Felipe Novaes Caldas

Propostas para Solução do Problema de Movimentação de *Tripper*

Ouro Preto
junho de 2018

Felipe Novaes Caldas

Propostas para Solução do Problema de Movimentação de *Tripper*

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para a obtenção do título de Mestre.

Universidade Federal de Ouro Preto – UFOP
Departamento de Computação – DECOM
Programa de Pós-Graduação

Orientador: Alexandre Xavier Martins
Coorientador: Marcone Jamilson Freitas Souza

Ouro Preto
junho de 2018

C145p Caldas, Felipe Novaes.
Propostas para solução do problema de movimentação de tripper [manuscrito]: MR / Felipe Novaes Caldas. - 2018.
75f.: il.: color; grafs; tabs.

Orientador: Prof. Dr. Alexandre Xavier Martins.
Coorientador: Prof. Dr. Marcone Jamilson Freitas Souza.

Dissertação (Mestrado) - Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Biológicas. Departamento de Computação. Programa de Pós-Graduação em Ciência da Computação.
Área de Concentração: Ciência da Computação.

1. Otimização combinatória. 2. Métodos heurísticos. 3. Beneficiamento de minério. I. Martins, Alexandre Xavier. II. Souza, Marcone Jamilson Freitas. III. Universidade Federal de Ouro Preto. IV. Titulo.

CDU: 004.023

Catalogação: www.sisbin.ufop.br

Ata da Defesa Pública de Dissertação de Mestrado

Aos 18 dias do mês de junho de 2018, às 14 horas na Sala de Seminários do DECOM no Instituto de Ciências Exatas e Biológicas (ICEB), reuniram-se os membros da banca examinadora composta pelos professores: **Prof. Alexandre Xavier Martins (presidente e orientador), Prof. Dr. Marcone Jamilson Freitas Souza (coorientador), Prof. Dr. Marco Antônio Moreira de Carvalho e Prof. Dr. Ricardo Saraiva de Camargo**, aprovada pelo Colegiado do Programa de Pós-Graduação em Ciência da Computação, a fim de arguirem o mestrando **Felipe Novaes Caldas**, com o título **"Propostas para Solução do Problema de Movimentação de Tripper"**. Aberta a sessão pelo presidente, coube ao candidato, na forma regimental, expor o tema de sua dissertação, dentro do tempo regulamentar, sendo em seguida questionado pelos membros da banca examinadora, tendo dado as explicações que foram necessárias.

Recomendações da Banca:

(X) Aprovada sem recomendações

( ) Reprovada

( ) Aprovada com recomendações: _____

Banca Examinadora:

_____
Prof. Dr. Alexandre Xavier Martins

_____
Prof. Dr. Marcone Jamilson Freitas Souza

_____
Prof. Dr. Marco Antônio Moreira de Carvalho

_____
Prof. Dr. Ricardo Saraiva de Camargo

_____
Prof. Dr. Joubert de Castro Lima
Coordenador do Programa de Pós-Graduação em Ciência da Computação
DECOM/ICEB/UFOP

Ouro Preto, 18 de junho de 2018.

Agradecimentos

À Deus por me iluminar e dar forças ao longo de toda esta jornada.

Aos meus pais, avós e toda minha família que sempre estiveram do meu lado me apoiando e incentivando por toda vida.

À Isabela por todas as suas sugestões importantes e por todo o carinho e paciência.

À Vale por ter propiciado todos estes anos de dedicação extraordinária à formação acadêmica, principalmente aos colegas e amigos Roberto Aquino e Thiago Rezende por todo apoio e motivação incondicionais que foram fundamentais para que eu chegasse até aqui.

À Mariana e aos gentis funcionários e professores que sempre foram solícitos e atenciosos em ajudar nos momentos de necessidade. Também agradeço à CAPES, FAPEMIG, CNPq e UFOP por todo o apoio a pesquisa que recebemos no Departamento de Computação. Especial agradecimento à PROGRAD por viabilizar a apresentação do artigo referente à este trabalho no exterior.

"A verdadeira sabedoria está em reconhecer a própria ignorância."
Sócrates

Resumo

O *tripper* é um equipamento frequentemente encontrado em uma planta de beneficiamento mineral. Sua função é distribuir o minério proveniente de uma correia transportadora sobre um silo de estocagem. A movimentação de *tripper* é um problema de sequenciamento definido pela determinação do posicionamento do equipamento sobre um silo ao longo do tempo. A escassez de referências na literatura científica que descrevam detalhadamente o tema em questão releva a importância deste trabalho em propor soluções a um problema que, apesar de receber pouca atenção do meio acadêmico, possui grande importância em muitas instalações de tratamento de minério ao redor do mundo. O primeiro passo é propor a modelagem do sistema silo-*tripper* na forma de um programa linear inteiro misto, de modo que seja possível determinar uma trajetória ótima de movimentação para o equipamento. Dois paradigmas foram utilizados para obter soluções exatas para este modelo: programação linear inteira mista e programação dinâmica. Embora tenham sido efetivas em solucionar instâncias pequenas, estas duas abordagens se mostraram ineficientes ao lidar com instâncias de dimensões mais elevadas, já que o tempo necessário para se alcançar a solução exata é muito alto, inviabilizando-se aplicações reais em silos com muitos compartimentos. Buscando-se alcançar soluções relativamente boas em relação ao ótimo, mas levando muito menos tempo, as meta-heurísticas GRASP e *Simulated Annealing* (SA) foram adaptadas como alternativa aos métodos exatos, representando esses algoritmos a segunda contribuição deste trabalho. O desempenho do GRASP se mostrou muito superior aos resultados obtidos pelo SA, tanto em relação ao tempo despendido quanto à assertividade em atingir soluções exatas. Os resultados importantes alcançados pela programação dinâmica e pelo GRASP os tornam fortes candidatos à implantação em aplicações reais, em situações que tanto precisão quanto tempo de resposta sejam pré-requisitos necessários.

Palavras-chave: *tripper*. otimização combinatória. meta-heurísticas.

Abstract

Tripper is an equipment usually found in a mineral processing plant, responsible for distributing the ore coming from previous stages of the process to a storage silo. The tripper movimentation is a scheduling problem defined by determining the positioning of the equipment of a silo over time. The few references of the scientific literature that describe the subject in detail highlights the importance of this work in proposing solutions to the problem that holds great importance in many ore treatment facilities around the world. The first step was to model the silo-tripper system as a mixed integer linear program to get optimal trajectories for the equipment. Two paradigms were used to obtain exact solutions for this model: mixed integer linear programming and dynamic programming. Although they were effective in solving small instances, these two approaches proved to be inefficient when dealing with large instances since the time required to reach exact solution was computationally high, making them possible to apply to real problems. To achieve relatively good solutions with respect to the optimum solutions but at cheaper computationally effort, the GRASP and Simulated Annealing meta-heuristics were adapted as alternative method. The GRASP algorithm performed better than the SA, both in terms of time spent and assertiveness to reach exact solutions. The important results achieved by dynamic programming and GRASP made them strong candidates for deployment in real applications, in which both precision and response time are prerequisites.

Keywords: tripper. combinatorial optimization. metaheuristics.

Lista de ilustrações

Sumário

1 Introdução

Uma planta de beneficiamento mineral é constituída de vários processos sequenciais cujo objetivo é transformar o minério em estado bruto, proveniente da frente de lavra de uma mina, em produtos com propriedades químicas e físicas particulares que atendam as especificações de mercado. As etapas necessárias a uma unidade de tratamento para atingir estes requisitos comerciais estão listadas na Figura 1 (WILLS; NAPIER-MUNN, 2015). Estas fases de beneficiamento estão dispostas de maneira sequencial, partindo-se do minério extraído e proveniente de etapas anteriores e prosseguindo-se sequencialmente até alcançar os produtos finais desejados que são diferenciados como concentrado e rejeito[1]. O primeiro atende as especificações físico-químicas estabelecidas para o processo e o segundo agrega todos os subprodutos indesejados e descartados ao longo de toda a cadeia de beneficiamento.

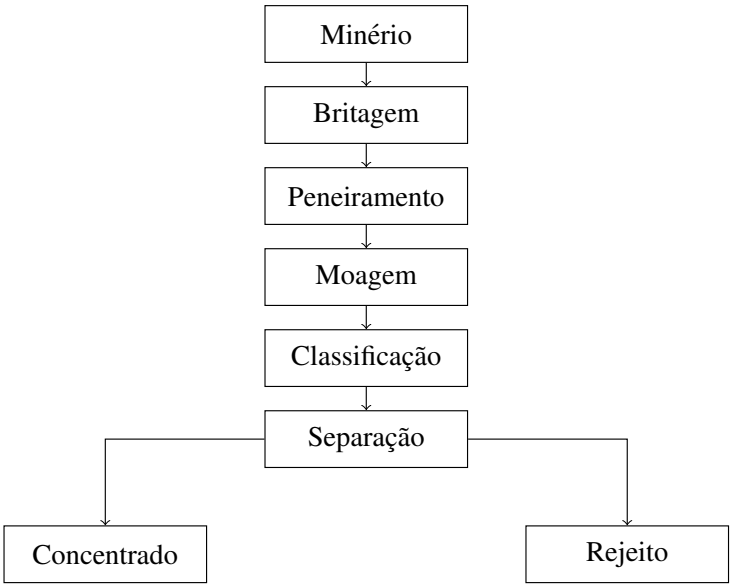


Figura 1 – Fluxograma simplificado ilustrando uma planta de beneficiamento mineral.

Os processos de britagem, peneiramento e parte da moagem frequentemente compõem a etapa de beneficiamento à seco de uma planta, não necessitando de utilização de água para realização das operações de tratamento. As ações aplicadas por estes processos se restringem a transformações físicas definidas por redução da granulometria, separação por faixa granulométrica e transporte do minério entre os processos. Sucessivas etapas de britagem e peneiramento são necessárias para redução da granulometria do minério proveniente da mina em estado bruto, de forma que as especificações granulométricas

[1] Após cada uma das etapas intermediárias, ao invés de prosseguir para a próxima etapa, a saída pode ser redirecionada para uma etapa anterior ou posterior do processo caso seja necessário. Estes desvios de rota possíveis não estão representados no fluxograma por fins de simplificação.

mínimas necessárias pelas etapas posteriores do processo sejam atingidas. O papel da britagem em uma planta de beneficiamento consiste na redução do tamanho das partículas por meio de compressão ou projeção do minério contra superfícies rígidas (WILLS; NAPIER-MUNN, 2015). O produto da britagem deve ser classificado para determinação da fração do material que prossegue ao longo da cadeia de beneficiamento, por atender aos requisitos granulométricos desejados, e de qual parte permanece acima da especificação mínima e deve ser novamente processada para assegurar o cumprimento do requisito de granulometria mínima definida para esta etapa.

Dentro de uma planta de beneficiamento a seco, o minério pode ser estocado em tanques, pilhas ou silos, dependendo da natureza do material processado e das características particulares do processo vinculado ao armazenamento. A necessidade de estocar o minério surge por razões variadas: diferentes partes do processo operam em taxas de vazão mássica diversas, alguns são intermitentes e outros contínuos, alguns apresentam nível elevado de indisponibilidade devido a manutenções frequentes. Desta forma, são necessários estoques intermediários entre as várias etapas do processo para que a operação da planta não seja irregular e não se torne inviável economicamente (WILLS; NAPIER-MUNN, 2015).

Os silos são estruturas civis projetadas para armazenar materiais granulados secos. Correias transportadoras depositam nos silos o minério proveniente de uma etapa anterior do processo. Posteriormente, este material é retirado por equipamentos chamados alimentadores, localizados abaixo de cada um dos compartimentos do silo, dando prosseguimento ao processo de beneficiamento. O número de alimentadores necessários em um sistema de estocagem é proporcional ao volume de minério que será processado. Logo, um silo deve ser subdividido em compartimentos menores, de acordo com o número de alimentadores estabelecido, disponibilizando-se uma área individualizada para instalação de cada um destes equipamentos sob a estrutura do silo.

É possível observar na Figura 2a vigas verticais presentes ao longo da estrutura inferior do edifício localizado no centro da imagem, indicando a localização aproximada das subdivisões do silo. A área definida por das vigas e preenchida por uma parede de concreto é a região que delimita um compartimento ou boca do silo. Existem dezesseis subdivisões neste processo apresentado. À direita da imagem há uma estrutura em formato curvo que se projeta para dentro da lateral direita do silo. Esta é a correia transportadora que conduz o material destinado a ser armazenado no silo.

O *tripper* é um sistema mecânico projetado para distribuir o minério ao longo da abertura superior de um silo de armazenamento. Este equipamento é constituído por uma estrutura metálica móvel que suporta fisicamente um ponto de descarga de uma correia transportadora. A movimentação do carro é realizada por rodas de aço situadas sob sua estrutura. Trilhos metálicos sustentam e guiam o *tripper* longitudinalmente ao longo do silo, permitindo a distribuição do minério conduzido pela correia transportadora convenientemente entre todas as subdivisões (SWINDERMAN, 2014). O fluxo de minério

(a) Silo SL-133A-9101

(b) *Tripper* TC-137A-9103

Figura 2 – Exemplos de um *tripper* na mina de Brucutu – Vale.

não é interrompido durante o processo de locomoção.

Um exemplo de *tripper* pode ser visto na Figura 2b. Este equipamento se localiza sobre o silo SL-133A-9101. A correia transportadora pode ser observada na parte inferior esquerda da imagem citada e percorre toda a parte superior do silo. A abertura do silo está situada abaixo da correia transportadora e atrás do guarda-corpo, o minério é depositado nessa região. O *tripper* é visto na área central da Figura 2b, sobre a correia transportadora. O ponto de descarga se estende da parte superior do equipamento e prossegue até a abertura do silo. Um dos trilhos do carro está localizado a direita do guarda-corpo, sobre ele estão as rodas direitas frontais do *tripper*.

O fluxograma mostrado na Figura 3 apresenta um exemplo ilustrativo de um processo de peneiramento à seco. O fluxo de alimentação é proveniente de uma correia transportadora e distribuído entre os quatro compartimentos do silo pelo *tripper*. Posteriormente, o minério é retomado pelos quatro alimentadores e direcionado para as peneiras. Nessa etapa, o material é classificado de acordo com sua granulometria. A parte acima da especificação mínima retorna ao processo anterior como subproduto 1 e o restante, dentro da especificação, prossegue adiante como subproduto 2.

O propósito deste trabalho é propor soluções para o problema de sequenciamento da movimentação de *tripper*. Esta metodologia é definida como um processo de tomada de decisão que lida com alocação de recursos para realização de tarefas ao longo de um período de tempo e sua meta é otimizar um ou mais objetivos (PINEDO, 2012). O objetivo, no caso do problema proposto, é a determinação do direcionamento do equipamento considerando-se o estado atual e as predições das consequentes ações tomadas ao longo do tempo. O sistema silo-*tripper* é representado por um modelo matemático que descreve de forma suficiente as características fundamentais do sistema.

A primeira contribuição deste trabalho é a proposição de métodos exatos, considerando-se a representação matemática como um problema de programação linear inteira mista. Neste caso, o objetivo é minimizar a variação dos níveis do silo ao longo do tempo, além da estabilização do processo, restringindo as variáveis aos limites estabelecidos. Em seguida,

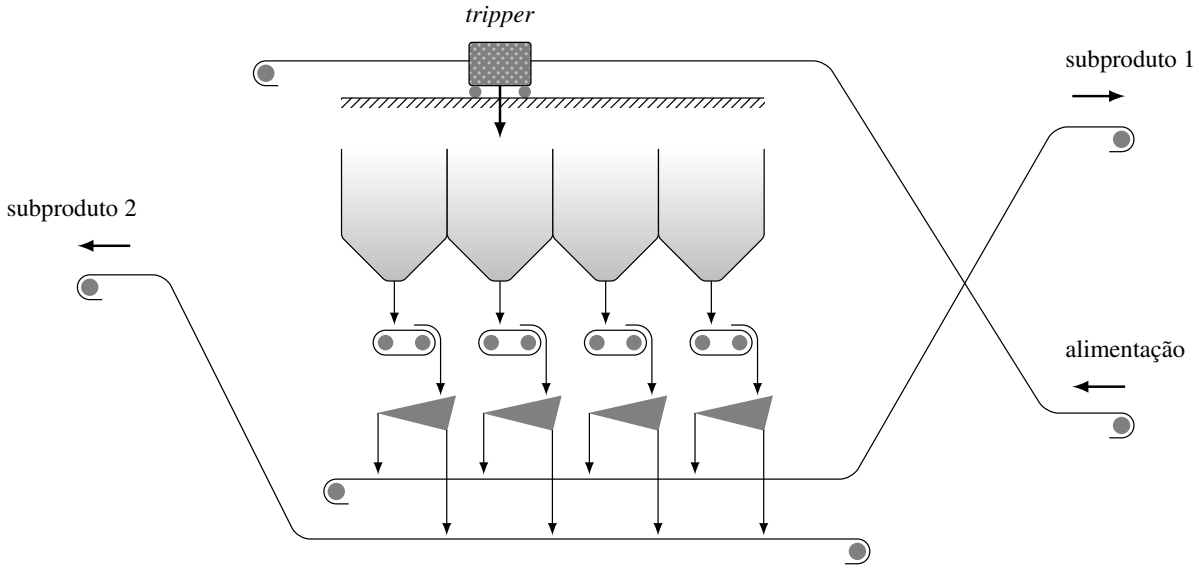


Figura 3 – Diagrama exemplificando um processo de peneiramento de minério granulado. O *tripper* é representado por um retângulo pontilhado com duas rodas. Um silo de quatro compartimentos é mostrado sob o *tripper*. Abaixo de cada uma das saídas do silo há um alimentador de correia. As peneiras são representadas por triângulos na cor cinza. As correias transportadoras estão sinalizadas pelo fluxo de minério que transportam: alimentação, subproduto 1 e subproduto 2.

um algoritmo de programação dinâmica é proposto para o problema com o propósito de reduzir o tempo necessário para se encontrar a solução exata em relação aos resolvedores de programação linear inteira mista. Finalmente, meta-heurísticas são propostas buscando-se reduzir o tempo despendido na busca de uma solução aceitável para o problema. Dois algoritmos heurísticos são descritos e testados: o GRASP e o *Simulated Annealing*.

1.1 Objetivos

Este trabalho está dividido em objetivo geral, aquele que define o propósito fundamental que norteou esta pesquisa, e específicos, que são objetivos secundários que proporcionam o cumprimento do objetivo geral ao caracterizar cada aspecto necessário para seu atingimento.

1.1.1 Objetivo Geral

O objetivo principal deste trabalho é *propor soluções para o problema de posicionamento de tripper*. A escassez de referências que abordam o tema na literatura faz necessário a busca por novas proposições que possam ser aplicadas em processos reais na indústria. O objetivo é o desenvolvimento de soluções para o problema utilizando-se de técnicas conhecidas de programação matemática e meta-heurísticas.

1.1.2 Objetivos Específicos

Os objetivos específicos são divididos em quatro partes:

1. O primeiro objetivo deste trabalho é criar uma descrição matemática para o problema de movimentação do *tripper*. As características físicas do processo são representadas de forma suficientemente detalhada para que a utilização de sistemas de otimização para solução do problema de movimentação seja viabilizada. Este modelo deve ser escalável, isto é, ser capaz de representar silos com qualquer número de compartimentos e iterações ao longo do tempo.

2. O segundo objetivo é resolver o problema de movimentação por meio de programação linear inteira mista. Resolvedores comerciais são escolhidos para a tarefa de encontrar a solução do modelo de otimização proposto para o problema.

3. O desenvolvimento de um algoritmo para solução do problema, que não requeira a utilização de resolvedores, será o terceiro objetivo deste trabalho. O propósito é reduzir o tempo gasto para se encontrar a solução exata de uma instância do problema. O tempo computacional despendido na resolução de uma instância é considerado um fator importante para a aplicação da metodologia no mundo real.

4. O último objetivo é o desenvolvimento de meta-heurísticas capazes de resolver o problema de posicionamento do *tripper* em tempo menor que os métodos exatos. Estes algoritmos, mesmo não necessariamente alcançando o ótimo, geram soluções boas que podem ser utilizadas em aplicações reais com algum tipo de compromisso.

1.2 Organização do Trabalho

O restante da estrutura deste trabalho está dividida em quatro partes: referencial teórico, metodologia, resultados e conclusão. A seguir uma descrição sucinta de cada destes capítulos.

O **Capítulo 2** apresenta uma revisão da bibliografia a respeito dos métodos adotados ao longo deste trabalho. Primeiramente, a programação linear inteira mista é descrita. Em seguida, um método alternativo para busca de soluções exatas é abordado, i.e., o paradigma de programação dinâmica. Finalmente, as meta-heurísticas adotadas neste texto são conceitualizadas. Inicialmente, o algoritmo GRASP é descrito. Em sequência, o algoritmo *Simulated Annealing* é apresentado.

O **Capítulo 3** descreve o desenvolvimento das metodologias discutidas ao longo deste trabalho. O primeiro passo é detalhar o funcionamento do sistema silo-*tripper*, com o objetivo de representá-lo matematicamente por meio de modelos que representem adequadamente o comportamento do sistema ao longo do tempo. Em seguida, os métodos

exatos adotados são apresentados. Primeiramente, o sistema é mapeado como um problema de programação linear inteira. Adiante, um algoritmo de programação dinâmica é proposto para solução do problema. Finalmente, as meta-heurísticas adaptadas para solução do problema são detalhadas: os algoritmos GRASP e *Simulated Annealing*.

O **Capítulo 4** mostra as metodologias desenvolvidas e apresentadas na seção anterior são testadas e validadas. O conjunto de instâncias escolhida para os testes é definido de acordo com as particularidades de cada teste. Os métodos exatos são comparados com as meta-heurísticas verificando-se a assertividade das soluções encontradas e a redução no tempo de busca por uma solução pelos algoritmos meta-heurísticos.

Por fim, no **Capítulo 5**, os resultados alcançados pelo trabalho são avaliados. É feita uma discussão sobre os objetivos atingidos por meio das metodologias adotadas. Concluise a respeito das contribuições resultantes dos desenvolvimentos propostos. Por fim, são feitos comentários sobre as oportunidades de desenvolvimento em aberto e os próximos passos sugeridos para continuidade desta pesquisa.

2 Referencial Teórico

Neste capítulo é feita uma revisão da literatura disponível a respeito das metodologias utilizadas neste trabalho. Primeiramente, os métodos exatos são apresentados, começando-se pela programação linear inteira mista e prosseguindo-se com a programação dinâmica. Em seguida, as meta-heurísticas *Greed Randomized Adaptive Search Procedure* e *Simulated Annealing* são detalhadas, ilustrando-se o funcionamento destes algoritmos.

2.1 Programação Linear Inteira Mista

O problema de programação linear (LP) é uma classe de problemas de otimização cujo propósito é encontrar o máximo ou o mínimo de uma função objetivo z com respeito a um conjunto de variáveis contínuas $(x_1, x_2, ..., x_n)$, respeitando-se um conjunto de restrições lineares que determinam os limites destas variáveis. A Equação (2.1) mostra a representação matemática do problema LP:

$$\begin{aligned} \text{Maximizar} \quad & z = \sum_j c_j x_j \\ \text{sujeito a} \quad & \sum_j a_{ji} x_j \leq b_i \quad (i = 1, 2, \ldots, m) \\ & x_j \geq 0 \qquad\qquad (j = 1, 2, \ldots, n) \end{aligned} \tag{2.1}$$

sendo que

z é a função objetivo

c_j são os coeficientes de custo

x_j são as variáveis de decisão

a_{ji} são os coeficientes das restrições

b_i são as constantes das restrições

m é o número de restrições

n é o número de variáveis

Caso ao menos uma das variáveis do problema de programação linear seja inteira, o problema é dito como de programação linear inteira mista (MILP) (WOLSEY, 1998).

Apesar de partir do mesmo princípio dos problemas contínuos, o fato de apresentar variáveis inteiras em sua formulação aumenta consideravelmente sua complexidade computacional.

O termo "programação" da expressão problema de programação linear não significa programação de computadores. Está, na verdade, relacionado ao planejamento de atividades que definem como recursos serão alocados em função de restrições que atendam necessidades específicas do problema. Recursos são frequentemente restritos no mundo real e precisam ser gerenciados de maneira ótima (SCHRINVER, 1999).

Uma nova formulação matemática pode ser reescrita a partir da Equação (2.1). Além das variáveis contínuas presentes no LP, variáveis discretas também estão representadas na formulação do MILP, como pode ser visto na Equação (2.2):

$$\begin{aligned} \text{Maximizar} \quad & z = \sum_j c_j x_j + \sum_k d_k y_k & \\ \text{sujeito a} \quad & \sum_j a_{ji} x_j + \sum_k g_{ik} y_k \leq b_i & (i = 1, 2, \ldots, m) \\ & x_j \geq 0 & (j = 1, 2, \ldots, n) \\ & y_k = 0, 1, 2, \ldots & (k = 1, 2, \ldots, p) \end{aligned} \tag{2.2}$$

além dos parâmetros presentes na Equação (2.1) temos que:

d_k são os coeficientes de custo das variáveis inteiras

y_k são as variáveis de decisão inteiras

g_{ik} são os coeficientes das restrições das variáveis inteiras

p é o número de variáveis inteiras

Esta formulação é geral de um problema MILP. Caso $p = 0$, não há variáveis discretas e a resultante é um problema LP. Já se $n = 0$, não existem variáveis contínuas e o problema é um IP, contendo apenas variáveis discretas. Uma outra possibilidade de formulação é considerar as variáveis y_k binárias, ou seja, restritas a valores entre 0-1. Esta forma é considerada como programação inteira binária (BIP) (CHEN; BATSON; DANG, 2010).

Um problema de otimização combinatória (COP, das iniciais em inglês *Combinatorial Optimization Problem*) é um problema de otimização discreta cujo objetivo é maximizar ou minimizar uma função de custo por meio da busca de uma solução em um conjunto finito de possibilidades de soluções. Em casos em que este conjunto é relativamente pequeno, é possível realizar uma busca exaustiva pela melhor solução dentro das várias opções possíveis. Contudo, o tamanho do espaço de busca frequentemente é muito grande impedindo esta abordagem de busca (PAPADIMITRIOU; STEIGLITZ, 1998). Todo COP pode ser formulado como um problema de programação binária.

2.2 Programação Dinâmica

A programação dinâmica é um paradigma de projeto de algoritmos inventado pelo matemático norte americano Richard Bellman na década de 50. O termo "programa-

ção", assim como no caso do problema de programação linear inteira mista, se refere a uma forma de *planejamento* e não necessariamente à programação de computadores. A programação dinâmica, como concebida por (BELLMAN, 1954), é baseada no princípio de otimalidade. Este define que uma política ótima pode ser construída para solução de um problema primeiro definindo uma política para solução do *subproblema de cauda*, estendendo-se essa política para o penúltimo subproblema e sucessivamente incorporando todos os subproblemas até que uma solução ótima seja estabelecida para o problema como um todo (BERTSEKAS, 1995).

Princípio da Otimalidade: *Uma política ótima tem a propriedade que qualquer que sejam o estado inicial e as decisões iniciais, as decisões restantes devem constituir uma política ótima com relação ao estado resultante das primeiras decisões* (BELLMAN, 1954).

Um analogia simples pode ser estabelecida para ilustrar o princípio de otimalidade. Como exemplo, podemos supor que alguém deseja viajar de carro saindo de Porto Alegre com destino à Belo Horizonte. Suponhamos que a rota mais rápida entre estas duas cidades passe pela cidade de São Paulo. O princípio da otimalidade nos diz que a porção da rota entre São Paulo e Belo Horizonte é também a rota mais rápida entre estas duas cidades (BERTSEKAS, 1995).

Após se firmar como uma técnica importante na área de matemática aplicada, especialmente no campo de controle ótimo, a programação dinâmica gradualmente se tornou conhecida como um paradigma de computação geral, não se restringindo apenas à problemas de otimização (LEVITIN, 2012). A aplicação desta técnica é caracterizada pela solução de um problema complexo por meio da solução de subproblemas menores. Cada um desses subproblemas é resolvido somente uma vez e o resultado armazenado em uma tabela para ser consultado posteriormente, ao invés de computá-lo novamente cada vez que seja requerido (CORMEN et al., 2001).

Para que a programação dinâmica seja aplicável a um problema, este deve apresentar um conjunto de requisitos necessários:

- *Formulação recursiva bem definida*: Em casos de estrutura recursiva, esta não deve possuir ciclos;
- *Subestrutura ótima*: A solução ótima para o problema deve ser construída por meio de subproblemas ótimos;
- *Subproblemas sobrepostos*: O espaço de subproblemas deve ser pequeno, ou seja, um algoritmo recursivo resolveria o mesmo subproblema muitas vezes antes de encontrar a solução final para o problema.

A técnica pode ser exemplificada por meio da série de números de Fibonacci, definida pela recorrência:

$$F(n) = \begin{cases} F(n-1) + F(n-2), \ \forall n > 1 \\ 0, \ \text{se } n = 0 \\ 1, \ \text{se } n = 1 \end{cases} \tag{2.3}$$

Existe um problema ao calcular diretamente o valor da série para o n-ésimo número de Fibonacci $F(n)$ utilizando-se a Equação 2.3, pois poderia ser necessário calcular os mesmos valores para esta função várias vezes. Isto significa que a série é definida por vários subproblemas sobrepostos, que é um requisito para aplicação da programação dinâmica.

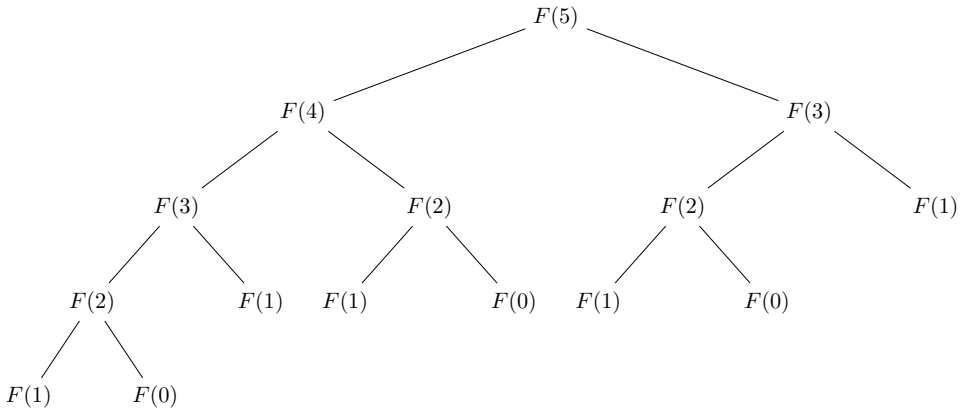


Figura 4 – Estrutura de subproblemas para um número de Fibonacci $F(5)$.

A Figura 4 mostra a expansão do número de Fibonacci para $F(5)$. Verificando-se a quantidade de vezes que os subproblemas ocorrem constata-se que neste exemplo há 1 ocorrência de $F(4)$, 3 ocorrências de $F(3)$ e $F(2)$, 5 vezes aparece $F(1)$ e 3 vezes $F(0)$. Especificamente as repetições de $F(3)$ e $F(2)$ são cálculos absolutamente desnecessários para se encontrar a solução do problema (as repetições de $F(1)$ e $F(0)$ podem ser desconsideradas nesse caso por se tratarem de constantes). Uma forma de evitar que os subproblemas sejam recalculados desnecessariamente é construir uma tabela com os todos os valores de $F(n)$, partindo-se de $n = 0$ até o valor de n desejado. O resultado procurado $F(n)$ é armazenado na n-ésima posição da tabela. A Figura 5 mostra a tabela montada para $F(5)$. A abordagem de se calcular todos os valores dos subproblemas de forma direta antecipadamente é chamada programação dinâmica *bottom-up*.

0	1	1	2	3	5
0	1	2	3	4	5

Figura 5 – Tabela com subproblemas para o número de Fibonacci $F(5)$.

Uma outra abordagem consiste em resolver apenas os subproblemas necessários recursivamente para se alcançar a solução, evitando-se assim o esforço computacional de

se calcular todos subproblemas possíveis. O procedimento consiste em se verificar caso o subproblema já tenha sido resolvido previamente: em caso afirmativo a computação que seria necessária é evitada; caso negativo o subproblema é resolvido e a solução memoizada, ou seja, guardada para consultas posteriores da solução deste mesmo subproblema. Esta variação é chamada programação dinâmica *top-down*.

2.3 GRASP

A meta-heurística *Greed Randomized Adaptive Search Procedure* (GRASP) consiste em um processo multi-*start* de duas fases: construção e busca local. A primeira fase é caracterizada pela construção de uma solução viável. Em seguida, busca-se o ótimo local em meio à vizinhança desta mesma solução. O Algoritmo 1 apresenta uma visão geral do procedimento GRASP. Ao longo de *MaxIterações* uma solução gulosa aleatorizada é gerada, e um ótimo local é encontrado para esta solução. Caso ele seja o melhor resultado alcançado até então, ele passa a ser a nova melhor solução (RESENDE; RIBEIRO, 2002a).

Algoritmo 1 Pseudocódigo do GRASP

```
1: função GRASP(MaxIterações,Semente)
2:     LeEntrada()
3:     para k ← 1 até MaxIterações faça
4:         Solução ← ContruçãoGulosaAleatorizada(Semente)
5:         Solução ← BuscaLocal(Solução)
6:         AtualizaSolução(Solução, MelhorSolução)
7:     fim para
8:     retorna MelhorSolução
9: fim função
```

O pseudocódigo da função *ContruçãoGulosaAleatorizada* está apresentado no Algoritmo 2 (FEO; RESENDE, 1995). A lista de elementos candidatos é formada por todos aqueles que podem entrar na solução sem a tornar inviável – linha 3. O elemento que será o próximo a fazer parte é escolhido por meio de um critério guloso entre os participantes da lista. A avaliação dos elementos pela função gulosa leva à criação de uma lista restrita de candidatos (RCL, das iniciais em inglês *Restricted Candidate List*). Somente os melhores elementos, ou seja, aqueles que acrescentam a maior parcela ao custo, são adicionados à lista. O elemento a ser adicionado é finalmente escolhido aleatoriamente na RCL – linha 6. Após esta seleção, a lista é atualizada para a nova vizinhança – linha 8. O algoritmo termina quanto a solução está completa – linha 4.

As soluções geradas pela construção gulosa aleatória não são necessariamente ótimas em sua vizinhança. Opcionalmente uma fase de busca local pode contribuir para melhorar a solução encontrada na etapa anterior. A efetividade desta busca depende de vários

Algoritmo 2 Pseudocódigo da fase de construção

```
1: função ContruçãoGulosaAleatorizada(Semente)
2:     Solução ← ∅
3:     Avalia os custos incrementais dos elementos candidatos
4:     enquanto Solução não é uma solução completa faça
5:         ConstruirRCL(RCL)
6:         s ← SelecionaElementoAleatoriamente(RCL)
7:         Solução ← Solução ∪ {s}
8:         ReavaliarFunçãoGulosa()
9:     fim enquanto
10:    retorna Solução
11: fim função
```

fatores, entre eles a estrutura da vizinhança, a técnica de busca e mesmo da solução inicial.

Uma característica essencial do GRASP é que ele é de fácil implementação e possui poucos parâmetros para ajuste. Isso permite que o esforço de desenvolvimento seja focado na criação de estruturas de dados eficientes para garantir que as iterações do GRASP sejam rápidas (FEO; RESENDE, 1995).

2.4 *Simulated Annealing*

O processo de recozimento ou *annealing* é um tratamento térmico aplicado a certos tipos de metal, vidro ou cristal, definido pelo aquecimento acima da temperatura crítica e, em seguida, resfriamento lento, resultando em materiais de estruturas cristalinas perfeitas. Este procedimento aumenta a ductibilidade e reduz a dureza dos objetos tratados. A simulação do processo de recozimento é chamado de *Simmulated Annealing* (SA) (KIRKPATRICK; GELATT; VECCHI, 1983). Neste caso, as propriedades físicas do processo de recozimento são relacionadas a características de um problema de otimização (DU; SWAMY, 2016):

- Os estados físicos do material são equivalentes às soluções do problema
- A energia de um estado representa o custo de uma solução
- A temperatura corresponde a um parâmetro de controle da simulação
- O estado de cristalização perfeita é equivalente ao custo ótimo global

O algoritmo de Metropolis é um procedimento simples que provê uma simulação eficiente de um conjunto de átomos em equilíbrio em um dada temperatura (METROPOLIS et al., 1953). SA é uma variação do algoritmo de Metropolis, sendo que a temperatura varia de alta para baixa ao longo da simulação (KIRKPATRICK; GELATT; VECCHI,

1983). O SA é composto basicamente de dois processos: geração e adoção de soluções. As soluções melhores do que a atual são sempre adotadas pelo algoritmo. A aceitação de soluções piores é feita com o propósito de escapar de situações de mínimo local em busca de uma solução ótima global. A temperatura de simulação determina a probabilidade com que soluções piores do que a atual são aceitas. A principal característica do algoritmo SA é a possibilidade de aceitar soluções piores à atual para não se ficar preso em ótimos locais. Essas aceitações se tornam mais raras à medida que a temperatura se resfria em direção a zero (HENDERSON; JACOBSON; JOHNSON, 2002).

A probabilidade de um sistema físico P_α estar em um estado α com energia E_α e sob temperatura T é dada pela distribuição de Boltzmann:

$$P_\alpha = \frac{1}{Z} \mathrm{e}^{-\frac{E_\alpha}{k_B} T} \tag{2.4}$$

em que k_B é a constante de Boltzmann, T é a temperatura absoluta e Z é a função de partição definida por:

$$Z = \sum_\beta \mathrm{e}^{-\frac{E_\beta}{k_B} T} \tag{2.5}$$

sendo β os estados que possuem energia E_β sob temperatura T. A distribuição de Boltzmann apresenta diferente probabilidade de ocorrência para os estados dependendo da temperatura. Todos os estados possuem chance uniforme sob temperaturas elevadas. À medida que T se aproxima do zero absoluto, apenas os estados com energia mínima possuem probabilidade diferente de zero.

O algoritmo SA simplifica as Equações (2.4) e (2.5). Primeiramente, o parâmetro k_β é eliminado. A seguir, a probabilidade de mudança entre estados é determinada pela diferença de energia entre eles:

$$P = \mathrm{e}^{\frac{\Delta E}{T}} \tag{2.6}$$

sendo T um parâmetro de controle chamado *temperatura computacional* e ΔE a diferença de energia entre dois estados. Em temperaturas elevadas, o sistema entra em equilíbrio rapidamente, repousando em mínimos locais. À medida que T diminui, pequenas mudanças de energia são percebidas e aumenta a eficácia da busca na vizinhança por soluções melhores. Sob temperatura zero, não há possibilidade de se escolher estados mais energéticos, estando o sistema próximo do equilíbrio.

O Algoritmo 3 apresenta o pseudocódigo do SA para um problema de minimização. O procedimento básico também é chamado de *Boltzmann annealing*. O planejamento de como a temperatura será reduzida é fundamental para a eficácia do SA. Caso T resfrie muito rapidamente, o estado pode ficar preso em um mínimo local. Se reduzir a temperatura muito devagar, o algoritmo leva muito tempo para convergir.

Algoritmo 3 Pseudocódigo *Simulated Annealing*

```
1: função SA
2:     x ← SoluçãoAleatória()
3:     T ← T_alta
4:     Iterações ← 0
5:     enquanto T > T_min faça
6:         enquanto Iterações < Iterações_max faça
7:             x ← x + Δx
8:             ΔE(x) ← ΔE(x + Δx) − E(x)
9:             se ΔE(x) < 0 então
10:                Move para o novo estado
11:            senão
12:                Recebe novo estado com probabilidade P = e^{−ΔE/T}
13:            fim se
14:        fim enquanto
15:        Atualize a temperatura
16:    fim enquanto
17:    retorna x
18: fim função
```

2.5 Detalhamento do problema

O *tripper* é um componente ativo no processo de beneficiamento, ele precisa ser operado apropriadamente para que cumpra sua função de distribuir o minério sobre o silo. É necessário que ele seja direcionado intencionalmente ao longo de seus trilhos, sobre todo o percurso de movimentação possível, de acordo com o estado atual do silo. Caso contrário, o sistema silo-*tripper* não será capaz de comportar toda a massa de minério proveniente das etapas anteriores do processo e a capacidade de produção da planta ficará comprometida. A movimentação do carro do *tripper* é conduzida por sistemas de automação, cuja função é acionar os motores elétricos vinculados às rodas de locomoção (BOYER, 2010). Com este aparato, é possível posicionar o equipamento determinando-se qual compartimento do silo receberá o material conduzido pela correia transportadora. Três ações de direcionamento são possíveis por meio da interface de automação: movê-lo para direita, para a esquerda ou pará-lo.

O problema de movimentação do *tripper* é caracterizado de forma abrangente como a *determinação de uma ou mais ações de direcionamento que conduzam o sistema silo-*tripper *para um estado desejado ao longo do tempo*. Duas questões fundamentais surgem a partir desta proposição: quem seria o agente causador das ações de movimentação e qual seria o estado desejável para o sistema. Dentre os agentes normalmente presentes em uma planta de beneficiamento real, se encontram o operador humano, que pode operar o equipamento localmente ou em uma sala de controle remota por meio de uma interface de automação, ou um sistema de intertravamento lógico implementado em um controlador industrial. O operador humano age de acordo com sua percepção do processo, por meio

de observação direta, por exemplo, avaliação visual da quantidade de minério dentro do silo ou da posição do *tripper* sobre os trilhos, ou indireta, por meio de uma interface computacional que disponibilize as informações correspondentes às variáveis de processo reais, tais como posição do *tripper*, vazão mássica da correia transportadora, etc. Contudo, estes dois agentes não são capazes de determinar o estado desejável para o sistema silo-*tripper*, ou seja, não podem definir qual seria o comportamento ideal para o sistema ao longo do tempo. O estado desejável é definido como *um conjunto de estados viáveis para as diversas variáveis do sistema que atenda a um ou mais objetivos ao longo do tempo.*

A partir desta proposição é possível verificar que o agente humano consegue, de forma limitada, determinar um objetivo para a operação do sistema, como por exemplo, não deixar faltar minério em nenhum compartimento do silo ao longo do tempo. Contudo, esse objetivo não é alcançável pois o operador não é capaz de lidar ao mesmo tempo com todas as variáveis de processo presentes e com o desdobramento de suas ações ao longo do tempo.

Diferentemente do agente humano, o intertravamento lógico é capaz de processar com facilidade todas as variáveis simultaneamente. Contudo, devido às restrições dos recursos computacionais de desenvolvimento disponíveis nestes sistemas de controle, este agente é capaz de considerar apenas poucas possibilidades de solução para o problema, não cumprindo o objetivo desejado na maioria dos cenários possíveis.

A deficiência na determinação e atingimento de um estado desejável satisfatório para o sistema silo-*tripper* é inerente aos dois agentes supramencionados. Uma terceira alternativa se faz necessária como opção viável para solucionar o problema de movimentação com mais eficiência do que as opções já discutidas. A literatura científica dedicada ao equipamento *tripper* é escassa e, quando encontrada, se limita a descrever as características e princípios de funcionamento fundamentais do sistema, tal como visto em (SWINDERMAN, 2014). Não foram encontrados modelos matemáticos que descrevam o comportamento do equipamento ou sugestões de sistemas que sejam capazes de resolver o problema adequadamente. Apenas uma referência de trabalho científico que menciona uma solução computacional para o problema foi encontrada. Um *framework* baseado em um sistema de controle preditivo baseado em modelo (MPC, do inglês *Model Predictive Controller*) híbrido foi utilizado por (KARELOVIC; PUTZ; CIPRIANO, 2015) para definir um conjunto de ações de movimentação de *tripper*. Contudo, há apenas a menção sobre o critério escolhido pelos autores do artigo para o estado desejável para o sistema, que foi definido como a estabilização do processo e minimização da movimentação do carro. No escopo do sistema descrito pelo artigo, o *tripper* é apenas um objeto incorporado a um modelo amplo de toda uma planta de britagem e não há detalhamento da implementação computacional realizada.

Controladores, tais como o MPC, são uma classe de sistemas dinâmicos capazes de determinar uma ação de controle a partir da previsão dos estados futuros do processo a

ser controlado. Essas previsões são realizadas por modelos dinâmicos que caracterizem o comportamento do sistema ao longo do tempo e são utilizadas por otimizadores para determinar a trajetória ótima para o sistema. O MPC define uma janela de previsão das variáveis de processo sobre a qual um conjunto de ações de controle ao longo do tempo são determinadas. A cada período de tempo apenas a ação imediata, dentre toda a janela de previsão da saída de controle, é enviada para a planta. No próximo ciclo de iteração as janelas de predição e controle são recalculadas e uma nova ação é direcionada para o processo (GARCíA; PRETT; MORARI, 1989). Como evolução desta proposta original do MPC, que contempla apenas variáveis contínuas, o algoritmo foi expandido para lidar adicionalmente com variáveis discretas, lógicas e regras. Essa forma expandida é conhecida como MPC híbrido (BEMPORAD; GIORGETTI, 2006). Esta técnica de controle é aplicável ao problema de movimentação do *tripper*, uma vez que este processo possui tanto variáveis contínuas (nível, vazão mássica) quanto discretas (posição).

3 Desenvolvimento

O detalhamento do problema é o ponto de partida para o desenvolvimento dos modelos matemáticos que descrevem o comportamento do sistema silo-*tripper*. São analisadas, em detalhe, as características do processo de movimentação, descrevendo-se as propriedades fundamentais necessárias para a compreensão do sistema. Em seguida, o processo é definido matematicamente e as diversas características do modelo, tais como os dados de entrada das instâncias, variáveis de decisão, restrições e função objetivo são detalhados. A formulação desenvolvida será resolvida posteriormente por meio de métodos de programação linear inteira mista e programação dinâmica. Adicionalmente, duas meta-heurísticas são adaptadas para a solução do problema de movimentação do *tripper*: GRASP e *Simulated Annealing*.

3.1 Modelo Matemático

A Figura 6 ilustra um grafo que representa as possibilidades de movimentação de um *tripper* com n compartimentos. Dada uma posição qualquer p_i, o equipamento pode locomover-se apenas para uma das posições adjacentes ou permanecer na posição em que estiver localizado. Por exemplo, caso esteja na posição p_2, as opções de movimentação serão $\{p_1, p_2, p_3\}$. Movimentar o carro para uma posição não adjacente à atual implica em atravessar todas as posições intermediárias entre os pontos de partida e destino. Consequentemente, o *tripper* encaminha o fluxo de minério para os respectivos compartimentos sobre os quais estiver transitando ao longo do deslocamento. O processo de seleção da posição de alimentação possui dependência temporal, já que a velocidade de movimentação do equipamento é finita.

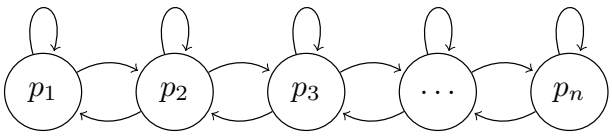


Figura 6 – Grafo representando as possibilidades de movimentação do *tripper*.

Eventualmente pode faltar material em algum dos compartimentos, independentemente das características físicas do equipamento. Isto acontece porque o *tripper* pode estar posicionado somente sobre uma determinada boca do silo em um determinado momento. Esta situação ocorre mesmo em momentos em que o balanço de massa do sistema esteja equilibrado, ou seja, a vazão mássica total alimentada é igual ao volume total retirado. Para contornar esse problema, é necessário que o *tripper* seja movimentado ao longo da abertura do silo, de modo que a adição de material, feita pela correia transpor-

tadora, não permita que falte minério. Além disso, os níveis dos compartimentos devem ser limitados de acordo com as restrições de capacidades máxima e mínima do silo.

O tempo necessário para que o *tripper* percorra a região correspondente à abertura de uma divisão do silo, movendo-se ininterruptamente, é proporcional ao comprimento da região transitada e inversamente proporcional à velocidade de deslocamento do carro. Para fins de simplificação do modelo, este tempo é considerado fixo nos casos em que as subdivisões do silo possuam o mesmo tamanho e o *tripper* tenha velocidade constante. A janela de tempo futuro, na qual a predição será realizada, é dividida em múltiplos deste tempo mínimo. O tamanho dessa janela caracteriza a primeira dimensão do modelo do problema de movimentação.

A segunda dimensão é definida pelo conjunto P de posições viáveis para posicionamento do *tripper*, cujo número de elementos corresponde à quantidade de compartimentos do silo. Neste contexto, a tomada de decisão de movimentação é definida como a escolha de uma sequência de ações de locomoção do carro ao longo de uma janela finita de tempo futuro. O objetivo destas ações é manter as variáveis controladas dentro das restrições operacionais estabelecidas para o problema. A janela de tempo futuro é mapeada como um conjunto finito T, composto pelo estado atual do sistema adicionado pelos $t-1$ períodos de tempo futuros. A inclusão dessas iterações adicionais é necessária, tornando o modelo capaz de predizer antecipadamente o resultado das ações que serão tomadas ao longo do tempo, assegurando-se de que as restrições estabelecidas não serão violadas em qualquer momento.

A Figura 7 apresenta um grafo representando o modelo desenvolvido definindo matematicamente a movimentação do *tripper*. As posições iniciais, relativas ao tempo e_1, estão contidas no conjunto $x_{1i} = \{1, 2, \ldots, n\}$, em que n é o número de compartimentos. A partir de uma destas posições iniciais, as posições subsequentes são escolhidas a cada iteração, respeitando-se a restrição de movimentação que limita o conjunto de posições possíveis a serem seguidas às posições adjacentes e à posição em que o *tripper* se encontra. Por fim, as posições possíveis para a última etapa de movimentação e estão contidas no conjunto $x_{ei} = \{1, 2, \ldots, n\}$.

Para fins de simplificação, a velocidade de translação do carro do *tripper* é constante em ambas as direções. Desta forma, pode-se considerar que o tempo mínimo de permanência sobre um compartimento do silo qualquer é a razão entre o comprimento desta abertura e a velocidade de translação. Esse tempo é o mesmo para todos os compartimentos que possuam a mesma dimensão. O intervalo de tempo será considerado no modelo como um período fixo Δt entre cada uma das iterações do *tripper* ao longo do tempo (PHILLIPS; PARR; RISKIN, 2007).

Desta forma, os parâmetros que definem o tamanho de uma instância do problema se restringem a:

- Número de compartimentos do silo n

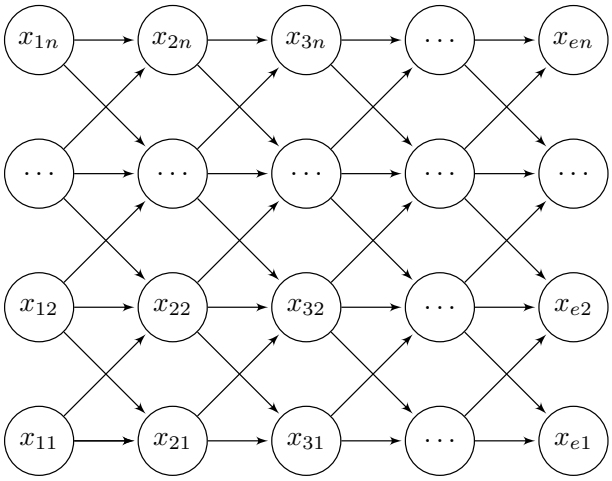


Figura 7 – Grafo representando as n possibilidades de movimentação do *tripper* ao longo das e iterações.

- Quantidade de ações futuras preditas e

3.1.1 Dados de Entrada

Os parâmetros de entrada para uma instância do problema de movimentação do *tripper* estão apresentados na Tabela 1. Ao longo deste trabalho, as taxas mássicas de entrada q e saída Q foram definidas, com propósito de simplificação, como constantes ao longo de todas as etapas do modelo. Contudo, estes parâmetros podem receber valores diferentes respectivos a cada uma das iterações caso seja necessário.

Tabela 1 – Parâmetros das instâncias.

Parâmetro	**Descrição**
T	Conjunto de períodos de tempo
P	Conjunto de compartimentos do silo
n	Número de silos
e	Número de períodos de tempo
I	Conjunto de níveis iniciais
$l_{\max}$	Nível máximo
$l_{\min}$	Nível mínimo
q	Massa de entrada
Q	Massa de saída
k	Taxa de esvaziamento
p	Posição inicial

O tamanho da instância é definido pelos conjuntos de movimentos T e de compartimentos P do silo. O nível inicial de cada um dos compartimentos é determinado pelo conjunto I. Os valores dos níveis dos silos são limitados entre $l_{\min}$ e $l_{\max}$. A posição de partida sobre a qual o carro do *tripper* inicia a movimentação é definida por p. A taxa

de esvaziamento do silo ao longo do tempo é definida pela constante k. Este parâmetro determina a quantidade de minério que o silo é capaz de armazenar por cada unidade de nível disponível.

3.1.2 Variáveis de Decisão

As variáveis de decisão para o problema são apresentadas na Tabela 2. A variável mais importante é a matriz de posicionamento do *tripper* $\mathbf{X}$. Trata-se de um conjunto de valores binários que determina o posicionamento do carro sobre as n posições possíveis ao longo dos e períodos de tempo. Esta variável binária é fundamental na definição do problema de movimentação do *tripper* como um problema de programação linear inteira mista (WOLSEY, 1998).

Tabela 2 – Variáveis de decisão.

Variável	**Descrição**
$\mathbf{X}$	Posições do *tripper*
$\mathbf{A}$	Folga de nível baixo
$\mathbf{B}$	Folga de nível alto
$\mathbf{L}$	Nível
Z	Auxiliar para *MinMax*

A Figura 8 mostra um exemplo de posicionamento de um *tripper* de $n = 4$ posições e $e = 10$ períodos. Os círculos na cor cinza escuro indicam uma posição dentro da trajetória definida para o equipamento, que representam os valores 1 na matriz $\mathbf{X}$, e os círculos vazios representam os valores nulos na matriz $\mathbf{X}$. As setas são ilustrativas e mostram a direção do movimento ao longo das possibilidades de movimentação, partindo-se das posições iniciais. A posição de partida do *tripper* é indicada pela letra branca "S".

Os estados dos níveis dos p compartimentos do silo ao longo dos e períodos de tempo são representados por $\mathbf{L}$. As matrizes $\mathbf{A}$ e $\mathbf{B}$ representam variáveis de folga para os níveis do silo. Elas garantem que o modelo permaneça viável mesmo em situações em que o silo estiver completamente cheio ou vazio. Nesses casos, acrescentar ou retirar minério acarretaria em violar as restrições máximas e mínimas absolutas dos níveis, tornando a solução inviável. Estas duas variáveis permitem que essa situação de violação das restrições ocorra sem que a viabilidade do modelo seja quebrada. O conjunto de variáveis Z são auxiliares para a função de custo *MinMax*, que será detalhada na próxima seção.

3.1.3 Função Objetivo

A função objetivo é baseada no princípio *MinMax* (RICH; KNIGHT, 1991). Busca-se maximizar o menor valor dentre os níveis dos compartimentos do silo a cada iteração,

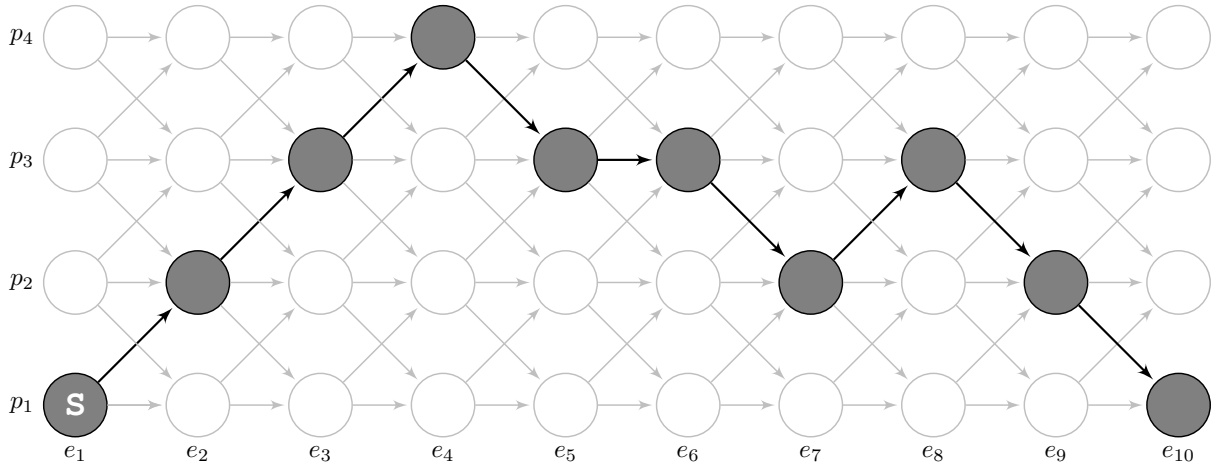


Figura 8 – Exemplo de posicionamento na matriz **X**. O eixo vertical é o tempo; o eixo horizontal é a posição.

não permitindo-se que falte minério em nenhum compartimento a qualquer momento. A variável Z representa o menor nível alcançado para cada período. A solução é encontrada ao maximizar-se o somatório do nível mínimo em cada uma das etapas. Além da própria função objetivo, o *MinMax* depende de restrições auxiliares que viabilizem a aplicação do princípio e que serão detalhadas na próxima seção. A função de custo para o problema pode ser vista na Equação (3.1). Além do custo do *MinMax*, as variáveis de folga **A** e **B** também aparecem na formulação.

$$\text{Maximizar} \sum_{j \in T} Z_j - \sum_{i \in P, j \in T} \mathbf{A}_{ij} - \sum_{i \in P, j \in T} \mathbf{B}_{ij} \tag{3.1}$$

3.1.4 Restrições

A equação dinâmica que define o nível de um silo ao longo do tempo é descrita pela Equação (3.2) (PAL; SANA; CHAUDHURI, 2014):

$$l(t) = k \int (q_e(t) - q_s(t))dt \tag{3.2}$$

sendo l_i o nível de minério em um silo, k a constante de esvaziamento, q_e a vazão mássica de entrada e q_s a vazão de saída.

A representação matemática adotada para o problema de movimentação do *tripper* considera um intervalo de tempo fixo Δt. Logo, faz-se necessário a discretização da Equação (3.2) de forma que o tempo possa ser indexado conforme a representação da matriz **X** (PHILLIPS; PARR; RISKIN, 2007). A nova formulação determinando a acumulação de nível para um compartimento do silo i no período j está mostrada na Equação (3.3).

$$\mathbf{L}_{in} = k \sum_{j=0}^{n} (q\mathbf{X}_{ij} - Q_i) \tag{3.3}$$

A Equação (3.15) define a acumulação de massa no silo a cada período de tempo baseando-se na Equação (3.3), acrescentando-se os componentes de folga de nível. A cada etapa j, o compartimento do silo definido por i em $\mathbf{X}_{ij}$ recebe acréscimo de massa adicional do *tripper*. O nível inicial é definido por (3.10). A variável de folga $\mathbf{A}$, para o nível abaixo de $l_{\min}$, é representada nas Equações (3.13) e (3.14). A variável de folga $\mathbf{B}$, para o nível acima de $l_{\max}$, nas Equações (3.11) e 3.12. As variáveis $\Delta\mathbf{A}$ e $\Delta\mathbf{B}$ representam a mudança das variáveis de folga entre dois períodos de tempo consecutivos.

A posição inicial do *tripper* é definida pela Equação (3.9). A Equação (3.5) determina que o carro somente pode estar sobre um compartimento do silo a cada iteração. A movimentação ao longo das etapas é definido pela Equação (3.6), que limita a posição do carro, em uma determinada iteração, à mesma posição ou às posições imediatamente adjacentes. As Equações (3.7) e (3.8) determinam a movimentação nos casos em que o equipamento esteja na primeira ou última posições.

$$Z_j \leq \mathbf{L}_{ij},\ \forall j \in T,\ i \in P \tag{3.4}$$

$$\sum_{i \in P} \mathbf{X}_{ij} = 1,\ \forall j \in T \tag{3.5}$$

$$\mathbf{X}_{ij} \leq \mathbf{X}_{i-1j-1} + \mathbf{X}_{ii-1} + \mathbf{X}_{i+1j-1},\ \forall i \in P, j \in T : n-1 \geq i \geq 2, j \geq 2 \tag{3.6}$$

$$\mathbf{X}_{1j} \leq \mathbf{X}_{1j-1} + \mathbf{X}_{2j-1},\ \forall j \in T : j \geq 2 \tag{3.7}$$

$$\mathbf{X}_{nj} \leq \mathbf{X}_{nj-1} + \mathbf{X}_{n-1j-1},\ \forall j \in T : j \geq 2 \tag{3.8}$$

$$\mathbf{X}_{p1} = 1 \tag{3.9}$$

$$\mathbf{L}_{i1} = I_i,\ \forall i \in P \tag{3.10}$$

$$\mathbf{A}_{ij} = \Delta\mathbf{A}_{ij} + \mathbf{A}_{ij-1},\ \forall i \in P, j \in T \tag{3.11}$$

$$\Delta\mathbf{A}_{i1} = 0, \forall i \in P \tag{3.12}$$

$$\mathbf{B}_{ij} = \Delta\mathbf{B}_{ij} + \mathbf{B}_{ij-1},\ \forall i \in P, j \in T \tag{3.13}$$

$$\Delta\mathbf{B}_{i1} = 0, \forall i \in P \tag{3.14}$$

$$\mathbf{L}_{ij+1} = \mathbf{L}_{ij} + K_i(q \cdot \mathbf{X}_{ij} - Q_i) + \Delta\mathbf{A}_{ij+1} - \Delta\mathbf{B}_{ij+1},\ \forall i \in P, j \in T : j \leq t-1 \tag{3.15}$$

$$\mathbf{L}_{ij} \leq l_{\max},\ \forall i \in P, j \in T \tag{3.16}$$

$$\mathbf{L}_{ij} \geq l_{\min},\ \forall i \in P, j \in T \tag{3.17}$$

3.2 Métodos Exatos

Os métodos exatos são os primeiros considerados como proposta de solução para o problema de movimentação do *tripper*. Por definição, estes métodos garantem que a solução encontrada para uma determinada instância seja a melhor possível dentro de todo o espaço de soluções do problema. Duas metodologias foram adotadas nesta seção: programação linear inteira mista e programação dinâmica.

3.2.1 Programação Linear Inteira Mista

As ações de movimentação do *tripper* são discretas e se restringem a duas possibilidades: manter-se na posição atual ou mover-se para uma das posições adjacentes. De acordo com a posição em que o equipamento se encontre, podem existir uma ou duas possibilidades de posições adjacentes. O posicionamento é definido por variáveis discretas representadas pela matriz $\mathbf{X}$. Variáveis contínuas também foram consideradas na formulação do modelo, tais como o nível de minério nos compartimentos do silo e a vazão mássica nas correias transportadoras. Algoritmos de programação linear inteira mista são apropriados no contexto deste modelo por lidarem naturalmente tanto com variáveis contínuas quanto discretas em sua formulação.

A função objetivo descrita pela Equação (3.1) e as restrições listadas pelas Equações (3.4) à (3.17), que definem o problema de movimentação do *tripper*, foram escritas em GNU Mat Prog. Esta é uma linguagem de código livre, baseada em AMPL, apropriada para elaboração de modelos de programação matemática linear.

Dois resolvedores foram utilizados para solucionar o modelo proposto para o problema de posicionamento de *tripper*. O primeiro foi o GLPK (GNU Linear Programming Kit), que processa nativamente a linguagem Mat Prog. O segundo escolhido foi o CPLEX Optimizer da IBM. A entrada para este resolvedor foi uma versão LP convertida do modelo Mat Prog original. O objetivo neste caso é comparar o resultado de um resolvedor de código aberto em relação a um proprietário na tarefa de solucionar o problema de movimentação de *tripper*.

3.2.2 Programação Dinâmica

Para que o paradigma de programação dinâmica seja aplicado ao problema de movimentação, é necessário primeiramente definir qual será a estrutura dos subproblemas derivados do problema principal. Nesta definição, os subproblemas são considerados como um conjunto de versões reduzidas do problema geral em que o número de períodos de tempo de cada um deles está contido no conjunto $\{n' \in \mathbb{N} \mid 1 \leq n' \leq n\}$. Logo, os subproblemas são versões menores do problema base, em que o número de períodos é restritos até o valor de n'. Como exemplo, a partir do subproblema mínimo com $n' = 1$, é possível gerar todos os subproblemas possíveis incrementando-se unitariamente o valor de n' até obter-se o problema principal em $n' = n$. O algoritmo desenvolvido propõe resolver cada um destes subproblemas e armazenar o resultado parcial de cada um deles em uma tabela, para serem consultados posteriormente caso o mesmo subproblema seja novamente encontrado. Duas tabelas foram criadas para armazenar os resultados parciais: a primeira com os valores de custo e a outra com o histórico das posições. Estas tabelas são identificadas como *Posições* e *MelhoresCustos* no Algoritmo 4. A variável *MelhoresCustos* e *MelhoresPosições* são globais e inicializadas vazias na partida da execução (CALDAS;

Algoritmo 4 Programação Dinâmica

```
1: função PD(custo, L, Q, q, Posições, p, etapa, e)
2:     L[p] ← L[p] + q
3:     L ← L − Q
4:     custo ← custo + min(L)
5:     se MelhoresCustos[etapa] = ∅ ou custo ≥ MelhoresCustos[etapa] então
6:         MelhoresCustos[etapa] ← custo
7:         Posições[etapa] ← p
8:         se etapa ≠ e então
9:             para i ← max(1, p − 1) até min(n, p + 1) faça
10:                PD(custo, L, Q, q, i, Posições, etapa + 1)
11:            fim para
12:        senão
13:            se MelhoresCustos[e] < custo então
14:                MelhoresPosições ← Posições
15:            fim se
16:        fim se
17:    fim se
18: fim função
```

MARTINS, 2018).

Inicialmente, os níveis dos compartimentos do silo L são atualizados de acordo com as equações do processo – linhas 2 e 3. Em seguida, o valor de *custo* é atualizado com o menor valor de nível – linha 4. Este valor é definido como a soma do custo do subproblema anterior adicionado do nível mínimo do período atual. Caso não haja algum valor de custo para a etapa atual registrado na *MelhoresCustos* ou o custo atual seja maior ou igual do que o valor armazenado atualiza-se *MelhoresCustos* e continua-se com a recursão –linha 6. Caso contrário, se o custo atual for menor do que o registrado, a função retorna imediatamente. A melhor solução possível é armazenada na variável global *MelhoresPosições* na última etapa de movimentação do *tripper* – linhas 13 e 14.

A Figura 9 apresenta um exemplo do funcionamento do algoritmo proposto para um silo de três compartimentos e uma janela de tempo de quatro unidades. As duas tabelas da esquerda, em qualquer uma das subfiguras, representam respectivamente as variáveis *MelhoresCustos* e *Posições*, e a seta sobre elas indica o período atual e a direção da recursividade. A seta com sentido à esquerda indica que a execução do algoritmo está voltando à etapa anterior, seta com sentido à direita significa que a execução está avançando à próxima etapa e a seta com sentido para baixo indica que a posição é a mesma da última iteração. Células que recebem atualização de valor são marcadas com a cor cinza. A tabela da direita representa os níveis atuais do silo. A seta superior à esta tabela indica a posição e a direção do movimento do *tripper*. Os níveis mínimos em uma etapa são indicados pelo preenchimento de suas células respectivas com a cor cinza.

A execução se inicia na Figura 9a. As três bocas são inicializadas com 50% de nível, a

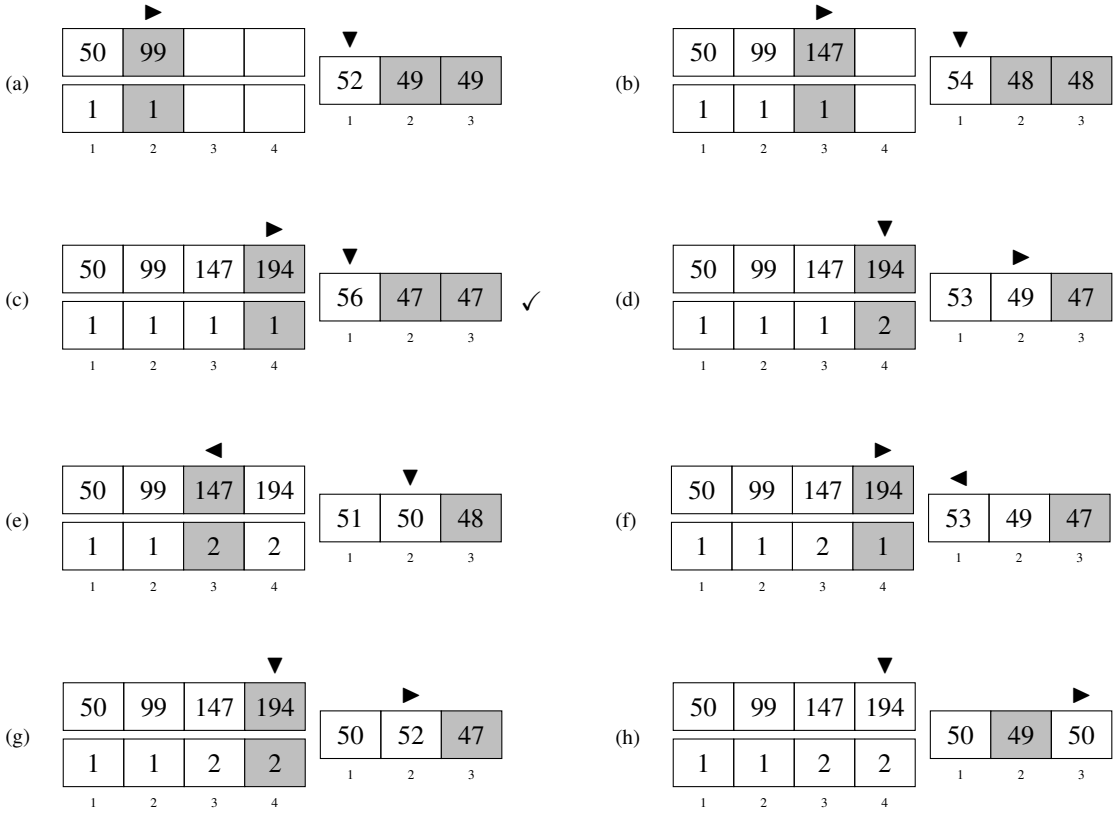


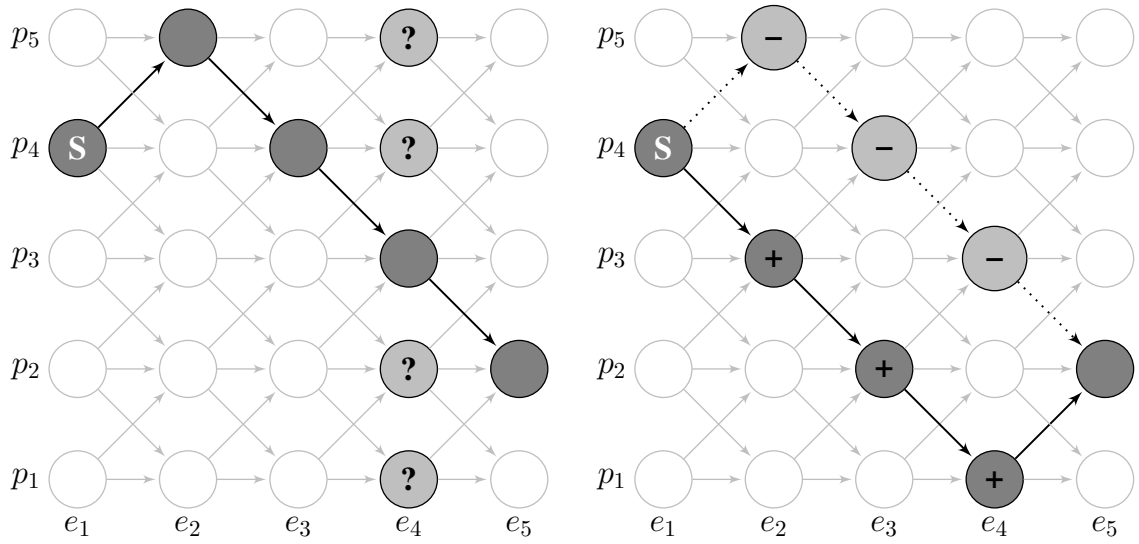
Figura 9 – Exemplo de funcionamento do algoritmo de programação dinâmica para um silo de três compartimentos.

alimentação do *tripper* é ajustada em 3 e a taxa dos três alimentadores em 1. A primeira ação do carro é permanecer na boca 1. Os níveis são decrementados de uma unidade e a boca 1 acrescida de 3 unidades. Os compartimentos com menor valor de nível são 2 e 3. O novo custo para a etapa é o custo inicial 50 adicionado custo atual 49, que é o menor valor de nível. A nova parcial de custo para a etapa é de 99. Este valor é registrado na tabela, juntamente com a posição do carro.

Este procedimento é repetido até que não haja novas adições na tabela ou até que a execução do algoritmo atinja o limite máximo de períodos $n' = n$. Neste caso, a solução se torna a nova melhor caso seu custo seja superior ao custo da melhor solução já encontrada até o momento. Uma marca ✓ é adicionada em uma nova solução melhor, como mostrado na Figura 9c.

3.3 Meta-heurísticas

Meta-heurísticas foram adotadas com o objetivo de reduzir o tempo gasto na geração de soluções boas para o problema em tempo relativamente menor do que os métodos exatos. Duas técnicas foram implementadas: o GRASP e o *Simulated Annealing*. Primeiramente, é descrito o algoritmo de correção de continuidade que se faz necessário para a busca



(a) Possibilidades de mudança de posição no período e_4.

(b) As posições das etapas e_3 e e_2 são corrigidas após a posição p_1 ser escolhida na etapa e_4.

Figura 10 – Correção de continuidade de uma nova solução.

local e para o *Simulated Annealing*. Em seguida as meta-heurísticas desenvolvidas neste trabalho são apresentadas.

3.3.1 Correção de continuidade

O algoritmo de busca local e o *Simulated Annealing* necessitam de gerar novas soluções a partir de uma solução inicial. Estes dois procedimentos alteram o posicionamento do *tripper* em um determinado período de forma que uma nova solução baseada na inicial seja encontrada. Contudo, de acordo com a estrutura da solução e do grau de modificação da nova posição, a nova solução pode não respeitar o critério de viabilidade para todo o conjunto de movimentos do *tripper* ao longo das e iterações. Ou seja, pode ocorrer uma descontinuidade no caminho entre os nós inicial e o final tornando a solução inviável.

Para solucionar este problema, foi desenvolvido um algoritmo para correção das soluções modificadas, garantindo-se a continuidade da trajetória. O Algoritmo 5 apresenta o pseudocódigo da solução proposta e que pode ser dividido em três partes: entre as linhas 2 e 9 é feita a correção das posições anteriores ao período de alteração, verificando-se e corrigindo-se a consistência em relação à posição inicial; entre as linhas 10 à 20 é feita a correção das posições anteriores ao período de alteração, corrigindo-se as descontinuidades; e entre as linhas 21 e 32, é feita a correção das posições posteriores ao período de alteração, corrigindo-se as descontinuidades.

A Figura 10 ilustra o funcionamento do algoritmo de correção de continuidade. Uma operação de alteração de posição no período e_4 é mostrada na Figura 10a. As posições relativas à solução inicial estão indicadas por círculos na cor cinza escuro e as novas posições

Algoritmo 5 Correção das posições de uma solução de forma a restituir sua continuidade

```
1: função CORREÇÃO(Solução, mudança, etapa)
2:     para j ← mudança até 0 faça
3:         se Solução[j] − Solução[0] > j então
4:             se Solução[j] > Solução[0] então
5:                 Solução[j] ← Solução[0] + j
6:             senão
7:                 Solução[j] ← Solução[0] − j
8:             fim se
9:         fim se
10:        diferença ← Solução[j − 1] − Solução[j]
11:        se ValorAbsoluto(diferença) > j então
12:            se diferença > 0 então
13:                Solução[j − 1] ← Solução[j] + 1
14:            senão
15:                Solução[j − 1] ← Solução[j] − 1
16:            fim se
17:        senão
18:            quebra laço para
19:        fim se
20:    fim para
21:    para j ← mudança+1 até etapa faça
22:        diferença ← Solução[j] − Solução[j − 1]
23:        se ValorAbsoluto(diferença) > 1 então
24:            se diferença > 0 então
25:                Solução[j] ← Solução[j − 1] + 1
26:            senão
27:                Solução[j] ← Solução[j − 1] − 1
28:            fim se
29:        senão
30:            quebra laço para
31:        fim se
32:    fim para
33:    retorna j
34: fim função
```

possíveis para o período na cor cinza claro, marcadas por um ponto de interrogação (?). A escolha de qualquer nova posição em e_4 torna a solução inicial descontínua, já que não há conexão direta entre o estado anterior x_{43} e posterior x_{25}. A Figura 10b mostra a escolha de uma nova posição para e_4 e a correção da continuidade do caminho. Neste exemplo, os novos nós estão marcados pelo sinal de adição (+) e os nós retirados da solução estão marcados por um sinal de subtração (–).

3.3.2 GRASP

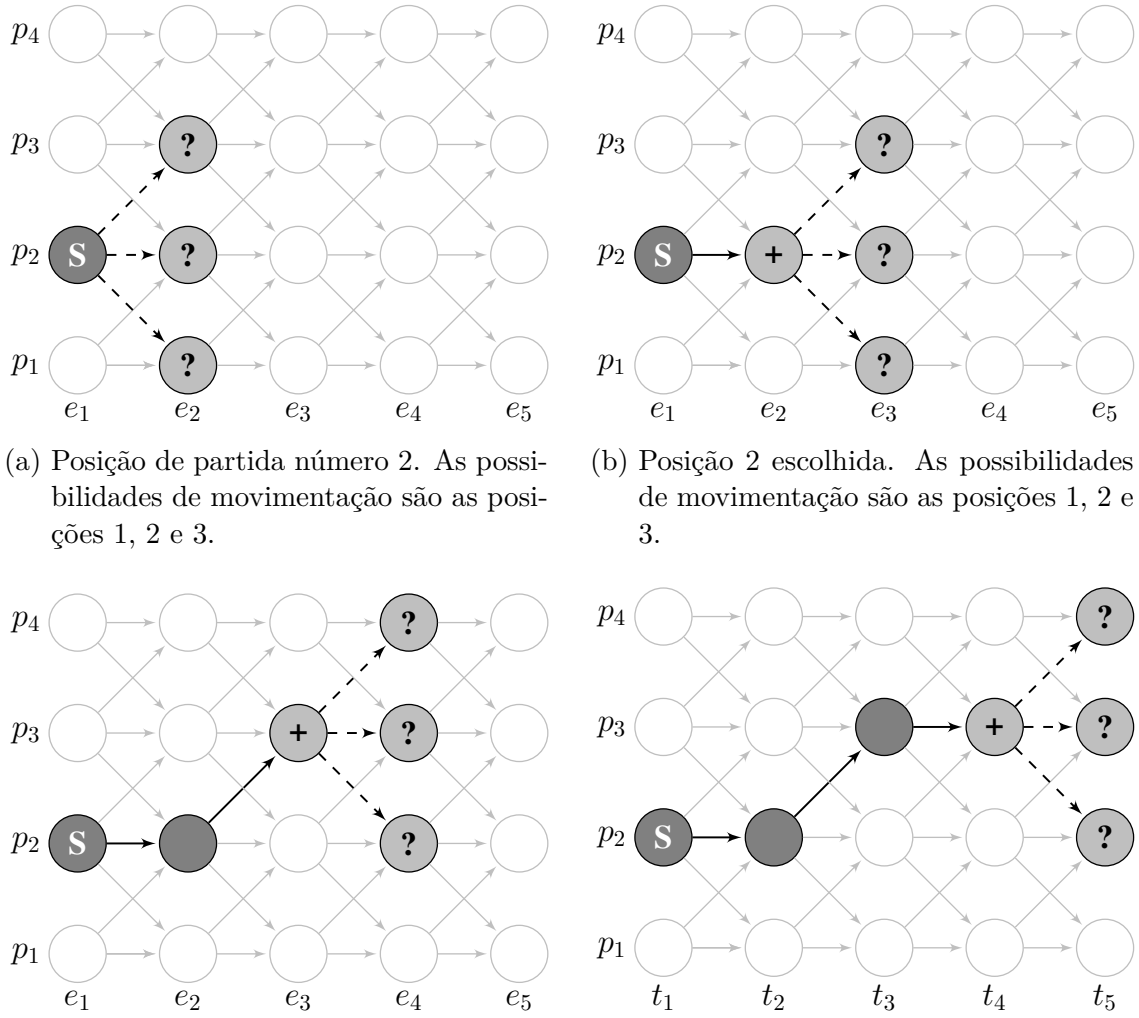
O GRASP é a primeira meta-heurística apresentada neste trabalho. A proposta deste algoritmo é aproveitar a busca por uma solução gulosa para o problema, adicionando-lhe uma componente de aleatoriedade. Esta combinação é eficiente e adequada para problemas puramente binários (RESENDE; RIBEIRO, 2002a), o que justifica sua utilização no problema de movimentação de *tripper*. O primeiro passo descrito para sua implementação é a definição da estrutura gulosa do problema. Em seguida, a metodologia de busca local utilizada é descrita em detalhe. Por fim, o algoritmo GRASP é apresentado.

3.3.2.1 Algoritmo guloso

O procedimento guloso é um paradigma de projeto de algoritmos que propõe a determinação de uma solução por meio da seleção sequencial de ótimos locais imediatos com o objetivo de se alcançar o ótimo global para um problema. O algoritmo guloso não garante o sucesso da operação de busca, muitas vezes encontrando soluções muito aquém do ótimo global (CORMEN et al., 2001).

A estrutura do problema de movimentação de *tripper*, descrita como um grafo acíclico direcionado, leva à definição da formulação do algoritmo guloso para este problema, conforme mostrado na Figura 11. A ideia é partir da posição inicial do equipamento, adicionando-se sequencialmente os nós na direção de movimentação ao longo das iterações. A Figura 11a mostra a seleção da segunda posição do *tripper*. Partindo-se do estado x_{21}, é possível movimentar o equipamento para três posições: x_{23}, x_{22} e x_{21}. A posição que oferecer o menor custo adicional dentre as três escolhas é acrescentada à solução parcial. A Figura 11b mostra a seleção do estado x_{22}, supostamente o menor custo das três opções apresentadas. A próxima escolha se dá entre os estados x_{31}, x_{32} e x_{31}. A terceira posição é escolhida, conforme mostrado pela Figura 11c. E, em sequência, a posição no tempo e_4 permanece a mesma do período anterior, como mostrado pela Figura 11d.

A função de custo definida para os algoritmos de busca de soluções exatas é caracterizada pelo princípio *MinMax*. Esta formulação determina que o valor do custo em uma etapa específica é o menor valor de nível de qualquer um dos compartimentos do silo. Entretanto, esta função de custo não se mostrou adequada para o algoritmo guloso. Como o custo é calculado a cada etapa, o algoritmo não é capaz de determinar a próxima posição a ser seguida em algumas situações. Por exemplo, supondo-se uma instância em que todos os valores de níveis iniciais sejam iguais em um silo com $n = 4$ compartimentos e cuja posição inicial seja $p = 1$. A primeira iteração do algoritmo não será satisfatória pois o nível mínimo dentre todos os compartimentos não será alterado independentemente da próxima posição escolhida para receber o incremento de massa. Isso ocorre pois os três compartimentos que não receberão adição de massa permaneceram com o mesmo valor de nível, restringindo-se a função de custo a um valor fixo para qualquer uma das opções



(a) Posição de partida número 2. As possibilidades de movimentação são as posições 1, 2 e 3.

(b) Posição 2 escolhida. As possibilidades de movimentação são as posições 1, 2 e 3.

(c) Posição 3 escolhida. As possibilidades de movimentação são as posições 2, 3 e 4.

(d) Posição 3 escolhida. As possibilidades de movimentação são as posições 2, 3 e 4.

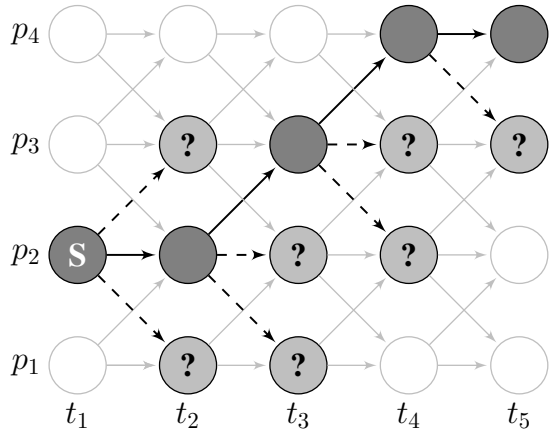
Figura 11 – Funcionamento do algoritmo guloso

de movimentação possíveis.

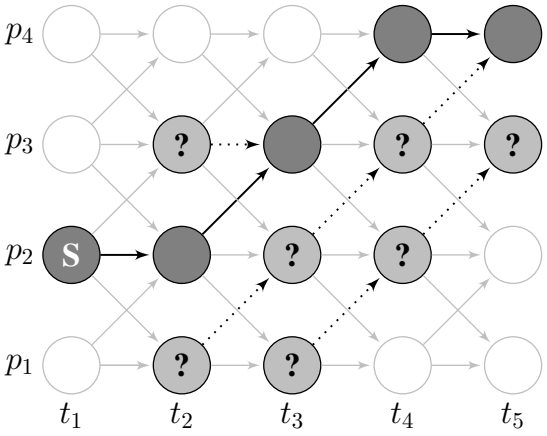
Uma alternativa imune a este inconveniente da função *MinMax* foi a utilização do cálculo de desvio padrão dos níveis do silo como função de custo ($\sigma = \sum_{i=0}^{n}(\bar{l} - l_i)^2/n$). O novo objetivo passa a ser minimizar o desvio padrão de todos os níveis dos compartimentos em cada etapa. Como esta função é contínua, qualquer alteração nos valores dos níveis altera o resultado da operação independentemente do valor dos outros níveis do silo. Este nova função permite que o algoritmo guloso se comporte conforme o esperado.

3.3.2.2 Busca Local

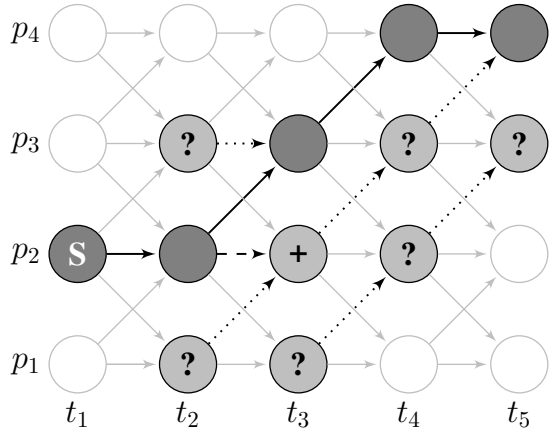
Uma das etapas da meta-heurística GRASP é a busca local. A finalidade deste procedimento é encontrar um máximo ou mínimo local a partir de uma solução inicial qualquer. Para cumprir este objetivo, a vizinhança ao redor da solução inicial é avaliada buscando-se algum vizinho próximo que consiga melhorar o custo em relação à solução inicial.



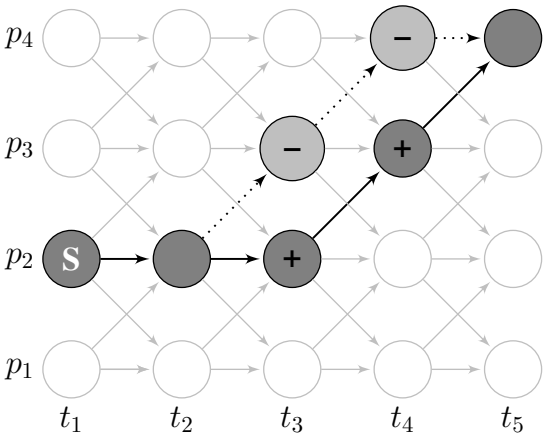
(a) Possibilidades de vizinhança para uma solução inicial.



(b) Arestas para reconstrução da nova solução.



(c) Escolha do nó x_{23} para formação de nova solução.



(d) Nova solução recontruída.

Figura 12 – Exemplo de funcionamento do algoritmo de busca local.

Neste trabalho, o método de primeira melhora (*First Improvement*) foi escolhido para a operação de busca local tendo em vista o aumento do desempenho do algoritmo GRASP. A Figura 12 ilustra o funcionamento da busca local para o problema de movimentação de *tripper*. O primeiro passo é definir a estrutura de vizinhança. A Figura 12a apresenta as possibilidades de vizinhança para uma solução inicial qualquer. A posição de partida é representada pela letra "S" na cor branca, as posições que fazem parte da solução inicial são representadas por círculos na cor cinza escuro e as posições possíveis para um vizinho são indicadas por círculos na cor cinza claro e são marcadas por um ponto de interrogação (?). O caminho percorrido ao longo da solução inicial está marcado por uma seta sólida (⟶).

O procedimento de busca começa da posição inicial do *tripper*. Uma ou duas opções podem ser escolhidas para a próxima etapa, estas opções de direcionamento são indicadas por uma seta tracejada (⇢) na Figura 12a. Por exemplo, para o estado x_{33} as opções possíveis para a próxima etapa são x_{34} e x_{24}, que complementam a posição inicial para

este período x_{44}. Este estado, que faz parte de solução atual, deixaria de ser considerada na estrutura de um novo vizinho. Se a nova posição seguinte à x_{33} for x_{34}, o novo vizinho terá exatamente uma posição de diferença em relação à solução inicial. A Figura 12b apresenta as arestas de correção de continuidade, representados por uma seta pontilhada (⋯➤). Caso o estado x_{23} seja escolhido, como é indicado por um nó marcado por um sinal de adição (+) na Figura 12c, o novo vizinho estará com uma descontinuidade no caminho entre as posições iniciais e finais, já que não há rota disponível entre os nós x_{23} e x_{45}. Para contornar esta situação, o nó x_{34} é adicionado à nova rota, substituindo-se o nó x_{44} que fazia parte da solução inicial até então. A Figura 12d mostra a nova rota corrigida e traçada desde o nó x_{22} até o nó x_{45}. Consequentemente, os nós x_{33} e x_{44} são eliminados da solução, como está indicado pelos sinais de subtração (–). Esta nova solução se torna viável após a correção de continuidade.

A busca local verifica todas as soluções corrigidas relativas a todas as posições vizinhas possíveis. Em caso de melhora no custo da nova solução, esta é imediatamente escolhida como a nova melhor solução local. Caso contrário, o algoritmo retorna após testar todas as possibilidades e não encontrar nenhuma alternativa melhor do que a solução atual.

Tabela 3 – Parâmetros de entrada do algoritmo de busca local.

Parâmetro	**Descrição**
I	Conjunto de posições iniciais
Q	Conjunto de vazões de saída
q	Vazão mássica de entrada
p	Posição inicial
n	Número de compartimentos
e	Número de períodos de tempo

O Algoritmo 6 apresenta o procedimento de busca local que foi detalhado anteriormente, a descrição das entradas do algoritmo está mostrada na Tabela 3. Primeiramente, a variável *melhorCusto* é inicializada com o valor do custo atual da solução de entrada na linha 2. Em seguida, entre as linhas 3 à 22, um laço verifica as possibilidades de movimentação das posições respectivas de cada uma das etapas, com exceção do primeiro passo que é a posição de partida e não pode ser alterado. O teste condicional que abrange as linhas 5 à 12 verifica se a posição atual nesta etapa j é a primeira posição possível para movimentação. Caso seja, a verificação é abortada. Senão, reduz-se uma posição com relação a posição atual, realiza-se a correção de continuidade e, por fim, verifica-se se o valor do novo custo é maior do que o valor do custo atual. Caso seja confirmado o teste, o algoritmo retorna na primeira melhoria, senão prossegue até encontrar um incremento do valor do custo ou após verificar todas as etapas possíveis. Entre as linhas 14 e 21 um novo teste condicional é realizado, contudo, neste caso, a movimentação é feita na direção oposta à da primeira verificação.

Algoritmo 6 Procedimento busca local

```
1: função BuscaLocal(I, Q, q, p, n, e)
2:     melhorCusto ← CalculaCusto(I, Q, q, p, n, e)
3:     para j ← 2 até e faça
4:         novoI ← I
5:         se novoI[j] > 0 então
6:             novoI[j] ← novoI[j] − 1
7:             novoI ← Correção(novoI, j, e)
8:             novoCusto ← CalculaCusto(novoI, Q, q, p, n, e)
9:             se novoCusto > melhorCusto então
10:                retorna novoCusto, novoI
11:            fim se
12:        fim se
13:        novoI ← I
14:        se novoI[j] < n − 1 então
15:            novoI[j] ← novoI[j] + 1
16:            novoI ← Correção(novoI, j, e)
17:            novoCusto ← CalculaCusto(novoI, Q, q, p, n, e)
18:            se novoCusto > melhorCusto então
19:                retorna novoCusto, novoI
20:            fim se
21:        fim se
22:    fim para
23: fim função
```

3.3.2.3 Algoritmo GRASP

A fase de construção GRASP é uma modificação da formulação gulosa. Durante o processo de construção da solução, os novos elementos são escolhidos de forma aleatória dentro de um conjunto com os melhores candidatos possíveis, expandindo-se a quantidade de soluções possíveis de serem geradas. Experimentos preliminares mostraram que repetindo-se este processo múltiplas vezes, é possível encontrar soluções melhores do que utilizando-se o método de construção de solução guloso. O Algoritmo 7 mostra o pseudocódigo proposto para o problema de movimentação de *tripper* e o Algoritmo 8 mostra o procedimento de construção da solução. Os parâmetros de entrada do procedimento estão apresentados na Tabela 4.

O Algoritmo 7 apresenta a visão geral do procedimento principal do GRASP. O laço de execução que contém o procedimento de construção da solução, mostrado entre as linhas 3 e 12, é executado enquanto o valor de custo da solução atual for maior do que o melhor custo da função objetivo encontrado. O procedimento *construçãoGrasp* realiza a construção da solução e será detalhado adiante.

O procedimento de construção da solução começa nas linhas 2 e 3 com a inicialização da primeira posição da solução parcial e de sua auxiliar, atribuindo-se a posição de partida do equipamento. Os valores dos níveis dos compartimentos do silo **L** são inicializados na linha

Tabela 4 – Parâmetros de entrada do algoritmo GRASP.

Parâmetro	**Descrição**
I	Conjunto de posições iniciais
Q	Conjunto de vazões de saída
q	Vazão mássica de entrada
p	Posição inicial
n	Número de compartimentos
e	Número de períodos de tempo
Pr	Probabilidade de escolha aleatória

Algoritmo 7 Procedimento principal GRASP

```
1: função GRASP(I, Q, q, p, n, e, Pr)
2:     melhoria ← verdadeiro
3:     enquanto melhoria faça
4:         construçãoGrasp(I, Q, q, p, n, e, Pr)
5:         novoCusto ← BuscaLocal(I, Q, q, p, n, e)
6:         se novoCusto > melhorCusto então
7:             melhorCusto ← novoCusto
8:             melhorI ← I
9:         senão
10:            melhoria ← falso
11:        fim se
12:    fim enquanto
13: fim função
```

4. Entre as linhas 5 e 25, o laço de repetição principal é executado para todas as etapas da instância, $2 \leq j \leq t-1$. O primeiro passo dentro do laço se define pela construção do conjunto de movimentos possíveis para a etapa atual, denominado Δ, na linha 6. A função Mov(p_i) define os valores possíveis para Δ, como mostra a Equação (3.18):

$$\text{Mov}(p_i) = \begin{cases} \{-1, 0, 1\}, & \text{se } 1 < p_i < n \\ \{0, 1\}, & \text{se } p_i = 1 \\ \{-1, 0\}, & \text{se } p_i = n \end{cases} \tag{3.18}$$

este conjunto pode conter 2 ou 3 elementos de acordo com a posição atual do *tripper*. As posições próximas à extremidade do silo $p_i \in \{1, n\}$ contêm dois elementos: permanecer na mesma posição ou se movimentar em direção ao centro do silo. Se o equipamento não estiver posicionado nas extremidades $1 < p_i < n$, três possibilidades de movimentação são possíveis.

Para cada uma das opções de movimentação contidas em Δ, é calculado o custo adicional gerado por cada uma destas escolhas possíveis – linhas 7 à 11. A função *SimulaNível* calcula os valores dos níveis do silo para cada período de tempo de acordo com as entradas

Algoritmo 8 Construção da solução GRASP

```
1: função ConstruçãoGrasp(I, Q, q, p, n, e, Pr)
2:     Solução[1] ← p
3:     Aux[1] ← p
4:     L ← SimulaNível(Aux, L, I, Q, q, n, 1, 2)
5:     para j ← 2 até t − 1 faça
6:         Δ ← Mov(Aux[j − 1])
7:         for all i ∈ Δ faça
8:             Aux[j] ← Solução[j − 1] + i
9:             L ← SimulaNível(Aux, L, I, Q, q, n, j, j + 1)
10:            custo[i] ← DesvioPadrão(L[j + 1], n)
11:        fim para
12:        x ← Aleatório([0, 1])
13:        se ∀(a, b) ∈ Δ : custo[a] = custo[b] então
14:            Solução[i] ← Solução[j − 1] + Aleatório(Δ)
15:        senão se ∃(a, b) ∈ Δ : custo[a] = custo[b] ou x < Pr então
16:            Solução[j] ← Solução[j − 1] + argMin(custo)
17:        senão
18:            Δ′ ← Δ
19:            se Tamanho(Δ) = 3 então
20:                Δ′ ← Δ′ \ argMax(custo)
21:            fim se
22:            Solução[j] ← Solução[j − 1] + Aleatório(Δ′)
23:        fim se
24:        Aux[j] ← Solução[j]
25:    fim para
26:    Solução[t] ← Solução[t − 1]
27:    retorna Solução
28: fim função
```

fornecidas em sua chamada. O passo seguinte é a determinação de qual será a próxima posição de movimentação, estabelecido pelas linhas 12 à 22. Três critérios de escolha são possíveis:

Linhas 13 e 14: Para todo par (a, b) de soluções pertencentes ao intervalo Δ, cujos valores de custo sejam iguais, a escolha da próxima posição é feita aleatoriamente.

Linhas 15 e 16: Se existe um par (a, b) de soluções pertencentes ao intervalo Δ, cujos valores de custo sejam iguais ou se uma variável pseudoaleatória x é menor do que o valor mínimo Pr, a escolha da próxima é feita de maneira gulosa.

Linhas 17 à 22: Caso contrário, a escolha se dá entre os dois elementos do conjunto Δ que implicam em maior custo para a função objetivo.

O procedimento proposto se diferencia da definição original do algoritmo GRASP estabelecida pela literatura (RESENDE; RIBEIRO, 2002b). Esta particularização se fez

necessária já que apenas duas ou três opções de escolha são possíveis para a seleção do próximo passo de movimentação independentemente do estado do sistema. O critério original para determinação de qual será o próximo elemento a ser incluído na solução parcial é definido pela escolha aleatória de um elemento a partir de uma lista restrita de melhores candidatos possíveis para inclusão na solução atual. A formulação feita para o problema de movimentação de *tripper* limita a RLC (*Restricted List of Candidates*) aos elementos de Δ', um subconjunto de Δ que contém as duas opções de movimentação que resultam no menor incremento de custo possível à solução parcial. Além disso, o critério adotado para o problema em questão exclui situações em que duas ou três opções de movimentação resultem em custo igual. Nestes casos, a escolha é feita semelhantemente ao algoritmo guloso.

A variável aleatória x é responsável por determinar se a escolha do próximo elemento será feita aleatoriamente dentro de Δ', caso $x \geq Pr$, ou se dará de forma gulosa, se $x < Pr$. Desta forma, o parâmetro Pr permite ajustar o desempenho do algoritmo, tornando-o totalmente guloso, caso $Pr = 1$, ou completamente aleatório, se $Pr = 0$.

A linha 24 do Algoritmo 8 copia o valor de posição definido para a etapa j da variável auxiliar *aux* para *solução*. Como o custo resultante de uma ação de posicionamento é caracterizado pelos valores de nível do silo na próxima etapa, não é possível selecionar um valor de posição para $j = t$. Desta forma, após a última iteração do laço em $j = t - 1$ a solução final é alcançada repetindo-se o último valor de posição para o *tripper*, como visto na linha 26 do pseudocódigo.

3.3.3 *Simulated Annealing*

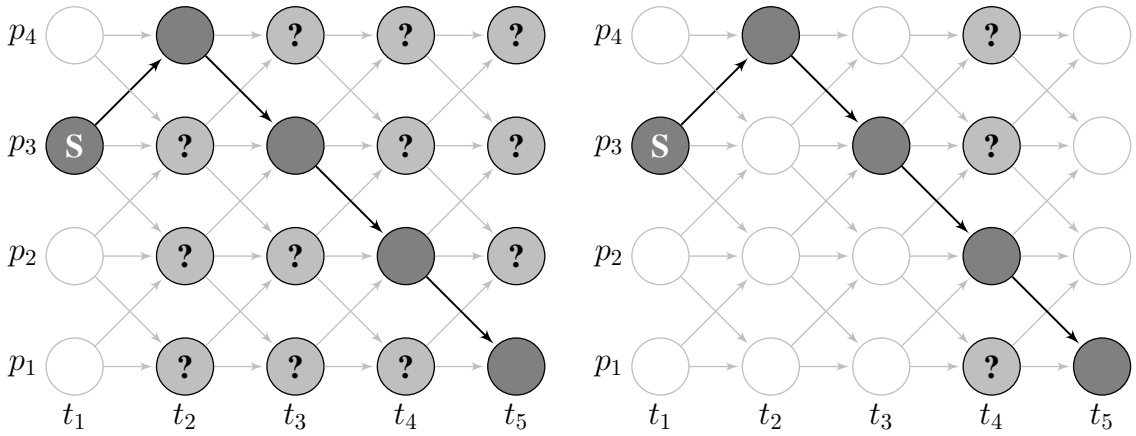
O *Simulated Annealing* é a segunda meta-heurística proposta por este trabalho para solução do problema de movimentação de *tripper*. A implementação deste procedimento foi baseada no pseudocódigo sugerido pela literatura, como discutido na Seção 2.4. A Tabela 5 apresenta os parâmetros de entrada do algoritmo proposto. Os parâmetros T, α e i_{max} determinam o desempenho do procedimento e são analisados em detalhe.

O Algoritmo 9 apresenta o pseudocódigo da adaptação do SA para solução do problema de movimentação de *tripper*. O procedimento começa com a inicialização das variáveis *solução*, i e *custo* nas linhas 2 à 4. A solução inicial é gerada aleatoriamente partindo-se da posição p. O laço principal do algoritmo é definido entre as linhas 5 e 28. Enquanto a temperatura T permanecer acima do limiar inferior 1, o laço é executado. Aninhado dentro desta estrutura, há outro laço ao longo das linhas 6 à 25. Neste caso, o objetivo é testar um conjunto de i_{max} estados sob uma mesma temperatura.

Primeiramente, é introduzida uma mudança aleatória na solução atual – linhas 9 e 10. Esta alteração é feita escolhendo-se uma etapa pivô aleatoriamente e, semelhantemente, atribuindo-lhe um valor aleatório de posicionamento. Esta nova solução é processada pelo algoritmo de correção de continuidade, como descrito na Seção 3.3.1, assegurando-se que

Tabela 5 – Parâmetros de entrada do algoritmo SA.

Parâmetro	**Descrição**
I	Conjunto de níveis iniciais
Q	Conjunto de vazões de saída
q	Vazão mássica de entrada
p	Posição inicial
n	Número compartimentos do silo
e	Número de períodos de tempo
T	Temperatura inicial
α	Taxa de resfriamento
i_{max}	Número máximo de iterações



(a) Possibilidades de mudança de estado. (b) Seleção aleatória do pivô e_4.

Figura 13 – Geração de um estado aleatório pelo algoritmo SA.

este conjunto seja contínuo e, portanto, viável ao longo de todas as etapas – linha 11. A Figura 13 mostra um exemplo do processo de geração de um novo estado a partir da solução atual do problema. Qualquer um dos estados possíveis, com exceção do período e_1 e dos presentes na solução, podem ser escolhidos, como pode ser visto na Figura 13a. Evidentemente, o estado x_{12} é inviável já que não há possibilidade de movimentação da posição inicial p_3 diretamente para p_1. Desta forma, os estados que não são alcançáveis a partir da posição inicial são forçados para um valor viável mais próximo. A Figura 13b mostra as opções de posicionamento relativas à escolha do período e_4. Qualquer uma das posições disponíveis para mudança requerem que a solução final seja corrigida para garantir sua continuidade.

Em seguida, o custo da nova solução é calculado na linha 12. Se for melhor do que a solução atual, este novo resultado é armazenado como o melhor alcançado até o momento – linhas 13 à 18. Senão, a nova solução é acolhida caso o valor da variável pseudoaleatória w seja menor do que o limiar $e^{\Delta/T}$ – linhas 20 à 25. Após o número máximo de iterações ser atingido, o laço externo é reiniciado reduzindo-se a temperatura por uma taxa α.

Algoritmo 9 *Simulated Annealing*

```
1: função SA(I, Q, q, p, n, e, T, α, i_max)
2:     Solução ← SoluçãoInicial(n, t, p)
3:     i ← 0
4:     custo ← CalculaCusto(Solução, L, I, Q, q, n, e)
5:     melhor ← custo
6:     enquanto T > 1 faça
7:         enquanto i < i_max faça
8:             i ← i + 1
9:             mudança ← Aleatório([2..e])
10:            Solução[mudança] ← Aleatório([1..n])
11:            Solução ← CorreçãoContinuidade(Solução, mudança, t)
12:            aux ← CalculaCusto(Solução, L, I, Q, q, n, e)
13:            Δ ← aux − custo
14:            se Δ > 0 então
15:                custo ← aux
16:                se custo > melhor então
17:                    melhor ← custo
18:                fim se
19:            senão
20:                probabilidade ← e^(Δ/T)
21:                w ← Aleatório([0, 1])
22:                se w < probabilidade então
23:                    custo ← aux
24:                fim se
25:            fim se
26:        fim enquanto
27:        i ← 0
28:        T ← T × α
29:    fim enquanto
30:    retorna Solução
31: fim função
```

Quanto menor o valor de α mais rápido o algoritmo converge para a temperatura mínima. Contudo, a probabilidade de se encontrar o ótimo global consequentemente se reduz (DU; SWAMY, 2016). Desta forma, a escolha dos parâmetros T, α e i_{max} é fundamental para garantir a eficiência do algoritmo SA.

4 Resultados

Os resultados estão divididos entre as metodologias exatas e as meta-heurísticas. O objetivo primeiro grupo de testes é avaliar estratégias de movimentação de *tripper* e comparar o desempenho dos resolvedores na tarefa de se encontrar uma solução exata para o problema. O segundo grupo de testes busca avaliar o desempenho das meta-heurísticas GRASP e *Simulated Annealing* por meio da análise da assertividade e do tempo despendido por estas metodologias na busca por soluções de qualidade. Os algoritmos dos experimentos foram desenvolvidos nas linguagens C/C++, inclusive as interfaces com os resolvedores GLPK e CPLEX, utilizando-se da interface de desenvolvimento Microsoft Visual C++ 2010 Express. Os experimentos foram realizados em um computador cujas características estão mostradas na Tabela 6.

Tabela 6 – Configuração do computador utilizado nos testes.

Parâmetro	**Valor**
Processador	Intel Core i5 4570 3,6 GHz
Memória	16GB DDR3

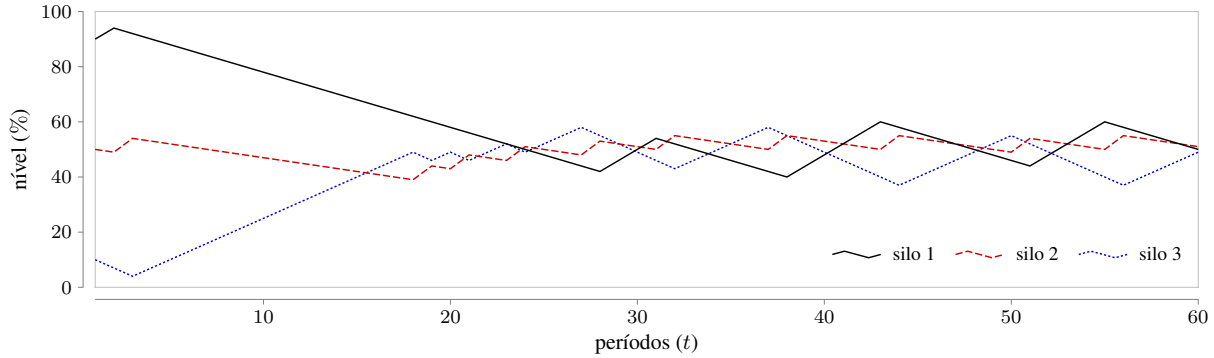
Cada instância é referenciada por um conjunto de três parãmetros: o número n de compartimentos de silo , o número e de períodos e a posição inicial p. A forma compacta de referência à uma instância qualquer se dá por:

$$tripper.n.t.p$$

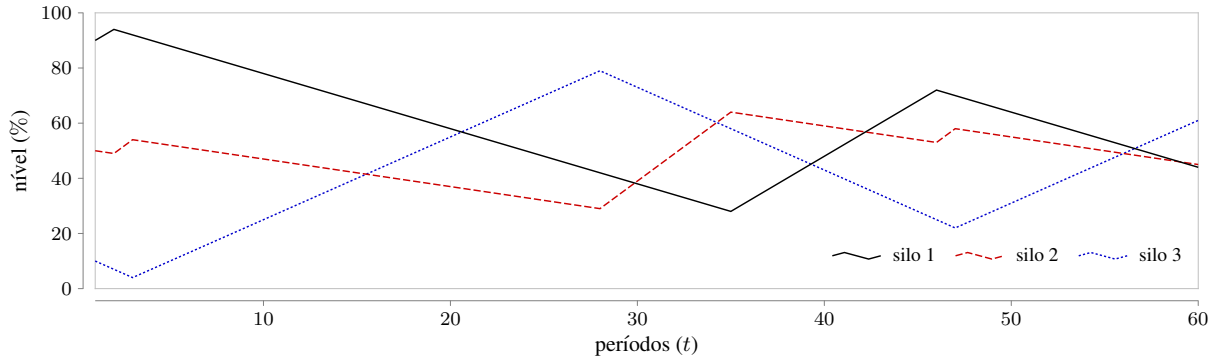
Por exemplo, uma instância com 3 compartimentos de silo, 10 períodos de tempo e com posição inicial 1 é representada por “tripper.3.10.1”.

4.1 Algoritmos Exatos

Três experimentos computacionais são apresentados nesta seção. O primeiro é uma comparação entre a proposta de controle por estabilização e minimização da movimentação do carro, como indicada por (KARELOVIC; PUTZ; CIPRIANO, 2015), e o algoritmo proposto por este trabalho para o problema de posicionamento. O segundo teste é feito sobre um silo com taxas de vazão desbalanceadas, o propósito é verificar o modelo proposto neste cenário. O último teste compara o desempenho dos resolvedores CPLEX e GLPK com relação ao algoritmo de programação dinâmica.



(a) Estratégia de maximização do menor nível a cada iteração.



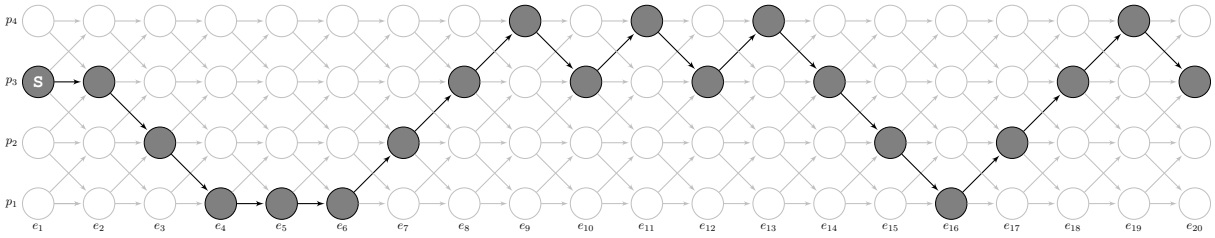
(b) Estratégia de minimização de movimentação.

Figura 14 – Testes comparativos entre estratégias de movimentação para uma instância tripper.3.60.1.

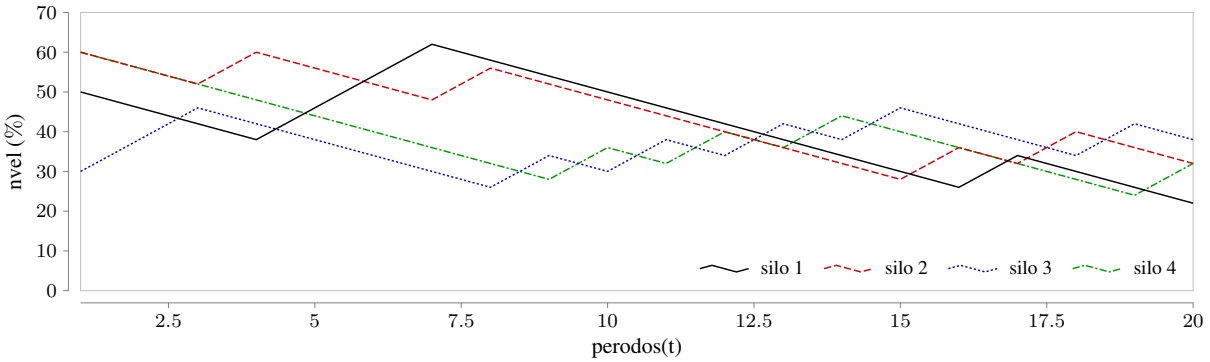
4.1.1 Comparativo de estratégias

O primeiro teste foi realizado sobre uma instância com silo de 3 compartimentos. Os níveis iniciais foram definidos como $L = \{90, 50, 10\}$, as taxas dos alimentadores $Q = \{3, 1, 2\}$, a taxa de alimentação do *tripper* é $q = 6$ e a posição inicial, $p = 1$. O objetivo é comparar as duas propostas de controle dos níveis do silo. A estratégia de estabilização foi ajustada para manter os níveis dentro de uma faixa aceitável de $20\% \leq L \leq 80\%$.

A Figura 14 mostra os resultados do teste. O gráfico superior apresenta o comportamento dos níveis do silo ao longo do tempo em resposta às ações para maximização do menor nível em cada iteração. Neste caso, após aproximadamente 20 períodos de tempo, os níveis se estabilizam dentro de uma faixa entre 40% e 60%. O teste da estratégia de minimização de movimentação, vista no gráfico inferior da Figura 14, os níveis ficaram dentro da faixa solicitada de 20% a 80%. A movimentação do carro é bem menor do que no primeiro caso, conforme pode ser visto pela pequena quantidade de inversões na taxa de variação dos níveis.



(a) Posicionamento do *tripper*.



(b) Resultados do comportamento do nível para cada compartimento do silo.

Figura 15 – Teste em uma instância desbalanceada tripper.4.20.3.

4.1.2 Sistema desbalanceado

O segundo teste foi realizado considerando-se uma instância com cargas desbalanceadas. O sistema com 4 compartimentos de silo, com taxas de vazão mássica de saída $\{Q_i = 4, \forall i \in P\}$ e taxa de vazão mássica de entrada de $q = 12$, os níveis iniciais definidos como $L = \{50, 60, 30, 60\}$. Neste caso, existe uma possibilidade real de que haja falta de material em algum dos compartimentos do silo caso o posicionamento do *tripper* não seja feito corretamente. A Figura 15a apresenta o posicionamento resultante para esta instância. A letra "`S`" na cor branca indica a posição de partida em $p = 3$. O carro é posicionado ao longo dos 20 períodos em todas as posições possíveis de modo a maximizar o nível mínimo em cada período.

O desbalanceamento entre as taxas de vazão de entrada e saída implicam em redução gradual dos níveis médios do silo ao longo do tempo, como mostra a Figura 15b. Os níveis médios inicial e final foram 50 e 31, confirmando o comportamento esperado. Contudo, o desvio padrão inicial dos níveis do silo foi $14,1$ contra o valor final de $6,6$, ou seja, não somente os níveis mínimos foram mantidos sob valores elevados mas suas variabilidades também foram reduzidas.

4.1.3 Comparativo entre resolvedores e programação dinâmica

O propósito do terceiro conjunto de testes é comparar o desempenho dos resolvedores com relação à programação dinâmica. Os testes foram realizados sobre instâncias com silo de $n \in \{3, 5, 7, 11, 16, 24\}$ compartimentos e com janela de previsão de períodos

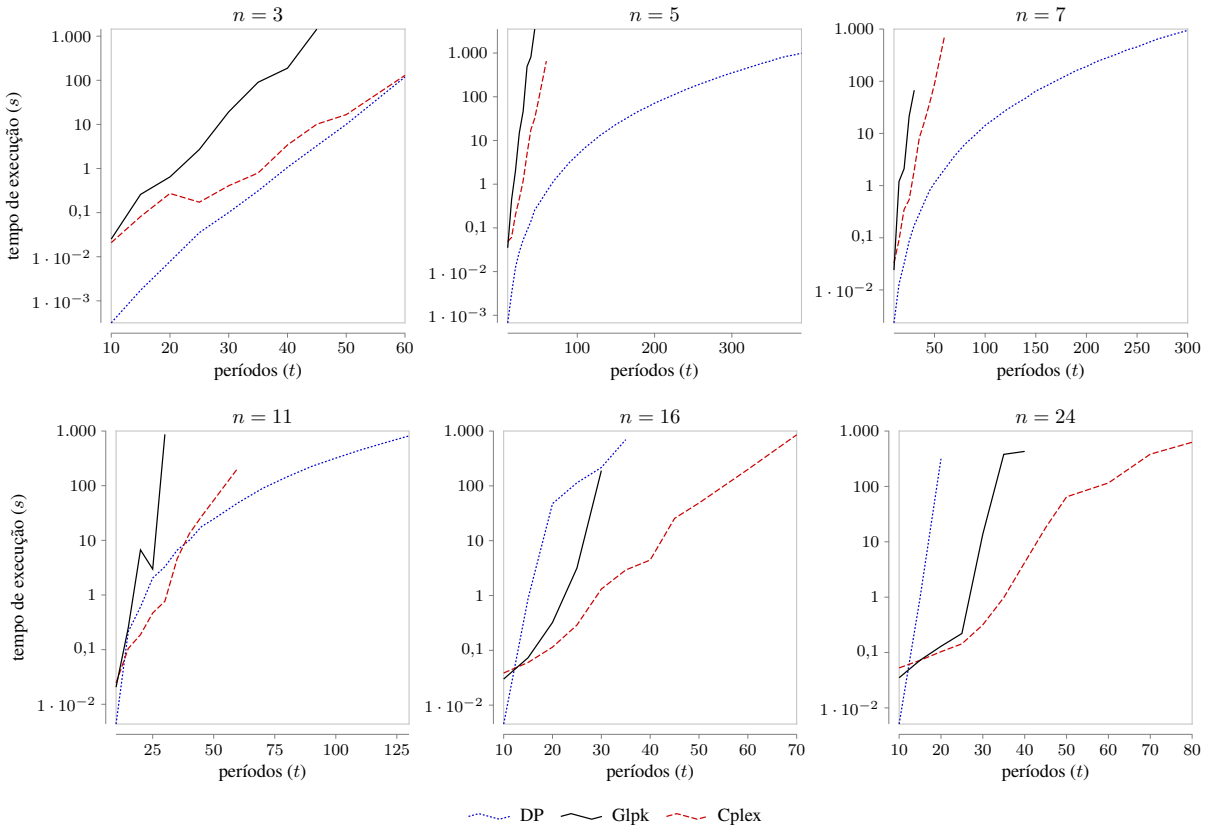


Figura 16 – Resultados dos testes dos resolvedores CPLEX e GLPK e do algoritmo de programação dinâmica.

em T. Estes parâmetros foram variados exaustivamente dentro de uma janela de tempo de execução de 1000 segundos, com o propósito de se verificar o tempo de resposta dos resolvedores. Este valor de tempo foi escolhido de forma que a avaliação possa ser realizada em uma janela de tempo muito maior do que o necessário para implantação dos métodos propostos em um processo real. Os parâmetros de taxa de vazão escolhidos para os alimentadores foram $\{Q_i = 1/n, \forall i \in P\}$, a taxa de vazão do *tripper* foi $q = 1$ e a posição inicial $p = 1$. É esperado que o tempo gasto para solução de uma instância seja proporcional ao número de compartimentos n. Esta variável representa uma das dimensões do problema, juntamente com o número de períodos e, e seu aumento é diretamente relacionado com o incremento de complexidade do problema.

Os resultados numéricos do experimento estão mostrados na Tabela 10 no Apêndice A e uma visão dos resultados é mostrado pela Figura 16. O algoritmo de programação dinâmica foi assintoticamente mais rápido nos testes com o número de compartimentos do silo $n \in \{3, 5, 7\}$ e em boa parte dos testes com $n = 11$. A discrepância entre o resultado da PD com relação aos resultados dos resolvedores nos testes com $n \in \{5, 7, 11\}$ sugere que nestes casos a programação dinâmica teve ordem de complexidade menor do que os resolvedores de programação linear inteira mista. Com $n = 16$, os resultados do CPLEX foram superiores ao algoritmo proposto, embora aparentemente o intervalo de tempo

tenha sido pequeno para avaliar os algoritmos para esta instância. Os dois resolvedores foram mais eficientes do que o algoritmo proposto em instâncias com $n = 24$. Em $n = 3$, embora o desempenho da PD tenha sido mais rápido do que os outros métodos, ele foi assintoticamente similar. Uma teoria para explicar este comportamento é que o algoritmo não foi capaz de tomar vantagem do nível maior de redundância de soluções presentes em uma instância de $n = 3$. O CPLEX foi mais rápido do que o GLPK em todas as instâncias testadas. Um ponto a favor do algoritmo de programação dinâmica é o fato de o CPLEX ser capaz de aproveitar todos os processadores disponíveis (quatro neste caso), o que incrementa o desempenho do resolvedor. De qualquer forma, este benefício não muda fundamentalmente a estrutura assintótica do método. Diferentemente, a implementação proposta para a PD somente lida com com uma linha de execução, não tomando proveito de toda capacidade de processamento disponível. O único método a esgotar a memória disponível na máquina na tentativa de se encontrar a solução exata foi o CPLEX, isto ocorreu na instância tripper.16.80.1.

4.2 Meta-heurísticas

O desempenho das duas meta-heurísticas em gerar soluções para o problema de movimentação de *tripper* foi avaliado por meio de experimentos computacionais. O objetivo é verificar o tempo despendido e a qualidade das soluções encontradas pelos algoritmos. Estes testes foram divididos em duas categorias: comparação da precisão e da rapidez dos algoritmos com relação aos métodos exatos e verificação do desempenho das meta-heurísticas frente à instâncias que não puderam ser solucionadas pelos métodos exatos dentro das restrições de tempo estabelecidas neste trabalho.

As instâncias escolhidas para realização dos testes possuem taxa de vazão mássica homogênea para todos os alimentadores do silo, semelhantemente às instâncias utilizadas no experimento comparativo dos métodos exatos. Esta opção permite que a comparação entre o desempenho das meta-heurísticas e dos métodos exatos seja consistente. A Tabela 7 apresenta os parâmetros gerais para as instâncias. Como se observa na tabela, o balanço de massa existente entre a alimentação proveniente do *tripper* e as saídas do silo é equilibrado, ou seja, a quantidade de material alimentado é igual ao total de minério retirado do silo. O valor dos níveis do silo são limitados entre $0 \leq L \leq 100$, estas limitações implicam que os compartimentos do silo podem estar no mínimo completamente vazios ou no máximo totalmente cheios. O nível inicial dos compartimentos do silo foi estabelecido em 50%.

Antes da realização dos experimentos computacionais, os valores dos parâmetros das meta-heurísticas foram ajustados por meio do procedimento de ajuste automático de configuração de algoritmos *irace* (LóPEZ-IBáñEZ et al., 2016). Um conjunto de instâncias utilizadas para a calibração foi constituído de todas as combinações possíveis de sistemas

Tabela 7 – Valores dos parâmetros dos testes.

Parâmetro	**Valor**
Limites dos níveis	$0 \leq L \leq 100$
Posição inicial	$p = 1$
Massa de saída	$Q_i = 1/n, \forall i \in P$
Massa do *tripper*	$q = 1$

com $n \in \{3, 5, 7, 11, 16, 24\}$ compartimentos de silo, com períodos entre $10 \leq t \leq 500$ e em múltiplos de 10 e posição inicial de partida $p \in \{1, \lfloor n/2 \rfloor\}$. Além do conjunto descrito pela Tabela 7, novas instâncias constituídas por valores de nível inicial e vazão de saída aleatórios também foi considerado para execução do *irace*.

O algoritmo GRASP possui apenas o parâmetro Pr para ser ajustado. O procedimento *irace* limitado à 10000 iterações resultou em um ajuste, relativo ao melhor conjunto de configurações, de $Pr = 0{,}9707$, a faixa de variação entre $\{Pr \in \mathbb{R} \mid 0{,}8 \leq Pr \leq 0{,}99\}$. No caso do *Simulated Annealing*, o número de iterações foi restrito à 1000 em função da maior quantidade de tempo de execução despendido pelo método. O SA possui três parâmetros para ajuste: temperatura inicial T, número de iterações máximo $i_{\max}$ e taxa de resfriamento α. Os parâmetros ajustados pelo *irace* para o *Simulated Annealing* estão mostrados na Tabela 8. Estes parâmetros conferidos pelo *irace* às duas meta-heurísticas foram utilizados na realização dos experimetos computacionais. As faixas de variação dos parâmetros foram definidas dentro dos intervalos: $\{T \in \mathbb{R} \mid 1 \leq T \leq 1000\}$, $\{\alpha \in \mathbb{R} \mid 0{,}85 \leq \alpha \leq 0{,}96\}$ e $\{i_{\max} \in \mathbb{N} \mid 1 \leq i_{\max} \leq 1000\}$.

Tabela 8 – Valores dos parâmetros do *Simulated Annealing* ajustados pelo *irace*.

Parâmetro	**Valor**
T	383,74
α	0,9592
$i_{\max}$	883

4.2.1 Comparação com o desempenho dos métodos exatos

A primeira meta-heurística testada foi o GRASP. O algoritmo foi executado 100 vezes para cada uma das instâncias que foram solucionadas pelos métodos exatos. A cada uma das execuções o método foi chamado seguidamente até que não houvesse melhora no custo da função objetivo. As seguintes avaliações foram realizadas para cada instância: tempo de execução, custo médio, melhor custo, percentual do melhor custo em relação à solução ótima e percentual de soluções ótimas com relação ao total de execuções. A

Tabela 11, localizada no Apêndice B, mostra os resultados numéricos do experimento. Diferentemente da implementação do algoritmo do GRASP sugerida, os custos médio e melhor para as soluções encontradas apresentados na tabela foram calculados por meio da função *MinMax*. Desta forma, a comparação entre os resultados da meta-heurística e dos métodos exatos pode ser realizada consistentemente.

Semelhantemente ao GRASP, o *Simulated Annealing* foi executado 100 vezes para cada uma das instâncias que foram solucionadas pelos métodos exatos neste trabalho. As mesmas avaliações feitas com o algoritmo GRASP também foram realizadas com o SA: tempo de execução, custo médio, melhor custo, percentual do melhor custo em relação à solução ótima e percentual de soluções ótimas com relação ao total de execuções. A Tabela 12, localizada no Apêndice C, mostra os resultados numéricos do experimento.

Comparando-se os dois experimentos pode-se verificar que os dois algoritmos estiveram muito próximos da solução exata. O GRASP teve grande superioridade em relação ao *Simulated Annealing* com respeito à precisão das soluções encontradas. O GRASP alcançou a solução exata em todas as instâncias testadas e o percentual do número de vezes em que a solução ótima foi encontrada em cada instância foi 99,65% na média. Em contrapartida, o SA alcançou a solução ótima em apenas 33,01% das vezes e o custo das melhores soluções encontradas ficaram em 99,44% do custo da solução ótima.

O tempo gasto para se encontrar uma solução é outro ponto importante da análise das meta-heurísticas com relação aos métodos exatos e que fornece a motivação necessárias para utilização destes algoritmos. A Tabela 9 apresenta os tempos despendidos pelas três metodologias para se encontrar a solução para a maior instância solucionada pelos métodos exatos para cada um dos valores de $n \in \{3, 5, 7, 11, 16, 24\}$. Nota-se a grande diferença dos tempos resultantes do processo de busca pela solução despendido pelas três metodologias. O GRASP foi praticamente três ordens de grandeza mais rápido do que o SA, que por sua vez foi praticamente três ordens de grandeza mais rápido do que o melhor método exato. Uma visão geral do tempo gasto por todas as instâncias está mostrado na Figura 17. Observa-se que o GRASP foi mais rápido em todas as instâncias testadas. Em contrapartida, o SA foi mais lento do que o melhor método exato para instâncias com poucas períodos e. Para valores maiores de e, o SA supera fortemente o melhor dos métodos exatos.

4.2.2 Comparação entre as meta-heurísticas

Um segundo conjunto de testes foi realizado com o objetivo de se comparar o valor das soluções encontradas pelas duas meta-heurísticas adaptadas para resolver o problema. As instâncias foram geradas com as mesmas propriedades dos testes comparativos anteriores, conforme definido pela Tabela 7. Diferentemente dos outros experimentos, as instâncias foram estendidas para além do valor máximo de iterações sob as quais os métodos exatos foram capazes de encontrar a solução exata. Neste caso, os valores de e

Tabela 9 – Tempos necessários para se encontrar a solução nas maiores instâncias resolvidas pelos métodos exatos.

Instância	**GRASP** (ms)	**SA** (ms)	**Melhor exato** (s)
tripper.3.60.1	0,184	32,8	119,03
tripper.5.500.1	15,012	1.220,8	992,43
tripper.7.300.1	9,743	1.224,5	943,79
tripper.11.130.1	3,237	810,9	815,32
tripper.16.70.1	1,347	611,3	853,68
tripper.24.80.1	2,598	1.022,5	627,35

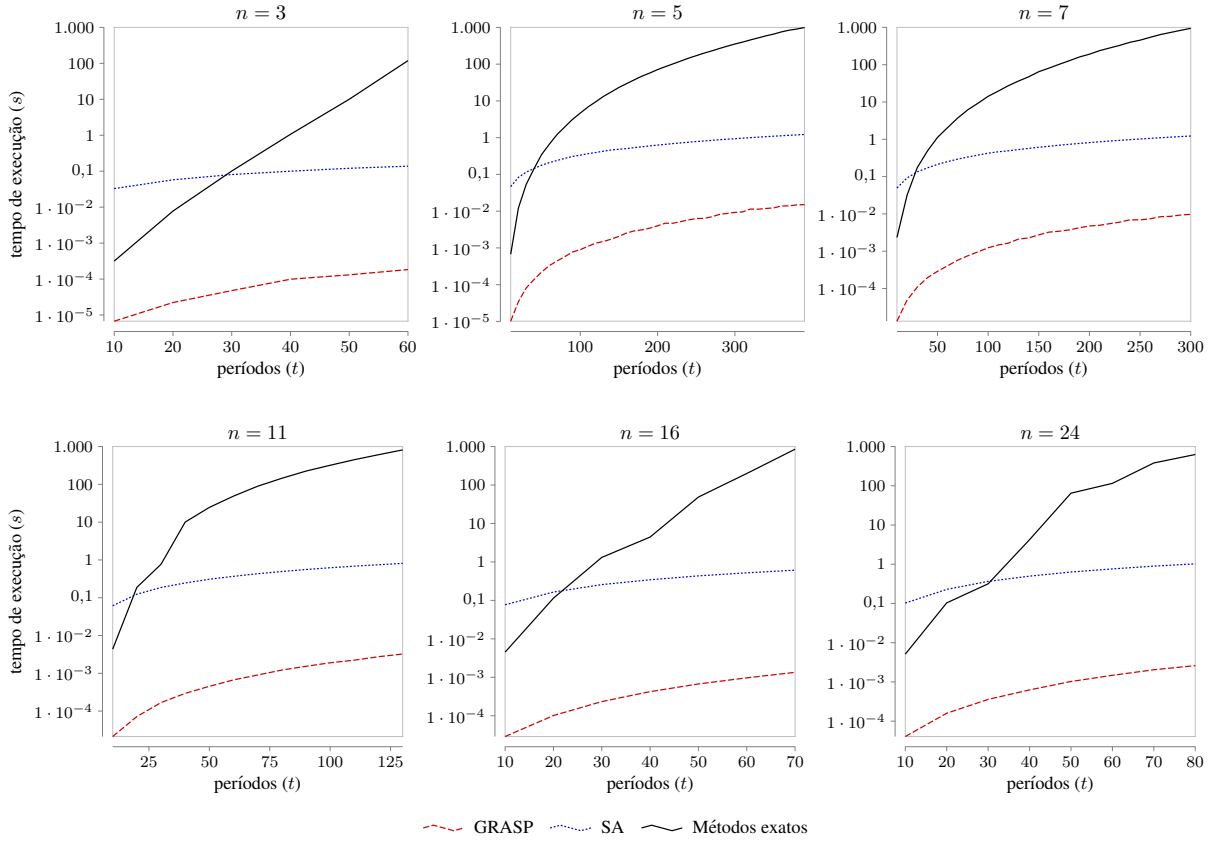


Figura 17 – Comparação entre o menor tempo para se encontrar uma solução dentre os melhores resultados dos métodos exatos e as meta-heurísticas.

foram restringidos à 50 e 1000 períodos, em variações de 50 iterações. Este conjunto de valores supera as maiores instâncias resolvidas pelos métodos exatos, que se limitaram aos valores de $t \in \{30, 500, 300, 130, 70, 80\}$ respectivos aos números de compartimentos do silo $n \in \{3, 5, 7, 11, 16, 24\}$, como mostrado pela Tabela 9.

Os resultados numéricos dos testes estão mostrados na Tabela 13 do Apêndice D. Foram avaliados o tempo médio, o custo médio e o melhor custo da solução encontrada para cada instância. O resultado do experimento é graficamente ilustrado pela Figura 18,

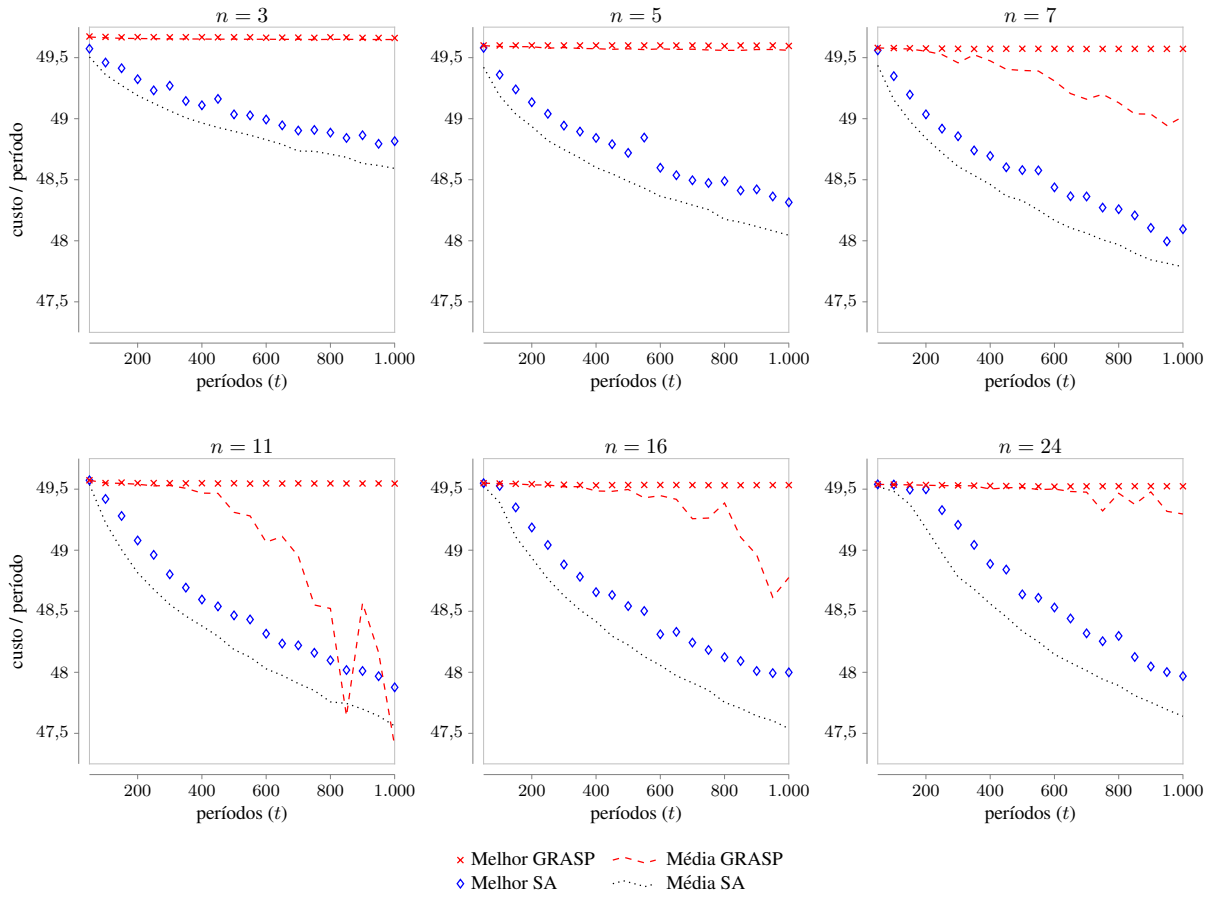


Figura 18 – Comparação das relações de custo por período das soluções encontradas pelos GRASP e *Simulated Annealing*

que apresenta os valores da razão entre o custo e o número de períodos a cada instância. O GRASP se mostrou superior em termos do custo da melhor solução encontrada em todos os experimentos. Particularmente, nas instâncias com valores de $n \in \{7, 11, 16\}$ o custo médio deste método apresentou alguma degradação em relação ao melhor custo alcançado. O custo das soluções encontradas para o SA gradualmente foi se reduzindo ao longo dos períodos, contudo a diferença do custo entre as soluções melhores e médias foi bastante estável ao longo das e iterações.

Uma observação interessante é a relação linear verificada entre o melhor custo encontrado pelo GRASP e o número de períodos. As instâncias que foram criadas para todos os testes comparativos realizados ao longo deste trabalho foram definidas de modo que a alimentação total, os níveis de silo e a posição inicial fossem iguais, além de que a diferença entre taxas de alimentação do *tripper* e saída do silo fosse nula. Estas propriedades aparentemente garantem que o custo da solução ótima varie linearmente com o número de compartimento do silo n.

5 Conclusão

Este trabalho propôs uma nova abordagem para solução do problema de posicionamento de *tripper*. A proposta foi modelar o sistema silo-*tripper* como um problema de otimização combinatória e resolvê-lo por meio de métodos exatos diretamente, sem a utilização de *frameworks* de MPC híbrido. O objetivo escolhido foi maximizar o menor nível do silo em cada periodo, de forma a alcançar a máxima estabilidade na operação do *tripper*. Este objetivo foi comparado com uma outra proposta indicada na literatura (KARELOVIC; PUTZ; CIPRIANO, 2015), neste caso o autor indicou a minimização da quantidade de movimentação do carro e a estabilização dos níveis. Algoritmos de programação linear inteira mista e programação dinâmica foram utilizados para resolver o problema.

O algoritmo exato proposto resultou em uma forma de maximização da estabilização dos níveis, tirando proveito da liberdade de movimentação do carro de forma que variações de nível fossem minimizadas ao extremo. Além disso, o desempenho da programação dinâmica foi superior ao desempenho dos resolvedores de programação linear inteira mista nos testes com instâncias inferiores a $n = 16$, em alguns casos, a PD chegou a apresentar ordem de complexidade sugestivamente inferior aos demais algoritmos dentro da janela de tempo avaliada.

Os resultados alcançados com os métodos exatos foram comparados com os resultados dos experimentos com as meta-heurísticas. A finalidade desta proposta foi encontrar alternativas à solução do problema de movimentação de *tripper* que despendam menos tempo na busca de uma solução para o problema de movimentação, viabilizando-se a aplicação da metodologia à um conjunto maior de problemas reais. Dois algoritmos foram adaptados com este objetivo: o GRASP e o *Simulated Annealing*. A resposta dos dois métodos foi comparada com as soluções exatas encontradas dentro das restrições de tempo de execução estabelecidas. Os resultados dos testes do GRASP foram bastante superiores aos resultados do SA, alcançando taxa de assertividade muito maior em tempo de execução médio insignificante.

As principais contribuições deste trabalho se dividem em descrição e elaboração do problema de movimentação de *tripper* e em propostas desenvolvidas para solução deste problema. A escassez de referências na literatura científica a respeito de metodologias para solução do problema em questão fomentam a importância do trabalho desenvolvido dada a relevância do equipamento estudado dentro do processo produtivo de uma planta de beneficiamento de minério. As proposições dos algoritmos de programação dinâmica e GRASP são opções certamente viáveis para aplicação em um processo real garantido-se tanto exatidão nos resultados como rapidez na execução computacional.

Como proposta de trabalho futuro deverá ser considerada a possibilidade de utilização

de outras meta-heurísticas para solução do problema de posicionamento de *tripper*, como por exemplo o *Variable Neighborhood Search* (GENDREAU; POTVIN, 2010; HANSEN; MLADENOVIC; N., 2010), de forma a consolidar as metodologias aqui estudadas e abrir espaço para verificação da aplicabilidade de novos métodos de solução para o problema. Além disso, novas funções objetivo deverão ser verificadas com o propósito de explorar outras possibilidades de funcionamento para o processo. O conjunto de cenários possíveis pode contemplar situações em que as taxas de vazões mássicas sejam variantes, silos possuam número variável de compartimentos ou que o volume nãos seja constante ao longo do tempo, simulando-se situações comumente encontradas em plantas de beneficiamento mineral. Nestes casos, o objetivo seria, por exemplo, minimizar a variabilidade dos níveis dos compartimentos do silo frente às variações e incertezas das taxas de vazão mássicas, ou mesmo considerá-las como variáveis de decisão, abrangendo-se desta forma o escopo de aplicabilidade das soluções propostas. O modelo matemático deverá ser expandido para inclusão de outros processos que possam influenciar diretamente o sistema silo-*tripper*, tais como correias transportadoras, alimentadores e retomadoras. Neste contexto, a função destes equipamentos é definir a vazão mássica de material destinada a alimentar o silo, portanto, modelar e prever a dinâmica destes processos ao longo do tempo permite que suas variações possam ser antecipadas pelo otimizador.

Referências

BELLMAN, R. The theory of dynamic programming. *American Mathematical Society*, 1954.

BEMPORAD, A.; GIORGETTI, N. Logic-based methods for optimal control of hybrid systems. *IEEE Transactions on Automatic Control*, v. 41, n. 6, p. 963–976, 2006.

BERTSEKAS, D. P. *Dynamic Programming and Optimal Control*. 1. ed. [S.l.]: Arena Scientific, 1995. v. 1. 387 p. ISBN 1886529124.

BOYER, S. A. *SCADA: Supervisory Control and Data Acquisition*. 4. ed. [S.l.]: International Society of Automation, 2010. ISBN 9781936007097.

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 2018, Funchal, Madeira, Portugal, March 21-24, 2018, Volume 1.* [S.l.: s.n.], 2018. p. 344–352.

CHEN, D.-S.; BATSON, R. G.; DANG, Y. *Applied Integer Programming: Modeling and Simulation*. 1. ed. [S.l.]: John Wiley & Sons, 2010. 468 p. ISBN 9780470373064.

CORMEN, T. H. et al. *Introduction to Algorithms*. 2. ed. [S.l.]: McGraw-Hill, 2001. 592 p. ISBN 0262032937.

DU, K.-L.; SWAMY, M. N. S. *Search and Optimization by Metaheuristics*: Techniques and algorithms inspired by nature. 1. ed. [S.l.]: Birkhäuser Basel, 2016. 434 p. ISBN 9783319411910.

FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–134, 1995.

GARCíA, C. E.; PRETT, D. M.; MORARI, M. Model predictive control: theory and practice – a survey. *Automatica*, v. 25, n. 3, p. 335–348, 1989.

GENDREAU, M.; POTVIN, J.-Y. (Ed.). *Handbook of Metaheuristics*. 3. ed. [S.l.]: Springer, 2010. ISBN 9781441916631.

HANSEN, P.; MLADENOVIC, N.; N., P. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, v. 175, p. 367–407, 2010.

HENDERSON, D.; JACOBSON, S. H.; JOHNSON, A. W. The theory and practice of simulated annealing. In: GLOVER, F.; KOCHENBERGER, G. A. (Ed.). *Handbook in Metaheuristics*. [S.l.]: Springer, 2002. p. 287–319. ISBN 9781402072635.

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

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 963–976, 1983.

LEVITIN, A. *Introduction to the design & analysis of algorithms*. 3. ed. [S.l.]: Pearson, 2012. ISBN 9780132316811.

LóPEZ-IBáñEZ, M. et al. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, v. 3, p. 43 – 58, 2016. ISSN 2214-7160.

METROPOLIS, N. et al. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, v. 21, n. 6, p. 1087–1092, 1953.

PAL, B.; SANA, S. S.; CHAUDHURI, K. A multi-echelon production-inventory system with supply disruption. *Journal of Manufacturing Systems*, v. 33, p. 262–276, 2014.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. *Combinatorial Optimization: Algorithms and Complexity*. 1. ed. [S.l.]: Dover, 1998. 528 p. ISBN 0486402584.

PHILLIPS, C. L.; PARR, J. M.; RISKIN, E. A. *Signals, Systems, and Transforms*. 4. ed. [S.l.]: Prentice Hall, 2007. 784 p. ISBN 0131989235.

PINEDO, M. L. *Scheduling*: Theory, algorithms, and systems. 4. ed. [S.l.]: Springer-Verlag New York, 2012. 676 p. ISBN 9781461423614.

RESENDE, M. G.; RIBEIRO, C. C. Greedy randomized aadaptive search procedures. In: GLOVER, F.; KOCHENBERGER, G. A. (Ed.). *Handbook in Metaheuristics*. [S.l.]: Springer, 2002. p. 219–218. ISBN 9781402072635.

RESENDE, M. G.; RIBEIRO, C. C. Greedy randomized aadaptive search procedures. *AT&T Labs Research*, 2002.

RICH, E.; KNIGHT, K. *Artificial Intelligence*. 1. ed. [S.l.]: McGraw-Hill Inc., 1991. ISBN 0071008942.

SCHRINVER, A. *Theory of Linear and Integer Programming*. 2. ed. [S.l.]: John Wiley & Sons, 1999. 484 p. ISBN 0471982326.

SWINDERMAN, R. T. *Belt Conveyors for Bulk Materials, w/Metric Conversion*. 7. ed. [S.l.]: Conveyor Equipement Manufactures Association, 2014. 829 p.

WILLS, B. A.; NAPIER-MUNN, T. *An Introduction to the Practical Aspects of Ore Treatment and Mineral Recovery*. 8. ed. [S.l.]: Butterworth-Heinemann, 2015. 512 p. ISBN 9780080970530.

WOLSEY, L. A. *Integer Programming*. [S.l.]: Wiley-Interscience, 1998. 288 p. ISBN 0471283665.

Apêndice A – Resultados dos experimentos computacionais com os métodos exatos

Tabela 10 – Resultados dos testes com os métodos exatos. O tempo de execução foi limitado em 1000 segundos.

Instância	**Tempo GLPK** (s)	**Tempo CPLEX** (s)	**Tempo PD** (s)	**Custo ótimo**
tripper.3.10.1	0,03	0,02	0,00	497,0
tripper.3.20.1	0,65	0,27	0,01	993,7
tripper.3.30.1	19,27	0,41	0,10	1.490,0
tripper.3.40.1	187,87	3,42	1,06	1.987,0
tripper.3.50.1	*	16,65	10,03	2.483,7
tripper.3.60.1	*	129,49	119,03	2.980,0
tripper.3.70.1	*	**	***	†
tripper.5.10.1	0,04	0,05	0,00	496,0
tripper.5.20.1	1,98	0,20	0,01	992,0
tripper.5.30.1	46,70	1,19	0,05	1.488,0
tripper.5.40.1	815,85	17,73	0,14	1.984,0
tripper.5.50.1	*	85,95	0,35	2.480,0
tripper.5.60.1	*	655,18	0,67	2.976,0
tripper.5.70.1	*	**	1,22	3.472,0
tripper.5.80.1	*	**	1,96	3.968,0
tripper.5.90.1	*	**	3,15	4.464,0
tripper.5.100.1	*	**	4,66	4.960,0
tripper.5.110.1	*	**	6,84	5.456,0
tripper.5.120.1	*	**	9,47	5.952,0
tripper.5.130.1	*	**	13,26	6.448,0
tripper.5.140.1	*	**	17,48	6.944,0
tripper.5.150.1	*	**	23,26	7.440,0
tripper.5.160.1	*	**	29,54	7.936,0
tripper.5.170.1	*	**	37,55	8.432,0
tripper.5.180.1	*	**	47,44	8.928,0

* A execução do GLPK excedeu o tempo limite.

** A execução do CPLEX excedeu o tempo limite.

*** A execução da Programação dinâmica excedeu o tempo limite.

† Nenhum método solucionou a instância dentro do tempo limite.

‡ CPLEX esgotou a memória disponível na máquina.

Instância	**Tempo GLPK** (s)	**Tempo CPLEX** (s)	**Tempo PD** (s)	**Custo ótimo**
tripper.5.190.1	*	**	57,45	9.424,0
tripper.5.200.1	*	**	71,14	9.920,0
tripper.5.210.1	*	**	85,35	10.416,0
tripper.5.220.1	*	**	101,70	10.912,0
tripper.5.230.1	*	**	121,37	11.408,0
tripper.5.240.1	*	**	145,43	11.904,0
tripper.5.250.1	*	**	170,04	12.400,0
tripper.5.260.1	*	**	199,18	12.896,0
tripper.5.270.1	*	**	229,27	13.392,0
tripper.5.280.1	*	**	266,72	13.888,0
tripper.5.290.1	*	**	308,06	14.384,0
tripper.5.300.1	*	**	352,90	14.880,0
tripper.5.310.1	*	**	398,11	15.376,0
tripper.5.320.1	*	**	455,91	15.872,0
tripper.5.330.1	*	**	516,93	16.368,0
tripper.5.340.1	*	**	588,38	16.864,0
tripper.5.350.1	*	**	656,73	17.360,0
tripper.5.360.1	*	**	752,60	17.856,0
tripper.5.370.1	*	**	838,73	18.352,0
tripper.5.380.1	*	**	901,85	18.848,0
tripper.5.390.1	*	**	992,43	19.344,0
tripper.5.400.1	*	**	***	†
tripper.7.10.1	0,02	0,03	0,00	496,6
tripper.7.20.1	2,10	0,35	0,03	991,9
tripper.7.30.1	67,26	1,96	0,17	1.487,9
tripper.7.40.1	*	16,62	0,48	1.983,6
tripper.7.50.1	*	88,06	1,12	2.479,0
tripper.7.60.1	*	743,83	2,06	2.975,1
tripper.7.70.1	*	**	3,73	3.470,0
tripper.7.80.1	*	**	6,19	3.966,6
tripper.7.90.1	*	**	9,28	4.461,9
tripper.7.100.1	*	**	14,20	4.957,9
tripper.7.110.1	*	**	19,61	5.453,6
tripper.7.120.1	*	**	27,39	5.949,0

* A execução do GLPK excedeu o tempo limite.

** A execução do CPLEX excedeu o tempo limite.

*** A execução da Programação dinâmica excedeu o tempo limite.

† Nenhum método solucionou a instância dentro do tempo limite.

‡ CPLEX esgotou a memória disponível na máquina.

Instância	**Tempo GLPK** (s)	**Tempo CPLEX** (s)	**Tempo PD** (s)	**Custo ótimo**
tripper.7.130.1	*	**	36,49	6.445,1
tripper.7.140.1	*	**	47,33	6.940,0
tripper.7.150.1	*	**	64,94	7.436,6
tripper.7.160.1	*	**	80,79	7.931,9
tripper.7.170.1	*	**	102,07	8.427,9
tripper.7.180.1	*	**	127,48	8.923,6
tripper.7.190.1	*	**	160,77	9.419,0
tripper.7.200.1	*	**	191,08	9.915,1
tripper.7.210.1	*	**	237,87	10.410,0
tripper.7.220.1	*	**	280,91	10.906,6
tripper.7.230.1	*	**	329,77	11.401,9
tripper.7.240.1	*	**	398,86	11.897,9
tripper.7.250.1	*	**	454,29	12.393,6
tripper.7.260.1	*	**	542,07	12.889,0
tripper.7.270.1	*	**	642,47	13.385,1
tripper.7.280.1	*	**	731,47	13.880,0
tripper.7.290.1	*	**	833,24	14.376,6
tripper.7.300.1	*	**	943,79	14.871,9
tripper.7.310.1	*	**	***	†
tripper.11.10.1	0,02	0,02	0,00	495,9
tripper.11.20.1	6,69	0,19	0,60	991,7
tripper.11.30.1	869,77	0,77	3,29	1.487,5
tripper.11.40.1	*	13,21	10,03	1.983,1
tripper.11.50.1	*	54,23	24,55	2.478,6
tripper.11.60.1	*	212,80	48,78	2.974,1
tripper.11.70.1	*	**	89,31	3.469,5
tripper.11.80.1	*	**	144,74	3.964,7
tripper.11.90.1	*	**	224,52	4.459,9
tripper.11.100.1	*	**	320,02	4.955,0
tripper.11.110.1	*	**	449,08	5.450,0
tripper.11.120.1	*	**	608,67	5.945,9
tripper.11.130.1	*	**	815,32	6.441,7
tripper.11.140.1	*	**	***	†

* A execução do GLPK excedeu o tempo limite.

** A execução do CPLEX excedeu o tempo limite.

*** A execução da Programação dinâmica excedeu o tempo limite.

† Nenhum método solucionou a instância dentro do tempo limite.

‡ CPLEX esgotou a memória disponível na máquina.

Instância	**Tempo GLPK** (s)	**Tempo CPLEX** (s)	**Tempo PD** (s)	**Custo ótimo**
tripper.16.10.1	0,03	0,04	0,00	497,2
tripper.16.20.1	0,32	0,11	47,44	992,1
tripper.16.30.1	189,94	1,31	216,15	1.486,8
tripper.16.40.1	*	4,44	***	1.983,3
tripper.16.50.1	*	48,69	***	2.477,4
tripper.16.60.1	*	199,26	***	2.973,4
tripper.16.70.1	*	853,68	***	3.469,1
tripper.16.80.1	*	‡	***	†
tripper.24.10.1	0,04	0,05	0,01	498,1
tripper.24.20.1	0,13	0,10	315,03	992,1
tripper.24.30.1	13,84	0,32	***	1.487,9
tripper.24.40.1	428,75	4,23	***	1.983,5
tripper.24.50.1	*	64,79	***	2.477,0
tripper.24.60.1	*	115,36	***	2.974,3
tripper.24.70.1	*	381,98	***	3.467,4
tripper.24.80.1	*	627,35	***	3.964,3
tripper.24.90.1	*	**	***	†

* A execução do GLPK excedeu o tempo limite.

** A execução do CPLEX excedeu o tempo limite.

*** A execução da Programação dinâmica excedeu o tempo limite.

† Nenhum método solucionou a instância dentro do tempo limite.

‡ CPLEX esgotou a memória disponível na máquina.

Apêndice B – Resultados dos experimentos computacionais com GRASP

Tabela 11 – Resultados dos testes comparativos entre o GRASP e o resultado dos métodos exatos.

Instância	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**	**Percentual do ótimo**	**Percentual de melhores**	**Custo ótimo**
tripper.3.10.1	0,007	497,0	497,0	100,0	100,0	497,0
tripper.3.20.1	0,022	993,6	993,7	100,0	100,0	993,7
tripper.3.30.1	0,048	1.489,9	1.490,0	100,0	100,0	1.490,0
tripper.3.40.1	0,099	1.986,9	1.987,0	100,0	100,0	1.987,0
tripper.3.50.1	0,131	2.483,4	2.483,7	100,0	100,0	2.483,7
tripper.3.60.1	0,184	2.979,6	2.980,0	100,0	100,0	2.980,0
tripper.5.10.1	0,010	496,0	496,0	100,0	100,0	496,0
tripper.5.20.1	0,035	992,0	992,0	100,0	100,0	992,0
tripper.5.30.1	0,081	1.488,0	1.488,0	100,0	100,0	1.488,0
tripper.5.40.1	0,137	1.984,0	1.984,0	100,0	100,0	1.984,0
tripper.5.50.1	0,221	2.479,8	2.480,0	100,0	100,0	2.480,0
tripper.5.60.1	0,330	2.975,8	2.976,0	100,0	100,0	2.976,0
tripper.5.70.1	0,441	3.471,7	3.472,0	100,0	100,0	3.472,0
tripper.5.80.1	0,578	3.967,8	3.968,0	100,0	100,0	3.968,0
tripper.5.90.1	0,776	4.463,6	4.464,0	100,0	100,0	4.464,0
tripper.5.100.1	0,897	4.958,8	4.960,0	100,0	99,0	4.960,0
tripper.5.110.1	1,106	5.455,6	5.456,0	100,0	99,0	5.456,0
tripper.5.120.1	1,349	5.951,1	5.952,0	100,0	100,0	5.952,0
tripper.5.130.1	1,497	6.446,7	6.448,0	100,0	100,0	6.448,0
tripper.5.140.1	1,731	6.942,7	6.944,0	100,0	100,0	6.944,0
tripper.5.150.1	2,065	7.438,6	7.440,0	100,0	99,0	7.440,0
tripper.5.160.1	2,542	7.933,8	7.936,0	100,0	100,0	7.936,0
tripper.5.170.1	2,914	8.430,8	8.432,0	100,0	100,0	8.432,0
tripper.5.180.1	3,105	8.925,7	8.928,0	100,0	100,0	8.928,0
tripper.5.190.1	3,453	9.421,4	9.424,0	100,0	99,0	9.424,0
tripper.5.200.1	3,984	9.916,8	9.920,0	100,0	100,0	9.920,0
tripper.5.210.1	4,693	10.413,9	10.416,0	100,0	99,0	10.416,0

Instância	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**	**Percentual do ótimo**	**Percentual de melhores**	**Custo ótimo**
tripper.5.220.1	4,693	10.907,0	10.912,0	100,0	100,0	10.912,0
tripper.5.230.1	5,203	11.404,8	11.408,0	100,0	99,0	11.408,0
tripper.5.240.1	5,767	11.899,2	11.904,0	100,0	100,0	11.904,0
tripper.5.250.1	6,312	12.395,6	12.400,0	100,0	96,0	12.400,0
tripper.5.260.1	6,345	12.891,1	12.896,0	100,0	100,0	12.896,0
tripper.5.270.1	7,022	13.387,7	13.392,0	100,0	100,0	13.392,0
tripper.5.280.1	8,145	13.883,1	13.888,0	100,0	100,0	13.888,0
tripper.5.290.1	8,683	14.376,0	14.384,0	100,0	98,0	14.384,0
tripper.5.300.1	9,247	14.873,6	14.880,0	100,0	96,0	14.880,0
tripper.5.310.1	9,532	15.369,2	15.376,0	100,0	100,0	15.376,0
tripper.5.320.1	11,403	15.862,4	15.872,0	100,0	100,0	15.872,0
tripper.5.330.1	11,277	16.361,5	16.368,0	100,0	100,0	16.368,0
tripper.5.340.1	11,837	16.854,6	16.864,0	100,0	92,0	16.864,0
tripper.5.350.1	12,229	17.352,4	17.360,0	100,0	95,0	17.360,0
tripper.5.360.1	13,825	17.848,1	17.856,0	100,0	100,0	17.856,0
tripper.5.370.1	13,879	18.344,6	18.352,0	100,0	100,0	18.352,0
tripper.5.380.1	14,832	18.839,1	18.848,0	100,0	95,0	18.848,0
tripper.5.390.1	15,012	19.333,2	19.344,0	100,0	100,0	19.344,0
tripper.7.10.1	0,013	496,6	496,6	100,0	100,0	496,6
tripper.7.20.1	0,049	991,9	991,9	100,0	100,0	991,9
tripper.7.30.1	0,108	1.487,9	1.487,9	100,0	100,0	1.487,9
tripper.7.40.1	0,194	1.983,6	1.983,6	100,0	100,0	1.983,6
tripper.7.50.1	0,288	2.479,0	2.479,0	100,0	100,0	2.479,0
tripper.7.60.1	0,407	2.975,0	2.975,1	100,0	100,0	2.975,1
tripper.7.70.1	0,574	3.470,0	3.470,0	100,0	100,0	3.470,0
tripper.7.80.1	0,754	3.966,4	3.966,6	100,0	100,0	3.966,6
tripper.7.90.1	0,959	4.461,5	4.461,9	100,0	100,0	4.461,9
tripper.7.100.1	1,235	4.956,2	4.957,9	100,0	100,0	4.957,9
tripper.7.110.1	1,466	5.453,2	5.453,6	100,0	100,0	5.453,6
tripper.7.120.1	1,645	5.948,3	5.949,0	100,0	100,0	5.949,0
tripper.7.130.1	2,113	6.444,7	6.445,1	100,0	100,0	6.445,1
tripper.7.140.1	2,267	6.934,3	6.940,0	100,0	99,0	6.940,0
tripper.7.150.1	2,737	7.434,3	7.436,6	100,0	100,0	7.436,6
tripper.7.160.1	3,241	7.930,6	7.931,9	100,0	100,0	7.931,9
tripper.7.170.1	3,431	8.423,9	8.427,9	100,0	99,0	8.427,9
tripper.7.180.1	3,718	8.918,4	8.923,6	100,0	100,0	8.923,6

Instância	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**	**Percentual do ótimo**	**Percentual de melhores**	**Custo ótimo**
tripper.7.190.1	4,253	9.416,6	9.419,0	100,0	100,0	9.419,0
tripper.7.200.1	4,742	9.910,3	9.915,1	100,0	100,0	9.915,1
tripper.7.210.1	4,952	10.400,3	10.410,0	100,0	100,0	10.410,0
tripper.7.220.1	5,465	10.895,5	10.906,6	100,0	100,0	10.906,6
tripper.7.230.1	6,015	11.394,2	11.401,9	100,0	100,0	11.401,9
tripper.7.240.1	6,927	11.891,1	11.897,9	100,0	100,0	11.897,9
tripper.7.250.1	6,920	12.385,1	12.393,6	100,0	100,0	12.393,6
tripper.7.260.1	7,395	12.876,2	12.889,0	100,0	100,0	12.889,0
tripper.7.270.1	8,395	13.374,3	13.385,1	100,0	100,0	13.385,1
tripper.7.280.1	8,592	13.855,6	13.880,0	100,0	100,0	13.880,0
tripper.7.290.1	9,335	14.365,3	14.376,6	100,0	100,0	14.376,6
tripper.7.300.1	9,743	14.847,8	14.871,9	100,0	100,0	14.871,9
tripper.11.10.1	0,021	495,9	495,9	100,0	100,0	495,9
tripper.11.20.1	0,072	991,7	991,7	100,0	100,0	991,7
tripper.11.30.1	0,168	1.487,5	1.487,5	100,0	100,0	1.487,5
tripper.11.40.1	0,294	1.983,1	1.983,1	100,0	100,0	1.983,1
tripper.11.50.1	0,452	2.478,6	2.478,6	100,0	100,0	2.478,6
tripper.11.60.1	0,669	2.974,1	2.974,1	100,0	100,0	2.974,1
tripper.11.70.1	0,901	3.469,4	3.469,5	100,0	100,0	3.469,5
tripper.11.80.1	1,219	3.964,5	3.964,7	100,0	100,0	3.964,7
tripper.11.90.1	1,532	4.459,9	4.459,9	100,0	100,0	4.459,9
tripper.11.100.1	1,898	4.954,9	4.955,0	100,0	100,0	4.955,0
tripper.11.110.1	2,224	5.449,8	5.450,0	100,0	100,0	5.450,0
tripper.11.120.1	2,751	5.945,5	5.945,9	100,0	100,0	5.945,9
tripper.11.130.1	3,237	6.441,6	6.441,7	100,0	100,0	6.441,7
tripper.16.10.1	0,029	497,2	497,2	100,0	100,0	497,2
tripper.16.20.1	0,101	992,1	992,1	100,0	100,0	992,1
tripper.16.30.1	0,234	1.486,8	1.486,8	100,0	100,0	1.486,8
tripper.16.40.1	0,425	1.983,3	1.983,3	100,0	100,0	1.983,3
tripper.16.50.1	0,671	2.477,4	2.477,4	100,0	100,0	2.477,4
tripper.16.60.1	0,969	2.973,4	2.973,4	100,0	100,0	2.973,4
tripper.16.70.1	1,347	3.469,1	3.469,1	100,0	100,0	3.469,1
tripper.24.10.1	0,040	498,1	498,1	100,0	100,0	498,1
tripper.24.20.1	0,159	992,1	992,1	100,0	100,0	992,1
tripper.24.30.1	0,358	1.487,9	1.487,9	100,0	100,0	1.487,9

Instância	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**	**Percentual do ótimo**	**Percentual de melhores**	**Custo ótimo**
tripper.24.40.1	0,625	1.983,5	1.983,5	100,0	100,0	1.983,5
tripper.24.50.1	1,024	2.477,0	2.477,0	100,0	100,0	2.477,0
tripper.24.60.1	1,472	2.974,3	2.974,3	100,0	100,0	2.974,3
tripper.24.70.1	2,040	3.467,4	3.467,4	100,0	100,0	3.467,4
tripper.24.80.1	2,598	3.964,3	3.964,3	100,0	100,0	3.964,3

Apêndice C – Resultados dos experimentos computacionais com *Simulated Annealing*

Tabela 12 – Resultados dos testes comparativos entre o *Simulated Annealing* e o resultado dos métodos exatos.

Instância	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**	**Percentual do ótimo**	**Percentual de melhores**	**Custo ótimo**
tripper.3.10.1	32,8	497,0	497,0	100,0	100,0	497,0
tripper.3.20.1	57,7	993,2	993,7	100,0	100,0	993,7
tripper.3.30.1	80,7	1.488,0	1.490,0	100,0	100,0	1.490,0
tripper.3.40.1	100,2	1.981,9	1.986,0	99,9	0,0	1.987,0
tripper.3.50.1	120,4	2.475,3	2.479,7	99,8	0,0	2.483,7
tripper.3.60.1	138,3	2.968,5	2.974,0	99,8	0,0	2.980,0
tripper.5.10.1	46,6	496,0	496,0	100,0	100,0	496,0
tripper.5.20.1	83,1	992,0	992,0	100,0	100,0	992,0
tripper.5.30.1	115,1	1.486,8	1.488,0	100,0	99,0	1.488,0
tripper.5.40.1	144,0	1.979,1	1.983,0	99,9	0,0	1.984,0
tripper.5.50.1	177,6	2.470,4	2.478,0	99,9	0,0	2.480,0
tripper.5.60.1	209,2	2.961,7	2.971,0	99,8	0,0	2.976,0
tripper.5.70.1	239,6	3.451,8	3.465,0	99,8	0,0	3.472,0
tripper.5.80.1	273,2	3.940,9	3.955,0	99,7	0,0	3.968,0
tripper.5.90.1	307,5	4.429,2	4.444,0	99,6	0,0	4.464,0
tripper.5.100.1	334,2	4.920,0	4.941,0	99,6	0,0	4.960,0
tripper.5.110.1	366,6	5.407,8	5.425,0	99,4	0,0	5.456,0
tripper.5.120.1	394,8	5.893,9	5.914,0	99,4	0,0	5.952,0
tripper.5.130.1	428,2	6.384,1	6.415,0	99,5	0,0	6.448,0
tripper.5.140.1	461,8	6.869,0	6.900,0	99,4	0,0	6.944,0
tripper.5.150.1	483,2	7.357,3	7.386,0	99,3	0,0	7.440,0
tripper.5.160.1	503,8	7.844,6	7.873,0	99,2	0,0	7.936,0
tripper.5.170.1	535,5	8.328,8	8.355,0	99,1	0,0	8.432,0
tripper.5.180.1	565,2	8.815,5	8.852,0	99,1	0,0	8.928,0
tripper.5.190.1	596,9	9.303,1	9.337,0	99,1	0,0	9.424,0
tripper.5.200.1	627,0	9.787,6	9.826,0	99,1	0,0	9.920,0
tripper.5.210.1	659,0	10.271,8	10.306,0	98,9	0,0	10.416,0

Instância	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**	**Percentual do ótimo**	**Percentual de melhores**	**Custo ótimo**
tripper.5.220.1	689,2	10.753,5	10.791,0	98,9	0,0	10.912,0
tripper.5.230.1	719,9	11.243,0	11.295,0	99,0	0,0	11.408,0
tripper.5.240.1	750,3	11.720,3	11.758,0	98,8	0,0	11.904,0
tripper.5.250.1	783,4	12.211,8	12.261,0	98,9	0,0	12.400,0
tripper.5.260.1	812,5	12.692,2	12.754,0	98,9	0,0	12.896,0
tripper.5.270.1	844,2	13.175,4	13.234,0	98,8	0,0	13.392,0
tripper.5.280.1	875,5	13.658,9	13.705,0	98,7	0,0	13.888,0
tripper.5.290.1	907,4	14.143,9	14.201,0	98,7	0,0	14.384,0
tripper.5.300.1	936,9	14.622,8	14.689,0	98,7	0,0	14.880,0
tripper.5.310.1	969,5	15.108,4	15.158,0	98,6	0,0	15.376,0
tripper.5.320.1	999,8	15.591,1	15.658,0	98,7	0,0	15.872,0
tripper.5.330.1	1.031,6	16.069,8	16.161,0	98,7	0,0	16.368,0
tripper.5.340.1	1.062,5	16.557,1	16.641,0	98,7	0,0	16.864,0
tripper.5.350.1	1.095,2	17.035,8	17.105,0	98,5	0,0	17.360,0
tripper.5.360.1	1.124,5	17.520,6	17.584,0	98,5	0,0	17.856,0
tripper.5.370.1	1.158,0	17.995,0	18.088,0	98,6	0,0	18.352,0
tripper.5.380.1	1.187,8	18.475,5	18.546,0	98,4	0,0	18.848,0
tripper.5.390.1	1.220,8	18.961,8	19.082,0	98,6	0,0	19.344,0
tripper.7.10.1	49,6	496,6	496,6	100,0	100,0	496,6
tripper.7.20.1	92,3	991,9	991,9	100,0	100,0	991,9
tripper.7.30.1	133,4	1.487,4	1.487,9	100,0	100,0	1.487,9
tripper.7.40.1	171,4	1.981,3	1.983,6	100,0	93,0	1.983,6
tripper.7.50.1	212,6	2.471,9	2.479,0	100,0	100,0	2.479,0
tripper.7.60.1	252,6	2.962,4	2.975,1	100,0	15,0	2.975,1
tripper.7.70.1	294,1	3.451,3	3.463,0	99,8	0,0	3.470,0
tripper.7.80.1	335,1	3.940,5	3.957,6	99,8	0,0	3.966,6
tripper.7.90.1	377,2	4.428,3	4.445,9	99,6	0,0	4.461,9
tripper.7.100.1	424,7	4.916,3	4.935,9	99,6	0,0	4.957,9
tripper.7.110.1	462,9	5.402,6	5.418,6	99,4	0,0	5.453,6
tripper.7.120.1	491,7	5.889,3	5.916,0	99,4	0,0	5.949,0
tripper.7.130.1	530,8	6.375,6	6.397,1	99,3	0,0	6.445,1
tripper.7.140.1	570,4	6.862,4	6.887,0	99,2	0,0	6.940,0
tripper.7.150.1	612,4	7.345,4	7.378,6	99,2	0,0	7.436,6
tripper.7.160.1	651,6	7.830,9	7.862,9	99,1	0,0	7.931,9
tripper.7.170.1	694,4	8.315,8	8.353,9	99,1	0,0	8.427,9
tripper.7.180.1	734,2	8.800,5	8.838,6	99,0	0,0	8.923,6

Instância	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**	**Percentual do ótimo**	**Percentual de melhores**	**Custo ótimo**
tripper.7.190.1	775,4	9.284,0	9.332,0	99,1	0,0	9.419,0
tripper.7.200.1	815,4	9.766,0	9.807,1	98,9	0,0	9.915,1
tripper.7.210.1	856,9	10.253,3	10.310,0	99,0	0,0	10.410,0
tripper.7.220.1	900,3	10.732,1	10.770,6	98,8	0,0	10.906,6
tripper.7.230.1	936,5	11.217,0	11.274,9	98,9	0,0	11.401,9
tripper.7.240.1	977,9	11.698,1	11.754,9	98,8	0,0	11.897,9
tripper.7.250.1	1.019,1	12.179,4	12.229,6	98,7	0,0	12.393,6
tripper.7.260.1	1.058,7	12.664,7	12.726,0	98,7	0,0	12.889,0
tripper.7.270.1	1.103,7	13.142,1	13.199,1	98,6	0,0	13.385,1
tripper.7.280.1	1.141,7	13.628,6	13.689,0	98,6	0,0	13.880,0
tripper.7.290.1	1.184,4	14.108,0	14.180,6	98,6	0,0	14.376,6
tripper.7.300.1	1.224,5	14.591,0	14.645,9	98,5	0,0	14.871,9
tripper.11.10.1	61,4	495,9	495,9	100,0	100,0	495,9
tripper.11.20.1	123,9	991,7	991,7	100,0	100,0	991,7
tripper.11.30.1	186,6	1.487,5	1.487,5	100,0	100,0	1.487,5
tripper.11.40.1	245,7	1.982,8	1.983,1	100,0	100,0	1.983,1
tripper.11.50.1	308,1	2.476,9	2.478,6	100,0	100,0	2.478,6
tripper.11.60.1	368,5	2.969,2	2.974,1	100,0	99,0	2.974,1
tripper.11.70.1	431,9	3.459,8	3.469,5	100,0	97,0	3.469,5
tripper.11.80.1	493,4	3.947,1	3.963,7	100,0	0,0	3.964,7
tripper.11.90.1	557,3	4.437,4	4.455,9	99,9	0,0	4.459,9
tripper.11.100.1	621,3	4.924,2	4.948,0	99,9	0,0	4.955,0
tripper.11.110.1	684,3	5.408,5	5.436,0	99,7	0,0	5.450,0
tripper.11.120.1	745,8	5.895,1	5.923,9	99,6	0,0	5.945,9
tripper.11.130.1	810,9	6.379,6	6.414,7	99,6	0,0	6.441,7
tripper.16.10.1	77,1	497,2	497,2	100,0	100,0	497,2
tripper.16.20.1	164,7	992,1	992,1	100,0	100,0	992,1
tripper.16.30.1	257,4	1.486,8	1.486,8	100,0	100,0	1.486,8
tripper.16.40.1	343,8	1.983,1	1.983,3	100,0	100,0	1.983,3
tripper.16.50.1	434,3	2.476,3	2.477,4	100,0	84,0	2.477,4
tripper.16.60.1	520,5	2.972,7	2.973,4	100,0	96,0	2.973,4
tripper.16.70.1	611,3	3.465,0	3.469,1	100,0	89,0	3.469,1
tripper.24.10.1	102,8	498,1	498,1	100,0	100,0	498,1
tripper.24.20.1	228,3	992,1	992,1	100,0	100,0	992,1
tripper.24.30.1	365,7	1.487,9	1.487,9	100,0	100,0	1.487,9

Instância	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**	**Percentual do ótimo**	**Percentual de melhores**	**Custo ótimo**
tripper.24.40.1	496,9	1.983,5	1.983,5	100,0	100,0	1.983,5
tripper.24.50.1	633,8	2.476,1	2.477,0	100,0	100,0	2.477,0
tripper.24.60.1	759,7	2.973,9	2.974,3	100,0	98,0	2.974,3
tripper.24.70.1	895,3	3.467,2	3.467,4	100,0	100,0	3.467,4
tripper.24.80.1	1.022,5	3.961,4	3.964,3	100,0	92,0	3.964,3

Apêndice D – Resultados dos experimentos computacionais com *Simulated Annealing* e GRASP

Tabela 13 – Resultados dos testes comparativos entre o *Simulated Annealing* e o GRASP.

	Simulated annealing			**GRASP**		
Instância	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**
tripper.3.50.1	0,126	2.483,5	2.483,7	113,8	2.475,3	2.478,7
tripper.3.100.1	0,525	4.966,3	4.967,0	211,2	4.936,1	4.946,0
tripper.3.150.1	1,240	7.448,8	7.450,0	316,0	7.390,9	7.412,0
tripper.3.200.1	2,139	9.931,4	9.933,7	415,7	9.838,0	9.864,7
tripper.3.250.1	3,380	12.413,8	12.417,0	526,0	12.281,5	12.308,0
tripper.3.300.1	4,805	14.896,2	14.900,0	624,6	14.719,4	14.781,0
tripper.3.350.1	6,805	17.378,7	17.383,7	729,0	17.152,6	17.200,7
tripper.3.400.1	8,715	19.861,0	19.867,0	831,0	19.586,2	19.644,0
tripper.3.450.1	11,155	22.343,0	22.350,0	932,5	22.017,6	22.123,0
tripper.3.500.1	14,088	24.825,9	24.833,7	1.041,2	24.448,3	24.517,7
tripper.3.550.1	16,371	27.307,7	27.317,0	1.157,9	26.875,7	26.965,0
tripper.3.600.1	21,411	29.790,5	29.798,0	1.262,0	29.296,7	29.396,0
tripper.3.650.1	22,449	32.272,3	32.281,7	1.371,1	31.713,1	31.814,7
tripper.3.700.1	27,284	34.755,5	34.765,0	1.469,7	34.114,6	34.232,0
tripper.3.750.1	29,912	37.235,8	37.247,0	1.608,2	36.549,9	36.681,0
tripper.3.800.1	33,265	39.720,0	39.733,7	1.697,1	38.967,5	39.108,7
tripper.3.850.1	38,958	42.202,4	42.216,0	1.817,9	41.381,4	41.516,0
tripper.3.900.1	43,659	44.684,4	44.696,0	1.921,9	43.770,1	43.978,0
tripper.3.950.1	47,043	47.165,9	47.178,7	2.039,2	46.186,2	46.354,7
tripper.3.1000.1	54,546	49.648,6	49.661,0	2.135,6	48.592,9	48.816,0
tripper.5.50.1	0,227	2.479,9	2.480,0	170,9	2.471,1	2.479,0
tripper.5.100.1	0,898	4.959,2	4.960,0	329,7	4.918,7	4.936,0
tripper.5.150.1	2,010	7.438,6	7.440,0	488,0	7.355,7	7.386,0
tripper.5.200.1	3,964	9.917,9	9.920,0	653,8	9.787,4	9.827,0
tripper.5.250.1	5,778	12.394,4	12.400,0	805,7	12.205,1	12.260,0
tripper.5.300.1	8,858	14.874,9	14.880,0	974,9	14.624,1	14.683,0

Instância	***Simulated annealing***			**GRASP**		
	Tempo médio (ms)	**Custo médio**	**Melhor custo**	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**
tripper.5.350.1	11,925	17.351,4	17.360,0	1.152,5	17.037,3	17.113,0
tripper.5.400.1	15,089	19.829,4	19.840,0	1.298,1	19.440,0	19.537,0
tripper.5.450.1	20,285	22.305,7	22.320,0	1.460,0	21.847,0	21.956,0
tripper.5.500.1	24,267	24.786,3	24.800,0	1.597,2	24.243,7	24.360,0
tripper.5.550.1	28,975	27.261,2	27.280,0	1.768,1	26.637,1	26.865,0
tripper.5.600.1	35,462	29.743,2	29.760,0	1.937,6	29.018,6	29.159,0
tripper.5.650.1	44,028	32.219,2	32.240,0	2.094,3	31.415,7	31.549,0
tripper.5.700.1	51,682	34.698,2	34.720,0	2.250,2	33.804,6	33.947,0
tripper.5.750.1	54,901	37.172,2	37.199,0	2.416,4	36.191,4	36.355,0
tripper.5.800.1	62,710	39.647,9	39.676,0	2.575,8	38.540,8	38.791,0
tripper.5.850.1	73,496	42.125,6	42.160,0	2.750,0	40.929,3	41.150,0
tripper.5.900.1	83,463	44.609,8	44.639,0	2.895,8	43.305,2	43.579,0
tripper.5.950.1	94,982	47.088,5	47.120,0	3.064,2	45.677,5	45.945,0
tripper.5.1000.1	98,643	49.560,2	49.596,0	3.210,4	48.045,5	48.315,0
tripper.7.50.1	0,301	2.479,0	2.479,0	218,1	2.471,8	2.478,0
tripper.7.100.1	1,185	4.957,5	4.957,9	433,9	4.915,5	4.934,9
tripper.7.150.1	2,801	7.435,6	7.436,6	652,2	7.347,1	7.379,6
tripper.7.200.1	4,684	9.910,6	9.915,1	863,9	9.767,7	9.807,1
tripper.7.250.1	7,351	12.381,4	12.393,6	1.084,2	12.180,2	12.229,6
tripper.7.300.1	10,299	14.837,3	14.871,9	1.232,6	14.582,7	14.656,9
tripper.7.350.1	14,564	17.333,4	17.350,0	1.439,4	16.987,4	17.059,0
tripper.7.400.1	17,122	19.789,5	19.829,0	1.649,0	19.384,6	19.478,0
tripper.7.450.1	21,629	22.232,1	22.307,9	1.864,3	21.765,2	21.870,9
tripper.7.500.1	26,956	24.697,6	24.786,6	2.074,8	24.164,2	24.289,6
tripper.7.550.1	32,277	27.164,7	27.265,1	2.286,2	26.538,5	26.717,1
tripper.7.600.1	38,307	29.585,6	29.743,6	2.496,9	28.900,4	29.062,6
tripper.7.650.1	47,225	31.983,9	32.221,9	2.704,8	31.268,6	31.436,9
tripper.7.700.1	49,035	34.411,7	34.700,0	2.911,0	33.644,0	33.854,0
tripper.7.750.1	56,075	36.900,0	37.179,0	3.122,7	36.006,0	36.204,0
tripper.7.800.1	64,177	39.304,7	39.657,9	3.333,2	38.374,9	38.606,9
tripper.7.850.1	67,190	41.683,4	42.136,6	3.540,3	40.713,9	40.976,6
tripper.7.900.1	78,806	44.133,9	44.615,1	3.748,4	43.059,5	43.295,1
tripper.7.950.1	82,668	46.497,2	47.093,6	3.960,0	45.425,2	45.595,6
tripper.7.1000.1	90,919	49.016,6	49.571,9	4.170,1	47.787,1	48.094,9
tripper.11.50.1	0,470	2.478,6	2.478,6	307,7	2.476,9	2.478,6

Instância	***Simulated annealing***			**GRASP**		
	Tempo médio (ms)	**Custo médio**	**Melhor custo**	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**
tripper.11.100.1	1,873	4.955,0	4.955,0	620,3	4.922,5	4.942,0
tripper.11.150.1	4,102	7.432,1	7.433,1	934,0	7.350,7	7.392,1
tripper.11.200.1	7,703	9.908,1	9.909,9	1.249,2	9.763,0	9.815,9
tripper.11.250.1	12,313	12.381,7	12.387,5	1.571,2	12.169,8	12.240,5
tripper.11.300.1	17,616	14.859,0	14.864,7	1.892,7	14.566,7	14.640,7
tripper.11.350.1	22,739	17.327,5	17.341,7	2.227,8	16.961,0	17.042,7
tripper.11.400.1	28,755	19.787,1	19.819,5	2.550,1	19.351,9	19.438,5
tripper.11.450.1	35,377	22.259,4	22.295,9	2.881,2	21.732,8	21.842,9
tripper.11.500.1	41,485	24.655,1	24.774,1	3.212,9	24.092,9	24.233,1
tripper.11.550.1	51,867	27.104,4	27.250,0	3.551,4	26.468,3	26.638,0
tripper.11.600.1	59,542	29.439,5	29.728,6	3.968,7	28.817,1	28.989,6
tripper.11.650.1	66,931	31.923,2	32.205,0	4.321,3	31.183,5	31.353,0
tripper.11.700.1	76,559	34.266,0	34.683,1	4.803,4	33.535,7	33.754,1
tripper.11.750.1	81,026	36.413,6	37.159,9	4.969,4	35.887,8	36.119,9
tripper.11.800.1	90,568	38.818,1	39.637,5	5.179,5	38.206,3	38.478,5
tripper.11.850.1	94,006	40.496,1	42.114,7	5.510,3	40.583,4	40.815,7
tripper.11.900.1	103,017	43.709,3	44.591,7	5.840,4	42.928,7	43.209,7
tripper.11.950.1	116,972	45.753,7	47.069,5	6.174,0	45.259,9	45.569,5
tripper.11.1000.1	120,940	47.395,9	49.545,9	6.501,7	47.560,3	47.876,9
tripper.16.50.1	0,661	2.477,4	2.477,4	430,2	2.476,4	2.477,4
tripper.16.100.1	2,768	4.954,6	4.954,6	872,8	4.938,9	4.952,6
tripper.16.150.1	6,319	7.431,6	7.431,6	1.321,1	7.366,3	7.402,6
tripper.16.200.1	10,834	9.907,0	9.908,3	1.770,1	9.787,5	9.837,3
tripper.16.250.1	17,349	12.383,5	12.384,7	2.227,9	12.191,2	12.260,7
tripper.16.300.1	24,553	14.856,0	14.860,9	2.678,5	14.588,2	14.664,9
tripper.16.350.1	32,780	17.332,2	17.336,8	3.140,1	16.978,8	17.073,8
tripper.16.400.1	42,614	19.794,1	19.812,5	3.592,0	19.366,5	19.462,5
tripper.16.450.1	54,176	22.267,0	22.289,9	4.051,6	21.733,3	21.884,9
tripper.16.500.1	64,855	24.748,7	24.767,1	4.511,2	24.112,0	24.271,1
tripper.16.550.1	80,681	27.185,8	27.244,1	4.992,8	26.471,1	26.676,1
tripper.16.600.1	93,571	29.668,2	29.720,8	5.452,8	28.834,3	28.986,8
tripper.16.650.1	103,605	32.120,9	32.197,2	5.920,6	31.181,6	31.415,2
tripper.16.700.1	125,427	34.480,4	34.673,4	6.378,5	33.539,5	33.770,4
tripper.16.750.1	151,282	36.947,5	37.149,3	6.849,4	35.889,5	36.137,3
tripper.16.800.1	169,664	39.509,0	39.625,0	7.311,4	38.203,4	38.499,0

Instância	***Simulated annealing***			**GRASP**		
	Tempo médio (ms)	**Custo médio**	**Melhor custo**	**Tempo médio** (ms)	**Custo médio**	**Melhor custo**
tripper.16.850.1	180,591	41.740,7	42.102,4	7.779,4	40.549,3	40.878,4
tripper.16.900.1	189,293	44.065,6	44.579,6	8.241,2	42.878,3	43.209,6
tripper.16.950.1	207,897	46.183,9	47.056,6	8.713,3	45.223,2	45.593,6
tripper.16.1000.1	241,896	48.776,2	49.533,3	9.142,9	47.538,7	47.999,3
tripper.24.50.1	1,040	2.477,0	2.477,0	627,0	2.476,1	2.477,0
tripper.24.100.1	4,102	4.953,7	4.953,8	1.277,0	4.948,2	4.953,8
tripper.24.150.1	9,497	7.430,3	7.430,4	1.939,4	7.406,3	7.424,4
tripper.24.200.1	17,191	9.906,5	9.906,8	2.593,3	9.836,1	9.899,8
tripper.24.250.1	27,329	12.382,2	12.383,1	3.259,0	12.244,1	12.332,1
tripper.24.300.1	38,285	14.858,7	14.859,3	3.918,2	14.634,5	14.762,3
tripper.24.350.1	52,237	17.334,5	17.335,2	4.589,2	17.038,2	17.165,2
tripper.24.400.1	63,082	19.800,3	19.811,0	5.256,3	19.424,4	19.555,0
tripper.24.450.1	85,634	22.280,1	22.286,6	5.930,9	21.804,4	21.978,6
tripper.24.500.1	101,251	24.757,4	24.762,1	6.596,6	24.168,0	24.319,1
tripper.24.550.1	124,570	27.223,2	27.237,4	7.279,8	26.537,5	26.735,4
tripper.24.600.1	147,647	29.700,5	29.712,5	7.937,1	28.888,2	29.118,5
tripper.24.650.1	182,229	32.162,6	32.189,5	8.618,8	31.251,2	31.486,5
tripper.24.700.1	194,941	34.632,8	34.666,3	9.279,0	33.609,6	33.823,3
tripper.24.750.1	210,735	36.991,3	37.142,9	9.954,1	35.957,1	36.190,9
tripper.24.800.1	260,148	39.573,0	39.619,3	10.637,5	38.313,2	38.638,3
tripper.24.850.1	291,947	41.969,2	42.095,6	11.304,3	40.637,9	40.906,6
tripper.24.900.1	341,854	44.529,8	44.571,8	11.976,2	42.978,5	43.242,8
tripper.24.950.1	340,621	46.852,5	47.047,7	12.652,0	45.309,6	45.601,7
tripper.24.1000.1	368,645	49.296,5	49.523,5	13.325,0	47.639,6	47.968,5