71
CM-P-DP-CI	14079,01
CM-Pup	10707,41
CM-CI	5852,52

Como se pode observar por meio do Algoritmo 5, durante o treinamento, a cada iteração m para convergência do valor, a ação a ser tomada é escolhida, e então calcula-se o custo, o erro, e os novos valores dos coeficientes θ das funções indicadoras. O custo representa o custo real da função objetivo para o estado, e o erro é calculado pelo método de diferenças temporais e representa a diferença entre a estimativa atual e a estimativa atualizada do valor de estar em um determinado estado. Ao fim das iterações, a política é atualizada.

A seguir são apresentados os gráficos do custo, do erro e dos coeficientes θ das funções indicadoras utilizadas, que descrevem o comportamento das variáveis envolvidas no treinamento ao longo das iterações para os testes escolhidos. Vale ressaltar que, para uma melhor visualização, os *outliers* dos gráficos não foram apresentados.

Salienta-se que o gráfico do coeficiente θ_1 representa o coeficiente da primeira função indicadora utilizada pelo teste, enquanto que o gráfico do coeficiente θ_2 refere-se ao coeficiente da segunda função. Além disso, os gráficos dos coeficientes são formados por duas séries: uma laranja e uma azul. A azul apresenta o valor do coeficiente θ a cada iteração m, representando a sua atualização a cada iteração do algoritmo. Já a laranja apresenta o valor de θ a cada iteração n do Algoritmo de Iteração de Política Aproximado *LSTD*, ou seja, seu valor é atualizado somente depois que as m iterações para convergência do valor forem realizadas.

5.1.1 Teste Custo Míope e Ciclo Mínimo

As figuras 4, 5, 6 e 7 apresentam os gráficos do custo, do erro e dos coeficientes das funções indicadoras, que descrevem o comportamento das variáveis ao longo das 15.000 iterações realizadas.

A partir da Figura 4, pode-se perceber que nas iterações iniciais as ações escolhidas pelo algoritmo geraram custos, o que implica que os limites dos silos foram violados. Entretanto, apesar de ter ocorrido um pico no início, logo estabeleceu-se um padrão, e já na iteração 1020 o custo convergiu para zero. Dessa forma, observa-se que o treinamento proporcionou ao sistema o controle dos níveis dos silos, em que os movimentos escolhidos não provocaram folgas.

Com relação ao erro, este é calculado a cada iteração em função do custo e dos retornos obtidos pela funções indicadoras antes e depois da transição. A Figura 5 exibe o comportamento do erro durante o treinamento ao longo do horizonte de tempo, em que, apesar das variações iniciais, rapidamente verifica-se uma tendência à convergência, sendo que o valor zero foi obtido na iteração 1021. Logo, infere-se que a função valor do sistema foi estabilizada.

O comportamento dos coeficientes θ das funções indicadoras utilizadas, Custo Míope e Ciclo Mínimo, é exibido nas figuras 6 e 7, respectivamente. Apesar dos valores de ambos os coeficientes terem convergido rapidamente, conforme demonstrado pela proximidade entre as linhas laranja e azul nos gráficos, as suas atualizações apresentaram diferenças entre si. Como se pode observar, o valor do coeficiente θ_1 da função indicadora Custo Míope sofreu alterações consideráveis nas iterações iniciais, mas convergiu para seu valor final em pouquíssimas iterações. Já o coeficiente θ_2 refente à função indicadora Ciclo Mínimo foi atualizado de forma menos brusca, em que após o pico inicial foram realizadas pequenas atualizações a cada iteração, o que fez com que a tendência à convergência ocorresse mais tardiamente se comparado à função Custo Míope.

Logo, constata-se que o treinamento retornou uma política estabilizada e capaz de controlar o sistema de forma satisfatória. No entanto, como a política foi estabilizada logo no início das iterações, visto que os seus coeficientes convergiram rapidamente, pode-se concluir que o treinamento do presente teste poderia ter sido realizado utilizando-se um número consideravelmente menor de iterações, o que implicaria em um expressivo ganho computacional.

Figura 4 – Teste Custo Míope e Ciclo Mínimo: custo ao longo das iterações

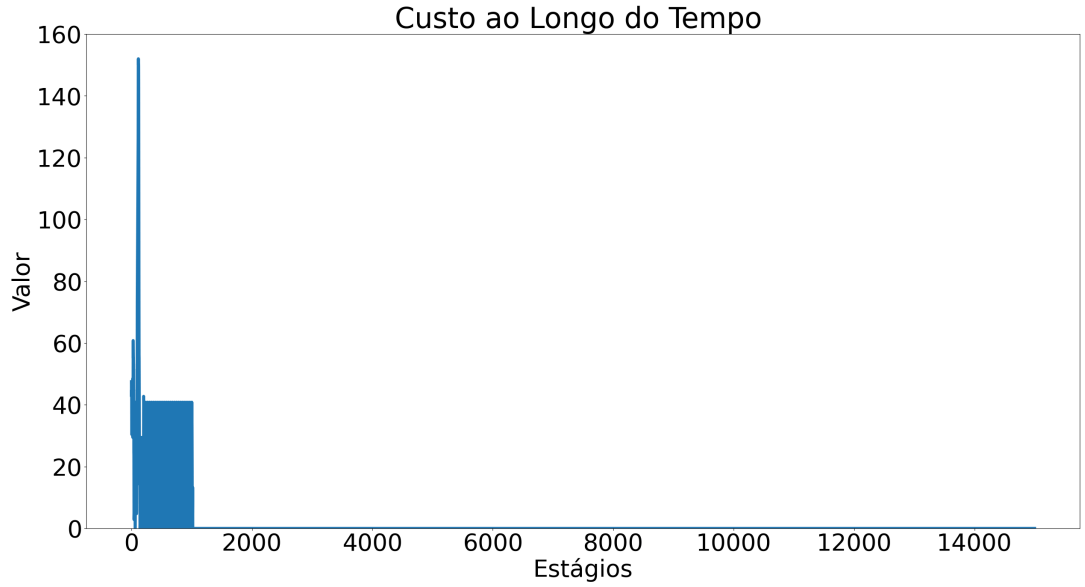


Figura 5 – Teste Custo Míope e Ciclo Mínimo: erro ao longo das iterações

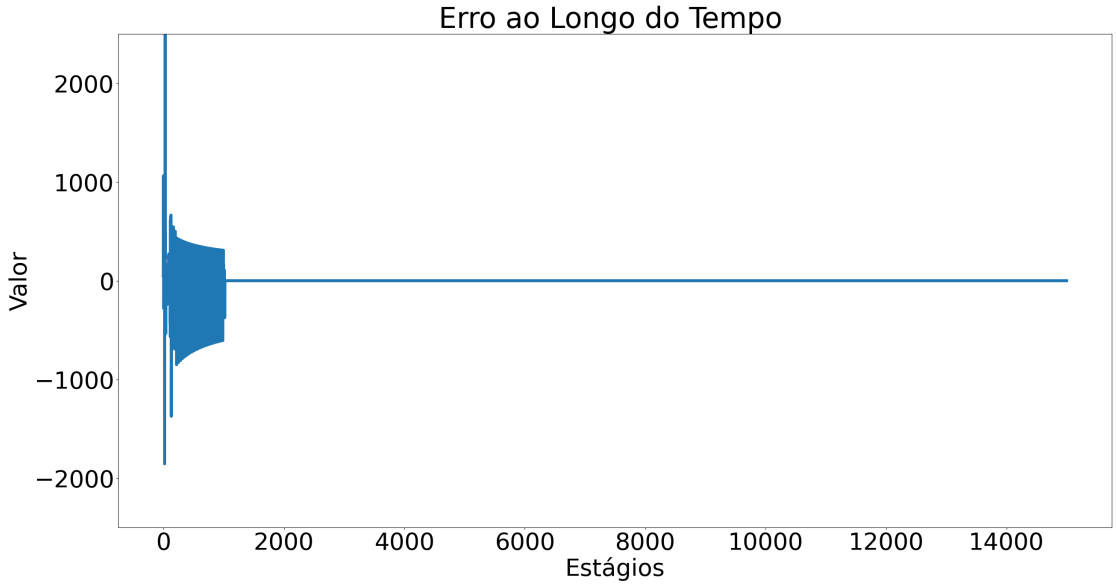


Figura 6 – Teste Custo Míope e Ciclo Mínimo: coeficiente θ_1 ao longo das iterações

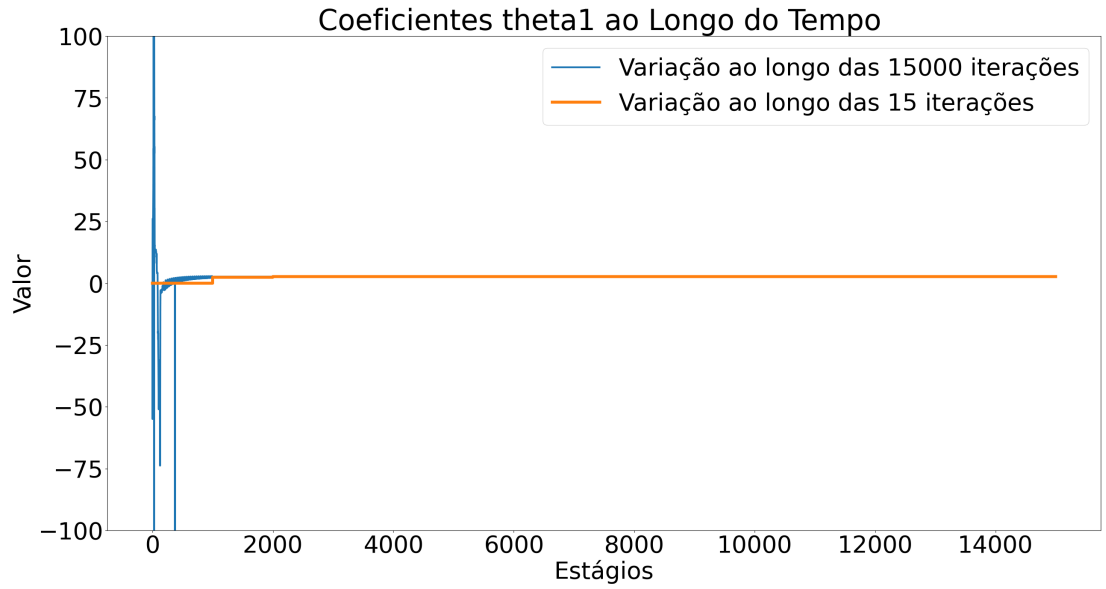
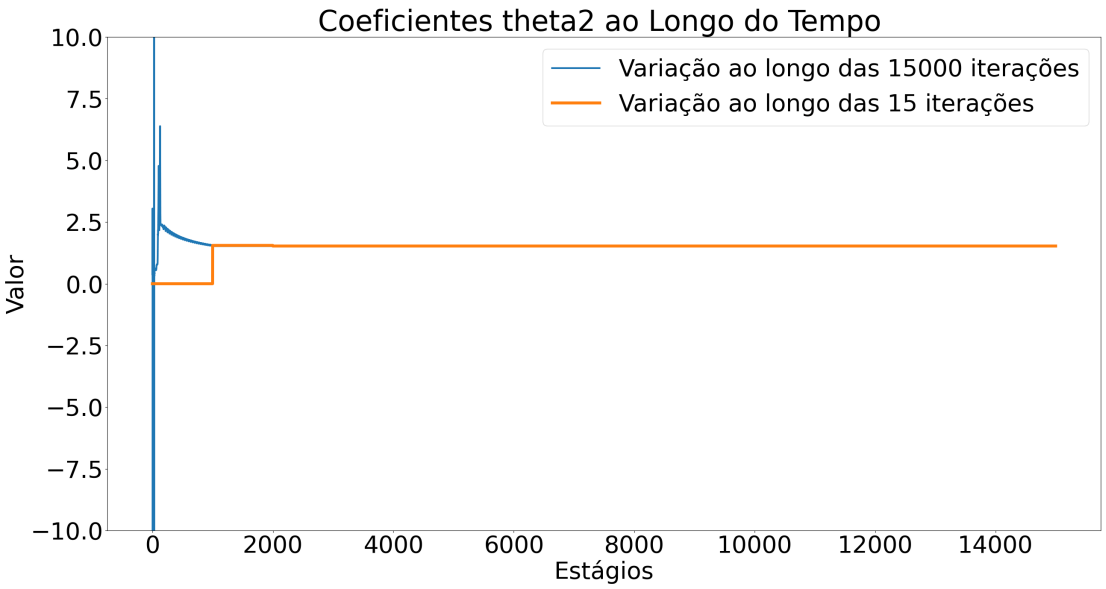


Figura 7 – Teste Custo Míope e Ciclo Mínimo: coeficiente θ_2 ao longo das iterações



5.1.2 Teste Custo Míope e Passos

As variáveis custo, erro e os coeficientes das funções indicadoras envolvidas no processo de treinamento são apresentadas nas figuras 8, 9, 10 e 11 a seguir.

Apresentado na Figura 8, o comportamento do custo ao longo das iterações mostra que, embora nas iterações iniciais tenha se conseguido diminuir o seu valor e atingir um

padrão, esse comportamento não foi mantido, resultando em um salto e na estabilização em um valor relativamente alto, 304, até mesmo maior que o pico inicial. Desse modo, nota-se que, durante o treinamento, não foi possível controlar os níveis dos silos de modo que não houvessem folgas.

O gráfico do erro, conforme mostra a Figura 9, aponta que nas primeiras iterações há uma grande variação no valor de se estar no determinado estado, mas logo percebe-se uma tendência para convergência de seu valor. Conforme o erro vai diminuindo, espera-se que tal comportamento seja mantido, convergindo para zero. Entretanto, ainda há dois saltos nas iterações posteriores, destoando do padrão que vinha sendo mantido. Comparando-se com o teste anterior, observa-se que a estabilização do erro do presente teste ocorreu mais tardiamente.

As figuras 10 e 11 mostram a atualização dos valores dos coeficientes θ_1 e θ_2 referentes as funções indicadoras Custo Míope e Passos a cada iteração. Como se pode observar pelos gráficos, aproximadamente nas 2000 iterações iniciais houve a atualização dos valores dos coeficientes, sendo que logo após já percebe-se a convergência. Além disso, a atualização dos coeficientes se deu de forma similar, em que, exceto pelos picos, as 1000 iterações iniciais não modificaram significativamente os valores, enquanto que nas próximas 1000 iterações as alterações foram mais expressivas, conforme mostram as séries azuis, convergindo para seus valores finais. Logo, verifica-se que foram necessárias apenas cerca de 2000 iterações para que o treinamento encontrasse uma política estabilizada.

Dessa forma, pode-se concluir que, de forma análoga ao teste anterior, o treinamento poderia ter sido realizado com um número significativamente menor de iterações, o que melhoraria o desempenho do teste no quesito tempo. Todavia, apesar de apresentar um tempo de execução competitivo, o custo da política gerada faz com que a sua utilização resulte em um desempenho que não seja satisfatório, em que o sistema não consegue ser capaz de controlar os níveis dos silos.

Figura 8 – Teste Custo Míope e Passos: custo ao longo das iterações

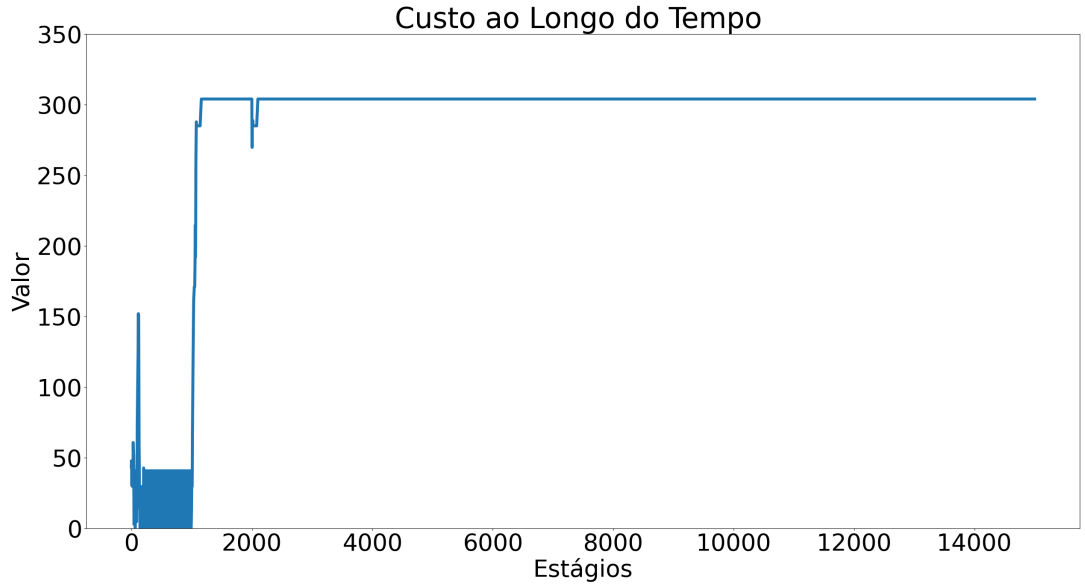


Figura 9 – Teste Custo Míope e Passos: erro ao longo das iterações

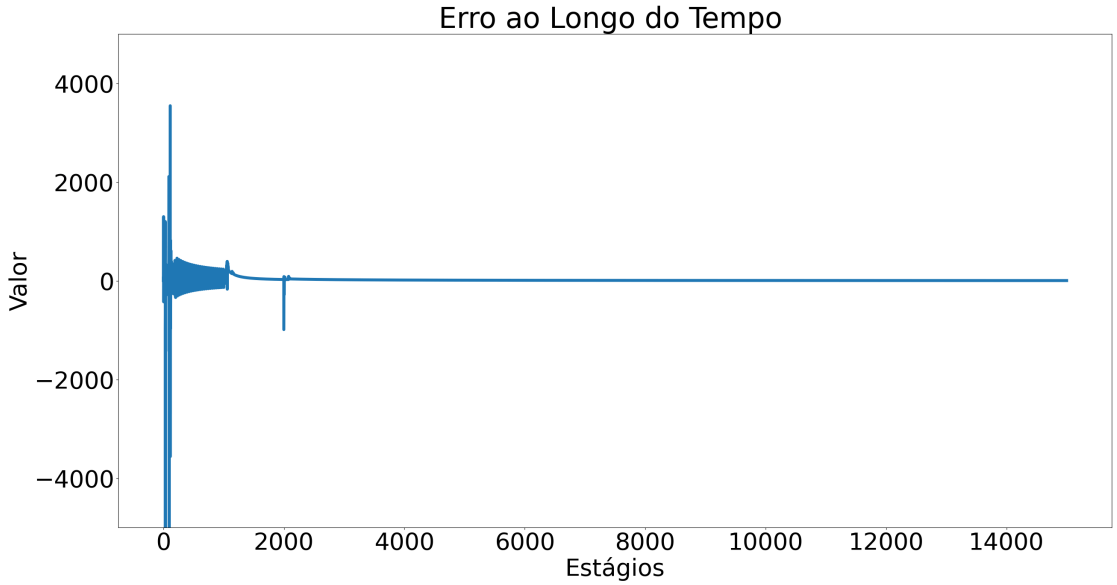


Figura 10 – Teste Custo Míope e Passos: coeficiente θ_1 ao longo das iterações

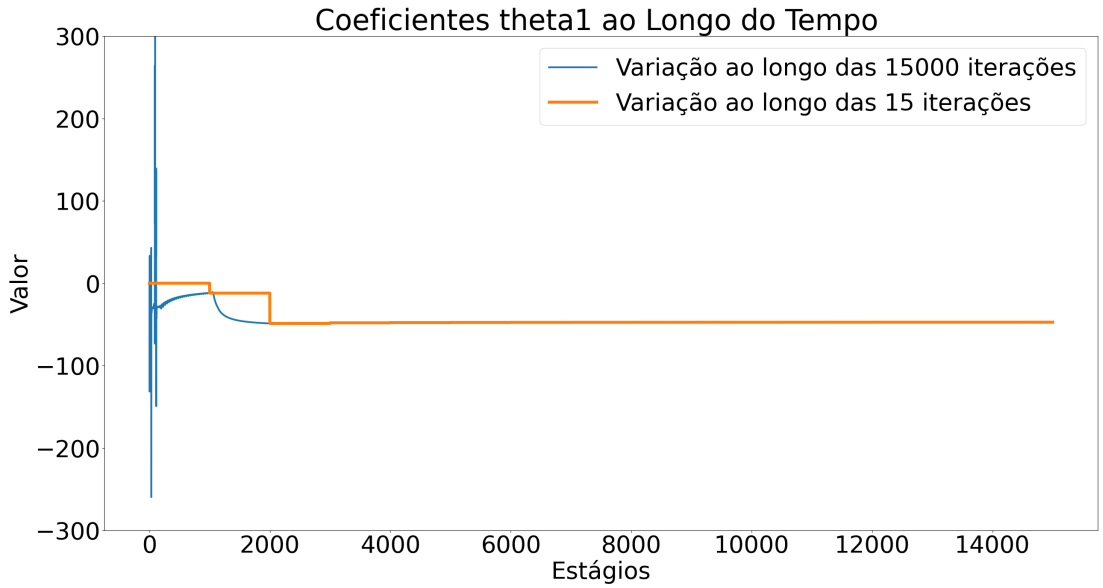
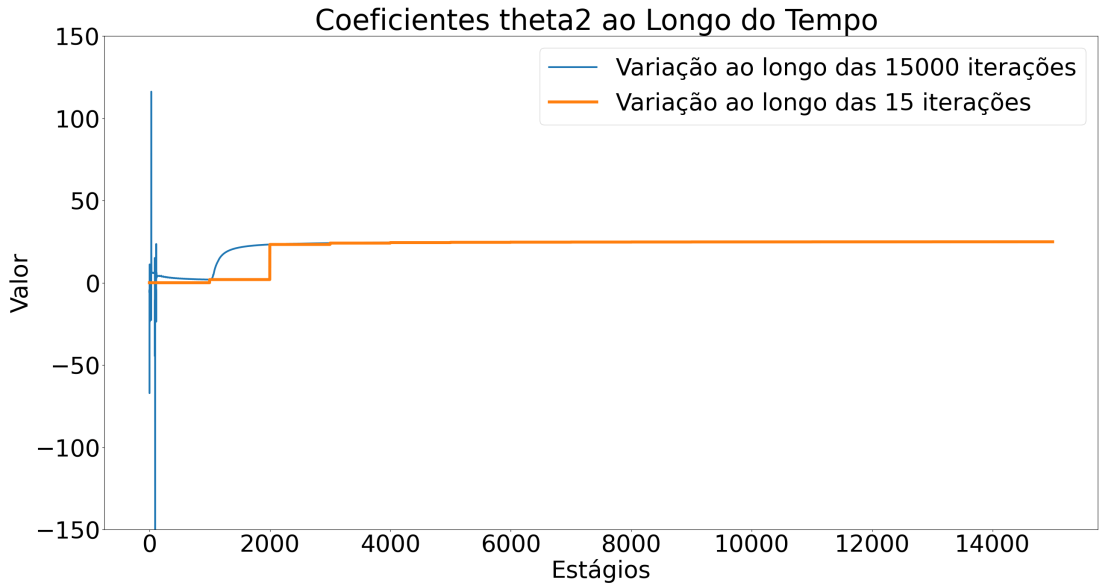


Figura 11 – Teste Custo Míope e Passos: coeficiente θ_2 ao longo das iterações



5.2 Simulação 1

A Simulação 1 consiste em utilizar a política resultante da etapa de treinamento do Algoritmo de Iteração de Política Aproximado *LSTD* na execução do algoritmo de

programação dinâmica desenvolvido para a resolução do problema utilizando-se a instância apresentada anteriormente.

Ou seja, a partir da política refinada e da instância *16_60_1_teste.xml*, a cada estágio escolhe-se qual movimento o *tripper* deve realizar, calcula-se o custo, e então atualiza-se os níveis de minério armazenado em todos os silos e suas respectivas folgas.

Foram realizadas 100 iterações, sendo que a cada iteração os níveis de minério dos silos foram armazenados. Como saída, gerou-se um arquivo no formato *.csv*, que armazena o custo, os coeficientes de cada uma das funções indicadoras utilizadas, os níveis de cada silo e a posição do *tripper* a cada iteração.

Para que uma análise gráfica pudesse ser realizada, os níveis dos silos armazenados a cada iteração foram exibidos ao fim da simulação, conforme mostram as figuras 12 e 13, correspondentes à simulação utilizando-se as políticas geradas pelo treinamento dos Teste Custo Míope e Ciclo Mínimo e Teste Custo Míope e Passos, utilizados como exemplo anteriormente.

Figura 12 – Teste Custo Míope e Ciclo Mínimo: níveis dos silos ao longo das iterações

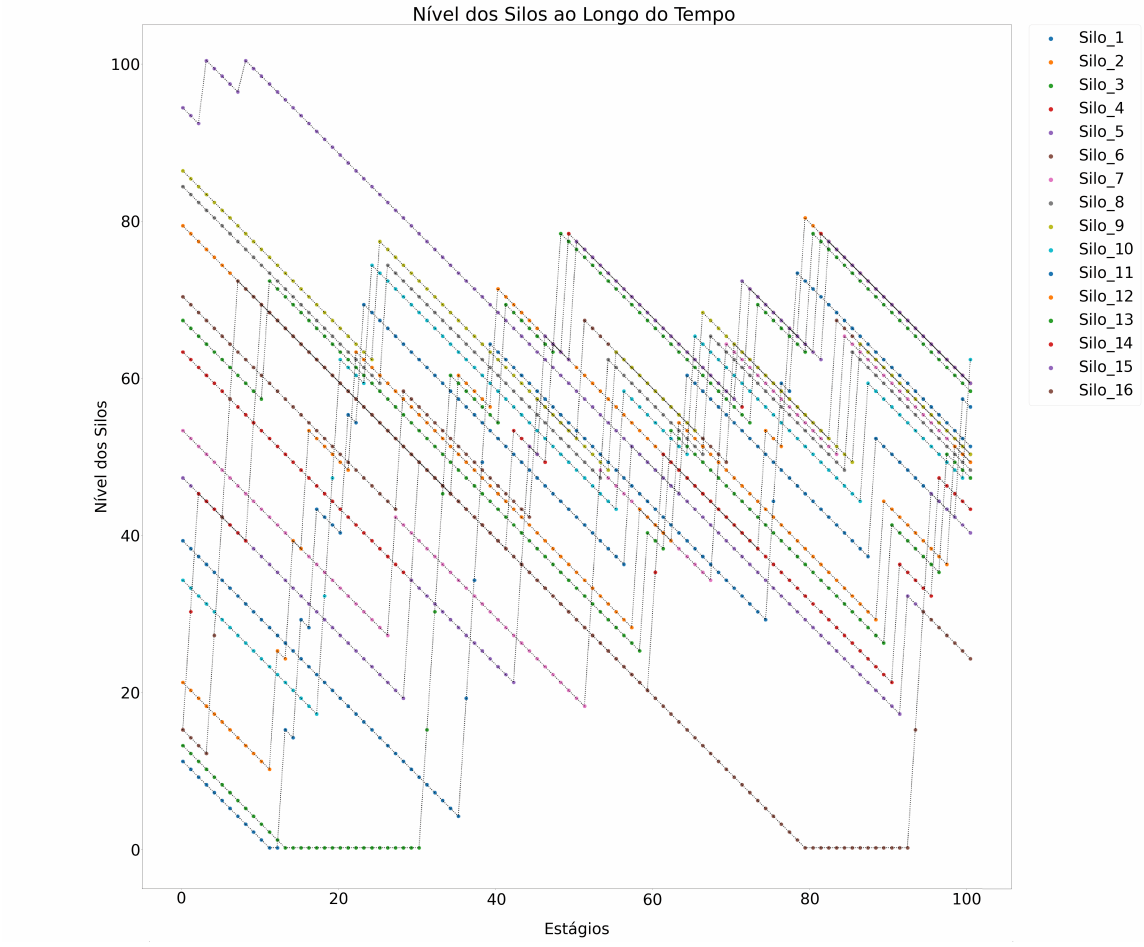
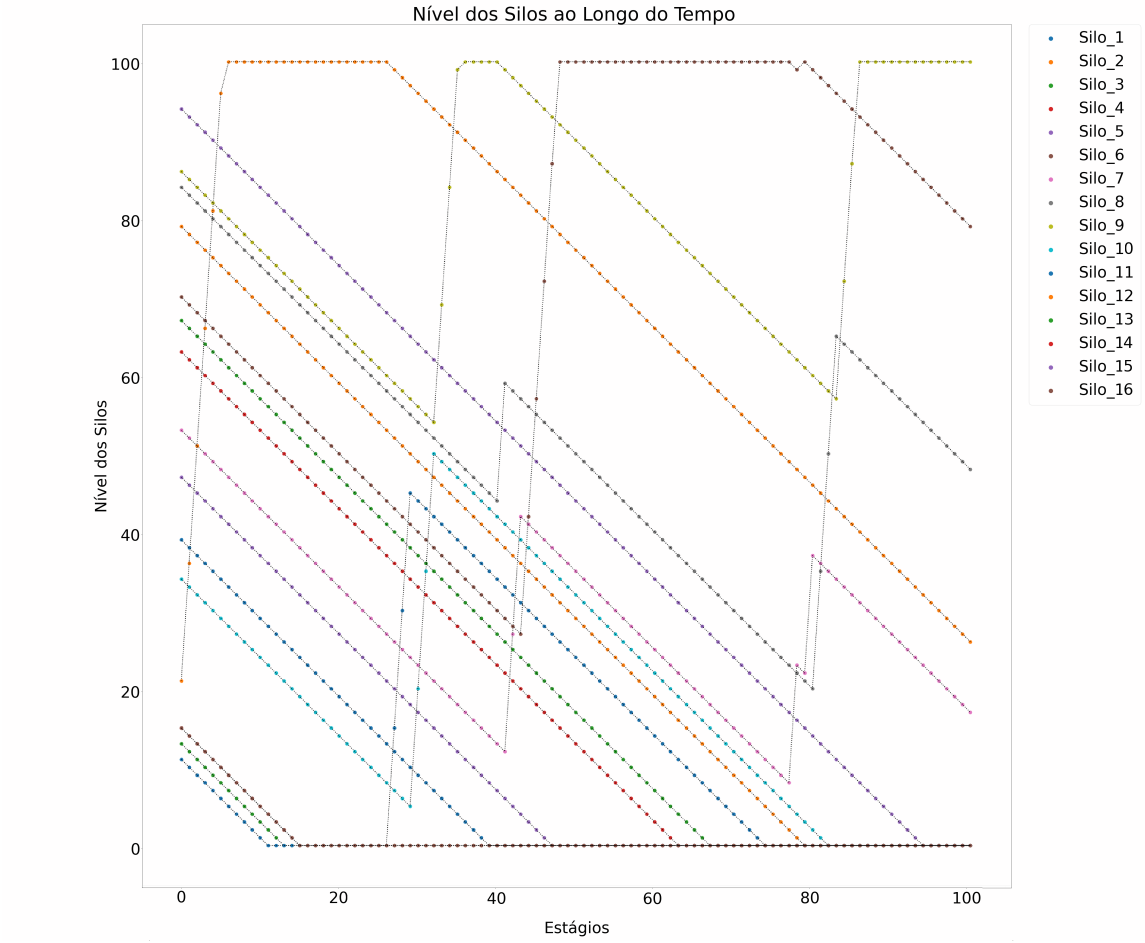


Figura 13 – Teste Custo Míope e Passos: níveis dos silos ao longo das iterações



A partir da Figura 12 pode-se observar que embora apresente alguns eventos indesejáveis, como a existência de silos em sua capacidade máxima ou mínima, de forma geral, houve uma tendência de estabilização dos níveis dentro dos limites máximo e mínimo permitidos ao longo do tempo quando a política gerada pelo treinamento das funções indicadoras Custo Míope e Ciclo Mínimo é utilizada.

Já a Figura 13 mostra que a política resultante do treinamento das funções indicadoras Custo Míope e Passos não é eficiente nem no critério de estabilização dos níveis ao longo do tempo, nem em evitar que os silos atinjam sua capacidade máxima ou mínima. Conforme exibido pelo gráfico, vários silos tiveram sua capacidade atingida, e o mais grave, mesmo essa condição sendo atingida, não foi possível revertê-la de forma imediata.

Dessa forma, conclui-se que a utilização das funções indicadoras Custo Míope e Passos não é apropriada para a instância, visto que o resultado apresentando ficou muito aquém do esperado. Por outro lado, a utilização das funções indicadoras Custo Míope e Ciclo Mínimo se apresenta como candidata à resolução do problema, uma vez que

apresentou resultados satisfatórios.

Para analisar a eficiência de todos os testes realizados, a partir do arquivo de saída gerado para cada teste calculou-se o custo total, a soma dos custos de cada uma das 100 iterações realizadas, e o desvio padrão total, constituído pela soma dos desvios padrões dos níveis dos silos a cada iteração. O custo permite analisar a eficiência dos testes por meio das folgas produzidas, enquanto que o desvio padrão permite analisar a homogeneidade dos níveis dos silos. Ademais, durante a realização de cada teste, computou-se e armazenou-se o seu tempo de simulação.

A Tabela 5 apresenta os resultados obtidos pela Simulação 1, provenientes da execução de todos os testes, em que são exibidos os valores iniciais dos coeficientes das funções indicadoras utilizadas, o tempo de execução da simulação (em segundos), o custo total e o desvio padrão total. As colunas CM, P, DP, Pup, CI e CMin representam as funções Custo Míope, Passos, Desvio Padrão, Passos up, Caminho Ideal e Ciclo Mínimo, respectivamente, enquanto que a coluna DP total refere-se ao desvio padrão total. Já as linhas resetB-cte, resetB-dn, resetB-up, B-cte, B-dn, B-up correspondem aos testes da Categoria 1, enquanto que CM, CM-DP, CM-CMin, CM-P, CM-P-DP, CM-P-DP-CMin, CM-P-DP-CMin-CI, CM-P-DP-CI, CM-Pup e CM-CI correspondem aos testes da Categoria 2. Além da tabela, os resultados são apresentados graficamente pela Figura 14 na mesma ordem em que são apresentados na Tabela 5, isto é, o Teste 1 do gráfico representa o teste resetB-cte, o Teste 2 representa o teste resetB-dn, e assim por diante.

Figura 14 – Resultados obtidos pela Simulação 1

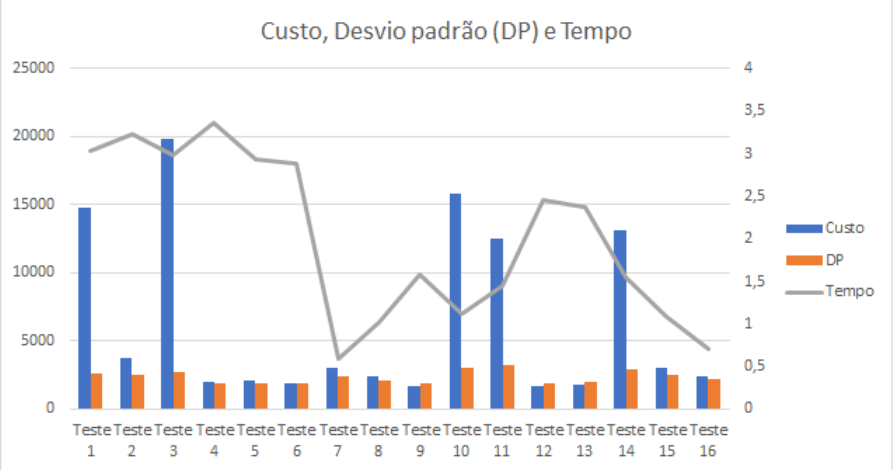


Tabela 5 – Resultados Simulação 1 para 100 períodos - Problema Determinístico

Teste	C M	P	D P	Pup	C I	C Min	Tempo(s)	Custo total	DP total
resetB-cte	-9,92	-0,14	2,96	-1,57	4,54	1,76	3,03	14787,70	2575,04
resetB-dn	-38,94	9,42	9,05	2,85	0,63	0,87	3,23	3734,45	2467,88
resetB-up	422,08	-9,79	-476,59	-17,89	229,82	8,70	2,98	19822,70	2711,05
B-cte	-9,18	8,54	-17,10	7,65	10,94	1,58	3,36	2025,40	1924,70
B-dn	9,09	4,86	-17,82	7,07	8,50	2,20	2,93	2074,80	1924,28
B-up	25,90	4,25	-20,29	15,75	8,68	1,87	2,88	1881,95	1908,06
CM	31,95	-	-	-	-	-	0,59	3023,85	2466,16
CM-DP	-17,57	-	62,00	-	-	-	1,02	2394,00	2087,59
CM-CMin	***2,68***	**-**	**-**	**-**	**-**	***1,53***	***1,59***	***1731,85***	***1876,50***
CM-P	-47.30	24,89	-	-	-	-	1,12	15835,55	3034,48
CM-P-DP	55,72	17,82	-53,47	-	-	-	1,45	12490,60	3280,18
CM-P-DP-CMin	***18,66***	***2,58***	***-32,70***	**-**	**-**	***4,79***	***2,46***	***1710,00***	***1872,49***
CM-P-DP-CMin-CI	31,58	9,84	-36,74	-	2,30	2,09	2,38	1764,15	1989,95
CM-P-DP-CI	101,88	17,49	-102,77	-	19,31	-	1,55	13098,60	2959,70
CM-Pup	30,21	-	-	13,30	-	-	1,09	2988,70	2490,23
CM-CI	44,00	-	-	-	3,24	-	0,70	2432,00	2215,01

Pode-se observar que o tempo necessário para a realização de 100 iterações do algoritmo de resolução do problema, partindo-se da instância *16_60_1_teste.xml*, é consideravelmente pequeno, atingindo o valor máximo de 3,36 segundos no teste B-cte, no qual todas as funções indicadoras são utilizadas. Ou seja, para que os próximos 100 movimentos a serem realizados pelo *tripper* fossem definidos, foram necessários apenas 3 segundos, aproximadamente, no pior caso. Além disso, constata-se que o tempo de execução é diretamente proporcional à quantidade de funções indicadoras utilizadas, como esperado.

No que tange ao custo total, verifica-se que há uma variabilidade entre os resultados obtidos, o que proporciona a comparação do desempenho das combinações de funções utilizadas. O menor custo, 1710, foi obtido pelo teste CM-P-DP-CMin, seguido pelo teste CM-CMin, cujo custo total foi de 1731,85. Entretanto, nota-se os testes resetB-cte, resetB-up, CM-P, CM-P-DP e CM-P-DP-CI apresentam valores muito discrepantes dos habituais, entre 12490,6 e 19822,7, indicando que foram ocasionadas um número elevado de folgas, isto é, que os limites dos silos foram extrapolados diversas vezes. Dessa maneira, infere-se que tais combinações das funções propostas não são eficientes para a resolução do problema.

Com relação ao desvio padrão, observa-se que o melhor resultado foi obtido pelo teste CM-P-DP-CMin, enquanto que o teste CM-P-DP apresentou o pior resultado. Analisando-se os custos e os desvios padrões de cada teste simultaneamente, percebe-se que, de forma geral, os testes que apresentam os menores custos também apresentam os menores desvios padrões e vice-versa, ou seja, os testes que são mais eficientes em manter os níveis dos silos dentro dos limites estabelecidos tendem a equilibrar os níveis mais rapidamente ao longo do horizonte de tempo.

Dentre a Categoria 1 de testes, nota-se que nos quais a função resetB não é utilizada, ou seja, cujos valores da matriz B são mantidos a cada iteração, os resultados são muito superiores aos dos testes que utilizam a função em ambos os quesitos custo e desvio padrão. Cabe ressaltar que dois dos três testes em que a função resetB é utilizada, resetB-cte e resetB-up, apresentaram custos muito elevados, destoando-se dos demais. Dessa forma, conclui-se que a função resetB não apresenta bom desempenho.

No que diz respeito as funções para cálculo do coeficiente λ, observa-se que duas funções obtiveram performance oposta quanto à utilização ou não da função resetB. A função stpsze_ln50up100 obteve o melhor desempenho no teste que não utiliza a função resetB e o pior desempenho no teste em que a função é utilizada. Em contrapartida, a função stpsze_ln100dn80 apresentou o melhor resultado quando a função resetB é utilizada e o pior resultado no teste em que não é utilizada. Sendo assim, não é possível concluir genericamente como a atribuição de prioridades aos estágios deve ser realizada.

Analisando-se a Categoria 2 dos testes, observa-se que o teste CM-P-DP-CMin

apresenta o melhor desempenho em ambos os quesitos. Já o pior desempenho no quesito custo foi obtido pelo teste CM-P, enquanto que o pior desempenho no quesito desvio padrão foi obtido pelo teste CM-P-DP.

Para a comprovação da efetividade das combinações de funções utilizadas, propôs-se utilizar estados gerados de forma aleatória ao invés de utilizar a instância *16_60_1_teste.xml*, a fim de verificar se os desempenhos apresentados são decorrentes das combinações realizadas ou do estado inicial utilizado. Dessa forma, realizou-se uma nova simulação, apresentada na seção 5.3.

5.3 Simulação 2

A Simulação 2 visa testar a capacidade de generalização do treinamento realizado por cada teste, e consiste em gerar estados iniciais de forma aleatória, contendo os níveis dos silos e a posição inicial do *tripper*, calcular o seu custo e executar o algoritmo de programação dinâmica a partir da política criada pelo treinamento do respectivo teste por um determinado número de iterações para escolher, a cada iteração, o próximo movimento a ser realizado pelo *tripper*, calcular o custo de realizar o movimento e realizar a transição para o próximo estado.

Em cada teste, foram gerados 50 estados aleatoriamente e realizadas 100 iterações do algoritmo para cada estado. De forma análoga à Simulação 1, gerou-se um arquivo no formato *.csv* para cada teste realizado, contendo o custo, os coeficientes das funções indicadoras utilizadas, os níveis dos silos e a posição do *tripper* a cada iteração.

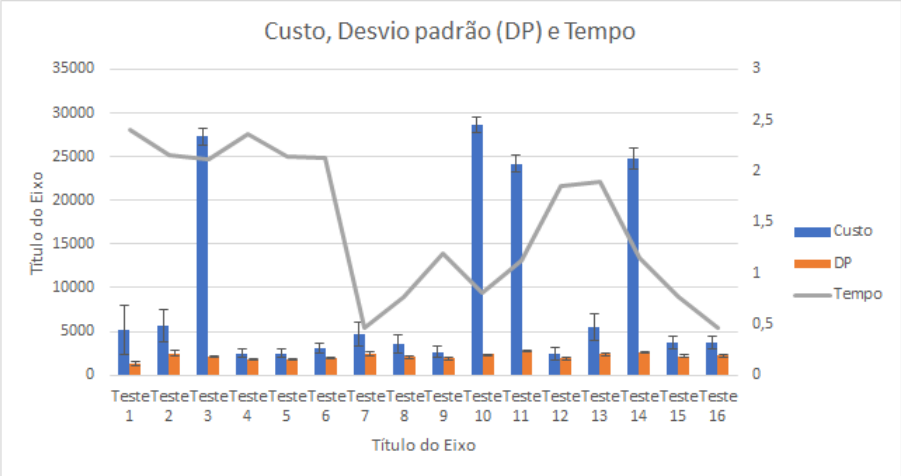
Para fins de análise, calculou-se para cada teste o custo médio, definido como a média do custo total dos 50 estados gerados, e o desvio padrão médio, calculado como a média dos desvios dos estados, que por sua vez é calculado como a soma dos desvios dos níveis dos silos a cada iteração. Além disso, ao final da realização dos testes, armazenou-se o tempo necessário à sua execução.

Na Tabela 6 são apresentados os coeficientes das funções indicadoras utilizadas pelos testes, o tempo médio de simulação de cada teste em segundos, o custo médio e o desvio padrão médio dos testes realizados. Similarmente à Tabela 5, as linhas correspondem aos testes realizados, enquanto que as colunas representam os variáveis analisadas. Já na Figura 15, os resultados são apresentados de forma gráfica, em que a margem de erro foi calculada utilizando-se um nível de confiança de 95% e os resultados obtidos pelos testes foram apresentados sequencialmente, conforme a ordem da Tabela 6.

Tabela 6 – Resultados Simulação 2 para 100 períodos - Problema Determinístico

Teste	C M	P	D P	Pup	C I	C Min	Tempo médio (s)	Custo médio	DP médio
resetB-cte	-9,92	-0,14	2,96	-1,57	4,54	1,76	2,40	5127,63	1364,09
resetB-dn	-38,94	9,42	9,05	2,85	0,63	0,87	2,16	5596,46	2580,82
resetB-up	422,09	-9,79	-476,59	-17,89	229,82	8,70	2,12	27310,35	2239,33
B-cte	-9,18	8,54	-17,10	7,65	10,94	1,58	2,37	2496,42	1902,16
B-dn	***9,09***	***4,86***	***-17,82***	***7,07***	***8,50***	***2,20***	***2,15***	***2493,74***	***1875,11***
B-up	25,90	4,25	-20,29	15,75	8,68	1,87	2,13	3128,93	1976,62
CM	31,95	-	-	-	-	-	0,47	4676,68	2549,07
CM-DP	-17,57	-	62,00	-	-	-	0,76	3580,98	2147,49
CM-CMin	2,68	-	-	-	-	1,53	1,19	2664,53	1999,20
CM-P	-47.30	24,89	-	-	-	-	0,81	28612,80	2433,56
CM-P-DP	55,72	17,82	-53,47	-	-	-	1,13	24169,38	2890,12
CM-P-DP-CMin	***18,66***	***2,58***	***-32,70***	**-**	**-**	***4,79***	***1,85***	***2408,75***	***1991,93***
CM-P-DP-CMin-CI	31,58	9,85	-36,75	-	2,30	2,09	1,90	5438,86	2511,77
CM-P-DP-CI	101,88	17,49	-102,77	-	19,31	-	1,15	24781,54	2695,68
CM-Pup	30,21	-	-	13,30	-	-	0,76	3738,60	2219,60
CM-CI	44,00	-	-	-	3,24	-	0,47	3695,04	2308,80

Figura 15 – Resultados obtidos pela Simulação 2



No que se refere à Categoria 1 dos testes, pode-se observar, por meio dos resultados obtidos, que o teste B-dn apresenta o menor custo e o menor desvio padrão, o que leva à conclusão de que a utilização da função stpsze_ln100dn80 e a não utilização da função resetB fazem com que os limites dos silos sejam ultrapassados menos vezes e que os níveis sejam mais equilibrados ao longo das iterações. Em contrapartida, o teste resetB-up apresenta os piores resultados tanto para o custo quanto para o desvio padrão.

Dentre as funções para o cálculo do coeficiente λ implementadas, pode-se verificar que a função stpsze_ln50up100 apresenta o pior desempenho, indicando que considerar o histórico de observações não é a melhor alternativa, enquanto que a função stpsze_ln100dn80, por apresentar os melhores resultados, indica que se deve priorizar as observações mais recentes. No que se refere à matriz B, os resultados mostram que quando seu valor não é alterado, ou seja, quando não é resetada como uma matriz identidade, os níveis obtidos são mais homogênos e são produzidas menos folgas.

No que se refere à Categoria 2 dos testes, constata-se que o teste CM-P-DP-CMin apresenta os melhores resultados em ambos os quesitos custo e desvio padrão. Já o teste CM-P apresenta o maior custo, enquanto que o teste CM-P-DP apresenta o pior desvio padrão. Entretanto, de forma geral, o comportamento dos testes é o mesmo para o custo e para o desvio padrão, ou seja, não há um teste que apresente um custo alto e um desvio padrão baixo, por exemplo.

Comparando-se com os resultados obtidos pela Simulação 1, os testes CM-P, CM-P-DP, CM-P-DP-CI e resetB-up também apresentam o mesmo comportamento em que o custo é muito elevado, destoando-se dos demais, indicando a ocorrência de folgas em muitas iterações. Entretanto, diferentemente do ocorrido na Simulação 1, na Simulação 2 o teste resetB-cte não apresentou essa discrepância de valor, gerando um resultado razoável. Sendo assim, pode-se perceber que o teste em questão é sensível ao estado utilizado, ou seja, pode apresentar comportamento bom ou ruim de acordo com o estado inicial.

Dessa forma, pode-se concluir que o teste resetB-cte é o único em que a eficiência é dependente do estado inicial. Além disso, de modo geral, os testes apresentaram desempenhos semelhantes em ambas as simulações, sendo que os testes CM-P-DP-CMin e CM-CMin apresentam os melhores resultados para todos os estados analisados, enquanto os testes CM-P e resetB-up apresentam os piores resultados.

Cabe ressaltar ainda que as diferentes combinações das funções implementadas influenciam diretamente na performance, fato comprovado pela variabilidade dos resultados. Por exemplo, determinadas combinações das funções indicadoras, utilizando a função λ stpsze_cte1, podem ser melhores do que utilizar todas as funções indicadoras simultanemamente, independentemente da função λ utilizada. Tal característica demonstra a importância das funções indicadoras, e ressalta a importância de que sejam modeladas e selecionadas de forma eficiente.

5.4 Comparação com os resultados de Caldas e Martins (2018)

Para que a qualidade dos resultados obtidos pudesse ser analisada, comparou-se o desempenho dos testes propostos por esse trabalho com o do trabalho proposto por Caldas e Martins (2018). Os resultados obtidos por Caldas e Martins (2018) são exibidos na Tabela 7, em que a coluna I representa a instância utilizada, $t(s)$ representa o tempo de execução em segundos e A/B representa a quantidade de folgas encontradas.

Tabela 7 – Resultados obtidos por Caldas e Martins (2018)

I	**t(s)**	**A/B**
1	1806,66	17/0
2	646,80	10/0
3	3170,03	2/12
4	40593,50	0/0
5	46315,20	0/0

As instâncias utilizadas possuem os mesmos níveis da instância utilizada por este trabalho, mas em cada uma a posição do *tripper* é gerada aleatoriamente. No entanto, vale ressaltar que o presente trabalho realizou uma modificação na instância que não foi considerada no trabalho de Caldas e Martins (2018), acrescentando as capacidades máxima e mínima dos silos e estipulando-as como 100 e 0, respectivamente, além de alterar os limites superior e inferior de 100 e 0 para 80 e 20. Dessa forma, a ocorrência de folgas se deu de formas diferentes nos dois trabalhos, e por tal maneira a quantidade de folgas produzida não foi comparada.

Comparando-se os tempos de execução da simulação 2 dos testes realizados com os tempos de execução obtidos por Caldas e Martins (2018), pode-se concluir que os testes

propostos por este trabalho apresentam resultados significativamente inferiores aos de Caldas e Martins (2018), que variam entre 646,80 e 46315,20 segundos.

Todavia, os tempos da simulação referem-se à media do tempo necessário para executar 100 iterações da política resultante da etapa de treinamento do Algoritmo de Iteração de Política Aproximado *LSTD* para 50 estados, ou seja, o tempo de treinamento não é considerado. Levando em conta o tempo de treinamento, observa-se que seriam gastos 25048,19 segundos no pior caso, teste B-dn, valor ainda menor do que o obtido pelas execuções das instâncias 3 e 4 de Caldas e Martins (2018).

Ao se analisar os tempos de execução de forma geral, deve-se considerar que o tempo total de execução do método proposto por este trabalho, composto pelo tempo de treinamento e pelo tempo de simulação, só deve ser levado em consideração uma única vez, pois uma vez realizado o treinamento, só é necessária a realização da simulação. Já o modelo de Caldas e Martins (2018) apresenta o mesmo tempo para todas as vezes que for executado. Ou seja, enquanto o tempo de execução do modelo de Caldas e Martins (2018) é fixo para cada execução, o método proposto apresenta o custo fixo de treinamento apenas na primeira execução, sendo que nas execuções posteriores utiliza apenas o custo variável referente ao tempo de simulação, valor consideravelmente baixo.

Exemplificando-se, ao executar o algoritmo deste trabalho 10 vezes, utilizando o teste B-dn, seriam gastos por volta de 25074,53 segundos, aproximadamente 7 horas, em que 25045,26 seriam gastos pelo treinamento e o restante pelas 10 simulações. Por outro lado, para executar o modelo de Caldas e Martins (2018) 10 vezes, utilizando a instância *16_60_5*, seriam gastos 463152 segundos, equivalentes a 129 horas aproximadamente.

Dessa forma, ainda que o tempo total de determinado teste proposto por este trabalho seja próximo ao tempo obtido por alguma instância de Caldas e Martins (2018), essa proximidade nos valores só ocorre quando é realizada apenas uma execução dos algoritmos.

6 Modelo 2 - Modelo Estocástico para o Problema de Movimentação do *tripper*

6.1 Descrição do Problema

Foram realizadas algumas adaptações no modelo determinístico para que ele possa representar melhor a situação real, retratando eventos que geralmente ocorrem nas plantas de beneficiamento de minério e que não foram considerados no modelo anterior.

Primeiramente, desconsiderou-se a restrição de movimentação imposta ao *tripper*. Dessa forma, o equipamento pode se locomover para qualquer silo ao longo da cadeia, independentemente de sua posição.

Assumiu-se que o fluxo de minério não é interrompido durante o processo de locomoção do *tripper*. Sendo assim, durante o deslocamento, ele deposita uma certa quantidade de minério sobre os silos que estão em sua trajetória, sendo a quantidade depositada dada em função da taxa de entrada e do tempo que o *tripper* gasta para percorrer um silo.

Foram incorporados ao modelo o tempo de deslocamento que o *tripper* gasta para se locomover do silo atual até o silo escolhido e o tempo de relógio do k-ésimo estágio. Considerou-se também uma nova decisão, o tempo em que o *tripper* permanecerá alimentando o silo escolhido a cada estágio. Além disso, foram consideradas incertezas nas taxas de entrada e de saída de minério dos silos, ou seja, considerou-se que tais taxas poderão assumir valores variáveis a cada estágio de acordo com uma determinada distribuição de probabilidade.

A Tabela 8, exibida a seguir, apresenta os parâmetros de entrada e as variáveis do modelo estocástico desenvolvido para o problema de movimentação do *tripper*, em que diferentemente do modelo determinístico, T_e e T_s representam, respectivamente, as distribuições de probabilidade e seus parâmetros que originam as taxas de entrada e de saída de minério dos silos a cada estágio, δ_{ij} representa o tempo necessário para o *tripper* locomover-se entre o silo atual i e o silo escolhido j, τ_k é a variável de estado que corresponde ao tempo de relógio do k-ésimo estágio, isto é, o instante de tempo em que o *tripper* iniciará seu movimento, e $d(x^k)$ equivale à variável de decisão que representa o tempo que o *tripper* irá permanecer sobre o silo, alimentando-o.

Tabela 8 – Parâmetros do problema estocástico

Parâmetro	**Descrição**
J	Conjunto de silos
γ_1	Coeficiente folga de excesso
γ_2	Coeficiente folga de falta
l_{max}	Limite máximo do Silo
l_{min}	Limite mínimo do Silo
cap_{max}	Capacidade máxima do silo
cap_{min}	Capacidade mínima do silo
P	Posição do *tripper*
V_j	Volume do silo j
T_e	Taxa de entrada dada por incerteza
T_s	Taxa de saída dada por incerteza
δ_{ij}	Tempo de deslocamento entre as posições i e j
Variável de Decisão	**Descrição**
x^k	Movimento a ser realizado pelo *tripper*
$d(x^k)$	Duração da alimentação do silo

Assim como no modelo determinístico, estipula-se que a velocidade do *tripper* é constante, e que o equipamento deposita a cada estágio toda a taxa de entrada no silo escolhido.

O objetivo do modelo estocástico consiste em determinar a sequência de movimentos ótimas e suas respectivas durações, a fim de que seja produzido o mínimo de folgas possível, por meio do estado atual do sistema e das predições das ações futuras, tomadas ao longo do horizonte de tempo.

Para tanto, a cada estágio são analisados todos os silos por meio do cálculo do custo, dado em função das folgas, e do tempo ótimo de duração da alimentação. A tomada de decisão é realizada com base no custo, ou seja, é escolhido o silo cujo nível mais se enquadre dentro de seus limites.

Os métodos de solução desenvolvidos para o problema são apresentados pelas próximas seções. São eles: Método Determinístico Baseado MILP, Método Zigue Zague e Método Dinâmico Estocástico. Como todos eles utilizam o mesmo método para o cálculo do tempo de alimentação do silo, tal método é apresentado na seção 6.5.

6.2 Método Determinístico Baseado MILP

Foi desenvolvido um modelo determinístico, isto é, sem a presença de incertezas, sob a forma de Programação Linear Inteira Mista para ser utilizado como método de resolução do problema.

Com base no modelo determinístico, foram criadas duas políticas de solução: na primeira, o modelo determinístico é executado uma única vez, gerando as ações de todas as etapas seguintes, enquanto que a segunda política implementa um horizonte de planejamento rolante, em que ao fim de cada etapa o modelo é executado para decidir as ações a serem tomadas na próxima etapa.

As variáveis e o modelo desenvolvido são apresentados a seguir.

Tabela 9 – Variáveis Modelo Determinístico Baseado MILP

Variável	**Descrição**
X_{ij}^t	Variável binária que indica se o *tripper* se locomoveu do silo i para o j no instante t
A_j^t	Folga de excesso do silo j no instante t
B_j^t	Folga de falta do silo j no instante t
d_t	Duração da alimentação do silo no instante t
V_j^t	Volume do silo j no instante t
dX_j^t	Variável de linearização

O modelo é dado por:

$$\min \sum_{t \in T} \sum_{j \in J} \left(\gamma_1 A_j^t + \gamma_2 B_j^t \right) \tag{6.1}$$

Subject to:

$$\sum_{j \in J: j <> p} X_{pj}^1 = 1 \tag{6.2}$$

$$\sum_{i \in J} \sum_{j \in J: j <> i} X_{ij}^t = 1, \qquad \forall t \in T : t \geq 2 \tag{6.3}$$

$$\sum_{j \in J: j <> i} X_{ij}^t - \sum_{j \in J: j <> i} X_{ji}^{t-1} = 0, \qquad \forall i \in J, \forall t \in T : t \geq 2 \tag{6.4}$$

$$A_j^t \geq V_j^t - L^{max}, \qquad \forall t \in T, \forall j \in J \tag{6.5}$$

$$B_j^t \geq L^{min} - V_j^t, \qquad \forall t \in T, \forall j \in J \tag{6.6}$$

$$V_k^t = V_k^{t-1} - \sum_{i \in J} \sum_{j \in J: j <> p} X_{ij}^t (\delta_{ij} \overline{T_s} + C(k,i,j) T_p) + dX_k^t \overline{T_e} - \overline{T_s} d^t, \qquad \forall t \in T, \forall k \in J \tag{6.7}$$

$$dX_k^t \leq D^u \sum_{i \in J: i <> k} X_{ik}^t, \qquad \forall t \in T, \forall k \in J \tag{6.8}$$

$$d^t - dX_k^t \leq D^u (1 - \sum_{i \in J: i <> k} X_{ik}^t), \qquad \forall t \in T, \forall k \in J \tag{6.9}$$

$$d^t \geq dX_k^t, \qquad \forall t \in T, \forall k \in J \tag{6.10}$$

$$V_k^t \geq cap^{min}, \qquad \forall t \in T, \forall k \in J \tag{6.11}$$

$$V_k^t \leq cap^{max}, \qquad \forall t \in T, \forall k \in J \tag{6.12}$$

$$d^t \geq 0.5, \qquad \forall t \in T \tag{6.13}$$

A função objetivo, expressa pela Equação 6.1, permanece a mesma, ou seja, minimizar as folgas produzidas ao longo do horizonte de tempo.

No que diz respeito as restrições, a Equação 6.2 assegura que no instante inicial o *tripper* saia de sua posição inicial e se locomova até outro silo, e a Equação 6.3 garante que o *tripper* esteja sobre somente um silo a cada período de tempo. Já a Equação 6.4 estabelece o fluxo de movimentação, em que o *tripper* deve partir de sua posição atual para uma nova posição. As equações 6.5 e 6.6 são responsáveis pela definição das folgas, enquanto a Equação 6.7 atualiza os níveis dos silos a cada instante de tempo. Como na Equação 6.7 ocorre uma não linearidade, as equações 6.8, 6.9 e 6.10 tem o objetivo de realizar a linearização. Para tanto, a equação 6.8 assegura que a multiplicação da variável binária X_k^t por d esteja limitada ao valor teto de d se o silo está ativado, isto é, se o *tripper* se encontra naquela posição, e que esteja limitada a zero caso o silo não esteja ativado. A Equação 6.9 garante que a diferença entre o tempo d^t, a duração da alimentação, e o tempo dX_k^t esteja limitada ao valor teto de d multiplicado pela diferença entre 1 e o acionamento do silo, o que implica que se o silo estiver ativado a diferença deve ser zero. Já a Equação 6.10 impõe que d^t seja sempre maior ou igual à dX_k^t. Dessa forma, a Equação 6.8 estabelece que o dX_k^t de todos os silos deve ser zero, exceto o do silo ativado, que pode ser maior do que zero, e as equações 6.9 e 6.10 estabelecem que o d^t seja exatamente igual ao dX_k^t do silo ativado. As equações 6.11 e 6.12 limitam os níveis dos silos as suas capacidades mínima e máxima, respectivamente. Por fim, a Equação 6.13 estabelece que a duração mínima da alimentação dos silos.

O modelo foi implementado utilizando-se *gurobipy*, a interface *Gurobi Python*, e resolvido pelo *Gurobi*, um *solver* de otimização matemática capaz de resolver problemas de programação linear inteira mista. A partir do modelo, foram gerados dois métodos de solução para o problema, apresentados a seguir.

6.2.1 Modelo Determinístico Baseado MILP - Política 1

Consiste em implementar no algoritmo de programação dinâmica desenvolvido uma política de solução cujo processo de tomada de decisão é estático, ou seja, uma vez que o processo de tomada de decisão é realizado, as decisões não podem ser modificadas.

No primeiro estágio, o modelo Determinístico Baseado MILP é executado, passando-se como parâmetro o número de iterações do problema. Dessa forma, o modelo é executado uma única vez, e logo no primeiro estágio são gerados e salvos o movimento e o tempo de alimentação para todos os estágios seguintes. A partir do segundo estágio, é necessário apenas acessar as decisões salvas.

De acordo com as decisões geradas, em cada estágio as ações são executadas e é calculado o custo, apresentado pela Equação 4.6, dado pelas penalizações das folgas produzidas por cada silo. Então, as ações devem ser tomadas de forma a minimizar o custo total conforme a Equação 6.1, isto é, de forma que os limites dos silos sejam extrapolados com a menor frequencia possível.

6.2.2 Modelo Determinístico Baseado MILP - Política 2

Consiste em implementar uma política de horizonte rolante, isto é, em que o planejamento é refeito ao fim de cada etapa.

Neste método, o modelo Determinístico Baseado MILP é executado a cada estágio do problema, gerando as ações, o movimento a ser realizado pelo *tripper* e o tempo que o *tripper* deve permanecer sobre o silo escolhido, do estágio seguinte baseadas em informações atuais do sistema. A partir das ações tomadas, calcula-se o custo de cada estágio pela Equação 4.6, sendo que as ações devem ser tomadas de forma a minimizar o total de folgas produzidas ao longo do horizonte de tempo.

Por realizar um replanejamento ao fim de cada estágio, espera-se que o desempenho obtido seja superior ao desempenho do Modelo Determinístico Baseado MILP - Política 1, que realiza o planejamento para todo horizonte de tempo baseado apenas nas informações iniciais. Tal suposição é verificada por meio da realização de testes, apresentados a seguir.

6.3 Método Zigue Zague

Consiste em fazer com que o *tripper* siga um caminho pré-estabelecido, ao invés do próprio algoritmo aprender e decidir de forma autônoma qual movimento deve ser realizado, e calcular quanto tempo o equipamento deve permanecer sobre o silo escolhido.

O posicionamento do *tripper* ao longo do horizonte de tempo deve ser semelhante à um zigue-zague. Tal situação foi proposta devido ao fato de que essa sequência de movimentos resulta em um controle razoável de acordo com o estado inicial em um ambiente sem a presença de incertezas, uma vez que deposita sequencialmente a mesma quantidade de minério em cada silo ao passo que retira de cada silo a mesma quantidade de minério.

Dessa forma, pode-se verificar se o desempenho da estratégia em um ambiente estocástico é similar ao desempenho em um ambiente determinístico. O Algoritmo 6 apresenta o pseudocódigo da política Zigue-Zague proposta.

A cada estágio, a escolha da próxima posição do equipamento e do sentido de movimentação é realizada por meio da função *next*, que recebe como parâmetros a posição atual do *tripper* e o sentido atual de movimentação, esquerda ou direita, e calcula a próxima posição. Se o *tripper* se encontra sobre o primeiro ou sobre o último silo, seu sentido de movimentação é alterado, e a próxima posição é calculada. O tempo d que o *tripper* deve permanecer sobre o silo é então calculado pelo método descrito ao fim deste capítulo, com o auxílio do módulo *least_squares* da biblioteca *scipy.optimize* do *Python* e das derivadas. Por fim, a posição atual e o sentido atual de movimentação são então

atualizados para serem utilizados no estágio seguinte.

Algoritmo 6: *Algoritmo de Movimentação Zigue-Zague*

função next(j^k, *sentido*)
$j = j^k$ + 2**sentido* - 1
se *j>|J| ou j<1* **então**
    sentido = 1 - sentido
    $j = j^k$ + 2**sentido* - 1
fim
Saída: sentido, j
incialize o estágio k
incialize a posição j^k
incialize o sentido de movimentação $sentido^k$
enquanto k **faça**
    $sentido^{k+1}$, j^{k+1} = next(j^k, $sentido^k$)
    calcular d
    $sentido^k = sentido^{k+1}$
    $j^k = j^{k+1}$
fim

A partir da escolha do próximo silo a ser visitado e da duração de sua alimentação, o custo produzido pelas ações é calculado a cada estágio por meio da soma das penalizações das folgas de cada silo, de forma semelhante ao cálculo da função objetivo do Modelo 1, exibido pela Equação 4.6.

6.4 Método Dinâmico Estocástico

Neste método, assim como no Modelo 1, a resolução do problema se dá por aproximação de programação dinâmica e pela utilização de funções indicadoras no Algoritmo de Iteração de Política Aproximado *LSTD*. Adicionalmente, é necessário um método para o cálculo da duração da alimentação do silo a ser utilizado pelo algoritmo, apresentado pela seção 6.5.

A seguir são apresentadas as alterações realizadas no Modelo 1 que resultam no novo modelo estocástico. Logo após, são apresentadas as funções indicadoras desenvolvidas para a resolução do problema. Como o algoritmo de aproximação de programação dinâmica é o mesmo do modelo determinístico, para fins de simplificação ele não foi apresentado novamente.

6.4.1 Modelo

6.4.1.1 Variável de Estado

A Equação 6.14 apresenta a nova variável de estado, em que considera-se que o tempo de relógio $\tau^{(k)}$ também é uma informação necessária à tomada de decisão. Dessa forma, a variável de estado é formada pela posição $P^{(k)}$ do *tripper* no k-ésimo estágio e pelos níveis $\mathrm{V}_j^{(k)}$ de cada silo $j \in J$ presentes no modelo anterior, e pelo acréscimo do tempo de relógio da k-ésima decisão $\tau^{(k)}$.

$$S^{(k)} = \begin{pmatrix} P^{(k)} \\ V_j^{(k)}, \forall j \in J \\ \tau^{(k)} \end{pmatrix} \tag{6.14}$$

6.4.1.2 Incerteza

Considerando-se o problema de movimentação do *tripper*, existem diferentes fontes de incertezas a serem consideradas no processo, sendo que as tratadas no presente trabalho são:

1. Incerteza na taxa de entrada, ou seja, na quantidade de minério que vai ser depositada pelo *tripper* no silo a cada estágio;
2. Incerteza na taxa de saída, isto é, na quantidade de minério que será retirada do silo a cada estágio.

A incerteza é representada no modelo por $\omega^{(k)}$, e sua realização determina os valores das taxas de entrada e de saída de minério dos silos a cada estágio de acordo com uma distribuição de probabilidade contínua, representadas por T_e e T_s. Assume-se que as distribuições de probabilidade são aplicadas a todos os silos a cada estágio.

6.4.1.3 Função de Transição

Como dito anteriormente, a função de transição é responsável por determinar a evolução do sistema do estado $S^{(k-1)}$ para o estado $S^{(k)}$ conforme as decisões $x^{(k)}$ e $d(x^{(k)})$ tomadas. No presente modelo, a função de transição é representada por $S^{(k)}\left(S^{(k-1)}, x^{(k)}, d(x^{(k)}), \omega^{(k)}\right)$, em que a decisão $x^{(k)}$ define o movimento a ser realizado pelo *tripper* e $d(x^{(k)})$ representa a duração da próxima alimentação do silo considerando a realização da incerteza $\omega^{(k)}$.

As equações a seguir são responsáveis por realizar a transição entre os estados, dada pela atualização dos valores da posição do *tripper*, dos níveis dos silos, da folga positiva, da folga negativa e do tempo de relógio, respectivamente.

$$P^{(k)} = p(x^{(k)}) \tag{6.15}$$

$$V_j^{(k)} = V_j^{(k-1)} - \delta(P^{k-1}, p(x^{(k)}))T_s + C_j^{(k)}T_p + \left(I_{[P_j]}T_e(\omega^{(k)}) - T_s(\omega^{(k)})\right) d(x^{(k)}) \tag{6.16}$$

$$A_j^{(k)} = \max\left\{V_j^{(k)} - cap_{max}, 0\right\} \tag{6.17}$$

$$B_j^{(k)} = \max\left\{cap_{min} - V_j^{(k)}, 0\right\} \tag{6.18}$$

$$\tau^{(k)} = \tau^{(k-1)} + \delta(P^{k-1}, p(x^{(k)})) + d(x^{(k)}) \tag{6.19}$$

A equação 6.15 determina uma nova posição para o *tripper*.

Na equação 6.16, os níveis $V_j^{(k)}$ de cada silo são atualizados pelo decréscimo da quantidade de minério que saiu do silo durante o processo de movimentação do *tripper*, pelo acréscimo de uma pequena taxa de minério T_p se o silo se encontra na trajetória a ser realizada pelo *tripper*, representada por $C_j^{(k)}$, e pelo decréscimo da taxa de saída (definida pela realização da incerteza ω) do estágio e pelo acréscimo da taxa de entrada (definida pela realização da incerteza ω) se o *tripper* se encontra sobre o silo corrente, situação representada por $I_{[P_j]}$, multiplicados pela duração da alimentação.

Já as folgas $A_j^{(k)}$ e $B_j^{(k)}$, a quantidade que extrapolou os limites superior e inferior respectivamente, são calculadas pelas equações 6.17 e 6.18, respectivamente. Por fim, a equação 6.19 atualiza o tempo de relógio de forma recursiva, adicionando à ele o tempo de deslocamento do *tripper* e a duração da alimentação no estágio.

6.4.1.4 Função Objetivo

Em um problema estocástico de minimização, o objetivo é minimizar o custo total de tomar as decisões no decorrer dos estágios, sujeito à uma incerteza.

O objetivo do modelo estocástico proposto continua sendo o objetivo do modelo determinístico, ou seja, minimizar as folgas, e é dado pela minimização do custo total, representado pela equação do retorno 6.20. Devido à presença de parâmetros aleatórios, o objetivo se torna minimizar o valor esperado do retorno total do processo, uma vez que a tomada de decisão é baseada na informação disponível e as novas informações se tornam disponíveis após a tomada de decisão. Sendo assim, a Equação 6.21 apresenta a

nova função objetivo do problema, dada em função do retorno total, em que $\mathbb{E}_\omega$ representa a esperança matemática para as incertezas ω.

$$R^{(k)}\left(S^{(k)}\left(S^{(k-1)}, x^{(k-1)}, d(x^{(k-1)}), \omega^{(k)}\right)\right) = \sum_{j \in J}\left(\gamma_1 A_j^{(k)} + \gamma_2 B_j^{(k)}\right) \tag{6.20}$$

$$\min_{x^{(k)} \in \chi^{(k)}} \left\{ \sum_{k=1}^{\infty} \mathbb{E}_{\omega^{(k)}} \left[R^{(k)}\left(S^{(k)}\left(S^{(k-1)}, x^{(k-1)}, d(x^{(k-1)}), \omega^{(k)}\right) | S^{(k-1)}\right)\right] \right\} \tag{6.21}$$

6.4.1.5 Restrições

As novas restrições do problema são apresentadas a seguir. A restrição 6.22 impõe não negatividade as variáveis $x_d^{(k)}$, enquanto que a 6.23 assegura que a posição assumida pelo *tripper* seja válida. Por fim, a restrição 6.24 garante que duração da alimentação pertença ao conjunto de números reais não negativos.

$$x_d^{(k)} \geq 0 \tag{6.22}$$

$$\chi_p = J \tag{6.23}$$

$$\chi_d = \mathbb{R}_+ \tag{6.24}$$

6.4.2 Métodos de Solução

A resolução do problema se dá pela utilização de funções indicadoras no Algoritmo de Iteração de Política Aproximado *LSTD*, apresentadas a seguir.

6.4.2.1 Funções Indicadoras

Foram adaptadas duas funções indicadoras, Gulosa e Desvio Padrão, a partir das funções desenvolvidas por Morais et al. (2020), e desenvolveu-se outras duas, Penalidade Exponencial - Capacidade Máxima, Penalidade Exponencial - Capacidade Mínima, apresentadas a seguir.

6.4.2.1.1 Função Gulosa

Assim como no modelo determinístico, o retorno da função é dado em função das folgas. Entretanto, como o modelo estocástico considera novas informações do sistema, como as incertezas $\omega^{(k)}$, o tempo de relógio $\tau^{(k)}$, o tempo de deslocamento δ_{ij} e a duração

da alimentação $d^{(k)}$, tais informações devem ser levadas em consideração no cálculo das folgas a cada estágio.

Dessa forma, utiliza-se a variável de estado, representada pela equação 6.14, para atualizar a posição atual do *tripper*, os níveis de cada silo e o tempo de relógio por meio das equações 6.15, 6.16 e 6.19 respectivamente. As folgas então são calculadas pelas equações 6.17 e 6.18.

Similarmente à função Gulosa do modelo determinístico, o retorno é apresentado pela equação 6.25:

$$\phi_{gulosa} = \sum_{j \in J} \left(\gamma_1 A_j^{(k)} + \gamma_2 B_j^{(k)} \right) \tag{6.25}$$

6.4.2.1.2 Função Desvio Padrão

Funciona de forma análoga à função Desvio Padrão do modelo determinístico, diferindo-se no fato que o cálculo dos níveis dos silos agora levam em conta as variáveis $\omega^{(k)}$ e $d_{(k)}$, conforme é mostrado na equação 6.16.

O cálculo do desvio padrão dos silos é realizado pela equação 6.26, em que V_j representa o nível do silo j, em que $j \in J$, $\bar{V}$ representa a média dos níveis dos silos, e |J| representa o número de silos da instância utilizada.

$$\sigma = \sqrt{\frac{\sum_{j \in J} \left(V_j - \bar{V} \right)^2}{|J|}} \tag{6.26}$$

O cálculo de V_j é realizado por meio da equação 6.16, definida anteriormente e reapresentada pela equação 6.27 a seguir.

$$V_j^{(k)} = V_j^{(k-1)} - \delta(P^{k-1}, p(x^{(k)}))T_s + \left(I_{[P_j]} T_e(\omega^{(k)}) - T_s(\omega^{(k)}) \right) d(x^{(k)}) \tag{6.27}$$

Por fim, a equação 6.28 mostra como o retorno da função utiliza o desvio padrão σ.

$$\phi_{desvio} = -\sigma \tag{6.28}$$

6.4.2.1.3 Penalidade Exponencial - Capacidade Máxima

Consiste em avaliar a penalização pela proximidade do nível no k-ésimo estágio à capacidade máxima dos silos, isto é, se o silo está prestes a transbordar. Como ultrapassar o limite superior dos silos resulta em custo, à medida que o nível ultrapassa o limite e se aproxima da capacidade máxima, a penalidade deve aumentar.

Para descrever tal comportamento, utilizou-se uma função exponencial. Para tanto, buscou-se uma função que represente a situação de forma satisfatória, ou seja, que aumente o seu valor de forma gradativa a medida em que se avança em direção à capacidade máxima.

Levando em consideração que a penalização é dada em função do nível V_j de cada silo j, o retorno da função indicadora na forma implícita é apresentado pela equação 6.29 e é apresentado na forma explícita pela equação 6.30, exibidas a seguir.

$$\phi_{cap\_max} = \sum_{j \in J} e^{0,1(V_j - 60)} \tag{6.29}$$

$$\phi_{cap\_max} = \sum_{j \in J} e^{0,1\left(\left[V_j^{(k-1)} - \delta(P^{k-1}, p(x^{(k)}))T_s + \left(I_{[P_j]}T_e(\omega^{(k)}) - T_s(\omega^{(k)})\right)d(x^{(k)})\right] - 60\right)} \tag{6.30}$$

6.4.2.1.4 Penalidade Exponencial - Capacidade Mínima

De forma similiar ao que acontece na função indicadora Penalidade Exponencial - Capacidade Máxima, consiste em penalizar de acordo com a proximidade entre o nível do silo e a capacidade mínima do silo, ou seja, atribuir uma penalidade maior quanto mais o nível do silo extrapola o limite inferior e se aproxima de zero, para tentar evitar que o silo se esvazie por completo.

Assim como na função indicadora anterior, também foi utilizada uma função exponencial, cujos parâmetros foram ajustados para que a situação possa ser representada de forma adequada, isto é, cujo valor aumente à medida que o nível do silo se aproxime de zero. A equação 6.31 apresenta a função exponencial encontrada em termos do nível V_j de cada silo.

$$\phi_{cap\_min} = -e^{-0,1(V_j - 40)} \tag{6.31}$$

Finalmente, a equação 6.32 apresenta de forma explícita o retorno da função.

$$\phi_{cap\_min} = \sum_{j \in J} -e^{-0,1\left(\left[V_j^{(k-1)} - \delta(P^{k-1}, p(x^{(k)}))T_s + \left(I_{[P_j]}T_e(\omega^{(k)}) - T_s(\omega^{(k)})\right)d(x^{(k)})\right] - 40\right)} \tag{6.32}$$

6.5 Método para Cálculo do Tempo de Alimentação

Conforme já dito, o presente trabalho utiliza funções indicadoras como aproximadores da função valor, e resolve o problema de movimentação do *tripper* por meio do Algoritmo de Iteração de Política Aproximado *LSTD*, cujo objetivo consiste em determinar

o direcionamento do equipamento ao longo do tempo e quanto tempo ele deve permanecer sobre o silo escolhido.

Para a tomada de decisão, implementou-se o seguinte método: a cada estágio, foram calculados o tempo de alimentação ótimo e o custo produzido pelas indicadoras para todos os silos da instância, e o silo que produziu o menor custo é escolhido como o próximo destino do *tripper*. Entretanto, para o cálculo do custo, o tempo que o *tripper* deve permanecer sobre o silo deve ser conhecido.

Como se pode observar por meio das equações 6.26, 6.29 e 6.31, as funções indicadoras Desvio Padrão, Penalidade Exponencial - Capacidade Máxima e Penalidade Exponencial - Capacidade Mínima são dadas em função dos níveis V_j de cada silo j, que por sua vez dependem do tempo $d(x^{(k)})$ de alimentação do silo, conforme mostram as equações 6.27, 6.30 e 6.32.

Embora não explicitamente como nas funções citadas acima, a função indicadora Gulosa também depende da duração da alimentação $d(x^{(k)})$, uma vez que, como apresentado pela equação 6.25, é dada em função das folgas, caracterizada pelos níveis V_j dos silos, que por sua vez são dependentes de $d(x^{(k)})$.

Sendo assim, como nas funções indicadoras só há o tempo $d(x^{(k)})$ de alimentação do silo como variável, conforme mostrado anteriormente, escolheu-se como método de cálculo do tempo de abastecimento do silo a resolução da primeira derivada das indicadoras utilizadas em relação à $d(x^{(k)})$. Tal escolha se deve ao fato de que resolver a primeira derivada das equações permite encontrar seu ponto mínimo, isto é, o menor custo de se utilizar as indicadoras ao longo do horizonte de tempo e o instante de tempo em que ele ocorre, o objetivo em questão.

Entretanto, como as funções não são estritamente côncavas ou convexas para todo o seu domínio, apenas uma análise da derivada primeira não é suficiente, pois mais de um ponto crítico é obtido. Portanto, necessita-se de uma condição suficiente, que nesse caso, é explicada pelo Teste da Derivada Segunda (Teste da Matriz Hessiana). Dados os pontos críticos, podemos afirmar que se todos os menores principais líderes da Matriz de Derivadas Segundas forem positivos quando aplicadas ao ponto, a mesma é classificada como positiva e com isso, podemos afirmar que o ponto crítico analisado é um mínimo local. Em casos em que a função retorne mais de um mínimo, pela classificação, devemos substituir tais pontos de mínimo na função custo original e verificar qual deles retorna o menor valor possível dentro da zona de factibilidade. Como as funções satisfazem esta condição, isto é, é possível encontrar o mínimo global, o método de solução por meio da resolução da primeira derivada das funções pode ser utilizado, uma vez que garante que a solução obtida seja ótima.

As equações obtidas pelas derivadas das funções indicadoras em relação à $d(x^{(k)})$

são apresentadas pelas equações 6.33, 6.34, 6.35 e 6.36, apresentadas a seguir:

1. Derivada Função Gulosa:

$$\frac{\partial \phi_{gulosa}(P_j, x)}{\partial x} = (\gamma_1 - \gamma_2)\left(I_{[P_j]}\overline{T_e} - \overline{T_s}\right) \tag{6.33}$$

2. Derivada Função Desvio Padrão:

$$\frac{\partial \phi_{desvio}(P_j, x)}{\partial x} = \frac{2T_e}{|J|}\left[V_p(p, x) - \overline{V}(p, x)\right] \tag{6.34}$$

3. Derivada Função Penalidade Exponencial - Capacidade Máxima:

$$\frac{\partial \phi_{cap\_max}(P_j, x)}{\partial x} = 0,1\left(I_{[P_j]}\overline{T_e} - \overline{T_s}\right)\phi_{cap\_max}(P_j, x) \tag{6.35}$$

4. Derivada Função Penalidade Exponencial - Capacidade Mínima:

$$\frac{\partial \phi_{cap\_min}(P_j, x)}{\partial x} = 0,1\left(I_{[P_j]}\overline{T_e} - \overline{T_s}\right)\phi_{cap\_min}(P_j, x) \tag{6.36}$$

Cabe ressaltar que no cálculo das derivadas das funções Função Penalidade Exponencial - Capacidade Máxima e Função Penalidade Exponencial - Capacidade Mínima, utilizou-se a variância no lugar do desvio padrão para fins de simplificação.

Para a resolução das derivadas obtidas, utilizou-se o módulo *least_squares* da biblioteca *scipy.optimize* do *Python*, que consiste em resolver um problema de mínimos quadrados não linear. Basicamente, foram passados como parâmetros a função de ajuste, formada pelas derivadas das indicadoras utilizadas, uma estimativa inicial da variável independente $d(x^{(k)})$, que assumiu o valor 1.0, os argumentos adicionais necessários à função e os limites inferior e superior da variável independente, definidos como 0.5 e 100.0, respectivamente. O algoritmo utilizado para realizar a minimização foi o algoritmo padrão *Trust Region Reflective*, e escolheu-se a função de perda padrão, a função linear. O módulo então minimiza a soma dos resíduos quadrados, isto é, da diferença entre o valor observado e o valor estimado da função passada como argumento, e encontra o mínimo local da função custo. Dessa forma, foi possível obter o tempo de duração da alimentação do silo $d(x^{(k)})$ por meio do instante de tempo em que o custo da função era mínimo.

7 Experimentos Computacionais para o Problema Estocástico

7.1 Modelo Dinâmico Estocástico

O algoritmo de aproximação de programação dinâmica foi implementado na linguagem *Python*, e os testes foram realizados em um computador com processador Intel Core i7-4790, 3,60GHz.

Para a realização dos testes criou-se a instância *16_60_2_teste.xml*, uma adaptação de uma das instâncias utilizadas por Pedrosa (2019), cujos parâmetros são apresentados na Tabela 10 a seguir, sendo que J representa o conjunto de silos, γ_1 e γ_2 são os coeficientes das folgas positiva e negativa, T_e representa a distribuição de probabilidade da taxa de entrada de minério e seus respectivos parâmetros e T_s a distribuição e os parâmetros da taxa de saída, cap_{max} e cap_{min} representam as capacidades máxima e mínima dos silos, l_{max} e l_{min} representam os limites superior e inferior dos silos, P_{ini} é a posição inicial do *tripper* e V_{ini} é a lista de níveis iniciais de cada silo.

Tabela 10 – Parâmetros Instância *16_60_2_teste.xml*

Parâmetro	**Valor**
J	16
γ_1	0.95
γ_2	0.95
T_e	normal, 17, 1
T_s	constante, 1
cap_{max}	100
cap_{min}	0
l_{max}	80
l_{min}	20
P_{ini}	13
V_{ini}	[39, 79, 13, 63, 47, 70, 53, 84, 86, 34, 11, 21, 67, 15, 94, 15]

De forma similiar ao Modelo 1, os testes realizados foram divididos em duas categorias, sendo que na primeira foram testadas diferentes combinações entre as funções implementadas o para cálculo do coeficiente λ e a utilização ou não da função resetB, utilizando-se todas as funções indicadoras simultaneamente, e na segunda categoria foram testadas diferentes combinações entre as funções indicadoras com a utilização da função lambda cte1. Em todos os testes, os coeficientes das funções indicadoras foram inicializados

como 0 e o fator de desconto γ assumiu o valor 0.99.

A seguir são apresentados os testes realizados e uma breve descrição sobre eles.

- Categoria 1 - Teste de combinações das funções lambda e da função resetB:
  - Teste resetB-cte1: habilita a função resetB e utiliza a função stpsze_cte1;
  - Teste resetB-ln100dn80: habilita a função resetB e utiliza a função stpsze_ln100dn80;
  - Teste resetB-ln50up100: habilita a função resetB e utiliza a função stpsze_ln50up100;
  - Teste B-ln100dn80: desabilita a função resetB e utiliza a função stpsze_ln100dn80;
  - Teste B-ln50up100: desabilita a função resetB e utiliza a função stpsze_ln50up100;
- Categoria 2 - Teste de combinações das funções indicadoras:
  - Teste Custo Míope: utiliza apenas a função indicadora Custo Míope;
  - Teste Custo Míope e Capacidade Máxima: utiliza as funções indicadoras Custo Míope e Capacidade Máxima;
  - Teste Custo Míope, Capacidade Máxima e Capacidade Mínima: utiliza as funções indicadoras Custo Míope, Capacidade Máxima e Capacidade Mínima;
  - Teste Custo Míope e Capacidade Mínima: utiliza as funções indicadoras Custo Míope e Capacidade Mínima;
  - Teste Custo Míope e Desvio Padrão: utiliza as funções indicadoras Custo Míope e Desvio Padrão;
  - Teste Custo Míope, Desvio Padrão, Capacidade Máxima e Capacidade Mínima: utiliza as funções indicadoras Custo Míope, Desvio Padrão, Capacidade Máxima e Capacidade Mínima.

Cada teste realizado foi dividido em duas etapas: treinamento e simulação. Na etapa de treinamento, o Algoritmo de Iteração de Política Aproximado *LSTD* foi executado 15 vezes, sendo que em cada uma foram realizadas 1000 simulações para convergência do valor, com o intuito de gerar uma nova política para solução do problema. Já na etapa de simulação foram realizadas duas simulações a partir da política obtida pelo treinamento, sendo que na Simulação 1 foram realizadas 100 iterações do algoritmo de programação dinâmica para a resolução do problema cujo estado inicial é dado pela instância *16_60_2_teste.xml*, e na Simulação 2 foram gerados 50 estados aleatoriamente e o algortimo de solução foi executado 100 vezes para cada estado.

A seguir, são detalhadas as etapas de treinamento do teste Custo Míope e Desvio Padrão e do teste Custo Míope, Capacidade Mínima e Capacidade Máxima, que obtiveram, respectivamente, o melhor e o pior desempenho. Logo após, são descritos os resultados

obtidos pela Simulação 1 dos dois testes e são apresentados, nas formas gráfica e de tabela, os resultados obtidos por todos os testes em cada simulação.

Por fim, será realizada uma comparação dos resultados obtidos pelos quatro métodos de resolução do problema do *tripper* desenvolvidos por este trabalho, com o objetivo de analisar o desempenho do método de aproximação de programação dinâmica proposto.

7.1.1 Etapa de Treinamento

A etapa de treinamento consiste em gerar novas políticas para a resolução do problema estocástico de movimentação do *tripper* através do Algoritmo de Iteração de Política Aproximado *LSTD*.

A Tabela 11 apresenta o tempo gasto, em segundos, para a realização do treinamento de todos os testes executados. As siglas resetB-cte, resetB-dn, resetB-up, B-dn e B-up representam os testes da Categoria 1, Teste resetB-cte1, Teste resetB-ln100dn80, Teste resetB-ln50up100, Teste B-ln100dn80, Teste B-ln50up100. Já as siglas CM, CM-C1, CM-C1-C2, CM-C2, CM-DP e CM-DP-C1-C2 referem-se aos testes Custo Míope, Custo Míope e Capacidade Máxima, Custo Míope, Capacidade Máxima e Capacidade Mínima, Custo Míope e Capacidade Mínima, Custo Míope e Desvio Padrão, e Custo Míope, Desvio Padrão, Capacidade Máxima e Capacidade Mínima da Categoria 2. Como se pode observar, o tempo de treinamento é diretamente proporcional à quantidade de indicadoras utilizadas, sendo que o teste CM-DP apresentou o melhor tempo, equivalente a aproximadamente 19,6 horas, enquanto que o teste resetB-cte apresentou o tempo mais alto, de aproximadamente 2,6 dias.

Tabela 11 – Tempo de Treinamento dos Testes - Problema Estocástico

Teste	**Tempo de Treinamento (s)**
resetB-cte	223440,32
resetB-dn	219181,52
resetB-up	219788,67
B-dn	220749,54
B-up	218922,00
CM	76028,44
CM-C1	127659,04
CM-C1-C2	176768,63
CM-C2	132178,67
CM-DP	70535,14
CM-DP-C1-C2	221968,52

Durante o treinamento, a cada iteração do algoritmo são escolhidas as ações a serem tomadas, são calculados o custo da ação e o erro, e os valores dos coeficientes θ das

funções indicadoras são atualizados. Ao fim de todas as iterações, o algoritmo retorna a nova política.

Serão apresentados a seguir os gráficos gerados pela execução do algoritmo dos dois testes escolhidos, referentes ao comportamento do custo, do erro e dos coeficientes das funções indicadoras ao longo do horizonte de tempo. A fim de proporcionar uma visualização mais clara dos dados, os *outliers* não foram apresentados.

7.1.1.1 Teste Custo Míope e Desvio Padrão

As figuras 16, 17, 18 e 19 descrevem, respectivamente, o comportamento do custo, do erro e dos coeficientes das funções indicadoras Custo Míope e Desvio Padrão ao longo das iterações realizadas.

De acordo com a Figura 16, observa-se que nas iterações iniciais do treinamento as ações escolhidas geravam um custo mais baixo, isto é, o fenômeno de exceder os limites dos silos ocorreram com uma menor frequência. No entanto, a partir da iteração 497 o custo estabilizou-se em 304, o que implica que, independentemente da ação tomada, o sistema não foi capaz de manter os níveis dos silos dentro de seus limites, gerando folgas.

Conforme já explanado, o gráfico do erro representa os erros na estimativa do valor se estar no estado a cada estágio ao longo das iterações. A partir da Figura 17, verifica-se que nas primeiras iterações as estimativas das funções valor atual e atualizada apresentavam uma diferença significativa. Entretanto, posteriormente essa diferença foi diminuindo, de modo que a função valor convergisse.

Por fim, as figuras 18 e 19 apresentam, respectivamente, o comportamento dos coeficientes das funções indicadoras Custo Míope e Desvio Padrão tanto ao longo das 15 iterações de atualização da política, representadas no gráfico pela cor laranja, quanto para as 15000 iterações realizadas para a convergência do valor, representadas pela cor azul. Como já era de se esperar, embasado pelo gráfico do erro apresentado anteriormente pela Figura 17, nas iterações iniciais ambos os coeficientes apresentam variações consideráveis em seus valores, mas à medida que as iterações avançam as variações diminuem e logo nota-se a convergência para seu valor final. Diante disso, infere-se que o treinamento foi capaz de estabilizar a nova política criada rapidamente.

Sendo assim, conclui-se que embora apresente custo, o treinamento criou uma política estabilizada, e como a convergência de seus valores aconteceu logo nas iterações iniciais, observa-se que o treinamento poderia ter sido realizado com um número menor de iterações.

Figura 16 – Teste Custo Míope e Desvio Padrão: custo ao longo das iterações

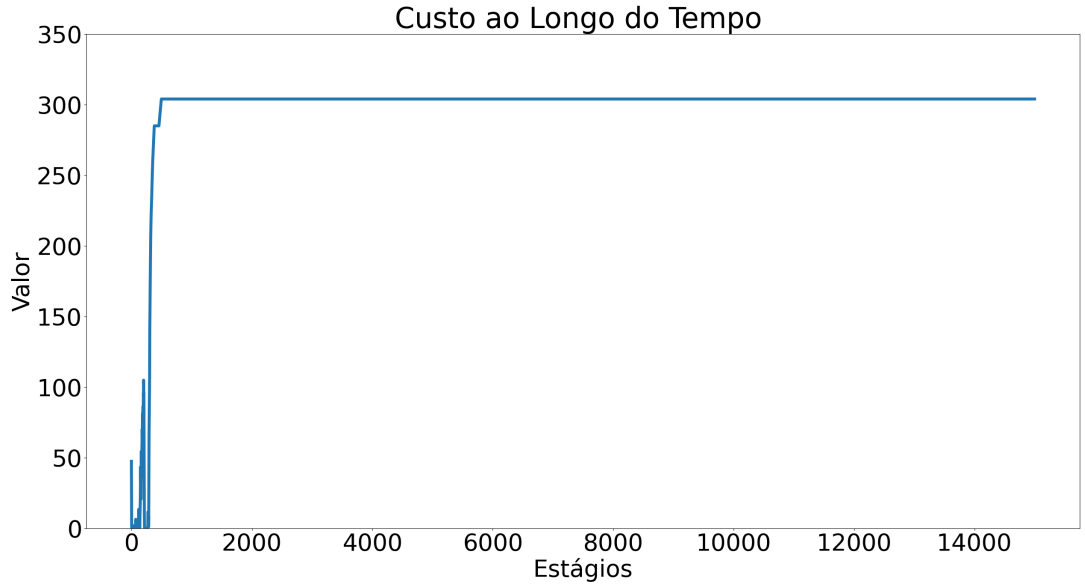


Figura 17 – Teste Custo Míope e Desvio Padrão: erro ao longo das iterações

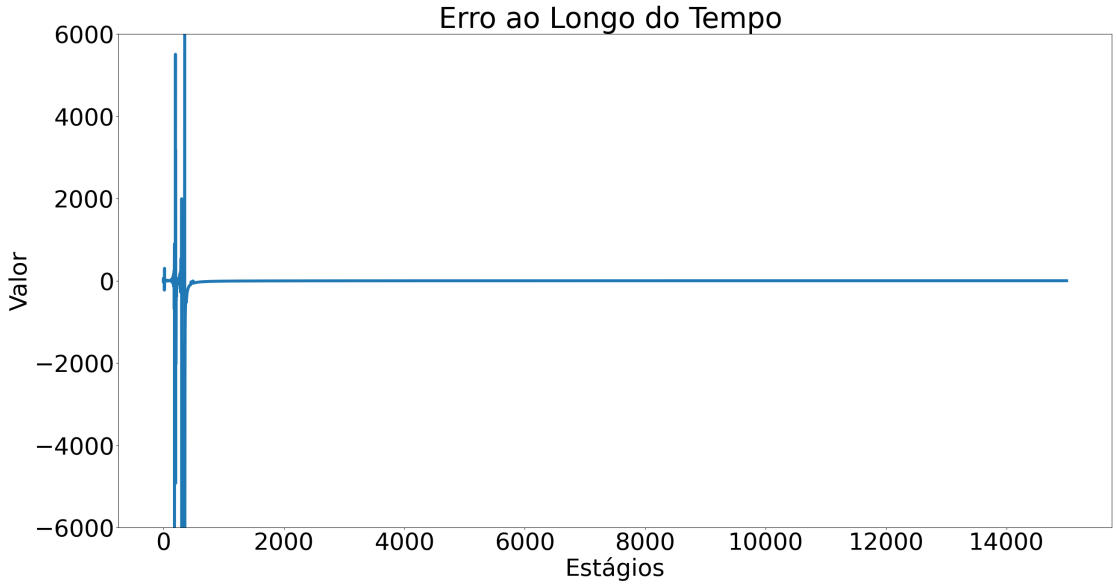


Figura 18 – Teste Custo Míope e Desvio Padrão: coeficiente θ_1 ao longo das iterações

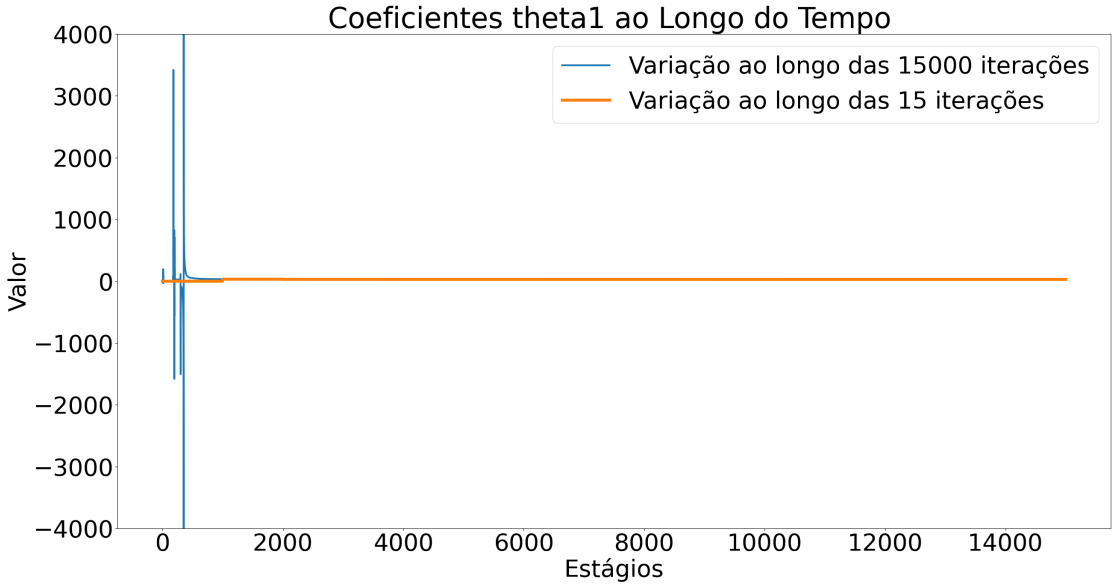
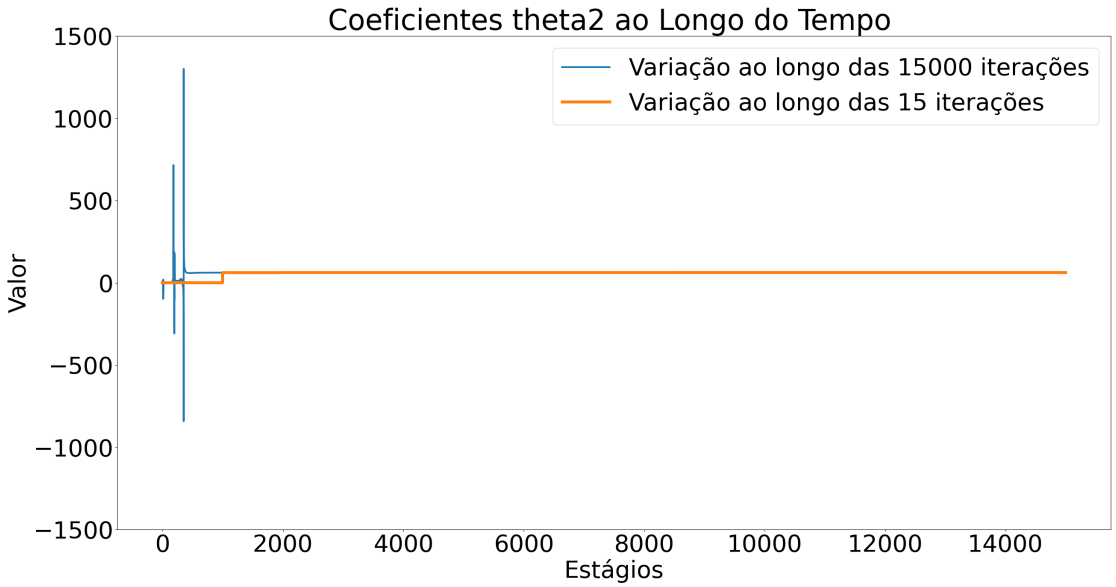


Figura 19 – Teste Custo Míope e Desvio Padrão: coeficiente θ_2 ao longo das iterações



7.1.1.2 Teste Custo Míope, Capacidade Mínima e Capacidade Máxima

As figuras 20, 21, 22, 23 e 24, exibidas a seguir, descrevem o comportamento das variáveis custo, erro, e dos coeficientes das funções indicadoras Custo Míope, Capacidade Mínima e Capacidade Máxima ao longo do tempo.

Como se pode observar pela Figura 20, as ações tomadas ao longo das iterações realizadas resultaram em diversos custos, e diferentemente do que ocorreu no teste Custo Míope e Desvio Padrão, não foi possível a obtenção de um padrão de comportamento e a consequente convergência de seu valor.

Conforme apresentado pela Figura 21, apesar de sua grande amplitude, o erro apresentou tendência à convergência, indicando que haviam diferenças entre a estimativa atual e a estimativa atualizada do valor de se estar no estado. Tal amplitude do erro é característica de problemas estocásticos, devido à aleatoriedade. Portanto, verifica-se que o treinamento realizado não conseguiu controlar a instância utilizada de forma que não houvesse erro.

Com relação ao comportamento dos coeficientes das indicadoras, nota-se que pelos gráficos 22, 23 e 24 que embora tenham apresentado uma tendência para convergir, as 15000 iterações realizadas não foram suficientes para a estabilização de seus valores, uma vez que as séries ainda apresentaram variações nas iterações finais. Observa-se que os picos presentes nas iterações iniciais ocorrem devido as maiores variações dos valores do custo e do erro ocorridas em tais iterações. Dentre as três indicadoras utilizadas, observa-se que os coeficientes da função Capacidade Mínima, exibidos pela Figura 23, apresentam uma variação mais atenuada de seus valores a partir do pico ocorrido por volta da iteração 330, enquanto que nas funções Custo Míope e Capacidade Máxima, conforme mostram as figuras 22 e 24, a atualização de seus coeficientes apresentou comportamento mais brusco, principalmente até a iteração 4000, aproximadamente.

Desse modo, conclui-se que, ao contrário do teste anterior, as iterações realizadas durante o treinamento não foram suficientes para criar uma política estabilizada, e consequentemente o número de iterações realizadas não poderia ter sido reduzido.

Figura 20 – Teste Custo Míope, Capacidade Mínima e Capacidade Máxima: custo ao longo das iterações

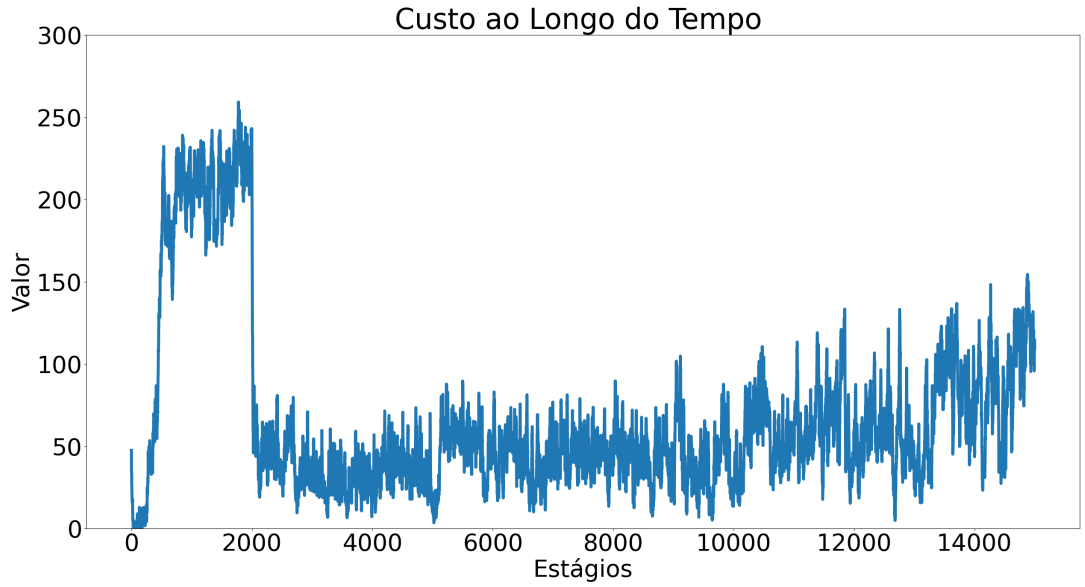


Figura 21 – Teste Custo Míope, Capacidade Mínima e Capacidade Máxima: erro ao longo das iterações

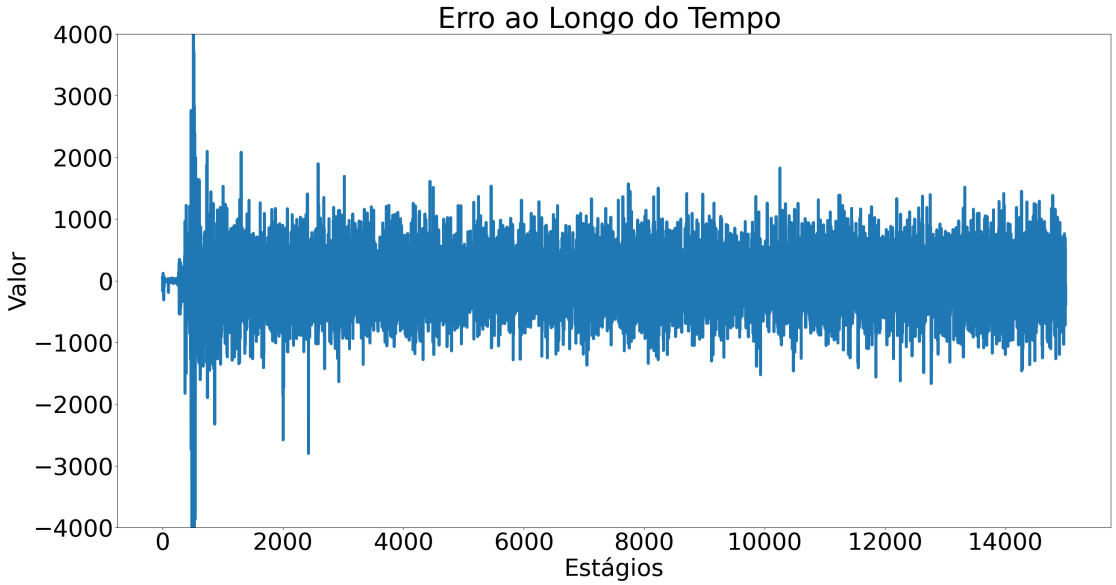


Figura 22 – Teste Custo Míope, Capacidade Mínima e Capacidade Máxima: coeficiente θ_1 ao longo das iterações

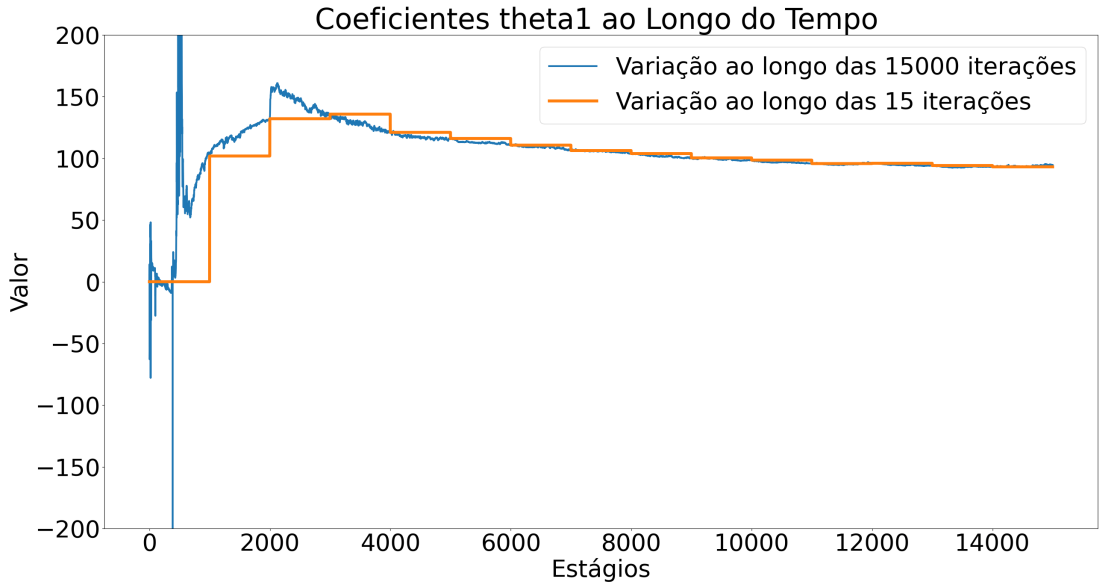


Figura 23 – Teste Custo Míope, Capacidade Mínima e Capacidade Máxima: coeficiente θ_2 ao longo das iterações

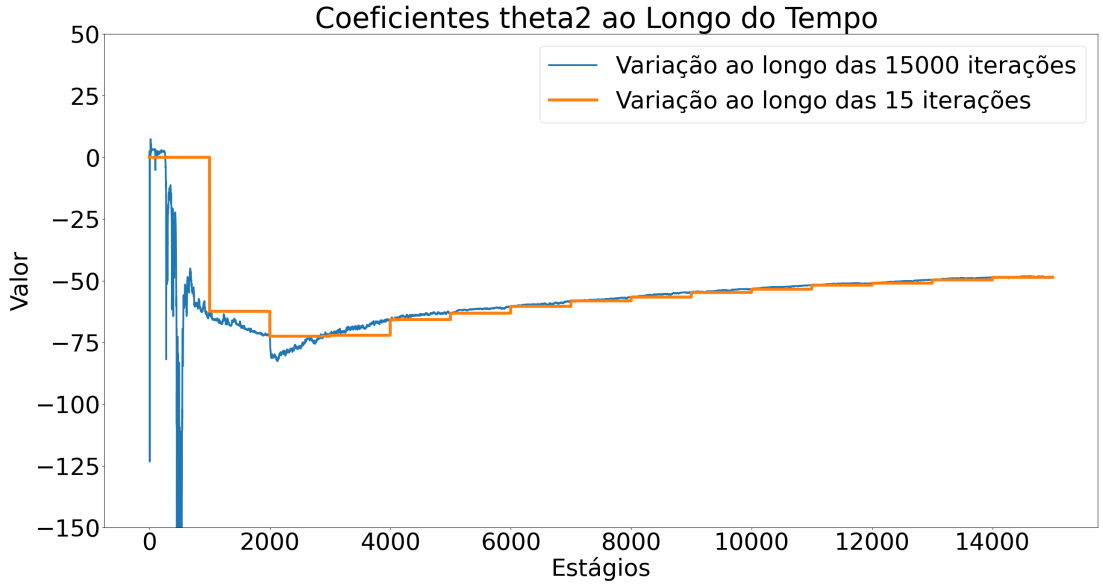
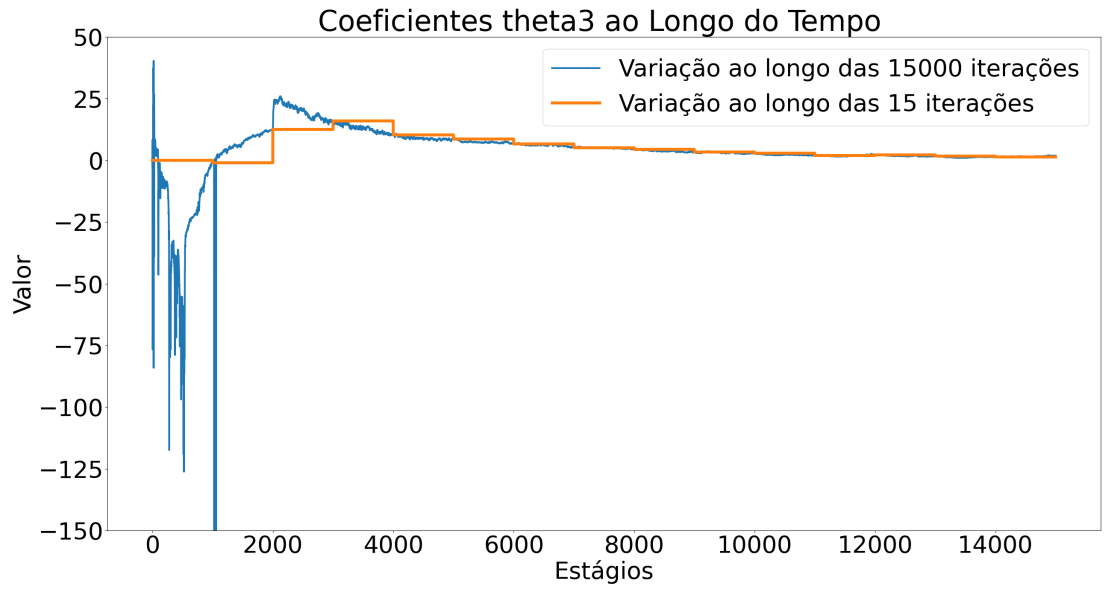


Figura 24 – Teste Custo Míope, Capacidade Mínima e Capacidade Máxima: coeficiente θ_3 ao longo das iterações



7.1.2 Simulação 1

A Simulação 1 utiliza a política obtida pela etapa de treinamento de cada teste para resolver o problema estocástico de movimentação do *tripper*, em que o estado inicial é dado pela instância *16_60_2_teste.xml*.

O algoritmo de programação dinâmica foi executado para 100 estágios em cada um dos testes apresentados anteriormente, sendo que a cada estágio foram escolhidas as ações, isto é, o movimento a ser realizado pelo *tripper* e o seu tempo de permanência sobre o silo escolhido, e foi calculado o custo relativo as ações tomadas.

Além disso, armazenou-se o custo, os coeficientes atuais das funções indicadoras, os níveis dos silos, a posição atual do *tripper* e a duração da alimentação referentes à cada estágio em um arquivo de formato *.csv*, para que uma análise dos resultados pudesse ser realizada.

No que diz respeito aos níveis dos silos, ao fim da simulação de cada teste plotou-se um gráfico que apresenta o comportamento dos níveis de cada silo ao longo dos 100 estágios. As figuras 25 e 26, exibidas a seguir, apresentam os gráficos dos níveis do teste Custo Míope e Desvio Padrão e do teste Custo Míope, Capacidade Mínima e Capacidade Máxima abordados nesta seção.

Por meio da análise da Figura 25, constata-se que, de forma geral, a política criada pelas indicadoras Custo Míope e Desvio Padrão foi eficiente em ambos os critérios de manutenção dos níveis entre os limites inferior e superior dos silos e de equilíbrio dos níveis,

uma vez que verifica-se por meio do gráfico que ao longo dos estágios os níveis de todos os silos tendem a ficar compreendidos entre 20 e 80 e também uniformes. A tendência de decrescimento dos níveis ao longo do tempo se dá pelo fato da instância utilizada ser desbalanceada.

Já a análise da Figura 26 mostra que a utilização da política criada pelas indicadoras Custo Míope, Capacidade Mínima e Capacidade Máxima fez com que o próprio sistema não fosse capaz de controlar os níveis dos silos, pois fez com que o *tripper* optasse por não se movimentar a partir da segunda iteração, resultando no extrapolamento do silo no qual o *tripper* está localizado e no esvaziamento dos demais silos ao longo do tempo.

Logo, observa-se que a política gerada pelas indicadoras Custo Míope e Desvio Padrão desponta como candidata à resolução do problema, enquanto a política gerada pelas indicadoras Custo Míope, Capacidade Mínima e Capacidade Máxima deve ser desconsiderada.

Figura 25 – Teste Custo Míope e Desvio Padrão: níveis ao longo das iterações

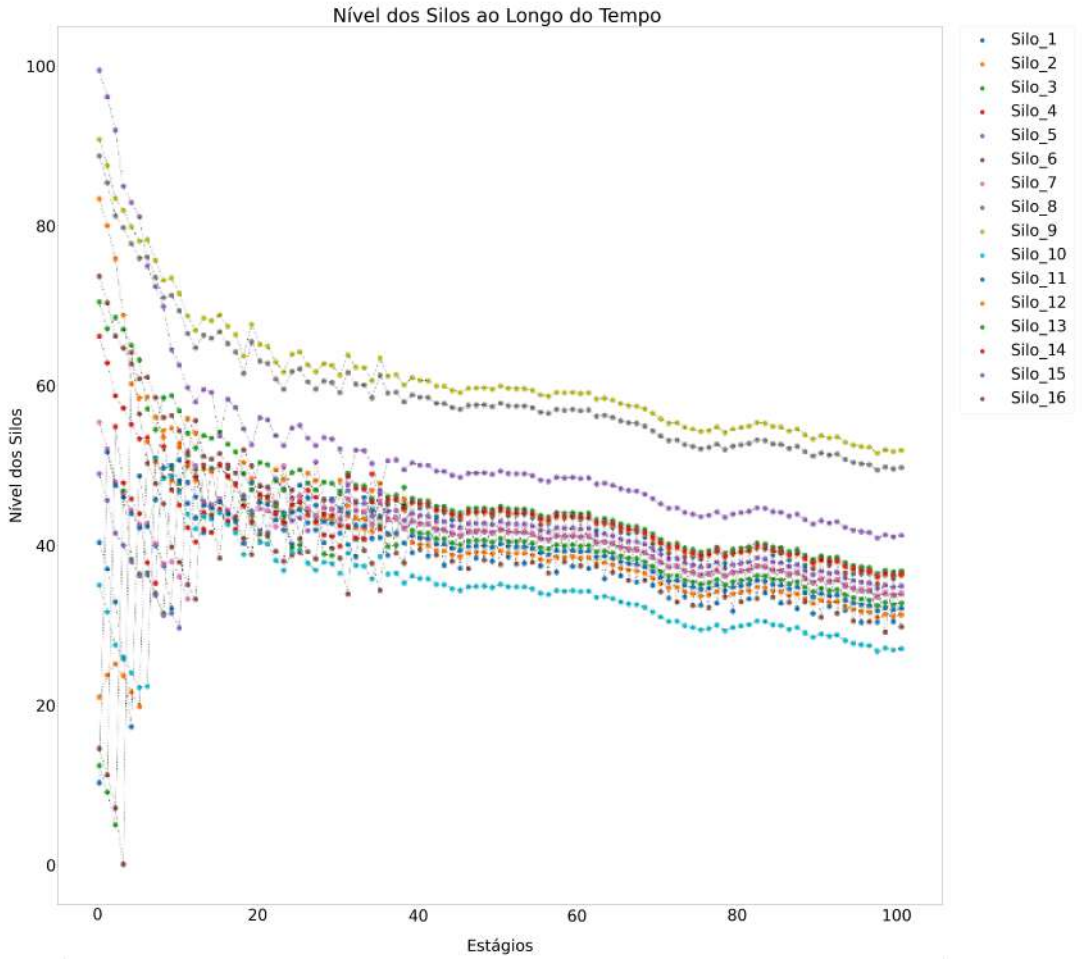
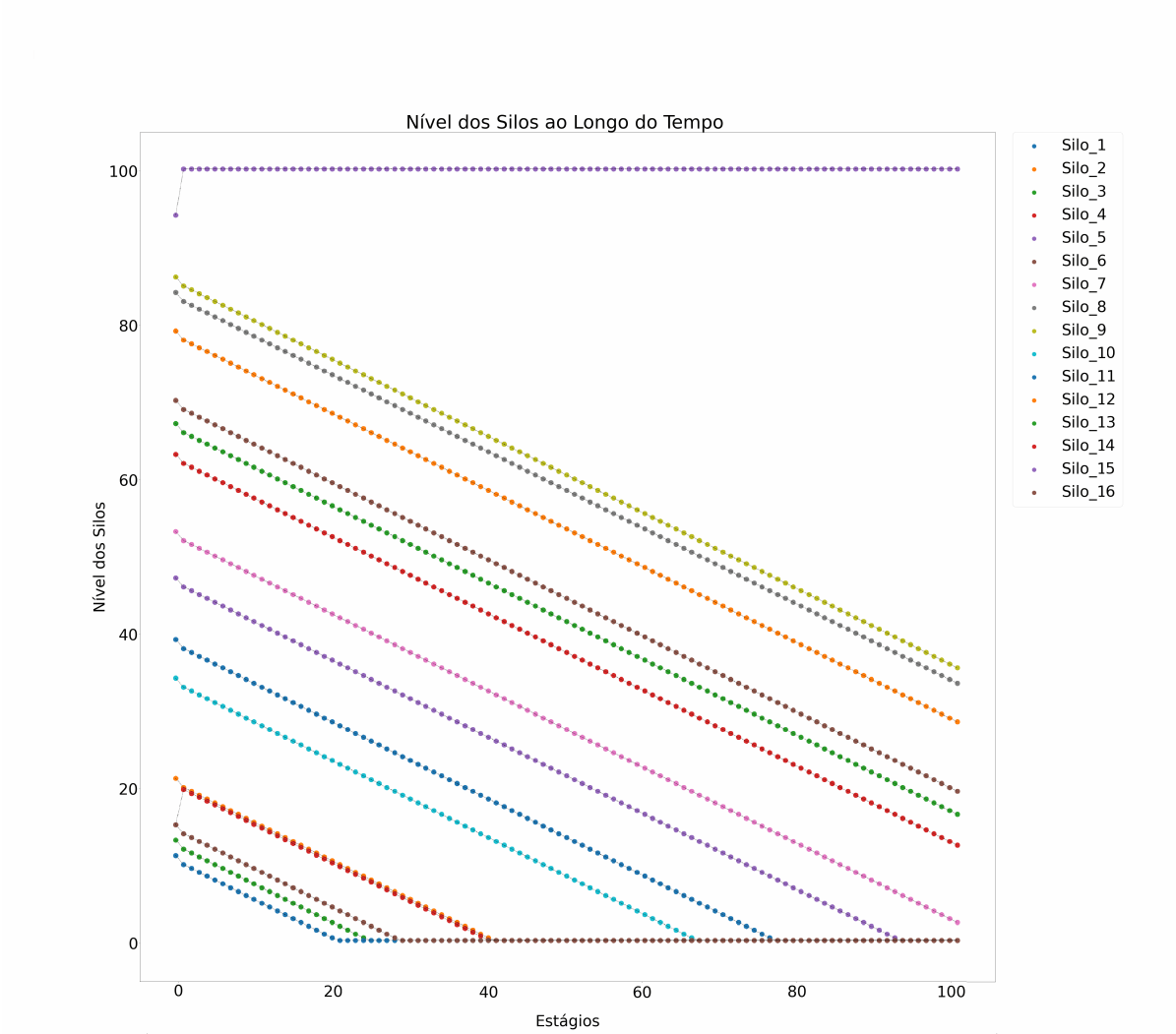


Figura 26 – Teste Custo Míope, Capacidade Mínima e Capacidade Máxima: níveis ao longo das iterações



Para que uma análise do desempenho de todos os testes pudesse ser realizada, construiu-se uma tabela com base nos arquivos de saída gerados por cada teste, contendo o custo total resultante de todos os estágios, o tempo despendido para realizar a simulação, o somatório dos tempos de alimentação dos silos e o desvio padrão total, caracterizado pela soma dos desvios dos niveis dos silos a cada estágio.

A Tabela 12 apresenta os resultados de cada teste mencionados acima e os coeficientes iniciais de cada função indicadora utilizada, sendo que as colunas C M, D P, Cap Máx e Cap Mín representam, respectivamente, as indicadoras Custo Míope, Desvio Padrão, Capacidade Máxima e Capacidade Minima, e as linhas resetB-cte, resetB-dn, resetB-up, B-dn e B-up representam os testes da Categoria 1, enquanto as linhas CM, CM-C1, CM-C1-C2, CM-C2, CM-DP e CM-DP-C1-C2 representam os testes da Categoria 2. Já as figuras 27 e 28 apresentam os resultados obtidos graficamente, de acordo com a sequência dos testes da Tabela 12.

Tabela 12 – Resultados Simulação 1 para 100 períodos - Problema Estocástico

Teste	C M	D P	Cap Máx	Cap Mín	Tempo (s)	Custo Total	DP Total	Tempo total de alimentação (s)
resetB-cte	134,37	-87,87	13462,10	-0,56	26,04	552,67	1109,44	53,27
resetB-dn	134,17	-89,45	13277,05	1,32	27,72	567,84	1149,04	54,81
resetB-up	122,28	-15,95	15719,33	7,97	26,51	637,36	1081,73	54,80
B-dn	116,75	-25,22	-10,22	-2,45	33,06	1205,04	2500,10	55,03
B-up	114,47	-23,46	-34,96	-2,07	36,37	13869,70	2831,62	50,50
CM	***98,38***	**-**	**-**	**-**	***4,87***	***193,68***	***1747,80***	***100,00***
CM-C1	67,29	-	52,50	-	17,51	327,23	1416,20	50,66
CM-C1-C2	94,63	-	-48,36	1,86	33,73	13966,52	2912,93	50,00
CM-C2	92,50	-	-	1,59	23,83	249,59	1930,97	119,19
CM-DP	***28,84***	***61,78***	**-**	**-**	***15,55***	***90,67***	***771,01***	***81,16***
CM-DP-C1-C2	110,96	-19,27	-11,63	-1,76	30,41	1007,06	2404,47	53,50

Figura 27 – Simulação 1 - Custo e Tempo

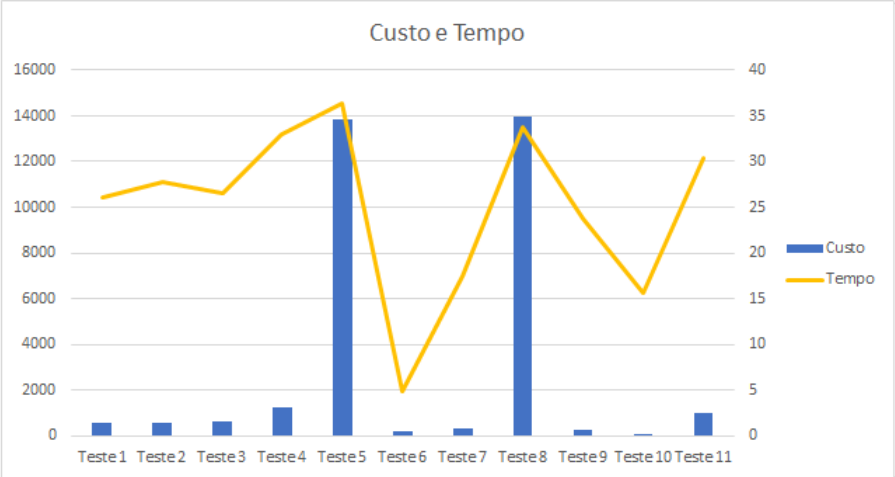
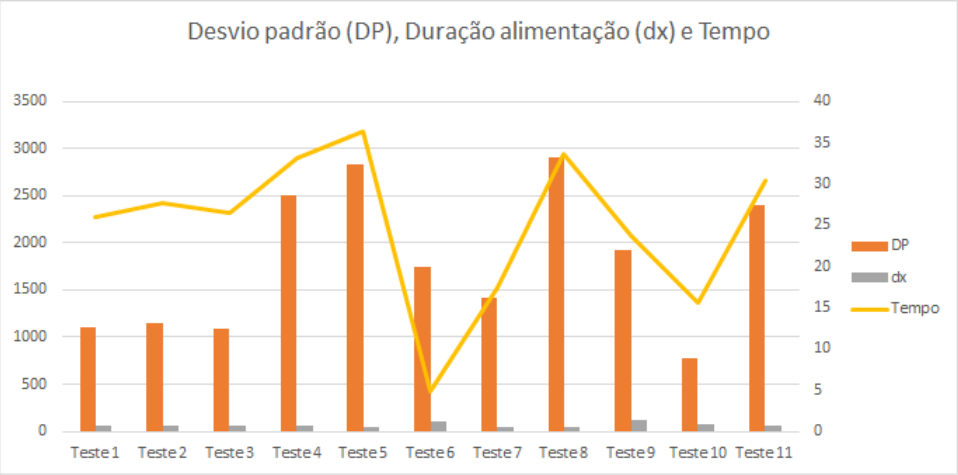


Figura 28 – Simulação 1 - Desvio padrão, Duração da alimentação e Tempo



Com relação ao tempo, observa-se que o tempo de execução da Simulação 1 de cada teste é diretamente proporcional ao número de funções indicadoras utilizadas, aumentando o seu valor à medida que o número de indicadoras utilizadas cresce. De forma geral, seu valor é significativamente pequeno, sendo 33,06 segundos no pior caso, teste B-up, e 4,87 segundos no teste CM, que apresenta o melhor valor, o que significa que mesmo no pior caso, uma vez já realizado o treinamento, é necessário menos de um minuto para resolver o problema utilizando-se a instância em questão para 100 estágios.

Comparando-se com o tempo de simulação do Problema Determinístico, apresentado no Capítulo 5, verifica-se que a execução da Simulação 1 do modelo estocástico é mais lenta. Entretanto, tal comportamento é justificado pelo fato de que, além de tomar mais uma decisão, o modelo estocástico utiliza um método numérico para o cálculo da duração da alimentação, lida com incertezas e analisa a cada estágio a possibilidade de movimentação para todos os silos, enquanto o problema determinísitico considera apenas a possibilidade

da realização de três movimentos: a locomoção para os silos imediatamente adjacentes à posição atual ou permanecer no silo atual.

No que se refere ao tempo total de alimentação dos silos, identifica-se que somente nos testes CM e CM-C1-C2 o tempo que o *tripper* permaneceu alimentado o silo escolhido manteve-se constante ao longo dos estágios, assumindo os valores 1,0 e 0,5 respectivamente, sendo que no restante dos testes a duração da alimentação do silo escolhido variou a cada estágio.

No que tange ao custo total, ou seja, ao desempenho dos testes em relação as folgas produzidas, destaca-se que foram obtidos resultados diversos, o que permite a comparação do desempenho em relação aos parâmetros utilizados por cada teste. Por meio da Tabela 12, observa-se que o teste CM-DP apresenta o menor custo, ou seja, violou menos vezes os limites dos silos, apresentando valor de 90,67, seguido pelo teste CM cujo valor obtido foi de 193,68. Já os testes CM-C1-C2 e B-up apresentaram resultados insatisfatórios, cujos valores 13966,52 e 13869,70 ficaram muito aquém dos demais.

Analisando-se o custo por unidade de tempo de alimentação, isto é, dividindo os valores, nota-se que o desempenho obtido é similar ao obtido pelo quesito custo, em que os testes CM-DP e CM apresentaram os melhores resultados, e os testes B-up e CM-C1-C2 apresentaram os piores resultados dentre todos os testes realizados. Tal comportamento deve-se ao fato da maioria dos testes apresentarem tempo total de alimentação similares, em torno de 50 segundos, sendo que os testes que apresentaram resultados divergentes, em que o tempo total de alimentação foi maior, também apresentaram os menores custos, refutando assim a superioridade destes testes. Analisando-se o comportamento do *tripper*, conclui-se que, nestas condições, o *tripper* deve permanecer alimentando os silos por um tempo superior ao tempo mínimo a cada estágio, estipulado como 0,5 segundos.

Por meio da análise dos desvios padrões obtidos, observa-se que o melhor resultado, 771,01, foi obtido pelo teste CM-DP, e o pior resultado, de valor 2912,93, foi obtido pelo teste CM-C1-C2, o que leva à conclusão de que o teste que mais controla os níveis dos silos em relação aos seus limites também é o mais eficiente quanto à homogeneização dos níveis, e vice-versa. De modo geral, a maioria dos testes seguem esse mesmo comportamento, exceto os testes CM e CM-C2 que são mais eficientes em produzir menos folgas e o teste resetB-up que é mais eficiente no quesito homogeneização dos níveis.

Através de análise dos arquivos de saída gerados pelos testes, é possível identificar a estratégia de movimentação assumida pelo *tripper* em cada um deles. No caso do teste CM-DP, que apresentou o melhor desempenho dentre todos, a seguinte estratégia foi utilizada: inicialmente, os movimentos eram feitos de forma a equilibrar os níveis de todos os silos, e uma vez equilibrados, o *tripper* se revezava em ir para o primeiro e para o último silo. Já o teste CM-C1-C2, de pior desempenho, optou por, uma vez escolhido o primeiro movimento, realizá-lo até o fim, justificando assim o alto custo obtido.

Quanto à Categoria 1 dos testes, percebe-se que os testes que utilizam a função resetB, isto é, que resetam a matriz B a cada iteração possuem, nos quesitos custo e desvio padrão, desempenho consideravelmente superior aos testes em que a função não é utilizada. Já no que diz respeito as funções para cálculo do coeficiente λ, constata-se que a utilização da função stpsze_cte1 ocasiona melhores performances, enquanto a função stpsze_ln50up100 gera piores resultados.

Analisando-se a Categoria 2 dos testes, constata-se que o teste CM-DP obteve melhor desempenho em ambos os quesitos, enquanto que o teste CM-C1-C2 obteve o pior desempenho também em relação à ambos quesitos.

Logo, infere-se que as funções utilizadas pelo algoritmo interferem diretamente no desempenho de sua resolução, como mostram os resultados obtidos pelos testes realizados. A fim de verificar se as funções utilizadas nos testes geram o mesmo desempenho para diferentes instâncias, propôs-se a realização de uma nova simulação, apresentada a seguir.

7.1.3 Simulação 2

Na Simulação 2, foram gerados aleatoriamente 50 estados iniciais para cada teste, e as simulações foram realizadas a partir de cada estado inicial gerado e da política obtida pela etapa de treinamento dos respectivos testes.

Foram realizadas 100 iterações do algoritmo para cada estado de cada teste, e de forma análoga as simulações apresentadas anteriormente, gerou-se um arquivo no formato *.csv* ao final de cada simulação contendo as informações referentes ao o custo, aos coeficientes das funções indicadoras utilizadas, aos níveis dos silos, à posição do *tripper* a cada iteração e à duração da alimentação do silo de cada iteração realizada.

Com base no arquivo gerado, calculou-se para cada teste realizado o tempo médio, isto é, a média de tempo necessária para realização da simulação, o custo médio, que representa a média do custo total dos 50 estados, o desvio padrão médio, calculado como a média dos desvios dos estados, que por sua vez é calculado como a soma dos desvios dos níveis dos silos a cada iteração, e o tempo médio de alimentação, constuído pela média do tempo de alimentação total. A Tabela 13 a seguir apresenta os resultados obtidos, em que as linhas correspondem aos testes realizados e as colunas correspondem as variáveis analisadas. As figuras 29 e 30 a seguir apresentam, de forma gráfica, os resultados, sendo que a margem de erro foi calculada utilizando-se um nível de confiança de 95% e a apresentação dos testes segue a ordem estabelecida pela Tabela 13.

Tabela 13 – Resultados Simulação 2 para 100 períodos - Problema Estocástico

Teste	C M	D P	Cap Máx	Cap Mín	Tempo médio (s)	Custo médio	DP médio	Tempo médio de alimentação (s)
resetB-cte	134,37	-87,87	13462,10	-0,56	28,62	7184,93	685,50	69,62
resetB-dn	***134,17***	***-89,45***	***13277,05***	***1,32***	***29,15***	***5900,97***	***643,16***	***70,17***
resetB-up	122,28	-15,95	15719,33	7,97	30,55	7191,99	505,31	61,14
B-dn	116,75	-25,22	-10,22	-2,45	25,34	14146,79	628,14	82,63
B-up	114,47	-23,46	-34,96	-2,07	24,10	14178,21	589,12	81,00
CM	98,38	-	-	-	4,74	21204,04	520,43	100,00
CM-C1	***67,29***	**-**	***52,50***	**-**	***19,94***	***5745,98***	***1291,06***	***233,50***
CM-C1-C2	94,63	-	-48,36	1,86	25,97	10559,95	562,51	67,03
CM-C2	92,50	-	-	1,59	18,20	10299,53	604,46	66,65
CM-DP	28,84	61,78	-	-	13,69	29944,14	2478,20	56,45
CM-DP-C1-C2	110,96	-19,27	-11,63	-1,76	24,41	15002,99	600,02	86,20

Figura 29 – Simulação 2 - Custo e Tempo

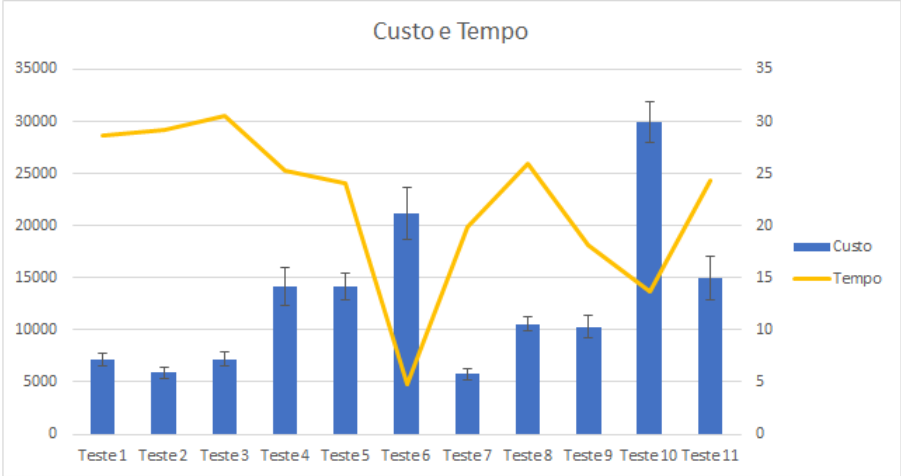
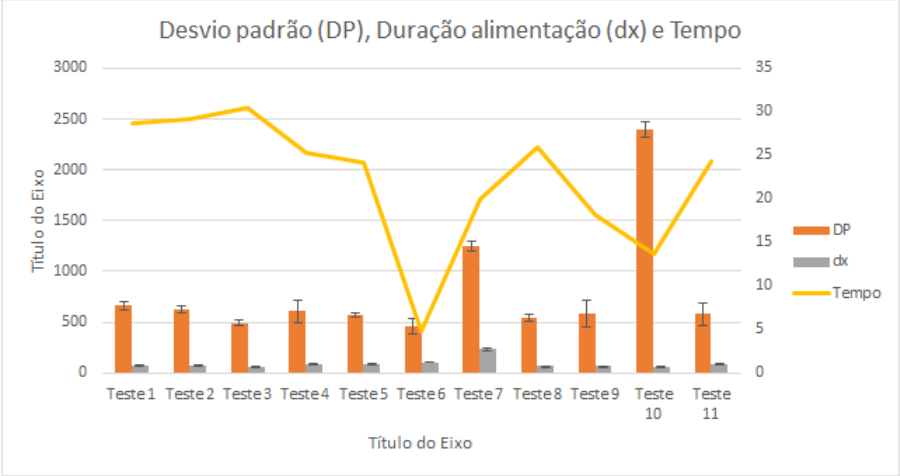


Figura 30 – Simulação 2 - Desvio padrão, Duração da alimentação e Tempo



De forma geral, por meio da análise dos resultados apresentados, observa-se que assim como na simulação anterior, o desempenho em ambos os quesitos tempo e custo depende da combinação de funções utilizada em cada teste.

No que diz respeito ao tempo médio de execução, infere-se que o tempo é diretamente proporcional ao número de funções indicadoras utilizadas e que os resultados obtidos foram similares aos obtidos pela Simulação 1 , em que o teste CM apresentou o melhor resultado, 4,74 segundos, e o teste resetB-up apresentou o pior desempenho, 30,55 segundos.

Com relação ao tempo médio de alimentação, constata-se que diferentemente do ocorrido na Simulação 1, apenas o teste CM apresentou o mesmo tempo de alimentação, equivalente a um segundo, em todas as iterações, sendo que nos demais testes a duração ótima da alimentação assumiu valores diferentes a cada iteração.

Os melhores custos foram obtidos pelos testes CM-C1 e resetB-dn, cujos valores foram de 5745,98 e 5900,97 respectivamente, enquanto que os piores resultados, 29944,14

e 21204,04 foram obtidos pelos testes CM-DP e CM. Entretanto, comparando-se com os custos obtidos pela Simulação 1, observa-se uma discrepância muito grande entre os valores em alguns testes, o que implica na ocorrência de um número elevado de folgas, sendo que apenas os testes B-up e CM-C1-C2 apresentaram valores semelhantes.

Analisando-se o desempenho dos testes por meio do custo por tempo de alimentação, observa-se que o testes obtiveram o mesmo desempenho obtido pelo quesito custo, em que os testes CM-C1 e resetB-dn alcançaram os melhores resultados, enquanto que os testes CM-DP e CM obtiveram os piores desempenhos. Como o desempenho do teste CM-C1 é notadamente superior, constata-se que, durante a tomada de decisão, o *tripper* deve priorizar alimentar o silo por uma quantidade de tempo significativa do que alimentar por um curto período e logo se locomover para o próximo silo.

Já o melhor desvio padrão foi obtido pelo teste resetB-up, enquanto o pior foi obtido pelo teste CM-DP. Com relação aos resultados obtidos pela Simulação 1, observa-se que os desvios obtidos pela Simulação 2 são inferiores, o que justifica-se pelo fato de alguns estados terem gerado os níveis iniciais mais uniformes do que os da instância *16_60_2_teste.xml*. Analisando-se o custo e o desvio padrão de cada teste simultaneamente, conclui-se que a maioria dos testes não apresenta o mesmo desempenho em ambos os quesitos, isto é, apesar dos níveis terem sido extrapolados com uma frequência menor, não foi possível fazer com que eles fossem equilibrados, e vice-versa.

Dentre os testes da categoria 1, observa-se que a utilização da função resetB gera custos consideravelmente menores, de aproximadamente metade dos custos dos testes em que a função não é utilizada. Com relação ao desempenho das funções para cálculo do coeficiente λ, constata-se que a função stpsze_ln100dn80 apresentou os melhores resultados.

Por fim, comparando-se os resultados obtidos pelas duas simulações, constata-se que, de forma geral, eles apresentam diferenças significativas. O desvio padrão médio dos testes da Simulação 2 apresenta valores consideravelmente inferiores do que a maioria dos testes da Simulação 1. Já o quesito custo apresenta diferenças ainda mais significativas, sendo que na Simulaçao 1 os quatro melhores testes apresentavam custos entre 90,67 e 327,23, e na Simulação 2 os quatro melhores testes apresentaram valores entre 5745,98 e e 7191,99. Além disso, os melhores testes na Simulação 1, CM-DP e CM, foram os piores testes na Simulação 2.

Dessa forma, conclui-se que além das funções utilizadas em cada teste, a instância utilizada também interfere no desempenho do algoritmo. Além disso, como a Simulação 2 utilizou o treinamento realizado para a instância *16_60_2_teste.xml* para resolver o problema para os estados gerados aleatoriamente, constata-se que o treinamento não é generalizável para outros estados iniciais, o que implica que para cada novo estado um novo treinamento deve ser realizado.

7.2 Modelo Zigue-Zague e Modelo Determinístico Baseado MILP

Foram realizados testes para os outros três métodos desenvolvidos, Zigue-Zague, Modelo Determinístico MILP - Política 1 e Modelo Determinístico MILP - Política 2, utilizando-se a mesma instância *16_60_2_teste.xml*.

Para o método Zigue-Zague, foram realizadas dez execuções, e os valores apresentados correspondem à média dos resultados obtidos. Já para o método Modelo Determinístico MILP, foram executados dois testes, em que o primeiro utiliza a Política 1, que executa o modelo uma única vez gerando todas as ações dos 100 estágios, e o segundo utiliza a Política 2, de horizonte rolante, isto é, que gera ao final de cada estágio as ações a serem tomadas no próximo estágio.

A Tabela 14 apresenta todos os testes realizados e os resultados obtidos por cada um, em que APIA representa o modelo de aproximação de programação dinâmica e Modelo Det. representa o Modelo Determinístico MILP. A coluna Trein. refere-se ao tempo gasto para se criar a nova política, a coluna Sim. representa o tempo de execução da simulação, e a coluna Total apresenta o total do tempo gasto na resolução, sendo que no método APIA é formado pela soma dos tempos de treinamento e simulação. Como os métodos Modelo Determinístico MILP e Zigue-Zague não possuem as etapas de treinamento e simulação, estas colunas são preenchidas pelo marcador "-", e o tempo de resolução é apresentado na coluna Tempo Total. Já a coluna Custo Total apresenta o custo referente ao total de folgas produzidas, e por fim a coluna Tempo alim. exibe o resultado obtido, em segundos, pela soma dos tempos de alimentação dos silos a cada estágio.

Tabela 14 – Resultados Obtidos pelos Métodos de Resolução

Método Resolução	**Teste**	**Tempo (s)**			**Custo Total**	**Tempo alim. (s)**
		Trein.	**Sim.**	**Total**		
APIA	**resetB-cte**	223440,32	26,04	223446,36	552,67	53,27
	resetB-dn	219181,52	27,72	219209,24	567,84	54,81
	resetB-up	219788,67	26,51	219815,18	637,36	54,80
	B-dn	220749,54	33,06	220783,61	1205,04	55,03
	B-up	218922,00	36,37	218958,37	13869,70	50,50
	CM	76028,44	4,87	76033,31	193,68	100,00
	CM-C1	127659,04	17,51	127676,55	327,23	50,66
	CM-C1-C2	176768,63	33,73	176802,36	13966,52	50,00
	CM-C2	132178,67	23,83	132206,50	249,59	119,19
	CM-DP	***70535,14***	***15,55***	***70550,69***	***90,67***	***81,16***
	CM-DP-C1-C2	221968,52	30,41	221998,93	1007,06	53,50
Modelo Det.	**Política 1**	-	-	440,38	491,68	110,17
	Política 2	**-**	**-**	***114710,36***	***357,68***	***113,85***
Zigue-Zague		-	-	2,85	7053,84	85,26

Por meio dos resultados obtidos, verifica-se que, no geral, o método Zigue-Zague apresenta tempo de execução substancialmente inferior, mas em contrapartida apresenta

um desempenho insatisfatório no quesito custo. Já o método APIA apresenta resultados variados tanto com relação ao tempo total quanto ao custo total, sendo o desempenho relativo à cada teste e dado pela combinação de funções utilizada. Por fim, o Modelo Determinístico apresenta um bom desempenho em ambas as políticas, sendo que a Política 2, que implementa o horizonte rolante, apresenta melhor desempenho no quesito custo e tempo de execução significativamente maior, como esperado.

No que diz respeito ao desempenho dos testes com relação ao custo por tempo de alimentação, observa-se que os testes CM-DP, CM, e CM-C2 do método APIA apresentaram os melhores resultados dentre todos os testes realizados, superando até mesmo os testes Política 1 e Política 2 do Modelo Determinístico MILP.

Dessa forma, conclui-se que o método Zigue-Zague não é eficiente, uma vez que apesar de resolver o problema consideravelmente rápido, possui custo muito elevado, o que significa que não foi possível controlar os níveis dos silos para que se mantivessem dentro dos limites e não provocassem folgas. Este desempenho é fundamentado pelo fato de que o método não analisa o estado atual do sistema para escolher para qual silo o *tripper* deve se locomover. Logo, conclui-se que a possibilidade que o método seja eficiente deve ocorrer em um ambiente determinístico cujos silos iniciais se encontrem minimamente equilibrados.

Quanto ao Modelo Determinístico, este apresenta bom desempenho em ambos os quesitos custo e tempo de execução, sendo que a Política 1 apresenta ótimo resultado, sendo notavelmente inferior, no quesito tempo dentre os demais testes, e resultado competitivo no quesito custo. Já a Política 2, apesar do tempo de execução ser muito aquém do encontrado pela Política 1, apresenta um custo menor do que o da Política 1, e os resultados obtidos em ambos os quesitos são compatíveis aos resultados obtidos pelos melhores testes do modelo APIA. Entretanto, vale ressaltar que o modelo não é uma representação exata do sistema, uma vez que não são consideradas as incertezas do processo.

No que se refere ao método de Aproximação de Programação Dinâmica, identifica-se que as combinações de parâmetros dos testes B-dn, B-up, CM-C1 e CM-DP-C1-C2 devem ser desconsideradas devido aos seus custos, e que as combinações dos testes CM-DP, CM, CM-C2 e CM-C1 despontam como alternativas à resolução do problema, gerando custos até menores dos que os obtidos pelo método Modelo Determinístico. Quanto ao tempo total, salienta-se que embora seja considerável, o valor é competitivo em relação ao valor obtido pela Política 2 do Modelo Determinístico, e é correspondente apenas à primeira execução do modelo para a respectiva instância, pois uma vez realizado o treinamento, a política é criada e nas próximas execuções só é necessária a realização da etapa de simulação, cujo tempo é notadamente baixo.

Sendo assim, conclui-se que os métodos Aproximação de Programação Dinâmica e Modelo Determinístico são eficientes em resolver o problema cujo estado inicial é dado pela instância *16_60_2_teste.xml*, sendo que o método de Aproximação de Programação

Dinâmica se destaca quando é necessária que a execução do algoritmo seja realizada mais de uma vez.

7.3 Análise Comparativa dos Métodos de Solução

Com o intuito de se analisar o comportamento de cada método para diferentes instâncias, para que se possa chegar a conclusões mais generalizadas sobre o desempenho de cada um, foram gerados cinco estados de forma aleatória e utilizou-se três métodos para resolução para cada um deles: a Política 2 do Modelo Determinístico PLIM, os parâmetros do teste CM-DP do método Aproximação de Programação Dinâmica (APIA), que apresentaram os melhores resultados relativos ao custo dos respectivos modelos, e o método Zigue-Zague. Ressalta-se que no método APIA foi realizada a etapa de treinamento para cada estado inicial gerado.

A Tabela 15 apresenta os resultados obtidos, em que a coluna Tempo Total corresponde ao tempo necessário para solucionar o problema, constituído pela soma dos tempos de treinamento e de simulação, quando houver, apresentados pelas colunas Trein. e Sim., a coluna Custo Total representa o custo em relação ao objetivo, isto é, relativo ao total de folgas produzidas, e a última coluna exibe a soma das durações da alimentação, obtidas a cada estágio, de cada teste realizado.

Tabela 15 – Resultados estados aleatórios para 100 períodos

Teste		**Tempo (s)**			**Custo Total**	**Tempo Total alimentação (s)**
		Trein.	**Sim.**	**Total**		
Estado 1	**APIA**	***58030,13***	***18,96***	***58049,09***	***163,29***	***83,7***
	Modelo Det.	-	-	86981,81	192,12	122,38
	Zigue-Zague	-	-	3,00	8083,40	73,90
Estado 2	**APIA**	***53897,83***	***15,65***	***53913,48***	***348,39***	***89,63***
	Modelo Det.	-	-	70947,28	374,12	124,06
	Zigue-Zague	-	-	3,20	9085,72	75,45
Estado 3	**APIA**	51289,21	15,67	51304,88	123,49	75,23
	Modelo Det.	**-**	**-**	***37416,48***	***111,45***	***127,87***
	Zigue-Zague	-	-	3,08	5437,79	74,33
Estado 4	**APIA**	49976,70	17,22	49993,92	873,56	94,91
	Modelo Det.	**-**	**-**	***62964,83***	***241,89***	***123,41***
	Zigue-Zague	-	-	3,01	7434,19	69,06
Estado 5	**APIA**	***68584,34***	***15,34***	***68599,68***	***104,08***	***73,39***
	Modelo Det.	-	-	38655,83	156,52	130,51
	Zigue-Zague	-	-	3,26	6007,05	81,14

A partir dos resultados apresentados pela Tabela 15, nota-se que, conforme o esperado, o método Zigue-Zague gera custos significativamente maiores em todos os estados, o que significa que o desempenho do método é intrínseco à ele mesmo e não

depende da instância utilizada. Logo, constata-se que tal método não deve ser utilizado para a resolução do problema.

No que tange ao tempo de resolução, verifica-se que os métodos APIA e Modelo Determinístico apresentaram resultados não muito divergentes, mesmo considerando-se o tempo gasto pela etapa de treinamento, em que em três dos cinco estados a resolução pelo algoritmo APIA foi mais rápida.

Com relação ao custo, constata-se que de forma geral, exceto no Estado 4, os resultados obtidos pelos métodos APIA e Modelo Determinístico foram similares, sendo que em três estados a utilização do algoritmo APIA resultou em melhores custos, enquanto que nos outros dois o Modelo Determinístico gerou custos menores.

Analisando-se o custo por unidade de tempo de alimentação, observa-se que, diferentemente do ocorrido quando o estado inicial é dado pela instância *16_60_2_teste.xml*, o método Modelo Determinístico apresentou o melhor desempenho em todos os testes realizados, mesmo naqueles em que o método APIA obteve melhor desempenho no quesito custo total.

Sendo assim, os resultados obtidos reforçam as premissas de que o desempenho do método APIA está relacionado à instância utilizada e à realização do treinamento específico para a instância. Já os tempos de execução mostram a superioridade do método APIA em situações em que são realizadas mais de uma execução. Em contrapartida, nota-se que o método Modelo Determinístico PLIM - Política 2 apresenta excelente desempenho independentemente da condição inicial do sistema.

Finalmente, analisando-se de forma geral todos os resultados obtidos, constata-se que o método de resolução APIA proposto foi capaz de gerar políticas eficientes em ambos os quesitos tempo e quantidade de folgas produzidas para a resolução do problema estocástico do *tripper*. Além disso, conclui-se que o desempenho do algoritmo de Iteração de Política Aproximado *LSTD* depende de diversos fatores, tais como os valores de seus parâmetros, a realização do treinamento para a instância a ser utilizada, as funções indicadoras e a instância utilizadas, sendo que no caso da instância não somente o seu tamanho importa, como também os parâmetros relacionados à ela.

Já o método de resolução Modelo Determinístico PLIM - Política 2 apresentou desempenho constante em todos os testes realizados, em que além de possuir boa performance nos quesitos custo e tempo de execução, apresentou melhor custo por unidade de tempo de alimentação em todos os testes realizados utilizando-se estados aleatórios, apresentando assim benefícios mesmo nos testes que não obteve o melhor desempenho no custo e tempo de execução.

Considerando-se que o tempo total de execução do algoritmo APIA, dado pela soma dos tempos de treinamento e simulação, só deve ser levado em conta uma única

vez, sendo que a partir de sua segunda execução só deve ser considerado o tempo de simulação, o modelo APIA desenvolvido apresenta larga vantagem frente ao método de resolução Modelo Determinístico PLIM - Política 2. Para clarificar tal dedução, supondo que seja necessário executar cada algoritmo de resolução 10 vezes utilizando-se a instância *16_60_2_teste.xml*, a execução do teste B-up do modelo APIA, que apresentou o maior tempo de simulação, levaria 219258,7 segundos, o que equivale a 60 horas. Já a execução da Política 2 do Modelo Determinístico despenderia de aproximadamente 318 horas.

Portanto, pode-se concluir que para a resolução do problema estocástico do *tripper*, o método APIA é competitivo quando é realizada a etapa de treinamento para a instância utilizada e se destaca para grandes períodos de tempo, devido à sua escalabilidade. No entanto, possui como limitação o fato do treinamento não ser generalizável a outros estados iniciais, o que faz com que seja necessário que o treinamento seja realizado a cada vez que o estado inicial do sistema se modificar. Já o método Modelo Determinístico, além de apresentar desempenho constante, independente do estado inicial utilizado, apresenta melhor custo por unidade de tempo de alimentação. Todavia, o seu tempo de execução cresce consideravelmente com o número de decisões.

Conclusões e Trabalhos Futuros

Por meio da realização deste trabalho, pode-se concluir que o algoritmo de aproximação de programação dinâmica desenvolvido, baseado no método de Iteração de Política Aproximado *LSTD*, foi eficiente em resolver tanto o problema determinístico quanto o problema estocástico de movimentação do tripper, uma vez que foi capaz de gerar políticas que resolveram os problemas de forma satisfatória.

No que se diz respeito ao problema determinístico, por meio da comparação dos resultados obtidos com os resultados encontrados na literatura, decorrentes do trabalho de Caldas e Martins (2018), constata-se que o presente trabalho apresenta ótimo desempenho no quesito tempo de execução. Tal conclusão ainda é válida mesmo considerando-se o tempo de treinamento, uma vez que o treinamento, que consome o maior tempo, só ocorre uma única vez, ao passo que no modelo de Caldas e Martins (2018) o tempo de execução é fixo em todas as execuções. Já a análise comparativa a respeito do controle dos níveis dos silos não pode ser realizada, pois a modificação na instância utilizada alterou o critério da ocorrência de folgas.

No que tange ao problema estocástico, os resultados obtidos mostram que o método de resolução por aproximação da programação dinâmica apresenta bom desempenho no quesito custo quando realizado o treinamento específico para a instância a ser utilizada. Com relação ao tempo de execução, o método proposto se destaca por sua escalabilidade, em que é possível que seja executado diversas vezes sem piorar consideravelmente o desempenho no quesito tempo. No entanto, verificou-se que o desempenho do algoritmo é altamente sensível à combinação de funções indicadoras utilizada, o que ressalta a importância da etapa de criação das funções, uma vez que são criadas a partir de perspectivas do programador em relação ao sistema. Dessa forma, pode-se concluir que a resolução de um problema também consiste em desenvolver boas funções indicadoras.

Com relação ao método Zigue-Zague, conclui-se que o mesmo não deve ser considerado como alternativa de resolução, uma vez que, apesar de apresentar tempo de execução significativamente inferior, apresenta um desempenho ruim no quesito custo, o que indica que não é capaz de controlar os níveis dos silos dentro dos limites. Em contrapartida, o método Modelo Determinístico Baseado MILP, sobretudo a Política 2, apresenta resultados competitivos nos quesitos custo e tempo de execução, além de apresentar melhores resultados relativos ao custo por unidade de tempo de alimentação, mas é caracterizado pelo fato do tempo de execução crescer significativamente com o aumento de decisões.

Em suma, por meio das análises realizadas, verifica-se que o método de aproximação de programação dinâmica é eficiente tanto em resolver o problema sob uma nova abordagem

quanto em apresentar melhores resultados no quesito tempo. No que diz respeito ao controle dos níveis dos silos, infere-se que o seu desempenho depende de fatores como as funções e a instância utilizadas, e que no caso do problema estocástico o método é poderoso em realizar o treinamento para um estado inicial, mas não consegue generalizar os coeficientes obtidos para outros estados. Além disso, infere-se que o método Modelo Determístico Baseado MILP, desenvolvido para solução do modelo estocástico, também desponta como método viável para a solução do problema.

Como proposta de trabalhos futuros, sugere-se a implementação de novas perspectivas de tomada de decisão, como a possibilidade do tripper não despejar minério sobre o silo em que está, a inutilização de um ou mais silos ou a possibilidade do tripper depositar apenas um parcela da quantidade minério proveniente da taxa de entrada.

Além disso, considerando-se a natureza dinâmica das plantas de beneficiamento de minério, e que a maioria de seus processos possuem demandas e consumos distintos, torna-se interessante que as taxas de saída de cada silo sejam dadas em função dos processos, sendo resultantes de um planejamento da produção.

Sugere-se também que a etapa de treinamento seja realizada de modo que o treinamento seja mais genérico, possibilitando que o controle do sistema seja realizado para qualquer estado, e não somente para o estado inicial que o treinamento foi realizado. Propõe-se ainda que as decisões sejam tomadas de forma gulosa, sem ser realizada a etapa de treinamento, a fim de que o efeito do treinamento possa ser analisado.

Por fim, sugere-se a inclusão no modelo de outros equipamentos e seus respectivos processos e dinamicidades, como por exemplo a inclusão de alimentadores e a modelagem refente ao seu funcionamento e incertezas, como uma situação de quebra, por exemplo.

Referências

AKYOL, D. E.; BAYHAN, G. M. A review on evolution of production scheduling with neural networks. *Computers & Industrial Engineering*, Elsevier, v. 53, n. 1, p. 95–122, 2007. Citado na página 7.

ALATARTSEV, S.; STELLMACHER, S.; ORTMEIER, F. Robotic task sequencing problem: A survey. *Journal of intelligent & robotic systems*, Springer, v. 80, n. 2, p. 279–298, 2015. Citado na página 7.

ALBUQUERQUE, K. et al. Averaging level control of bulk solid material using a tripper car. *IFAC-PapersOnLine*, Elsevier, v. 52, n. 14, p. 147–152, 2019. Citado 2 vezes nas páginas 12 e 23.

ALLAHVERDI, A. et al. A survey of scheduling problems with setup times or costs. *European journal of operational research*, Elsevier, v. 187, n. 3, p. 985–1032, 2008. Citado na página 6.

ARENALES, M. et al. *Pesquisa Operacional*. [S.l.]: Elsevier, 2007. Citado na página 13.

BARRETO, A. d. M. S. *Soluções Aproximadas para Problemas de Tomada de Decisão Sequencial*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2008. Citado na página 21.

BELHE, U.; KUSIAK, A. Dynamic scheduling of design activities with resource constraints. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, IEEE, v. 27, n. 1, p. 105–111, 1997. Citado na página 10.

BELLMAN, R. E. *Dynamic Programming*. [S.l.]: Princeton UniversityPress, 1957. Citado 2 vezes nas páginas 13 e 15.

BELLMAN, R. E. *Adaptive Control Processes: A Guided Tour*. [S.l.]: Princeton University Press, 1961. Citado na página 19.

BERTSEKAS, D. P. *Dynamic programming and optimal control*. [S.l.]: Athena scientific Belmont, 1995. v. 1. Citado na página 19.

BERTSEKAS, D. P.; TSITSIKLIS, J. N. *Neuro-dynamic programming*. [S.l.]: Athena Scientific, 1996. Citado na página 28.

BRADTKE, S. J.; BARTO, A. G. Linear least-squares algorithms for temporal difference learning. *Machine learning*, Springer, v. 22, n. 1-3, p. 33–57, 1996. Citado na página 21.

CALDAS, F. N. *Propostas para Solução do Problema de Movimentação de Tripper*. Dissertação (Mestrado) — Universidade Federal de Ouro Preto, 2018. Citado 4 vezes nas páginas 1, 2, 11 e 23.

CALDAS, F. N.; MARTINS, A. X. Proposed solutions to the tripper car positioning problem. In: *Proceedings of the 20th International Conference on Enterprise Information Systems - ICEIS*. [S.l.]: SciTePress, 2018. p. 344–352. Citado 10 vezes nas páginas 3, 4, 6, 8, 2, 11, 37, 53, 54 e 92.

ÇALIŞ, B.; BULKAN, S. A research survey: review of ai solution strategies of job shop scheduling problem. *Journal of Intelligent Manufacturing*, Springer, v. 26, n. 5, p. 961–973, 2015. Citado na página 7.

CAUMOND, A. et al. An milp for scheduling problems in an fms with one vehicle. *European Journal of Operational Research*, Elsevier, v. 199, n. 3, p. 706–722, 2009. Citado 2 vezes nas páginas 8 e 9.

CHRYSSOLOURIS, G.; SUBRAMANIAM, V. Dynamic scheduling of manufacturing job shops using genetic algorithms. *Journal of Intelligent Manufacturing*, Springer, v. 12, n. 3, p. 281–293, 2001. Citado na página 10.

COWLING, P.; JOHANSSON, M. Using real time information for effective dynamic scheduling. *European journal of operational research*, Elsevier, v. 139, n. 2, p. 230–244, 2002. Citado na página 10.

DANG, Q.-V. et al. Scheduling a single mobile robot for part-feeding tasks of production lines. *Journal of Intelligent Manufacturing*, Springer, v. 25, n. 6, p. 1271–1287, 2014. Citado 2 vezes nas páginas 7 e 9.

ERNST, D.; GEURTS, P.; WEHENKEL, L. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, v. 6, n. Apr, p. 503–556, 2005. Citado na página 16.

FATTAHI, P.; FALLAHI, A. Dynamic scheduling in flexible job shop systems by considering simultaneously efficiency and stability. *CIRP Journal of Manufacturing Science and Technology*, Elsevier, v. 2, n. 2, p. 114–123, 2010. Citado na página 10.

GIFFLER, B.; THOMPSON, G. L. Algorithms for solving production-scheduling problems. *Operations research*, INFORMS, v. 8, n. 4, p. 487–503, 1960. Citado na página 6.

GORDON, V.; PROTH, J.-M.; CHU, C. A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, Elsevier, v. 139, n. 1, p. 1–25, 2002. Citado na página 6.

GUPTA, S. K.; KYPARISIS, J. Single machine scheduling research. *Omega*, Elsevier, v. 15, n. 3, p. 207–227, 1987. Citado na página 6.

HE, Y. et al. Segment set-based part input sequencing in flexible manufacturing systems. *International Journal of Production Research*, Taylor & Francis, v. 53, n. 17, p. 5106–5117, 2015. Citado 2 vezes nas páginas 7 e 9.

HE, Y.; STECKE, K. E.; SMITH, M. L. Robot and machine scheduling with state-dependent part input sequencing in flexible manufacturing systems. *International Journal of Production Research*, Taylor & Francis, v. 54, n. 22, p. 6736–6746, 2016. Citado 2 vezes nas páginas 7 e 9.

HURINK, J.; KNUST, S. A tabu search algorithm for scheduling a single robot in a job-shop environment. *Discrete applied mathematics*, Elsevier, v. 119, n. 1-2, p. 181–203, 2002. Citado na página 7.

HURINK, J.; KNUST, S. Tabu search algorithms for job-shop problems with a single transport robot. *European journal of operational research*, Elsevier, v. 162, n. 1, p. 99–111, 2005. Citado 2 vezes nas páginas 8 e 9.

INSTITUTO BRASILEIRO DE MINERAÇÃO. *Informações sobre a economia mineral brasileira 2020*: Ano base 2019. Brasília, 2020. 80 p. Citado na página 4.

JOHNSON, S. M. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, Wiley Online Library, v. 1, n. 1, p. 61–68, 1954. Citado na página 6.

KARELOVIC, P.; PUTZ, E.; CIPRIANO, A. A framework for hybrid model predictive control in mineral processing. *Control Engineering Practice*, Elsevier, v. 40, p. 1–12, 2015. Citado na página 11.

KOULAMAS, C. The single-machine total tardiness scheduling problem: review and extensions. *European Journal of Operational Research*, Elsevier, v. 202, n. 1, p. 1–7, 2010. Citado na página 6.

LEI, D. Multi-objective production scheduling: a survey. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 43, n. 9-10, p. 926, 2009. Citado na página 6.

LI, Z.; IERAPETRITOU, M. Process scheduling under uncertainty: Review and challenges. *Computers & Chemical Engineering*, Elsevier, v. 32, n. 4-5, p. 715–727, 2008. Citado na página 6.

METAXIOTIS, K. S.; ASKOUNIS, D.; PSARRAS, J. Expert systems in production planning and scheduling: A state-of-the-art survey. *Journal of Intelligent Manufacturing*, Springer, v. 13, n. 4, p. 253–260, 2002. Citado na página 7.

MINISTÉRIO DE MINAS E ENERGIA. *Boletim do Setor Mineral*. Brasília, 2020. 28 p. Citado na página 1.

MORAIS, B. S. d. *Problema de movimentação de tripper abordado por programação dinâmica.* 2019. 43 p. Trabalho de Conclusão de Curso (Bacharel em Engenharia de Producão), Universidade Federal de Ouro Preto, João Monlevade, Brasil. Citado na página 11.

MORAIS, B. S. d. et al. Uma abordagem dinâmica para o problema de movimentação do carro tripper. In: *Proceedings of the Brazilian Symposium on Operations Research*. [S.l.]: SOBRAPO, 2020. Citado 7 vezes nas páginas 2, 4, 11, 23, 24, 28 e 63.

NÚÑEZ, W.; SOLEDAD, S. *Mejoramiento de la gestión de carga viva en acopio Los Colorados, Minera Escondida Ltda.* 2013. 93 p. Memoria (Ingenieria Civil de Minas), Universidad de Chile, Santiago, Chile. Citado 2 vezes nas páginas 1 e 22.

OUELHADJ, D.; PETROVIC, S. A survey of dynamic scheduling in manufacturing systems. *Journal of scheduling*, Springer, v. 12, n. 4, p. 417, 2009. Citado na página 10.

PEDROSA, L. O. M. *Problema de Movimentação do Carro Tripper - Comparação de Novas políticas*. 2019. 39 p. Trabalho de Conclusão de Curso (Bacharel em Engenharia de Producão), Universidade Federal de Ouro Preto, João Monlevade, Brasil. Citado 4 vezes nas páginas 2, 11, 35 e 68.

PINEDO, M. *Scheduling*: theory, algorithms, and systems. [S.l.]: Springer Science & Business Media, 2012. Citado na página 6.

POWELL, W. B. *Approximate Dynamic Programming: Solving the curses of dimensionality*. [S.l.]: John Wiley & Sons, 2007. v. 703. Citado 10 vezes nas páginas 4, 13, 14, 15, 16, 17, 19, 20, 28 e 33.

PUTERMAN, M. L. *Markov decision processes: discrete stochastic dynamic programming*. [S.l.]: John Wiley & Sons, 2014. Citado 2 vezes nas páginas 13 e 14.

RAJA, P.; PUGAZHENTHI, S. Optimal path planning of mobile robots: A review. *International journal of physical sciences*, Citeseer, v. 7, n. 9, p. 1314–1320, 2012. Citado na página 7.

ROTTER, J. M. Silos and tanks in research and practice: state of the art and current challenges. In: *Evolution and Trends in Design, Analysis and Construction of Shell and Spatial Structures*. [S.l.]: Editorial Universitat Politècnica de València, 2009. Citado na página 22.

SANTOS, M. S. et al. Simheuristic-based decision support system for efficiency improvement of an iron ore crusher circuit. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 94, 2020. Citado na página 12.

SURESH, V.; CHAUDHURI, D. Dynamic scheduling—a survey of research. *International journal of production economics*, Elsevier, v. 32, n. 1, p. 53–63, 1993. Citado na página 10.

SUTTON, R. S. *Temporal Credit Assignment in Reinforcement Learning*. Tese (Doutorado) — University of Massachusetts Amherst, 1985. Citado na página 20.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018. Citado na página 18.

TEIXEIRA, H. T. *Aprendizado por reforço e programação dinâmica aproximada com máquinas kernel para controle de sistemas não lineares*. Tese (Doutorado) — Universidade Estadual de Campinas, Campinas, 2016. Citado 2 vezes nas páginas 18 e 28.

VIANA, S. A. A. Autonomous positioning advisor for trippers in mineral processing facilities. *Journal of Applied Instrumentation and Control*, v. 6, n. 1, p. 20–30, 2018. Citado na página 12.

WAGNER, H. M. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, v. 6, n. 2, p. 131–140, 1959. Citado na página 6.

WATKINS, C. J. C. H. *Learning from delayed rewards*. Tese (Doutorado) — King's College, 1989. Citado na página 18.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. *Machine learning*, Springer, v. 8, n. 3-4, p. 279–292, 1992. Citado na página 18.

WILLS, B. A.; FINCH, J. *Wills' mineral processing technology: an introduction to the practical aspects of ore treatment and mineral recovery*. [S.l.]: Butterworth-Heinemann, 2015. Citado na página 1.

ZACHARIA, P. T.; ASPRAGATHOS, N. Optimal robot task scheduling based on genetic algorithms. *Robotics and Computer-Integrated Manufacturing*, Elsevier, v. 21, n. 1, p. 67–79, 2005. Citado na página 7.